UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Propagating information like waves in GNNs

Tobias S. Myrmel Antonsen

FYS-3941 Master's thesis in applied physics and mathematics 30 SP
June 2024

Abstract

Various deep learning architectures are appearing in the field of machine learning with the goal of being able to handle various types of data, or solving inherent problems within the networks. In this thesis, we propose the idea of creating architectures based on physics partial differential equations (PDEs), where we transfer the known properties of PDEs as a method of introducing inductive bias to the model architectures. We test this idea by comparing the oversmoothing process in graph neural networks to heat diffusion, and constructing new architectures based on the wave equation to reduce the effects of oversmoothing. The experiments suggests that the proposed architectures posses similar properties to wave propagation, implying that the idea of inheriting properties from physics PDEs is a viable method.

Acknowledgements

First, I would like to thank my supervisor for the assistance with this thesis, Benjamin Ricaud. Your indisputable excitement for deep learning, and especially graphs, has been an inspiration for me during this project. Our numerous meetings and discussions has critical for enhancing my own understanding and strengthening my excitement for this field.

Secondly, I would like to acknowledge value of my friends during my time as a student. Be it spending our time discussing machine learning, politics or some random banter, you have been critical in giving a valuable and memorable experience as a student. I would like to mainly thank Iver Nørve, Christian Salomonsen and Sigurd Hanssen for the considerable assistance and support during this thesis. And of course, I can not forget to thank my partner Sandra Walle for the unyielding support.

Contents

List of Figures

List of Tables

1 Introduction

In deep learning, the architecture of a network has a significant impact on the models performance and capabilities. Different architectures allow networks process data in unique and complex manners, allowing the models to learn complex patterns within verious types of data. Various architectures have been emerging over the years, each with different properties that allows for the creation of models specialised for different tasks. Amongst these are some renowned architectures like ResNet [15] and transformers [31]. Many architectures are constructed based on what the properties of the models should be, often based on reasoning rather than a theoretical reasoning. In this thesis, we have an idea regarding a method of constructing model architectures where we can know the properties of a model architecture before testing. With this idea, we can construct architectures tailored for whatever problem we encounter. The idea is basing the properties of a model architecture in physics. By using partial differential equations (PDEs) representing physical systems, we can create model architectures that inherit the properties of the physical system.

Graph neural networks (GNNs) [11, 30] are increasing in popularity as a powerful deep learning model for handling graph structured data. Graph data is more complex than simple table data, as the different instances in the data has some relation to each other. GNNs are capable of utilising these relations, making them very popular in various fields of science. GNN models can be used in any case where data-points has some relation to other data-points, be it some direct connection, dependency or physical adjacency. Some examples of where

GNNs are used are social media [24], chemistry [7, 18], weather prediction [20], data analysis [9] and much more. A known problem within GNNs is oversmoothing [5, 26], where the ability to handle relational information also results in loss of unique information with the increase of aggregation layers. Oversmoothing is not a bug or an unintended flaw, but rather an inherent effect that occurs with the aggregation of vertices in GNNs. When aggregation performed, each vertex combines the information of the neighbouring vertices, along with itself, and is updated. By performing multiple aggregation steps, the vertices aggregate information of a larger range within the graphs. This results in each vertex converging to a state of similarity, meaning that the vertices become indistinguishable with large amount of aggregation layers. This is an unavoidable problem in standard GNN models, as the aggregation of vertices is vital for the utilisation of the relational information in graphs.

Numerous attempts has been made to reduce the effects of **oversmoothing**. Various methods like changing the aggregation method, or the architecture of the GNNs have been attempted to solve the oversmoothing problem. Amongst these methods ideas like applying gates [21, 27], applying attention on relational embeddings [19], manipulating the weights [23], applying skip-connections [35], using differential equations [8] and many other ideas have been presented. This thesis presents the idea of using differential equations that represent physics, as an inspiration for the architectures we will propose. Applying differential equations to deep learning models is not a new concept, as it has been researched and used in many cases [8, 12, 29]. The difference is that we will use partial differential physics equations to construct architectures with an inductive bias to handle the oversmoothing problem.

In this thesis, we present the **idea of using physics PDEs** construct architectures that inherit the properties of the PDE, and therefore the properties of the physical system. We will present this idea through the oversmoothing problem in GNNs. By comparing graph convolutional network (GCN) layers to heat diffusion, we find that the properties of GNNs are similar to the properties of heat diffusion system. With this, we will establish a connection between the partial differential equation for heat diffusion with the standard GCN layer. By investigating the properties, we are inspired to create a model layer function that applies an other PDE, with different properties. The architectures we are proposing to reduce oversmoothing in GNNs based on this method are the Neural bølge operator and Bølgenet, where both are created to inherit properties from the wave equation. Neural bølge operator is directly based on the wave equation, while Bølgenet is a modified version, but is still constructed to inherit the properties. The reasoning for choosing the wave equation is the systems property of not converging to some equilibrium state, in contrast to heat diffusion. The property of waves propagating indefinitely is the inspiration to attempt to reduce the effects of oversmoothing.

We show how to design neural network architectures such that the properties can be related to physical concepts. This project thesis show that by creating model architectures directly from, or inspired by, physical systems, we can have the models inherit the properties of the systems. By doing this, we have a method of introducing inductive bias when constructing model architectures to handle specific tasks. This allows us have a theoretical background in the choice of architecture for a task, and therefore have knowledge the properties of the models before testing. We use oversmoothing in GNNs to validate the method, but this idea can be applied arbitrarily. The are numerous PDEs that can be used introduce inductive bias into model architectures by inheriting their properties.

2

Background theory

2.1 Graph data

Graphs are structures that represents sets of objects that have some relation to each other. In the field of data and machine learning, graphs is used to represent data that consisting of data-points (vertices), and the distinctive property of graphs is that the vertices have relations to other vertices. A graph is defined by these sets of vertices, where the relations (edges), are represented as connections between the vertices. An example of a graph structure is an online social media network, where a user has an account and they can follow other accounts. In this social media graph network, each vertex would represent a user account, and the relations between the users is the following. Each user has an outgoing connection to the users they follow, and incoming connections to users that follow them. This is an example of a directed graph, where the connections only goes in the direction of following, as following a user does not require them to follow you. Undirected graphs are also common, an example of an undirected graph in the same setting is a social media network where the users can add friends instead of following. This is a case where adding a friend requires them to add you as well, meaning that each connection is required to go both ways. This means that each connection is undirected. An example of an undirected social media graph can be viewed in figure 2.1

In graph theory, a graph is defined as a set of vertices V and edges E [33]. The vertices represents the objects in a graph, and the edges are the relations

between them.

$$\mathbf{G} = (V, E)$$

G is a graph, defined by V (a set of vertices) and E (a set of edges) [33]. A requirement for a graph is that all vertices has to be connected to some degree. If a set of vertices are not connected to the remaining vertices, the set is essentially its own graph.

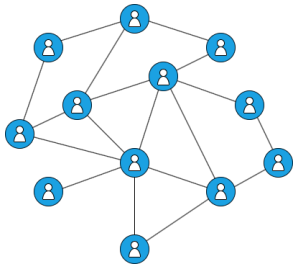


Figure 2.1: Undirected social media graph. Each vertex (blue circle) denotes an account and the edges (lines between vertices) denotes that the account are friends on the social media.

Note that visual representations of graphs are arbitrary. The only important factor is that the edges between specific vertices are conserved, they are vital to the graph structure. Graph can be illustrated in any form, including isomorphisms such as rotations or translations of any vertices.

2.1.1 The Graph Laplacian

To define the graph Laplacian, we first need to define the adjacency and degree matrix. The adjacency matrix **A** is a matrix that contains the relational information in a graph structure. The entries in the matrix are boolean values that denote which vertices are connected to each other. The adjacency matrix can also be modified to contain values between 0 and 1 to weigh the connections. For an undirected graph, all vertices are connected two ways, making the adjacency matrix symmetric. The degree matrix is diagonal matrix, created from the adjacency matrix

$$\mathbf{D}_{ii} = \sum_{j} \mathbf{A}_{ij}$$

where each entry denotes the degree of each vertex. The degree of a vertex denotes the total amount of outgoing connections a vertex has. The graph Laplacian contains structural properties of a graph, and it is calculated based on the adjacency and degree matrix.

$$\mathcal{L} = \mathbf{D} - \mathbf{A} \tag{2.1}$$

In Equation 2.13 is the definition of the graph Laplacian matrix, where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the degree matrix. The adjacency and degree matrices are calculated from the edge indices from a specific graph structure. The graph Laplacian is used for various purposes, amongst them is machine learning, where it is used to incorporate the data structure in the learning process. In Graph Neural Networks (GNNs), the graph Laplacian is used to create the Graph Convolutional Network (GCN) layer, where the GCN layers performs aggregation based on the graph Laplacian, along with the application of weights.

2.2 Neural networks

Neural networks is a concept of information processing that is inspired by the human brain [10] and is a fundamental in the field of deep learning. Neural networks consists of a network of neurons where each neuron contains parameters, and is trained to process data to perform some desired task. Neural networks can be used to perform a huge range of tasks, with many variations to the network architecture. In a simple fully-connected network, data is processed through layers that contain a matrix representing the learnable parameters, referred to as weights. Equation 2.2 represent a layer in a fully-connected neural network. The learnable parameters are the weight matrices and the biases. The weight matrices are denoted as $\mathbf{W}_i$, where i is a layer index, and biases $\mathbf{b}_i$. The data is denoted with $\mathbf{X}$ where the input data is $\mathbf{X}_0$ and the intermediate data (latent data) is $\mathbf{X}_i$ when $i \neq 0$. In a layer of the network there is an activation function σ, this function is non-linear. With a non-linear function, the network is able to learn and model more complex patterns within the data. Without non-linear activation functions a neural network would essentially behave like a linear model.

$$\mathbf{X}_i = \sigma(\mathbf{W}_i\mathbf{X}_{i-1} + \mathbf{b}_i) \tag{2.2}$$

The neural network models learn by a method called backpropagation [10]. This method is the idea of propagating the changes of parameters backwards from the last layer to the first. The backpropagation process in initialised based on a measure of how accurate the network performed. This measure is referred to as the loss, and is calculated from a loss function. The loss function compares the predicted values of the network with the true values of the training data. During the training of a network, it is normal to perform the backpropagation numerous times. Each training iteration, called epochs, the network predicts some values and the loss is calculated. Based on the loss, the parameters of the network are tuned through backpropagation.

There are numerous variations to neural networks that allow us solve a large

range of tasks, some tasks are predictions and classifications of data, image processing, language models, generative tasks, transforming of data and lots more. Some known variations that can perform some of these tasks are recurrent networks, convolutional networks, transformer networks and graph networks. This thesis uses graph neural networks (GNNs), which will be further explained in section 2.2.5.

2.2.1 Loss functions

Loss functions are the functions that calculate the measure that initialises the backpropagation method of tuning the network parameters. There are various loss functions with different properties that are used depending on the goal of the network.

Cross-entropy loss

Cross-entropy loss is a common loss function used within machine learning for multi-class classification tasks. A loss function is a function that provides the loss measure for the learning process of the model. This measure is used to initiate the backpropagation of a model in order to update the weight parameters. The loss of a model is calculated by evaluating the discrepancy between the predictions of the model and the actual targets (true labels).

$$L = -\sum_{c}^{M} y_c \log(\hat{y}_c) \tag{2.3}$$

Equation 2.3 is the cross-entropy loss function for a model with M classes. In the equation y_c represents the target at position c in the label vector for a specific data-point, and $\hat{y}_c$ represents the prediction at position c in the prediction vector. This loss function sums the loss at each position in the vector and provides a total value for the model during training.

Binary cross-entropy loss

Binary cross-entropy loss is a specific case of cross-entropy loss, where there are only two classes. Due to the fact that there are only two classes, the probability of a data-point belonging to a class has probability p, and thus the probability of belonging to the other class is $1 - p$. Using this fact the loss function for a specific data-point is defined as

$$L = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \tag{2.4}$$

This is the same function as equation 2.3, when $M = 2$, but PyTorch provides a special function for binary cross-entropy loss.

2.2.2 Activation functions

Activation functions is a critical part of machine learning algorithms as they add a non-linear capability to the neural networks [10]. There are numerous activation functions that are actively used within the field, but we will only present the ones that are used in this thesis project.

$$\sigma_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

Equation 2.5 is the **Sigmoid** activation function. It is a relatively popular activation function with a continuous derivative. This activation function scales all input values in the range $(0, 1)$.

$$\sigma_{\tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.6}$$

Tanh is the hyperbolic tangent activation function (equation 2.6). It is also relatively popular due to its continuous derivative and the range of output values. The function scales all input values to an output domain in the range $(-1, 1)$. This allows the network to use negative values in the processing.

Classification functions

Classification functions are used at the end of a network to classify data-points. There are numerous classification functions, but we will only present the ones used in this thesis project.

Sigmoid is an activation function that is also used as a logistic classification function in a binary class case (equation 2.5). The output of the function can be viewed as the probability p of belonging to a specific class, making the probability of belonging to the other class $1 - p$.

$$\sigma_{\text{softmax}}(\mathbf{x}) = \frac{e^{x}}{\sum_{j}^{K} e^{x_j}} \tag{2.7}$$

Softmax is similar to Sigmoid, but it is used for multi-class cases (equation 2.7). The output of a softmax function is a vector, with the size of the amount of classes K, where all values sum to 1. This makes the output a probability vector, where the classifications follow the position of the highest probability.

2.2.3 Optimiser and learning rate scheduler

There are many optimisers and learning rate schedulers used in machine learning algorithms, but we will only present the ones relevant for this thesis project.

The **ADAM** optimiser is an optimisation algorithm that computes individual adaptive learning rates for different weights, based on estimated of first and second order moments of the gradients [16]. This optimisation algorithm is based on two known optimisation algorithms: Adagrad and RMSProp. The name "ADAM" is an acronym for "adaptive moment estimation" [10].

Linear learning rate scheduler is a simple algorithm for scaling the learning rate during training. The linear scheduler simply changes a coefficient of the learning rate linearly throughout the training.

2.2.4 Inductive bias

Inductive bias is a concept that refers to a set of assumptions we make when constructing model architectures. This is done to generalise the architectures capabilities within specific settings. Models architectures with specific inductive biases are enabled to better handle certain types of data, resulting in improved learning and performance. An example of this are convolutional neural networks (CNNs) where the convolutions are used as an inductive bias. This inductive bias results in, amongst other things, CNNs being capable of object detection in images, independent from translation and transformation of the object or image [6].

Introducing inductive bias requires some prior knowledge of the task at hand. With this knowledge, model architectures are constructed to address specific challenges in a more generalised and effective manner.

2.2.5 Graph neural networks

Graph neural networks (GNNs) are a specialised version of neural networks, made to be able to handle graph-structured data [11, 30]. The graph data contains a set of nodes with features and a set of edges that describe the relationship between the nodes. These edges provide a new aspect to the data, compared to non-graph data. The edges provide relational information of the data structure, and that needs to be retained throughout processing. There are various methods that are used in GNNs to utilise the relational information in a graph. Methods like Graph Attention Networks (GATs) [32],

Graph Convolutional Networks (GCNs) [17], Graph Isomorphism Networks (GINs) [34], and GraphSage (Graph Sample and AggregatE) [13] are amongst the most popular methods. This thesis will use the GCN method, as it is based on the graph Laplacian, and we will use this to relate the method to PDEs. Otherwise, we use this method as it is an effective, widespread and the aggregation method is easily comprehensible. GCN layers aggregates the information of a vertex and its neighbours, in addition to applying the weights. The aggregation is also weighted with regards to the degree of the vertices. The degree of a vertex is the amount of vertices it is has edges with.

$$x_i^{(k)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \cdot \left(\mathbf{W}^T \cdot \mathbf{x}_j^{(k-1)} \right) + \mathbf{b} \tag{2.8}$$

Equation 2.8 shows the mathematical expression of a convolutional process for a specific node i and with itself and its neighbours $\mathcal{N}(i) \cup \{i\}$. The weights are represented as $\mathbf{W}$, the bias $\mathbf{b}$ and the features $\mathbf{x}_i^k$ at latent state k.

$$\mathbf{X}_i = \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \mathbf{X}_{i-1} \mathbf{W}_i \tag{2.9}$$

Equation 2.9 is the expression for the entire GCN layer in matrix form [17]. $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-connections and $\tilde{\mathbf{D}}$ is its diagonal matrix, meaning the degree matrix of the graph. The matrices $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{A}}$ is based on the graph Laplacian to perform the aggregation process [17].

Graph neural networks can be used to perform various tasks. These tasks are node classification, node regression, graph classification and edge prediction. In this thesis, we only use node classification during our experiments.

The oversmoothing problem

Oversmoothing is a known concept within the applications of graph convolutional networks (GCNs). Oversmoothing is the problem where the loss of unique vertex information and increase of similarity between the vertices. When data is processed in a GCN layer, the data within a vertex and its neighbouring vertices are aggregated, in addition to the application of the trained weights [17]. The aggregation of the vertices and their neighbours are also scaled by the amount of edges they have, meaning that the vertices somewhat experiences a weighted averaging, in each GCN layer (equation 2.8).

As described, each vertex is updated through the aggregation with its neighbours and the application of the weights. This results in each vertex containing information of the local neighbourhood after one layer. This is similar to convolutional neural networks (CNNs), where filters are applied on a node and

its neighbours, for a given range, to update the node. A major difference from CNNs is that the vertices can have from one to an arbitrary amount of neighbours. As described in section 2.1, the edges denotes a relationship between the vertices, and the application of GCN layers allows us to use this relation when processing the data.

However, this leads to the oversmoothing issue. For each GCN layer, the information within each vertex affects the update of a larger amount of vertices, which is seemingly is an advantage. The information spreading means that each vertex is aggregating more information, and due to this, the original unique information of each vertex is weakened. Additionally, as the information spreads further throughout the graph, each vertex collects information from more vertices in the graph. This results in each vertex collecting information from all other vertices at some point. When this occurs, all the information contained within the vertices become similar. By applying more GCN layers, the vertices become more similar over time and the "feature vectors on nodes in a graph go indistinguishable as we increase layers" [22].

The application of multiple GCN layers have two main effects. The first being that the vertices collect similar information as its neighbours, resulting in an increase of similarity. The second being that by aggregating information multiple times, the influence of vertex-specific information is lost due to being weakened repeatedly. This is what we call oversmoothing. It is not a special case or a bug, but rather the inherent effect of the aggregation in graph networks.

Dirichlet energy

There are various measures on oversmoothing in GNNs, but a renowned measure is the Dirichlet energy [5, 25–28, 37]. Dirichlet energy is not a perfect measure for oversmoothing, as it does not confirm whether or not a model will oversmooth or not. If the Dirichlet energy decreases throughout the network, the model will oversmooth, but if it is stable, we can not know if it will oversmooth or not. This means that the measure can be used to filter out models that will experience oversmoothing.

$$\mathcal{E}(\mathbf{X}^n) = \frac{1}{v} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} || \mathbf{X}_i^n - \mathbf{X}_j^n ||_2^2 \tag{2.10}$$

$$\mathcal{E}(\mathbf{X}^n) = \langle \mathbf{X}^n, \mathcal{L}_{\mathbf{X}} \cdot \mathbf{X}^n \rangle \tag{2.11}$$

$$\mathcal{E}(\mathbf{X}^n) = \frac{1}{2} \sum e_{ij} || \frac{\mathbf{X}_i^n}{\sqrt{1+d_i}} - \frac{\mathbf{X}_j^n}{\sqrt{1+d_j}} ||_2^2 \tag{2.12}$$

Equation 2.10, 2.11 and 2.12 all represent a measure of Dirichlet energy for a graph, where the state of the latent graph data is denoted $\mathbf{X}^n$ and a specific

vertex $\mathbf{X}_i^n$. Equation 2.10 and 2.12 sums the norm of the difference of vertex i and its neighbours $j \in \mathcal{N}_i$ squared for all vertices. Equation 2.10 normalises with the amount of nodes $v = |\mathcal{V}|$ [26–28]. Equation 2.12 works similarly to equation 2.10, but it is also capable of evaluating edge attributes, meaning that the strength of the connections are taken into account [5, 37]. Additionally, each node is scaled internally by its degree. Equation 2.11 uses the Laplacian to calculate the Dirichlet energy [25].

As presented in equations 2.10, 2.11 and 2.12, there are various methods of calculating the Dirichlet energy measure. In this thesis we will use equation 2.10 during the experiments.

2.3 Partial differential equations (PDEs)

A partial differential equation (PDE) is a multivariate equation that contains partial derivatives. PDEs are used in multiple different fields like mathematics, physics, and various engineering fields. A multivariate function takes multiple different variables as the input. A partial derivative is a multivariate function derived with respect to one specific variable.

2.3.1 The Laplacian operator in PDEs

While the Laplacian matrix is observed within graph theory, it is also seen in PDEs. Within the domain of differential equations, the Laplacian operator is the divergence of the gradient, and serves as the second spatial ($\mathbf{X}$) derivative of a system (u).

$$\mathcal{L} = \sum_i^n \frac{\partial^2 u}{\partial x_i^2} \tag{2.13}$$

In Equation 2.13 is a definition of the Laplacian operator matrix. The n-dimensional space of the system u is denoted as $x_i \in \mathbb{R}^n$.

2.3.2 Heat equation

The heat equation is a specific partial differential equation that describes the diffusion of heat in a continuous physical system.

$$\frac{\partial u}{\partial t} = \alpha \sum_i^n \frac{\partial^2 u}{\partial x_i^2} \tag{2.14}$$

Equation 2.14 shows the partial differential heat equation where u denotes a system of temperatures, $\frac{\partial}{\partial t}$ denotes the derivative with respect to time, $\sum_i^n \frac{\partial^2}{\partial x_i^2}$ denotes the second spacial derivative of n-dimensional space and α denoting the diffusivity of the medium. The equation shows how the temperature distribution u changes over time by the heat diffusion.

Discrete heat equation

By applying the Laplacian as the second spatial derivative and the approximation of the time derivative, the system can be discretised so that a numerical calculation of the diffusion process of temperature is possible. By doing this, we can calculate the discretised process of heat diffusion on a surface u.

$$\begin{aligned} \frac{\partial u}{\partial t} = -\alpha \mathcal{L}_u \cdot u[t] \approx \frac{u[t+1] - u[t]}{m} = -\alpha \mathcal{L}_u \cdot u[t] \\ \implies u[t+1] = u[t] - \gamma \mathcal{L}_u \cdot u[t] \end{aligned} \tag{2.15}$$

where

$$\gamma = \alpha \cdot m$$

In Equation 2.15, m denotes a chosen time-step for the calculation and α is the diffusion coefficient that determines how fast the diffusion can be performed through the medium. These constants are combined to a coefficient γ. The equation shows how the Laplacian operator is used to create a discrete mathematical representation of the heat diffusion process.

2.3.3 Wave equation

The wave equation is a partial differential equation that describes wave propagation in continuous physical systems. The wave equation describes the propagation of multiple types of waves such as water waves, sound waves etc.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \sum_i^n \frac{\partial^2 u}{\partial x_i^2} \tag{2.16}$$

Equation 2.16 shows the wave equation where u denotes the amplitude of each point in the system, $\frac{\partial}{\partial t}$ denotes the derivative with respect to time and $\sum_i^n \frac{\partial^2}{\partial x_i^2}$ denotes the second spacial derivative of n-dimensional space. The constant c describes the characteristics of the medium as the wave speed, as the wave speed will vary depending the type of wave and medium.

Discrete wave equation

By applying the Laplacian to the wave equation, the temporal and spatial perspectives become discretized. This allows for a numerical calculation of the states of the propagation in the discrete system. By doing this, the evolution of a system u can be calculated.

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} = -c^2 \mathcal{L}_u u[t] \approx \frac{u[t+1] - 2u[t] + u[t-1]}{m^2} = -c^2 \mathcal{L}_u u[t] \\ \implies u[t+1] = 2u[t] - \gamma \mathcal{L}_u u[t] - u[t-1] \end{aligned} \tag{2.17}$$

where

$$\gamma = c^2 \cdot m^2$$

In Equation 2.17, m denotes the chosen time-step for the approximation, and c denotes the wave speed of the specific wave type with regards to the medium. These constant are combined to a coefficient γ. The equation shows how the Laplacian operator can be applied to create a discrete form of the wave equation.

2.4 The Cora citation network

The Cora dataset is graph that represents machine learning papers [36]. Each vertex represents a specific paper, and it connects to the papers it cites, making the citations edges. There are 2701 vertices with a total of 10 556 edges. Each vertex contains boolean dictionary-style features. There are 1433 features that contains either 0 or 1, depending if a specific word in contained within the paper. The dataset is used for supervised node classification, where there are 7 classes. Each class represents a category in which the paper is contained.

Label	Category
0	Theory
1	Reinforcement Learning
2	Genetric Algorithms
3	Neural Networks
4	Probabilistic Methods
5	Case based
6	Rule Learning

Table 2.1: Classes in the Cora citation graph

Table 2.1 shows the different categories the papers are classified as. All papers

are machine learning related, so all the categories are fields within machine learning.

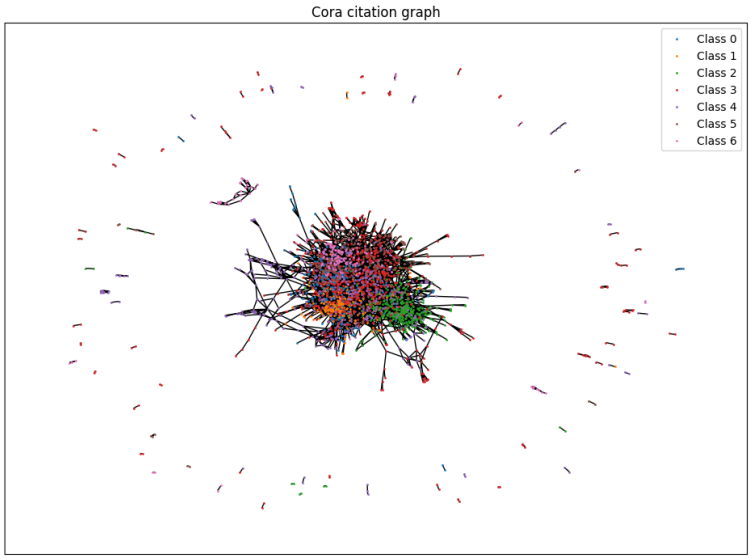


Figure 2.2: Cora citation network. Each colour denotes a specific label, the labels meaning are presented in table 2.1.

Figure 2.2 shows the citation graph from the Cora dataset. Each colour represents a label, by using table 2.1 we observe the different types of papers and their relations in the figure. The dataset has been used in different manners, resulting in PyTorch having different splits for the dataset. The splits denotes how the data has been distributed into training, test and validation. In this thesis project, the "public" split as been used. With the "public" split, the training data is a subset of the total dataset, where all classes have an equal amount of vertices.

2.5 PyTorch

PyTorch is a machine learning framework that is used in python. It is a practical tool used for the creation and usage of machine and deep learning algorithms. It offers methods in which to create networks and practical flexibility with tensors (multi-dimensional arrays). It is an open-source software released

under the modified BSD license [1].

2.5.1 PyTorch geometric

PyTorch geometric is a library built on PyTorch specified for creating neural networks that are able to handle irregular data-structures, like graph data. It includes tools for GNNs, including a specific type of GNN known as graph convolutional networks (GCNs) [2].

2.6 Declaration of previous work and AI assistance

ChatGPT has been used during the work that the author have performed. It has not been used for writing this thesis or to generate content in any manner. It has been used as a code reviewer, information source and for explanations of unfamiliar technical concepts. No information has been directly collected from ChatGPT, as it is not a reliable source of information. Any information collected has been fact-checked and sourced from original papers.

The **author** has written a project paper for UiT with similar subject matters [4]. This thesis is written with partial inspiration from the project thesis. The amount of common content should be little to none, but there is certainly an unavoidable similarity, as the author has written both within a relatively short time period.

3

Physics-inspired architectures

3.1 Motivation

The main motivation of this thesis is to investigate the idea of inheriting physical properties to solve problems in machine learning. By creating model architectures based on PDEs representing physical systems, the idea is that the models will inherit the properties of the system. By doing this we should be able to introduce inductive bias so that we can construct architectures that are tailored to handle problems we encounter.

The specific task we will be investigating to test this idea is **oversmoothing** in GNNs. Oversmoothing is an inherent effect of the aggregation property in GNNs. This does not mean that the problem is unsolvable. This thesis will present a potential solution to reduce the effects of oversmoothing in GNNs, by creating model architectures with physics as an inspiration. By creating architectures based on physics PDEs, we hope that the architectures are able to inherit the properties of the physical system. By inheriting the properties of the system, we utilise inductive bias by choosing some system that are able to handle oversmoothing better than the standard GCNs.

3.1.1 Diffusion in Graph Convolutional Networks

To establish the idea of physics properties in machine learning, we need to relate the properties of the networks to physics. In our case, we are using GNNs and we want to test if we can use this idea to reduce oversmoothing with GCN models. Partial differential equations can be used to describe the evolution of physical systems, with this in mind, we want to see if we can use a PDE to model the processing inside GCNs. Through reasoning of how oversmoothing occurs, we want to investigate the similarity between oversmoothing in GNNs and heat diffusion systems. Heat diffusion is a physical phenomenon that can be described with the use of a partial differential equation. Equation 2.14 shows the general n-dimensional representation of the PDE that governs the evolution of heat diffusion in a temperature system. Now, how does this relate to GNNs? Graphs can be represented through a matrix that describes the properties of the graph. This matrix is called the adjacency matrix. The adjacency matrix $\mathbf{A}$ describes how the vertices in a graph are connected to each other. This matrix contains all the information that is useful when handling graphs. With this matrix, we can calculate a degree matrix $\mathbf{D}$, which tells us how many connections each vertex has. With these matrices, we can calculate the Laplacian operator $\mathcal{L}$ (equation 2.13). Back to the heat equation, the Laplacian operator can also be used to represent the second spatial derivative of a system, this means that we can replace the second spatial derivative in the heat equation to create a discretised version, as shown in section 2.3.2.

Now we have the Laplacian operator in the heat equation and in graphs, but this is not enough. We want to relate the process of the information propagation in GNNs to the heat equation. By observing equation 2.15, we know that the heat equation takes a state of the system as an input, and it is used to calculate the next state of the system. This is similar to a GCN layer, where the current state of the latent data is the input to a GCN layer, and the next state of the latent data is computed (equation 2.9). The GCN layer equation uses a altered version of the Laplacian to aggregate the vertices. To perform our comparison, we need to manipulate the Laplacian. We do this by changing the Laplacian, so that is becomes similar as in a GCN layer. An important factor in GCNs, is that each vertex has a self-connection. This means that the first step is to add self-connections to the adjacency matrix, and therefore the degree matrix as well:

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$$

and subsequently, the degree matrix

$$\hat{\mathbf{D}}_{ii} = \sum_j \hat{\mathbf{A}}_{ij}$$

With the altered adjacency and degree matrices, we calculate the Laplacian

operator with self-connections:

$$\hat{\mathcal{L}} = \hat{\mathbf{D}} - \hat{\mathbf{A}} \tag{3.1}$$

The GCN layer (equation 2.9) is based on a normalised Laplacian with self-connections [17], meaning that the next step is to normalise the Laplacian:

$$\begin{aligned} \tilde{\mathcal{L}} &= \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{D}}\tilde{\mathbf{D}}^{-1/2} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2} \\ &= \mathbf{I} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2} \end{aligned} \tag{3.2}$$

The paper "Semi-supervised classification with graph convolutional networks" by Kipf et al. [17], applies this method along with other tricks to define the standard GCN layer. However, we want to show the similarity between the heat equation and the GCN layer, so we will not perform all the methods that they do. With this definition of the normalised Laplacian with self-connections, we can apply this to the discretised heat equation. We show the similarity by replacing the Laplacian in the discretised heat equation with the normalised Laplacian with self-connections:

$$\begin{aligned} u[t+1] &= u[t] - \gamma\tilde{\mathcal{L}}u[t] \\ &= u[t] - \gamma\left(\mathbf{I} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\right)u[t] \end{aligned} \tag{3.3}$$

For simplicity, we set $\gamma = 1$, then our expression starts to look familiar

$$\begin{aligned} u[t+1] &= u[t] - u[t] + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] \\ &= \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] \end{aligned} \tag{3.4}$$

Now equation 3.4 shows how GCN layers can be compared to the heat equation. We can now view this equation as the GCN layer function, where $u[t]$ is the latent input data to the layer and $u[t+1]$ is the output of the GCN layer. If we replace the heat system $u[t]$ with the latent data $\mathbf{X}_i$ and apply some weights $\mathbf{W}$, we get equation 2.9, which mathematically represents a GCN layer.

We can also **argue the similarity with reasoning**. Lets say we have a case of a graph representing some data, where all vertices contain the same data, except for one. When we apply a GCN layer, the unique vertex aggregates its information with the neighbourhood, resulting in it collecting common information and losing some of its unique information. The neighbouring vertices receives some of the unique information during the aggregation. Now we switch over to a temperature system, where we have a similar case of a surface, where all point have the same temperature, except for one position with a different temperature. This position have a noticeable different temperature than the rest, regardless of higher or lower temperature. In this example we perform the diffusion in a discretised manner. When one diffusion step occurs,

the unique position loses some of its uniqueness as the temperature spreads to its neighbours. The neighbouring positions collect some of the unique temperature. In both cases, we now have a local area that are unique to the rest of the system.

As the process continues, **the uniqueness of the original vertex and position decrease** as they are aggregated with the neighbours. After some amount of aggregation/diffusion steps, all points in the systems have received the unique data/temperature. After this point, each aggregation/diffusion step only re-scales the data/temperatures so that all entries become more similar. This is the convergence to a state of equilibrium, where all entries eventually become equal. By viewing the graph as a representation of the temperature surface, where the vertices are positions that have edges to the neighbouring positions, we observe that the cases are incredibly similar.

With this comparison we observe that **the propagation of information in GNNs is very similar to diffusion**, and with this we observe that the process of oversmoothing is very similar to the convergence to a state of equilibrium. This idea is supported by the reasoning of various research papers, where they state that the node features converge to similar values, or a state where they are indistinguishable [5, 8, 23, 26, 27], as mentioned in section 2.2.5. An illustration of the heat diffusion process can be observed in some animations generated based the discretised heat equation on the authors GitHub page [3].

We have now **explained the connection** between the physical system of heat diffusion and the GCN layers in GNNs. This means that process of information propagation in GNNs, and therefore oversmoothing, can be viewed having the same properties as heat diffusion. Now we ask the question, if a standard GCN layer can be viewed as diffusion, can we change how a layer is constructed so that it can be viewed as an other physical system? This is our motivation for this thesis.

3.1.2 GNN layers as waves

Now that GCN layers can be viewed as having the properties of a heat diffusion system, we conclude that the oversmoothing is the property of converging to some state of equilibrium. With this, we want to change the properties of a standard GNN to create an architecture that does not have this property, but rather one that does not induce oversmoothing. Whilst researching we came up with the idea of using the system of wave propagations. A large difference in a wave propagation system compared a heat diffusion system is that the waves does not converge towards an equilibrium, given that the system remains uninterrupted. The wave will start with a different amplitude than the baseline,

and the amplitude will decrease as the waves spread over a larger area, but the propagation will not stop. The waves will always propagate as long as the system is uninterrupted. This is the reason of our motivation to apply the wave equation to our problem.

To be able to inherit the properties of a wave propagation system, we need to perform the same process as with the heat equation. We need the Laplacian operator to relate the wave propagation equation to graphs. In section 2.3.3 the wave equation is presented and the application of the Laplacian operator is used to define the discretised version in equation 2.17. A noticeable difference in between the heat PDE (equation 2.14) and the wave PDE (equation 2.16) is time. In the wave equation, we are using the second derivative with regards to time, instead of the first derivative. This results in an additional time-step being used to calculate the next state of the discretised wave propagation equation (equation 2.17). However, this can be advantageous for us, let us apply the normalised graph Laplacian with self-connections to the discretised wave equation.

$$\begin{aligned} u[t+1] &= 2u[t] - \gamma\tilde{\mathcal{L}}u[t] - u[t-1] \\ &= 2u[t] - \gamma\left(\mathbf{I} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\right)u[t] - u[t-1] \end{aligned} \tag{3.5}$$

By setting $\gamma = 1$ for simplicity, we receive an equation that can inspire model layers:

$$\begin{aligned} u[t+1] &= 2u[t] - u[t] + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] - u[t-1] \\ &= u[t] + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] - u[t-1] \end{aligned} \tag{3.6}$$

Now we have equation 3.6 that represents how we can apply **GCN layers in a wave equation inspired model layer**. This is our inspiration for two model architectures that we propose in this thesis. As explained previously, we want our architecture to have similar dynamical properties that do not converge to some equilibrium. We want it to keep evolving throughout the layers. This equation inspires a model layer function that uses the current state and previous state of the latent data to compute the next. This is known as a skip-connection, where data is sent into a layer and a copy of the data is sent to the layers output, to be combined with the data that was processed in the GCN layer. This is a known method that is employed in numerous deep learning algorithms. A renowned architecture that uses this, is the ResNet [15], it is a deep convolutional network, used for computer vision tasks, that employs residual connections.

We will construct **two architectures which will employ skip-connections** in their own unique manner. As skip-connections is a known method, we want to propose an architecture that has a unique twist to the method. We want to create architectures that inherit the properties of wave propagation, and

equation 3.6 can be converted to a model layer function for an architecture that inherits the properties. Based on this, we create a model architecture named *Neural bølge operator*. We also want to create a model with a different architecture, but we still want it to have the properties. By observing equation 3.6, we observe that an aggregated version of the current state, along with the current state and the previous state, is used to calculate the next. Based on this, we modify the equation, to create an other model architecture named *Bølgenet*.

The two architectures are called Neural bølge operator and Bølgenet. Neural bølge operator is the architecture created directly from the equation 3.6, and Bølgenet will be modified to create a unique architecture that has similar properties, but in a different manner. We have named these models around the word "bølge", which is the Norwegian word for wave. We wanted to give the architectures unique names, and to add some Norwegian touch to them.

3.2 Neural bølge operator

The first architecture we propose is the Neural bølge operator, based directly on equation 3.6. We create the architecture by using this equation as the architectures model layer function. The hypothesis is that this architecture will directly inherit the properties of a wave propagation system, and due to the lack of convergence to a state of equilibrium, it should be able to handle oversmoothing in an improved manner compared to standard GCN models. The equation is modified so that it becomes a model layer function:

$$\mathbf{X}_{t+1} = \sigma(\mathbf{X}_t + \mathbf{F}_{\mathbf{W}_t}(\mathbf{X}_t) - \mathbf{X}_{t-1}) \tag{3.7}$$

Equation 3.7 is Neural bølge operators model layer function. This equation is a modified version of equation 3.6. It is modified to take in $\mathbf{X}$, which is latent data within the network. The function $\mathbf{F}_{\mathbf{W}^t}$ represents a GCN layer as in equation 3.8.

$$\mathbf{F}_{\mathbf{W}_t}(\mathbf{X}_t) = \hat{\mathbf{D}}^{-1/2}\,\hat{\mathbf{A}}\,\hat{\mathbf{D}}^{-1/2}\,\mathbf{X}_t\,\mathbf{W}_t \tag{3.8}$$

With this, we now have a model architecture based on the wave equation, and should therefore inherit its properties. For implementation, we have one issue that comes with the wave equation. As with the wave equation, this model requires two initial states of the latent data to compute the next states. In the implementation we are not able to perform the skip-connection to calculate the latent state after the first GCN layer. To handle this issue, we set the data input to the first linear layer $\mathbf{X}_0$ equal to the output of the linear layer $\mathbf{X}_1$. This is done due the fact that the input and the output of the linear layer has different embedding sizes. By doing this, we essentially remove the skip-connections as

they will negate each other after the first GCN layer, following equation 3.7. The model architecture is illustrated in figure 3.1 with this in mind, which is why the skip-connections are not illustrated after the first GCN layer.

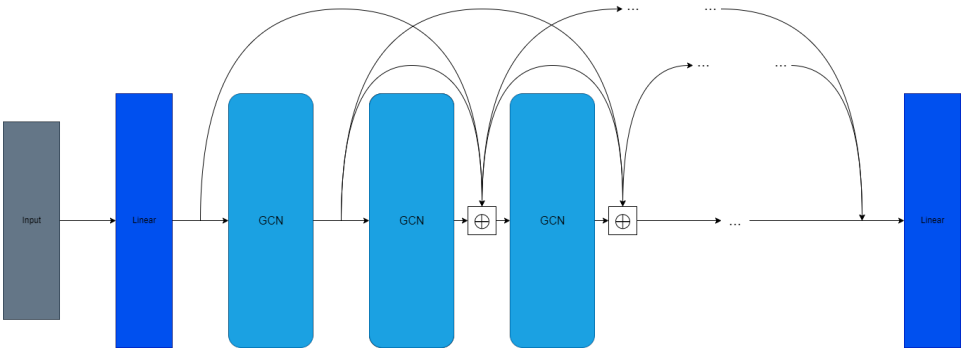


Figure 3.1: Our first proposed architecture: Neural bølge operator. This is an illustration of the the architecture. The different components are labelled within the figure. The skip-connections following the bølge GCN operator (equation 3.7) is denoted as $\bigoplus$.

Figure 3.1 is an illustration of the Neural bølge operator architecture. In the figure, we observe that the initial learnable layer is a linear layer, this is to reshape the input features to the correct embedding size. With the correct embedding size the summation is possible between all GCN layers. The last layer is a linear layer for classification purposes.

Weight matrix	Shape
$\mathbf{W}_{\text{linear},1}$	Input size $\times$ Embedding size
$\mathbf{W}_1$	Embedding size $\times$ Embedding size
$\mathbf{W}_2$	Embedding size $\times$ Embedding size
$\vdots$	$\vdots$
$\mathbf{W}_N$	Embedding size $\times$ Embedding size
$\mathbf{W}_{\text{linear},2}$	Embedding size $\times$ Class amount

Table 3.1: Parameter table for the Neural bølge operator architecture. Due to the addition and subtraction in the skip-connection, the embedding size have to remain constant.

Table 3.1 shows the dimensions for the weight matrices throughout the network, from this we observe that the total amount of parameters $M_{\text{parameters}}$ for this architecture is

$$M_{\text{parameters}} = M_{\text{input}} \cdot M_{\text{embedding}} + N \cdot M_{\text{embedding}}^2 + M_{\text{embedding}} \cdot M_{\text{class amount}}$$

3.3 Bølgenet

We have created an architecture based on equation 3.6 that should inherit the properties of the wave equation, but it is of interest to see if an other architecture can be created with the equation as inspiration. We propose a different architecture that uses the similar components as equation 3.6. Instead of summing or subtracting the data with the skip-connections, as the equation suggests, this architecture will concatenate the latent data. This results in the network having direct access to the unmodified previous states of the latent data. Equation 3.10 shows the model layer function for Bølgenet. To construct our second architecture, we need to define a variation of the GCN layer function :

$$\mathbf{X}_t = \mathbf{F}_{\mathbf{W}^t}(\mathbf{f}_{t-1}) = \hat{\mathbf{D}}^{-1/2}\,\hat{\mathbf{A}}\,\hat{\mathbf{D}}^{-1/2}\,\mathbf{f}_{t-1}\,\mathbf{W}_t \tag{3.9}$$

Equation 3.9 is a slight variation of a standard GCN layer, but is functionally the same. The input is denoted as $\mathbf{f}_{t-1}$, the only change is that $\mathbf{f}$ is a concatenated vector, from the latent vectors $\mathbf{X}_t$ and $\mathbf{X}_{t-1}$. Its components are the output of the previous layer and the output of the layer before. With this, we construct our model layer function:

$$\begin{aligned} \mathbf{f}_t &= \sigma(\mathbf{X}_{t-1} \,||\, \mathbf{X}_t) \\ \mathbf{X}_{t+1} &= \mathbf{F}_{\mathbf{W}^t}(f_t) \end{aligned} \tag{3.10}$$

The bølgeGCN equation 3.10 is the mathematical representation of the Bølgenet model layer function. The concatenation of the latent data outputs is denoted as $||$. Observe that the concatenation is only performed on the $\mathbf{X}$ vectors, which are the outputs of the GCN layers. $\mathbf{f}$ denotes the activated concatenated vectors that are the inputs to all the GCN layers, with the exception of the first GCN layer.

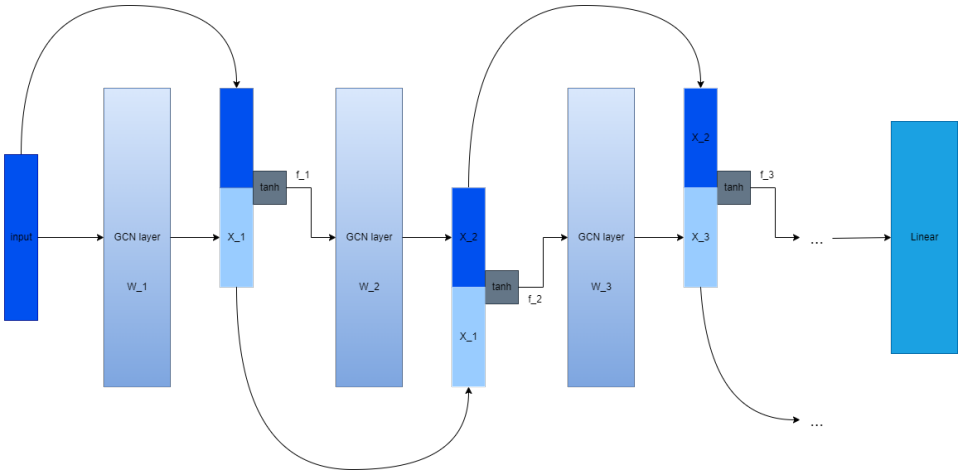


Figure 3.2: Our second proposed architecture: Bølgenet. The different components of the architectures are labelled within the figure. In the figure, the data is input to a GCN layer, where the output is concatenated with the output of the previous layers. The activation function is applied after the concatenation. Note that the skip-connection is performed before the concatenation and the activation.

The architecture is somewhat complicated, but figure 3.2 visualises the architecture. In the figure, the layers, output data and concatenation data are labelled for a clearer understanding of the architecture. Visible in the illustration of the architecture (figure 3.2), the output of each GCN layer is concatenated with the output of the previous GCN layer, as opposed to the input data. Bølgenet is constructed in this manner so that all data has to be processed through GCN layers, resulting in information not being able to skip many layers at a time. This architecture can be a parameter heavy model, given that the embedding size is rather large and constant. Given a constant embedding size, the parameters of the architecture can be viewed in table 3.2.

Wight matrix	Shape
$\mathbf{W}_1$	Input size $\times$ Embedding size
$\mathbf{W}_2$	(Input size + Embedding size) $\times$ Embedding size
$\mathbf{W}_3$	2$\cdot$ Embedding size $\times$ Embedding size
$\vdots$	$\vdots$
$\mathbf{W}_N$	2$\cdot$ Embedding size $\times$ Embedding size
$\mathbf{W}_{\text{linear}}$	2$\cdot$ Embedding size $\times$ Class amount

Table 3.2: Parameter table for the Bølgenet architecture, given a constant embedding size. Due to the concatenation, the input to each GCN layers is 2 times the embedding size, resulting in large weight matrices.

Table 3.2 shows the sizes of the weight matrices within the architecture. Based on this table, we can calculate the total amount of parameters $M_{\text{parameters}}$:

$$\begin{aligned} M_{\text{parameters}} =& M_{\text{input}} \cdot M_{\text{embedding}} + (M_{\text{input}} + M_{\text{embedding}}) \cdot M_{\text{embedding}} \\ &+ (N-1) \cdot 2 \cdot M_{\text{embedding}}^2 + 2 \cdot M_{\text{embedding}} \cdot M_{\text{class amount}} \end{aligned}$$

Note that this is significantly larger than the parameters count of Neural bølge operator, given the amount of GCN layers.

3.4 Hypothesised effect on oversmoothing with wave equation properties

Our hypothesis is that the proposed architectures are capable of reducing the effects of oversmoothing, due the inheritance of the wave propagation properties. It is uncertain if they are able to remove the effects of oversmoothing completely, or simply reduce the effects. In any case, the architectures will allow for more GCN layers to be applied in a model, resulting in models being able to handle more complex tasks. We believe that the wave equation inspired skip-connections allow the networks to reinforce the unique information of the vertices, which should result in oversmoothing being less likely. With the specific information in the vertices being reinforced, this removes that property of converging to some equilibrium state. In any case, both of these architectures are expected to handle oversmoothing better than any standard GCN models.

With the **wave equation** based architectures, we believe that they should inherit the wave propagation properties. This provides an expectation of performance, based on the properties. As described in section 3.1.2, wave propagation does not converge towards an equilibrium. This leads us to believe that the information can be propagated indefinitely, which is advantageous. However, there are drawbacks, as the waves spread in a wave propagation system, the amplitudes decrease. By relating this to information being propagated in a GNN, the uniqueness of information will decrease as the information is propagated further through the graph. This is the reason for our us not being certain whether or not the architectures will negate oversmoothing or reduce its effects.

In summary, our hypothesis is that both proposed architectures will, at least, be able to reduce the effects of oversmoothing. They should be able to retain their performance with the increase of GCN layers with a noticeable difference from traditional GCN models. The likelihood of exceptional performance in the tests seems low, but the goal of this thesis is not to provide some revolutionary

architecture. The goal is to explore the concept of relating the mechanisms of machine learning models to real-world physical systems, and to show that the idea inheriting properties can be advantageous. We hope that this idea will inspire researchers, or students, to continue exploring this. We welcome people to continue exploring this idea, by improving it or applying other physical systems to problems where the properties would be advantageous. We will discuss some ideas of further work and improvements of testing regarding our idea in the further work section (6.5).

4

Method of testing

Now we have described the theoretical background of our idea and proposed two architectures as an attempt to show its validity. We need to test Neural bølge operator and Bølgenet. To validate our idea of inheriting properties from physics, we need to perform some testing to confirm our hypothesis in section 3.4. In this thesis, oversmoothing is the problem the architectures are constructed to handle. Based on this, two experiments have been performed to test the architectures ability to handle oversmoothing. Each experiment tests the architectures in their own manner, meaning that the results of each experiment tells us different aspects of the architectures. The first experiment is performed on noise data, where we have created a graph structure consisting mostly of noise. This experiment will test the architectures capability in preserving information with regards to oversmoothing and noise. The second experiment is performed on the Cora citation network. The citation network consists of vertices that represents machine learning papers, and the edges are the citations. This experiment is performed so that we can observe the architectures capabilities in non-synthetic data and so that they can be benchmarked.

4.1 Noise graphs experiment

The first experiment is performed on a **noise dataset**, where features in almost all vertices consists of noise. Figure 4.1 shows a representation of the graph that is used in this dataset. This dataset will be referred to as the noise dataset. All

vertices in the graph contains 10 features, and all of them contain randomised values limited by the noise range, with the exception of the red node. The red node contain either 1 or -1 in all features. The goal is to train a network to propagate the information from the red node to the purple node, in the graph illustrated in figure 4.1. The models will be trained to classify the purple node, where the ground truth is the initial values in the red node. This should show the models ability to propagate the information through noise. The noise range refer to the boundaries of a uniform distribution that defines the range of the noise values. The boundaries are $0, \pm 0.1, \pm 0.2, ..., \pm 1$ and the noise range is denoted by the absolute value of the boundaries.

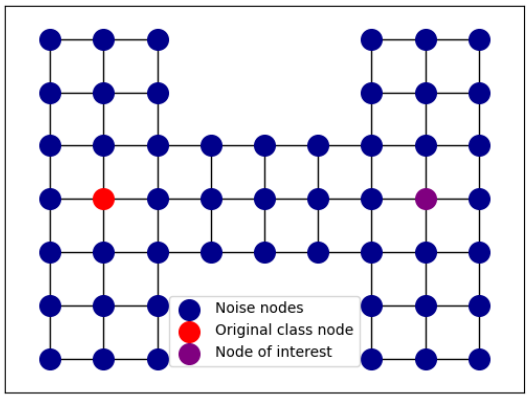


Figure 4.1: Noise graph: The graph constructed by the author to test the architectures capabilities in propagating information through noise. The noise nodes (blue) are vertices containing only noise. The original class node (red) is the node containing only 1 or -1 is all features, and the node of interest (purple) is the node we want to propagate the information to.

The goal of this experiment is to investigate the architectures ability to handle oversmoothing with noise. The information will be smoothed with noise during the aggregation, so the experiment will test the architectures ability reinforce the propagated information of interest. For a model to be able to propagate information from the red node to the purple node in figure 4.1, at least 6 GCN layers are necessary, simply due to the least amount of edges between them. Each GCN layer aggregates information of a local area, with the range of one edge. This experiment will be performed on the proposed architectures, but also on other models. These models are used to compare and benchmark the proposed models. ResGraph is a ResNet inspired architecture for GNNs and basic GCN represent a standard simple GCN model, which has the diffusion properties as described in section 3.1.1.

Data designation	Amount of graphs
Training	2000
Validation	300
Test	500

Table 4.1: Table contains the amount of training, validation and test graphs for the Noise experiment.

Table 4.1 shows the training, validation and test split of the data. To investigate the performance of the architecture properly, multiple models are trained for each noise range. To reduce the effect of the random initialisations of the data on the performance, 10 models are trained with different initializations for each noise range. This means that the data for each iteration of each noise range is randomly generated independently from the other iterations. This allows us to measure an average accuracy, and a standard deviation for each noise range.

The **parameters** need to be determined for all models. In this experiment the parameters will be equal for all models where possible. They are determined through some hyperparameter tuning, resulting in somewhat satisfactory results. The parameters are presented in table 4.2.

Parameter	Value
Epochs	250
GCN layers	8
Learning rate	0.001
Start factor	1
End factor	0.001
Embedding size	25
Batch size	15

Table 4.2: Training parameters for the noise experiment. These parameters are used for all models, where possible.

This is a short **explanation of the parameters** provided for the Noise experiment in table 4.2. GCN layers refers to the amount of GCN layers used in the models. Learning rate is the initial learning rate, however the learning rate is modified by a linear learning rate scheduler, where the start and end factors are provided in the table. Batch size is the amount of graphs being evaluated for the loss calculation for each back propagation. Note that it is not viable to perform actual batching in a graph network where there is only one graph, but for this experiment we have multiple graphs as the training data. This means that for this experiment, batch size refers to the amount of graphs that are

being evaluated at once during the training.

4.2 Cora experiment: The Cora citation network

As described in section 2.4, the Cora citation network consists of 2708 vertices, where each vertex represent a paper, and the edges are the citations. There are 7 labels that categorise the papers in fields within machine learning. The Cora dataset is quite popular within graph neural network related research papers.

The reason for this experiment is to test our architectures on a renowned dataset. This experiment allows us to analyse the performance of the proposed architectures on some real-world data. With this experiment, the architectures can be compared to other models that are attempting to handle the oversmoothing problem. As this thesis is not attempting to solve the oversmoothing problem, the comparisons will be a small part of the discussion (section 6.3).

The **setup for the experiment** is as follows, the training is done with the 'public' split of the Cora dataset. This means that the training data has equally split the 7 classes, while the rest is not balanced. Additionally, the 'public' split also reduces the training data, meaning that there is a lot of unused data during the training. All three models are trained with the same parameters. To induce oversmoothing, the models are trained in multiple instances with an increasing amount of layers. The reasoning for this is that by applying multiple GCN layers we induce oversmoothing, as described in section 2.2.5. The models are trained with $2, 4, 8, 16, 32, 64$ and 128 GCN layers. Note that we are not counting the linear layers.

The **parameters** are determined for all models. They are determined through some hyperparameter tuning, resulting in somewhat satisfactory results. The explanations of the parameters are the same as the Noise experiment, provided in section 4.1.

Parameter	Value
Epochs	1000
Learning rate	0.001
Start factor	1
End factor	0.001
Embedding size	50
Batch size	35

Table 4.3: Table containing the training parameters for the Cora experiment for all models.

Dirichlet energy of the models

As described in section 2.2.5, Dirichlet energy is a renowned measure used to quantify oversmoothing. We will calculate the Dirichlet energy for the last layer of each model for the Cora experiment. With this we can investigate if the models follow the same conclusions as other research papers. The experiment is simply collecting the latent state of the data at the end of each model, before the classification. The Dirichlet energy will be calculated based the method in "A survey on oversmoothing in graph neural networks" by Rusch et al. [26] where we calculate the log of the square root of equation 2.10.

4.3 Comparing to other architectures

As previously mentioned, we will **compare our architectures to other models** in both experiments. To properly investigate the performance of the proposed architectures, it is interest to observe how our architectures performs with regards to other architectures. This is the reasoning for training other architectures along with our wave equation inspired architectures. These architectures are a ResNet inspired architecture, referred to as ResGraph, and a standard GCN. The ResGraph architectures is shown in figure 4.2 and the standard GCN in figure 4.3.

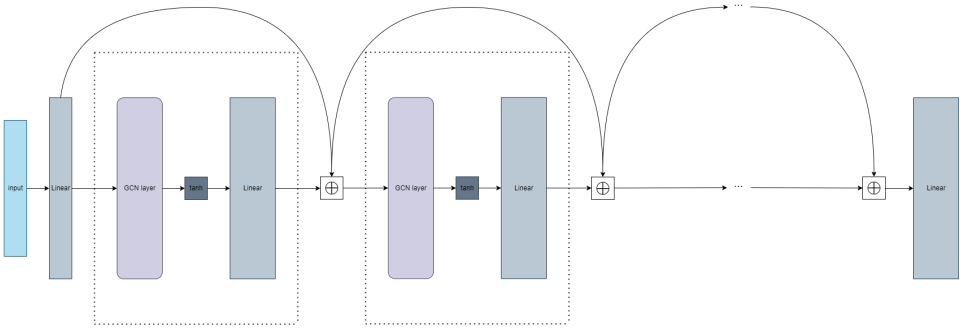


Figure 4.2: ResGraph: A ResNet inspired GNN architecture. This architecture contains blocks, represented by the dotted lines, where each block contains a GCN layer and a linear layer. The skip-connections are performed over each block and the latent data is summed at the connection points, denoted as $\bigoplus$. In each connection point and block, tanh is used as the activation function. Note that the data sent through skip-connections are before the activation functions.

The ResGraph architectures starts with a linear layer to transform the data into the correct embedding size. The other trained architecture is a basic GCN model. It is simply a straight forward network with no skip-connections. An illustration can be viewed in figure 4.3.

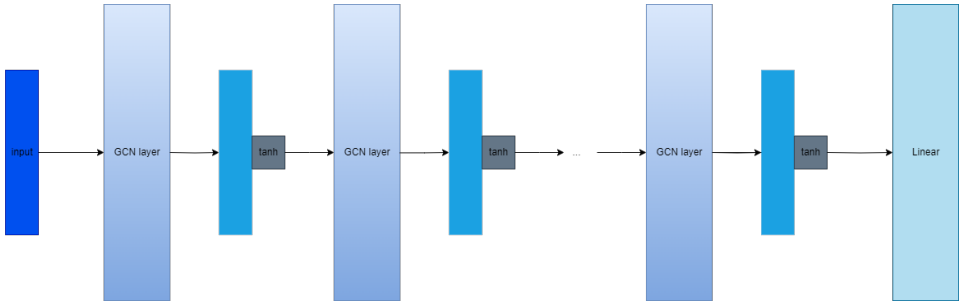


Figure 4.3: Standard simple GCN architecture. The layers are simply chronological with no skip connections, each layer consists of a GCN layer and a tanh activation. The network ends with a linear layer for classification purposes.

ResGraph is the ResNet [15] inspired architecture, and it employs skip-connections similar to our proposed architectures. The main difference between ResGraph and Bølgenet is the manner of which the skip-connections are performed. The skip-connections in Bølgenet is performed by concatenation, while the ones in ResGraph is performed with addition. The concatenation should allow the network access to unmodified previous states of the latent vertices, while ResGraph transforms the vertices with addition. Neural bølge operator differ from

ResGraph by also subtracting the latent state before, other than this, they are very similar.

5 Results

This chapter presents the results of the two experiments: The Noise experiment and the Cora experiment. Both experiments are performed to test whether or not the proposed model architectures have inherited the properties of wave propagation. This will be observed through the architectures capabilities in handling oversmoothing.

5.1 Noise experiment results

As described in section 4.1, this experiment is performed on graphs containing noise at various ranges. The goal of this experiment is to test the capabilities of the model architecture and its resilience to noise. To properly test this, 10 models are trained for independently for each noise range. With this we can investigate the mean accuracies along with the maximum, minimum and standard deviation. Tables containing detailed results for Neural bølge operator and Bølgenet is provided in the appendix (section 8.1).

5.1.1 Neural bølge operator model

Neural bølge operator is one of the architectures we have proposed in this thesis. The architecture is created directly from the wave equation based model layer function 3.6. The purpose of this architecture is to have inherited properties from wave propagation, so that it can reduce, or negate, the effects of oversmoothing in GNNs. The model architecture is presented in section 3.2.

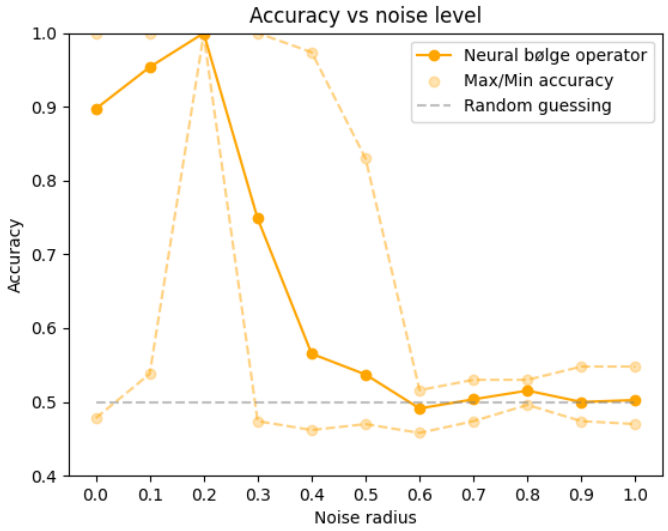


Figure 5.1: Neural bølge operator: Average accuracy per noise range. The dotted lines above and below the average accuracy line denotes the maximum and minimum accuracy achieved in the 10 iterations of each noise range, respectively.

The noise experiment results for the Neural bølge operator model is fascinating, as the architectures performance in clearly declining with the increase of the noise range. Figure 5.1 shows the results of the experiment. In the figure, we notice the swift decline of the average accuracy with regards to the noise range. Initially, it learns in most cases, as the average accuracy is higher than 85%. However, the minimum accuracy for the lowest noise ranges is very close to random guessing, meaning that some iterations were not able to learn. All 10 iterations at noise range 0.2 seems to have learned perfectly, but as the noise range increases, it quickly loses its performance. We observe that the average accuracy swiftly decreases after noise range 0.2, but due to the maximum accuracy, some of the iterations are able to learn up to noise range 0.5. After noise range 0.5, it seems that none of the iterations are able to learn at all. The specific metrics for this plot can be viewed in table 8.1.

5.1.2 Bølgenet model

Bølgenet is the second architecture based on the wave equation. This model is not directly created from equation 3.6, but rather inspired by it, and constructed in a manner where it should inherit the wave propagation properties. The architecture is created for the same purposes as the Neural bølge operator architecture. The Bølgenet architecture is presented in section 3.3.

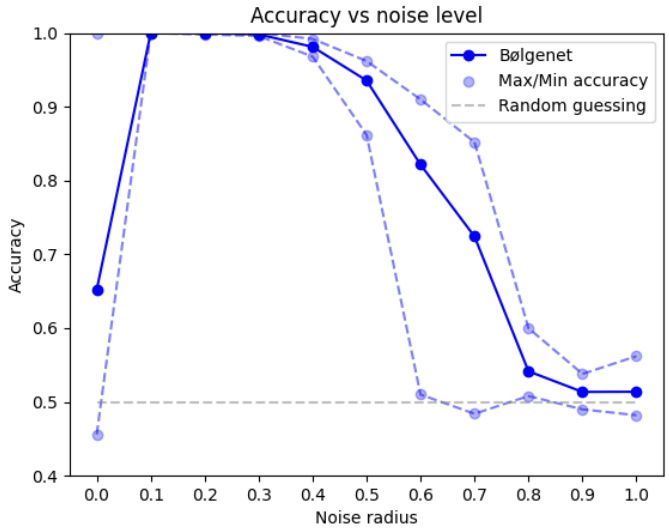


Figure 5.2: Bølgenet: Average accuracy per noise range. The dotted lines above and below the average accuracy line denotes the maximum and minimum accuracy achieved in the 10 iterations of each noise range, respectively.

Figure 5.2 shows the average accuracy per noise range along with the maximum and minimum accuracy. Bølgenet seems to be performing well overall, however it still experiences a decline towards the higher noise ranges. Interestingly, Bølgenet is having trouble with no noise (noise range 0). Based the maximum accuracy, some models are able to learn, but most are not due to the low average. The models are able to learn well up to noise range 0.5, where some instability occurs. By observing the maximum and minimum accuracy of the iterations, we observe that after noise range 0.5, some models are still able to learn, but the amount that are not able to learn at all is increasing. Above noise range 0.8, all models seem to be unable to learn. Specific values that are used to generate this plot can be viewed in table 8.2.

5.1.3 ResGraph model

ResGraph is the model architecture inspired by ResNet, as described in section 4.3. We have trained ResGraph so that we can compare the the performance of the proposed architectures to other known architectures that employ similar methods. ResNet employs a renowned method of skip-connections [15], and by converting it to GNNs, we have ResGraph.

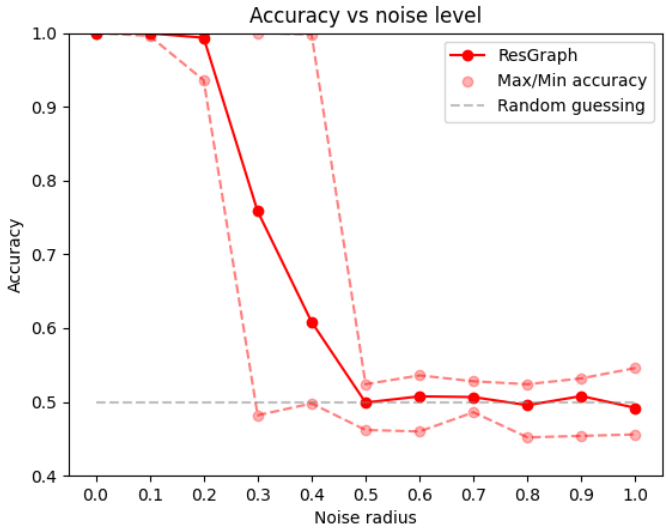


Figure 5.3: Resgraph: Average accuracy per noise range. The dotted lines above and below the average accuracy line denotes the maximum and minimum accuracy achieved in the 10 iterations of each noise range, respectively.

Figure 5.3 shows the mean accuracy per noise range, along with maximum and minimum accuracies. ResGraph is able to learn very well for the noise ranges up to 0.3, where some instability starts occurring. Based on the maximum accuracies, some of the iterations are able to perfectly learn up to noise range 0.5, but from there, no iterations are able to learn at all. From noise range 0.3 to 0.5, the average accuracy swiftly declines, making this a the limit for ResGraphs ability to learn on the noise graphs.

5.1.4 Basic GCN model

The basic GCN model is a traditional simple GCN architecture. The structure of the architecture is presented in section 4.3. We test this architecture to be able to compare the performance of our proposed architectures with regards to to the standard GCN architecture so that we can observe that different properties of the model architectures. Due to our architectures theoretically having the properties of wave propagation, and the standard GCN having the properties of heat diffusion, the standard GCN should be more susceptible to oversmoothing due to the difference in properties as described in section 3.1.1 and 3.1.2.

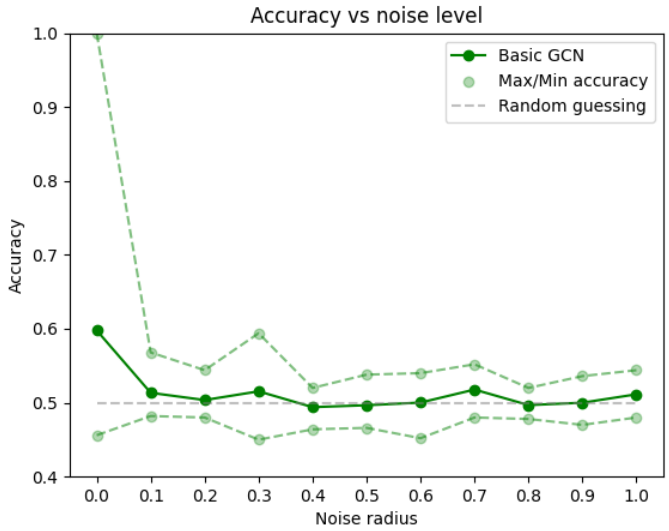


Figure 5.4: Basic GCN model: Average accuracy per noise range. The dotted lines above and below the average accuracy line denotes the maximum and minimum accuracy achieved in the 10 iterations of each noise range, respectively.

Based on figure 5.4, we observe that the basic GCN architecture is not able to learn for almost all noise ranges. The only exception is noise range 0, where the maximum accuracy tells us that at least one iteration is able to learn perfectly. But in all other noise ranges, the average, maximum and minimum accuracies are close enough random guessing that we assume that the deviations are simply due to random initialisation.

5.2 Cora experiment results

This section presents the second experiment, performed on the Cora citation network. The experiment is presented and described in section 4.2. We have trained 7 models for each architecture, and recorded the performance for each model. Each model has a different amount of GCN layers, and with the increase of GCN layers, we induce oversmoothing. Specific accuracy measures can be found in the appendix, in table 8.3 in section 8.2.

5.2.1 Neural bølge operator

Neural bølge operator is the proposed architecture that is directly created from equation 3.6. The accuracy for each model is shown in figure 5.5, where the horizontal axis represents each model with the amount of GCN layers within the trained model. The vertical axis represents the accuracy on the test data post training. Based on the results, the Neural bølge operator models with a low amount of GCN layers are performing adequately, and as the amount of GCN layers increase the performance declines. However, after 16 GCN layers, the performance is stabilising. This implies that the architecture is able to retain the performance to some degree.

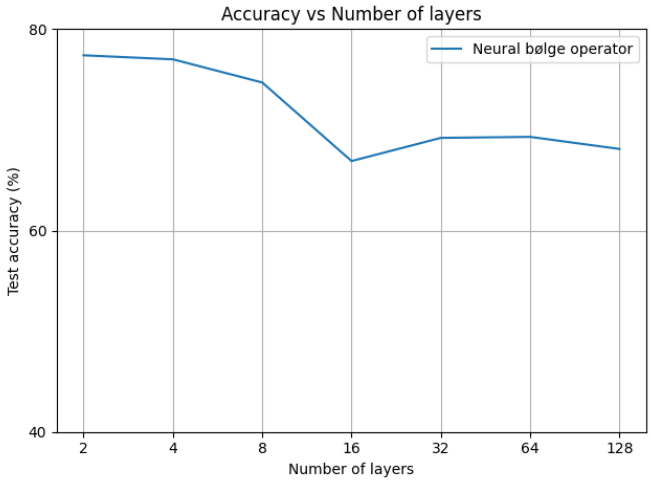


Figure 5.5: Neural bølge operator: Accuracy vs amount of GCN layers. Each tick on the horizontal axis represents a model with the amount of GCN layers given by the value.

We can observe the loss curves for each of the Neural bølge operator models in figure 5.6. The horizontal axis represents the epochs and the vertical is the calculated cross-entropy loss. We observe that all models are able to learn, but

the models with higher amount of GCN layers generally have higher loss values than the models with a lower amount of layers. We also observe the instability of the loss curve for the model with 128 GCN layers, this is likely due to the more complex loss surface with the increase of parameters.

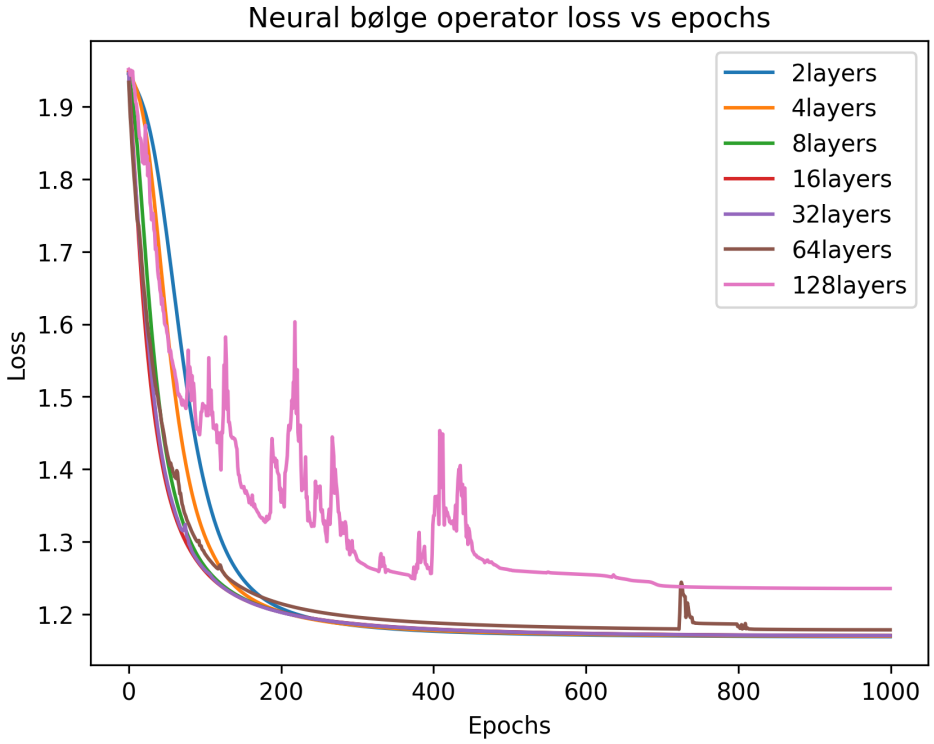


Figure 5.6: Loss curves for each neural bølge operator model. Each model is denoted by the amount of GCN layers in the figure.

5.2.2 Bølgenet model

Bølgenet is the second proposed model, inspired by the wave equation. Figure 5.7 shows the accuracy for each model. We observe that the performance is generally decreasing as the amount of GCN layers increase. The architecture is still able to retain a relatively high accuracy with the increase of the amount of GCN layers, meaning that it is able so handle oversmoothing somewhat.

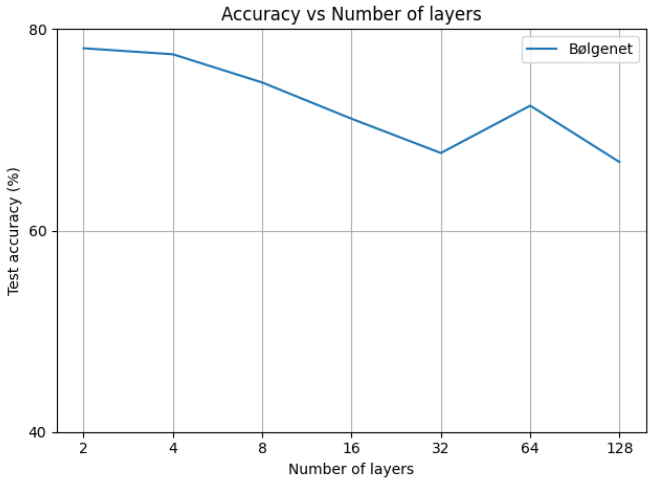


Figure 5.7: Bølgenet: Accuracy vs amount of GCN layers. Each tick on the horizontal axis represents a model with the amount of GCN layers given by the value.

Figure 5.8 shows the loss curves for each model with regards to the epochs during training. We observe similar results to the Neural bølge operator models from figure 5.6, where the models with a higher amount of GCN layers generally have higher loss values. We also observe the instability of the same models, related to the more complex loss surface.

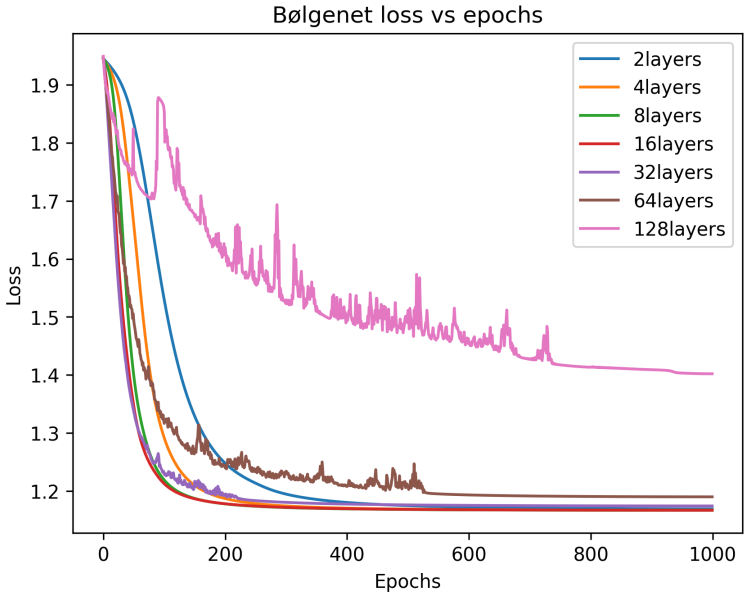
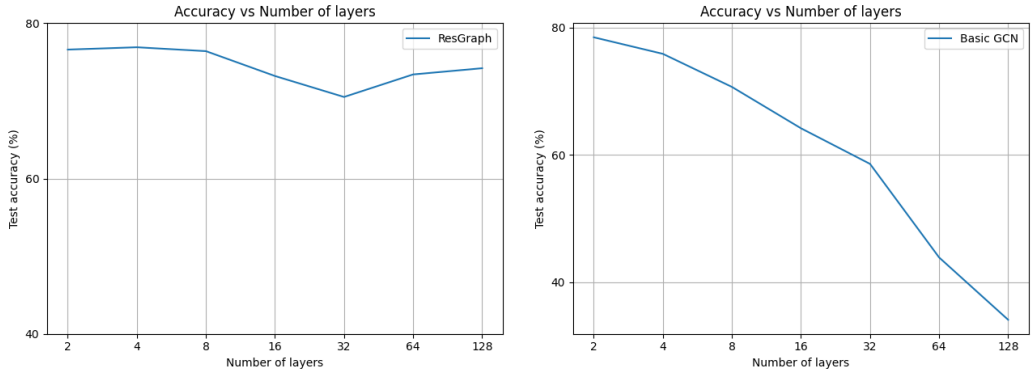


Figure 5.8: Loss curves for each Bølgenet model. Each model is denoted by the amount of GCN layers in the figure.

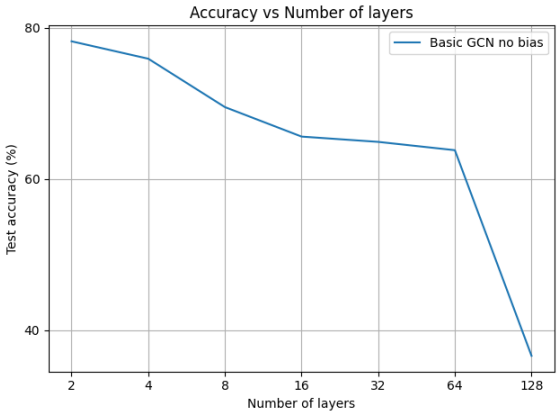
5.2.3 ResGraph and basic GCN model

As previously described, the ResGraph and the basic GCN models are trained only for the purpose of comparison to the Neural bølge operator and Bølgenet models. For this experiment, we have also trained a basic GCN model without a bias term. The reason for this is that some research papers investigate the effect of the bias term on oversmoothing and the Dirichlet energy [26].



(a) ResGraph: Accuracy vs amount of GCN layers

(b) Basic GCN: Accuracy vs amount of GCN layers



(c) Basic GCN with no bias: Accuracy vs amount of GCN layers

Figure 5.9: Test accuracy for each model for ResGraph (top left), the basic GCN model (top right) and the basic GCN model without bias (bottom). Each tick on the horizontal axis represents a model with the amount of GCN layers given by the value.

Figure 5.9 contains the accuracy for each model for the ResGraph, Basic GCN and Basic GCN without bias. Sub-figure 5.9a shows the accuracy for the ResGraph models. We observe that the models are able to retain their performance

with the increase of the GCN layers similarly to Neural bølge operator. The performance decline with the increase of GCN layers, but stabilise somewhat for the higher amounts. We observe from sub-figure 5.9b and 5.9c that the performance for both Basic GCN architectures are decreasing as the amount of GCN layers increase. The performance for the Basic GCN models seems to decline slowly. The basic GCN with bias seem to be declining slower than the one without bias, until the model with 64 layers. The decrease of performance between 64 and 128 for the basic GCN model without bias is very significant.

5.2.4 Dirichlet energy

Dirichlet energy is a common measure used as an attempt to quantify oversmoothing within GNNs. This has been described in section 2.2.5. Dirichlet energy is not a perfect measurement for oversmoothing. Based on the paper "A survey on oversmoothing in graph neural networks" by Rusch et al. [26], the persistence of Dirichlet energy with the increase of GCN layer is not informative, but rather an exponential decrease. If the Dirichlet energy were to exponentially decrease with the increase of GCN layers, we can expect oversmoothing, but without any exponential decrease, oversmoothing is uncertain.

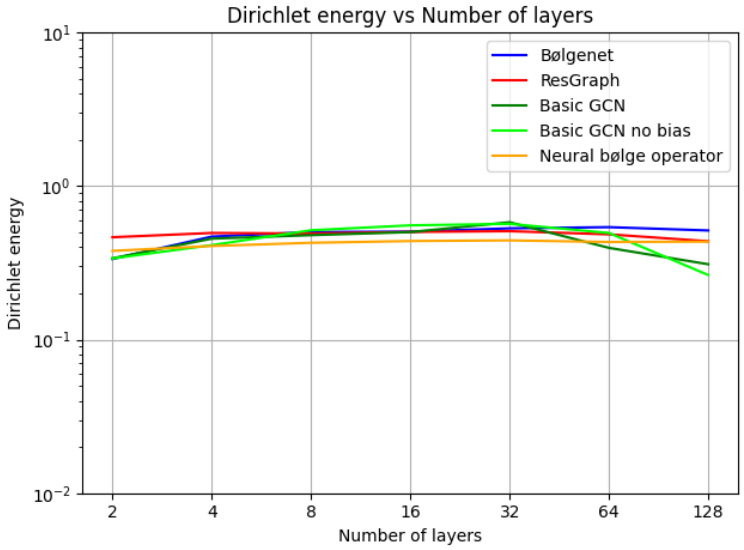


Figure 5.10: Dirichlet energy for the output of the last linear layer of each model. The horizontal axis denotes the total amount of GCN layers in a model. It is clear that all of the models are able to retain the Dirichlet energy measure, but both basic GCN models are decrease for the higher GCN layer amounts.

Our results from figure 5.10 is not very informative, as none of the architectures are decreasing in any significant manner. We note that the Dirichlet energy for the basic GCN models are decreasing slightly towards the higher amount of GCN layers. The results are contradicting the results of the basic GCN models in the research paper by Rusch et al. [26], and we will discuss the difference of results in section 6.1.1.

6 Discussion

To properly evaluate the validity of our idea of inheriting properties for physics into model architectures, we need to test the proposed architectures and investigate whether or not they are working as intended. In these sections we will discuss and evaluate the capabilities of Neural bølge operator and Bølgenet, and conclude whether or not our idea of inheriting properties from physics PDEs is valid.

6.1 The performance of Bølgenet and Neural bølge operator

Based on the results presented from both experiments (section 5.1 and 5.2), the Neural bølge operator and Bølgenet architectures are behaving as expected. They are able to reduce the effects of oversmoothing, but the performance is not being retained in an optimal fashion. The results currently imply that the architectures are working to some degree, which supports our hypothesis in section 3.4. We will discuss the results in the following sections.

6.1.1 The experiments

This section contains discussion of the experiments with regards to performance of the proposed architectures and comparisons to the other architectures. We will discuss and speculate as to why the proposed architectures are performing as presented in the results of the experiments. The experiments are performed as described in section 4.1 and 4.2.

Evaluating the Noise experiment results

The goal of the Noise experiment is to test the architectures capabilities in handling oversmoothing and noise. With the goal of propagating information from the red vertex to the purple vertex in the graph structure shown in figure 4.1. In this graph structure, oversmoothing should occur for any traditional graph neural network.

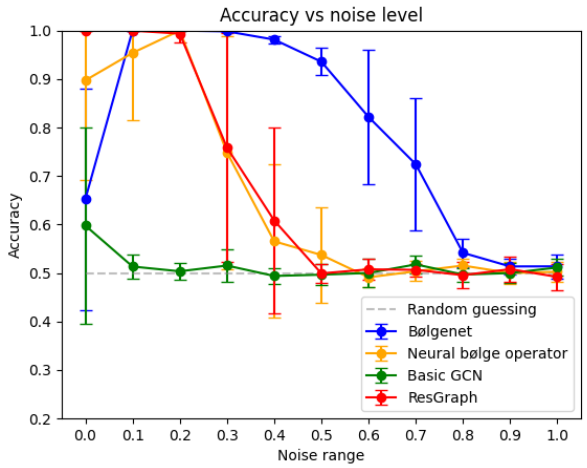


Figure 6.1: Noise experiment results for all models. Each architectures average accuracy and standard deviation per noise range. Neural bølge operator (yellow) and Bølgenet (blue) are the models with our proposed architectures, ResGraph (red) is the ResNet inspired architecture and Basic GCN is the traditional GNN (green). The vertical lines represent the standard deviation from each average accuracy. Observable, Bølgenet is able to retain performance longer than all other models. Neural bølge operator and ResGraph is able to retain performance for lower noise ranges, while the basic GCN model is not able to learn at all.

Neural bølge operator and Bølgenet is seemingly performing as expected. Section 3.4 describes our hypothesis that the Neural bølge operator and Bøl-

genet architectures will be able to reduce the effects of oversmoothing, and not necessarily completely negate the problem. Figure 6.1 seems to support this hypothesis. In the figure it is clear that Bølgenet is performing better than the other architectures. Bølgenet and ResGraph is performing similarly for noise range 0.1 and 0.2, but with the increase of the noise range, the performance of both models decrease. However, this is more apparent for the ResGraph models. From noise range 0.2 and further, ResGraph and Neural bølge operator is performing very similarly. Both lose accuracy very quickly, and converges to random guessing from noise range 0.5. Neural bølge operator and ResGraph is losing accuracy much faster than Bølgenet. Bølgenet is able to retain some performance for higher noise ranges than both of them, which implies better robustness to oversmoothing and/or noise. Noticeably, all architectures seems to be unable to learn when the noise ranges are close enough to the class values. This is likely due to the noise values being able to contain values very close to -1 and 1, which makes it difficult for the networks to learn which information to propagate. This makes it very difficult for the models to learn which vertex they should propagate information from, as the noise is too strong, or similar, to the values of the initial vertex.

The **basic GCN** is performing poorly for all noise ranges, the average accuracy is quite close to random guessing in almost all instances. This is due to the effect of oversmoothing. With the instances being so close to random guessing, with only slight deviations, we assume that the deviations occur due to random initialisation of the weights. With this in mind, we assume that the models have failed to learn in these instances. When the noise range is zero, the standard deviation of the accuracy is high. Based on this and the maximum accuracy from figure 5.4, some models are actually able to learn. We assume that this is due to the lack of noise to interfere with the propagation of information. If all blue vertices in figure 4.1 contain zeros, and the red vertex contains something different from zeros, the purple vertex only needs to receive some information different than zero. Due to the classes being -1 and 1, the classification might only depend on the sign of the values received in the purple vertex, making the classification possible even if the values are small.

A **curious occurrence** is the difference between the Neural bølge operator, ResGraph and Bølgenet at noise range 0. ResGraph is performing perfectly, while Bølgenet and Neural bølge operator are having trouble with learning. For Bølgenet, we assume that this is related to the higher parameter count and how the skip-connections are performed. Due to the nature of the Bølgenet architecture, the amount of parameters is rather large compared to the other architectures. Due to the high parameter count, the loss surface is more complex, resulting in a more difficult learning process. Because of the more complex loss surface, Bølgenet is likely able to achieve better performance at noise range 0, given more training and improved tuning of the training parameters. We believe that

the ResGraph has an easier time learning at this noise range due to the nature of its residual connections. Due to the fact that the data is summed, rather than concatenated, the spread of positive or negative values are reinforced for each skip-connection. Due to the lack of noise, there is nothing impeding the propagation of information. As mentioned, the Neural bølge operator models are also having some trouble with learning at noise range 0. This is also likely due to the nature of the skip-connections. The skip-connections for the Neural bølge operator architecture follows equation 3.7. In this equation, we observe the addition of the current latent state, but also the subtraction of the previous latent state. This is likely causing problems for the models in this experiment, as the skip-connections contain the addition of the current state and subtraction of the previous state. When the information initially reaches a vertex, the skip-connections will have no effect at noise range 0, as the previous state of the vertex is only zeros. After one more layer, the skip-connections are able to reinforce the information that has been propagated to it. The problem arrives when even more layers are performed, as the subtraction of the previous state will invert the signs of the propagated values. This can have a negative effect on the propagated information as this is performed multiple times in this experiment. The scale of this problem is uncertain, but it likely has an effect on the training at noise range 0. This, in addition to the adopted wave equation property of needing two latent data states to perform the skip-connections, can impede the models ability to propagate information. This inherited wave equation property was presented as an issue in section 3.4, and this means that for the initial layers of the architecture, the models will behave as basic GCN models. By not being able to reinforce the propagated information early, the propagation of the information will likely be slowed.

We reason that **reinforcement of the vertex of origin** has a noticeable impact when we employ the skip-connections, at least for noise range 0. Due to the fact that we use 8 GCN layers in all models, the necessary information is not required to be sufficient when it is first propagated to the vertex of interest (after 6 GCN layers). The necessary information can be propagated to the vertex at GCN layer 7 or 8. This means that being able to reinforce the necessary information in the vertex of origin results in it being able to propagate the information with a greater strength. A visualisation of this would be a heat diffusion system, where a point has a higher temperature, but it does not change. It remains constant through the process, while the diffusion process is still occurring. It is important to note that this seems to only be beneficial in a case where there is no noise. In the other noise ranges, ResGraph is performing approximately equally to the Neural bølge operator and worse than Bølgenet.

The **summary of the Noise experiment** is the Bølgenet is able to handle oversmoothing and noise data equally or better than all the other architectures, for all noise ranges other than 0. This is likely due to the nature of

its skip-connections: concatenation, where the propagated information is not transformed by skip-connections. Neural bølge operator is performing approximately equally to ResGraph for all noise ranges, with the exception of noise range 0. Noise range 0 seems to be a special case, where ResGraph is performing perfectly, while the other models are having trouble. The performance of this case is not as interesting, as having no noise in data is very unrealistic, but is still relevant to investigate as we can extract the behaviour of the architecture from analysis. The results of our proposed architectures has generally not been surprising with regards to our hypothesis, meaning that the results of this experiment is supporting the idea of that the proposed architectures have inherited the properties of the wave equation.

Cora experiment results evaluation

The Cora citation network is a commonly used dataset within GNN research papers, however, it is not a very good dataset to investigate the effects of oversmoothing. The reason for this is that the GNNs require few GCN layers to perform reasonably well. A small amount of GCN layers, for example two or three, is sufficient to achieve satisfactory performance on the Cora dataset. The reason for testing our architectures on this dataset is that the dataset is non-synthetic and quite popular. As previously described, we train the model for multiple amounts of GCN layers, to induce oversmoothing.

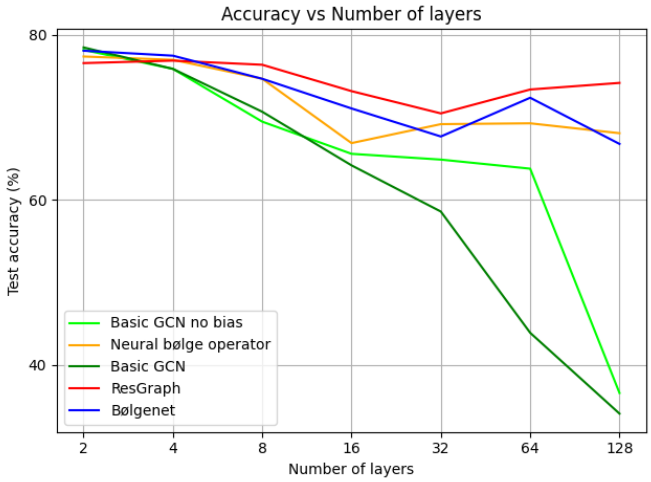


Figure 6.2: Total results for the Cora experiment, accuracy per model with the amount of GCN layer denoted on the horizontal axis. The basic GCN layer (green), basic GCN without bias (lime), ResGraph (red), Neural bølge operator (yellow) and Bølgenet (blue) are included in this plot. Basic GCN with, and without, bias loses performance with in the increase of GCN layers. ResGraph, Neural bølge operator and Bølgenet is able to retain their performance, but not without some decrease.

Figure 6.2 shows the the combined results from section 5.2. From the results we observe how our proposed models are able to preserve their performance throughout the increase of GCN layers. In the figure it is obvious that the basic GCN architectures, with (green) and without bias (lime), performance declines with a higher rate than the other architectures throughout the increase in GCN layers. This is due to the effect of oversmoothing, this gives us the benchmark on whether or not our architectures are capable of handling oversmoothing. Based on the figure, Neural bølge operator and Bølgenet is able to retain a higher performance than the basic GCN models. This means that the proposed architectures are working as hoped. Also with regards to random guessing. As the Cora dataset has 7 classes, random guessing would result in a probability of guess correctly being $p(x)\frac{1}{7} \approx 14.28\%$, meaning that our architectures being able to retain around 70% accuracy is very good.

From figure 6.2, we observe that **three architectures are able to retain some performance** with the increase of GCN layers. These architectures are our proposed architectures: Neural bølge operator and Bølgenet, and ResGraph. We observe that ResGraph is able to retain a higher accuracy than Bølgenet and Neural bølge operator throughout the increase of layers. All three architectures loses some performance with the increase of layers, but they are able to retain performance to a significantly higher degree than the basic GCN models,

meaning that they are able to reduce the effects of oversmoothing.

ResGraph is able to retain a better performance than Neural bølge operator and Bølgenet in the Cora experiment. We speculate that this is related to the nature of the skip-connections in all architectures, and the forced GCN processing in Bølgenet. ResGraph is likely able to retain a better performance due to it being able to skip information over multiple GCN layers. The model might be able to "ignore" the induced oversmoothing, by training multiple GCN layers have little effect on the data. This is resulting in the models being trained to essentially only use few GCN layers. As previously mentioned, this dataset requires a low amount of GCN layers to perform adequately.

Neural bølge operator and ResGraph applies skip-connections in a similar manner, the main difference is the Neural bølge operator uses two skip-connections for each layer. The current and previous state of the latent data is combined with the output of the GCN layer, while ResGraph only combines the current latent state with the output of the block. This might be the main reason for ResGraphs better performance. The complexity in Neural bølge operator is higher than in ResGraph, as multiple stages of the latent data are combined (equation 3.7). This might have the effect that the Neural bølge operator models require more training and tuning to perform well, however this does not suggest that it could be better than ResGraph. Adding multiple previous states will give the model more information and could be able to learn more complex patterns, but it could also be counter-productive. As the latent states are transformed, by the addition and subtraction, with the multiple previous states. If multiple previous states are not necessary, this might hinder the models ability to learn.

Bølgenet and ResGraph employs skip-connection in different manners. Bølgenets drawback is likely the forced processing of the data, although intended. While ResGraph and Neural bølge operator is able to skip data directly between all layers, being able transfer vertex specific data with little modification, Bølgenet is incapable of doing this. Due to the the architecture of Bølgenet, all latent data will be processed through GCN layers, the latent data can only skip GCN layers once without being processed. This means that the latent data will be processed through GCN layers for half the total amount of GCN layers in a model. This means that Bølgenet can experience oversmoothing to some degree, however at a much slower rate than any traditional GNN. The forced processing will induce some sort of smoothing, which is disadvantageous in this experiment, as the amount of required GCN layers is low for a reasonable performance. This forced processing can have noticeable effects with a large amount of layers, meaning that this could be the reason for a decrease in performance. As this is a drawback in this experiment, it is indented. Bølgenet is constructed for cases where multiple GCN layers are required to solve some

problem, and as this dataset does not require many layers, the results of the Cora experiment is not really representative of Bølgenets capabilities.

Dirichlet energy

The Dirichlet energy values in our experiment was unfortunately not very informative. The results of the Dirichlet energy can be viewed in figure 5.10. Based on the explanation in section 5.2.4, without an exponential decrease, oversmoothing is uncertain. We would like to note a difference in results for the basic GCN models without any bias terms. In general, the results of some research papers [26, 27] suggests that the Dirichlet energy of a basic GCN model, with and without bias, should decrease, but in our case the energy is relatively stable. This is likely due to the difference in the implementations of the models. In our experiment, the models have been implemented with a different embedding size and activation function from the the models in the research papers. This is probably the reason for the difference in outcome of the experiment.

6.2 Problems with the experiments

The first experiment is able to give us information in the architectures capabilities in handling oversmoothing and noise. However, there are some disadvantages with this experiment. The noise experiment is filled with noise. With only pure noise and no pattern in the data, the models have a difficult time differentiating the noise for the actual data of interest. Although, the results indicate that the nature of the skip-connections does give Bølgenet an advantage in handling pure noise, as it is capable of preserving performance for higher noise ranges than the other architectures. An improvement to this experiment, could be to have some actual data with patterns, and then investigating the models capabilities in propagating information from a vertex of origin to a vertex of interest, but this is essentially finding a dataset more suited to the task. If some of the models are weaker to noise, this experiment might not properly represent the capability of handling oversmoothing, as there are no patterns for the networks to differentiate.

The second experiment has one main issue, as previous mentioned, the Cora citation network is not a good dataset to test the capabilities of handling oversmoothing. This dataset requires a small amount of GCN layers to perform reasonably well, which is likely why the ResGraph is performing better than the proposed architectures, following the argument in section 6.1.1. ResGraph is able to, in some sense, ignore multiple layers and perform the necessary

computations with few GCN layers to reach satisfactory performance. By this we mean that ResGraph might only properly tune the weights in a small amount of layers to actually perform the prediction. We assume that in a case where we have a dataset that requires a higher amount of GCN layers to perform well, ResGraph will likely perform worse than Bølgenet and possibly Neural bølge operator, however we can not be certain. We believe that Bølgenet might be able to perform better due to its ability to skip information without modifying it. Note that this is just speculation, as an experiment like this this has not been performed. The reason for this remaining untested is the lack of time and the availability of reasonable datasets.

6.3 Other models with similar ideas: PDE-GCN

There are some other research that have explored similar ideas to ours, amongst these one research paper proposes the PDE-GCN model. This paper, "PDE-GCN: Novel Architectures for Graph Neural Networks Motivated by Partial Differential Equations" by Eliasof et al., proposes a model architecture that is based by partial differential equations. This is a similar to our process where we are also motivated by PDEs. There are however some rather large differences. The PDE-GCN transforms the vertex features into edge attributes during in the network to handle the time derivative of the PDEs, in contrast to using the discretised PDEs like we have. This results in their model learning edge weights instead of feature weights. The major difference between their proposed architecture and ours is that theirs it restricted to specific graph structures. A meaningful property of GNNs is the capability of handling different graph structures, however the PDE-GCN method removes this attribute. While the experiments in this thesis has not contained various graph structures, the proposed architectures of our method still has this attribute. However, based on the results in the research paper, their architecture achieves better performance than ours on the Cora dataset.

6.4 The effect on the physics-based models idea

The results of both experiments imply that the proposed model architectures are able to reduce the effects of oversmoothing. In both cases, they did not perform in a revolutionary manner, but they clearly performed better than the standard GCN architecture models. Based on this, we conclude that the architectures are able to inherit the properties of wave propagation. Some properties of wave propagation is the lack of convergence to some equilibrium, but also the decrease in amplitude as the waves spread over a larger area. The

results of both experiments show behaviour similar to these properties. The lack of convergence is visible by observing how the architectures are able to retain a higher performance than the standard GCN architecture. This implies that the models are able to keep the vertices from converging to similar values, which would have made it difficult, and in some cases impossible, to classify specific vertices correctly. The second property, the decrease in amplitude, can be compared to the decline of performance in the Cora experiment. As the models are able to retain a higher performance than the standard GCN models, they still decline to some degree. We speculate that this is due to the "strength" (amplitude) of the information is decreasing as it is propagated through the graph.

Based on this reasoning, it seems that **the architectures are able to inherit the properties of wave propagation**. This supports the idea of creating architectures that inherit the properties of physics PDEs, giving us the possibility to employ inductive bias in the choice of architecture stage, in a different manner to other methods [14].

6.5 Further work

The capabilities of **Neural bølge operator and Bølgenet should be explored further**. The parameters for the Cora experiment and for the Noise experiment can likely be tuned to achieve better performance. Additionally, the models we trained during the experiments, uses tanh as the activation function throughout the entire network. Other activation functions have not been explored for these experiments. The reasoning for tanh in the Noise experiment is the negative class, so the negative relations should be preserved. For the Cora experiment, there are no negative values, as all the features are boolean, and thus other activation functions like ReLU and Sigmoid can be applied, and might result in improved performance.

Due to time constraints and lack of availability of reasonable datasets, an other future work idea is more **extensive testing** of the architectures. As mentioned in section 6.2, the Cora citation graph dataset is not a good dataset to test the capability of the architectures. The goal of the architectures is to be able to propagate information to vertices far away, while being able to reduce the effects of oversmoothing. Due to the fact that the Cora dataset requires few GCN layers to perform adequately, the dataset is not really sufficient to test the capabilities properly. This is why some further work should be finding datasets that is more fit for the task that the architectures constructed to perform.

We also want to emphasise that there are **other PDEs** than the ones we have

used. In this thesis, we have used the heat diffusion PDE to describe the process of oversmoothing, and the wave propagation PDE to inspire the creation of new architectures to help reduce the effects of this problem. Applying other PDEs to create architectures with the goal of solving some problem is something that should be investigated. Whether or not there is an inherent problem with the network, or solving specific problems with classification, regression etc., applying PDEs can be a viable method to construct architectures. While applying differential equations to construct architectures is not a novel idea, using physics to create models with specific properties is. Another thing that would be of interest to investigate, is the different components of various PDEs and how to translate these into architectures. Some PDEs contain components that are more difficult to implement.

An interesting idea that can be investigated based on our architectures, is **the constant** γ. It appears in the initial inspiration of the model layer equations. That being the wave model layer equation 3.5 and the heat model layer equation 3.3. We continued both these equations by setting $\gamma = 1$ for simplicity. What would happen if $\gamma \neq 1$? In the heat equation case, the model layer function would look like this:

$$\begin{aligned} u[t+1] &= u[t] - \gamma u[t] + \gamma \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] \\ &= (1-\gamma)u[t] + \gamma \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] \end{aligned}$$

and in the wave equation case:

$$\begin{aligned} u[t+1] &= 2u[t] - \gamma u[t] + \gamma \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] - u[t-1] \\ &= (2-\gamma)u[t] + \gamma \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}u[t] - u[t-1] \end{aligned}$$

In both of these cases the current state, and the GCN layer is also scaled with γ in some manner. This can be something of interest to investigate. How would this affect the performance of the proposed model architectures? The constant γ is generated from PDE specific parameters, along with the time-step of the discretisation of the PDEs. In the heat equation, γ is a combination of the time-step and the diffusion constant, and in the wave equation, γ is the combination of the time-step and the wave-speed constant. With the application of the PDEs to GNNs, there constants become arbitrary. It would be of interest to investigate their effects on the architectures. By observing both equations above, they essentially scale the GCN layer with regards to the skip-connections.

7

Conclusion

The **goal of this thesis** was to present the idea of using physical systems to create model architectures tailored to solve some problem. This idea was tested by creating two model architectures with the goal of negating the effects of oversmoothing in graph convolutional networks. The results of the experiments indicate that the model architectures are able to reduce the effects of oversmoothing, however not negate the effect completely. This implies the that the idea of using physical systems to inherit properties to model architectures is promising.

Two experiments were performed to test the capabilities of the architectures, along with their resilience to oversmoothing. The first experiment was testing the architectures on noise data. The noise data was constructed so that oversmoothing will occur, meaning that the experiment will indicate the architectures ability to propagate information through noise. The second experiment was performed on the Cora citation network. The goal of this experiment was to test the architectures on non-synthetic data and observe their capabilities in handling oversmoothing with regards to existing architectures. Both experiments indicate that both architectures are able to handle oversmoothing to some degree. Based on the comparisons, there are existing models that achieve better performance, but the proposed architectures can be improved. Meaning that they are likely capable of better performance, given more extensive testing.

The idea of using physics for oversmoothing is a good idea that should be

explored further. By relating heat diffusion to oversmoothing, and comparing the process of a GCN layer with the discretised heat equation, we observed a strong similarity. With this we applied the discretised wave equation to create a new model layer function that resulted in the architecture Neural bølge operator. Additionally, we used the model layer function to inspire an other architecture, Bølgenet. Based on the results and the discussion, we conclude that this method of introducing an inductive bias to model architectures as a method of handling oversmoothing is viable. We hope that this conclusion inspires further investigation of the method.

While this thesis demonstrates the viability of inheriting properties from physics, **further investigations are encouraged** to uncover the true potential of this approach. There are numerous physical systems out there, offering unique characteristics that could provide unforeseen capabilities for machine learning models. We hope that this thesis inspires further exploration of this idea and the potential it holds.

Bibliography

[1] Pytorch: An open source deep learning platform. https://pytorch.org/. Accessed: 10.12.2023.

[2] Pytorch geometric: Geometric deep learning extension library for pytorch. https://pytorch-geometric.readthedocs.io/. Accessed: 10.12.2023.

[3] Antonsen, T. (2023a). Project paper illustrations. https://github.com/axdeux/project-paper-illustrations.

[4] Antonsen, T. S. M. (2023b). Performance of graph neural networks on simulating heat and wave systems. *UiT Capstone Project*.

[5] Cai, C. and Wang, Y. (2020). A note on over-smoothing for graph neural networks. *CoRR*, abs/2006.13318.

[6] Cao, Y.-H. and Wu, J. (2022). A random cnn sees objects: One inductive bias of cnn and its applications. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):194–202.

[7] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

[8] Eliasof, M., Haber, E., and Treister, E. (2021). Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3836–3849. Curran Associates, Inc.

[9] Fey, M., Hu, W., Huang, K., Lenssen, J. E., Ranjan, R., Robinson, J., Ying, R., You, J., and Leskovec, J. (2023). Relational deep learning: Graph representation learning on relational databases.

[10] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[11] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2.

[12] Haber, E. and Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004.

[13] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[14] Han, J., Rong, Y., Xu, T., and Huang, W. (2022). Geometrically equivariant graph neural networks: A survey.

[15] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[16] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[17] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

[18] Klicpera, J., Becker, F., and Günnemann, S. (2021). Gemnet: Universal directional graph neural networks for molecules. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

[19] Koishekenov, Y. (2023). Reducing over-smoothing in graph neural networks using relational embeddings.

[20] Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. (2023). Learning skillful medium-range global weather forecasting. *Science*, 0(0):eadi2336.

[21] Lukovnikov, D. and Fischer, A. (2021). Gated relational graph attention networks.

[22] Oono, K. and Suzuki, T. (2019). On asymptotic behaviors of graph cnns from dynamical systems perspective. *CoRR*, abs/1905.10947.

[23] Oono, K. and Suzuki, T. (2021). Graph neural networks exponentially lose expressive power for node classification.

[24] Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., and Tang, J. (2018). Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2110–2119, New York, NY, USA. Association for Computing Machinery.

[25] Ricaud, B., Borgnat, P., Tremblay, N., Gonçalves, P., and Vandergheynst, P. (2019). Fourier could be a data scientist: From graph fourier transform to signal processing on graphs. *Comptes Rendus Physique*, 20(5):474–488. Fourier and the science of today / Fourier et la science d'aujourd'hui.

[26] Rusch, T. K., Bronstein, M. M., and Mishra, S. (2023a). A survey on oversmoothing in graph neural networks.

[27] Rusch, T. K., Chamberlain, B. P., Mahoney, M. W., Bronstein, M. M., and Mishra, S. (2023b). Gradient gating for deep multi-rate learning on graphs. In *The Eleventh International Conference on Learning Representations*.

[28] Rusch, T. K., Chamberlain, B. P., Rowbottom, J., Mishra, S., and Bronstein, M. M. (2022). Graph-coupled oscillator networks. *CoRR*, abs/2202.02296.

[29] Ruthotto, L. and Haber, E. (2018). Deep neural networks motivated by partial differential equations. *CoRR*, abs/1804.04272.

[30] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

[31] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[32] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks.

[33] Wilson, R. J. (1996). *Introduction to graph theory*. Prentice Hall, forth

edition.

[34] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *CoRR*, abs/1810.00826.

[35] Xu, K., Zhang, M., Jegelka, S., and Kawaguchi, K. (2021). Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11592–11602. PMLR.

[36] Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861.

[37] Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. (2021). Dirichlet energy constrained learning for deep graph neural networks.

8 Appendix

8.1 Table of results for the Noise experiment

This appendix section contain some specific results from the Noise experiment.

8.1.1 Neural wave operator

Noise range	Mean	Std	Max	Min	Median
0.0	0.897	0.206	1.0	0.478	1.0
0.1	0.954	0.139	1.0	0.538	1.0
0.2	1.0	0.001	1.0	0.998	1.0
0.3	0.749	0.24	1.0	0.474	0.77
0.4	0.565	0.158	0.974	0.462	0.5
0.5	0.537	0.099	0.83	0.47	0.51
0.6	0.491	0.019	0.516	0.458	0.493
0.7	0.504	0.021	0.53	0.474	0.508
0.8	0.515	0.013	0.53	0.496	0.521
0.9	0.5	0.022	0.548	0.474	0.498
1.0	0.503	0.02	0.548	0.47	0.499

Table 8.1: Noise experiment results: accuracy measures for Neural bølge operator. Table contains that mean accuracy, standard deviation, maximum accuracy and minimum accuracy. Additional the median accuracy is included.

8.1.2 Bølgenet

Noise range	Mean	Std	Max	Min	Median
0.0	0.652	0.228	1.0	0.456	0.519
0.1	1.0	0.0	1.0	1.0	1.0
0.2	1.0	0.001	1.0	0.998	1.0
0.3	0.998	0.001	1.0	0.996	0.998
0.4	0.981	0.007	0.992	0.968	0.982
0.5	0.936	0.028	0.962	0.862	0.942
0.6	0.822	0.138	0.91	0.51	0.895
0.7	0.724	0.137	0.852	0.484	0.796
0.8	0.542	0.03	0.6	0.508	0.529
0.9	0.514	0.016	0.538	0.49	0.511
1.0	0.514	0.025	0.562	0.482	0.509

Table 8.2: Noise experiment results: accuracy measures for Bølgenet. Table contains that mean accuracy, standard deviation, maximum accuracy and minimum accuracy. Additional the median accuracy is included.

8.2 Table results for the Cora experiment

This section contains specific accuracy results for the Cora experiment, performed as explained in section 4.2.

Model	2	4	8	16	32	64	128
Basic GCN no bias	78.2	75.9	69.5	65.6	64.9	63.8	36.6
Basic GCN	78.5	75.9	70.7	64.2	58.6	43.9	34.1
Bølgenet	78.1	77.5	74.7	71.1	67.7	72.4	66.8
Neural bølge operator	77.4	77.0	74.7	66.9	69.2	69.3	68.1
ResGraph	76.6	76.9	76.4	73.2	70.5	73.4	74.2

Table 8.3: Accuracies for the models in the Cora experiment. Left side denotes which models the row of results are generated from. The top row denotes the amount of GCN layer contained within the models.