Institut für
Automatisierungstechnik

Master Project

Imitation learning for power-flow study

Hrushikesh Mantri

October 25, 2023

Project Number: M1050

Tutor:
Msc. Markus Miller
Msc. Nasratullah Mohseni

Examiner:
Prof. Dr.-Ing. Johanna Myrzik

Universität
Bremen

Institut für Automatisierungstechnik

Master Project/Thesis

Imitationslernen für die Leistungsflussanalyse

Hintergrund Die Leistungsflussanalyse berechnet nicht beobachtbare Größen im Stromnetz (Wirk- und Blindleistung, Spannung, Winkel). Sie ist die Grundlage für weitergehende Optimierungen wie zum Beispiel den optimalen Leistungsfluss oder für die Planung von Generatoren. Der Rechenaufwand steigt jedoch quadratisch mit der Anzahl an Knoten im Stromnetz an. Dies hat zur Folge, dass die Lösung großer Stromnetze rechenintensiv wird und daher in vielen Fällen ein vereinfachtes Modell verwendet wird. In dieser Arbeit soll ein Algorithmus entwickelt werden, das die Leistungsflussanalyse verschiedener Stromnetze imitiert und dabei die Rechenzeit reduziert. Die Stromnetze sollten eine variable Anzahl von erneuerbaren Energien (Windkraftanlagen, Solarpanel) enthalten.

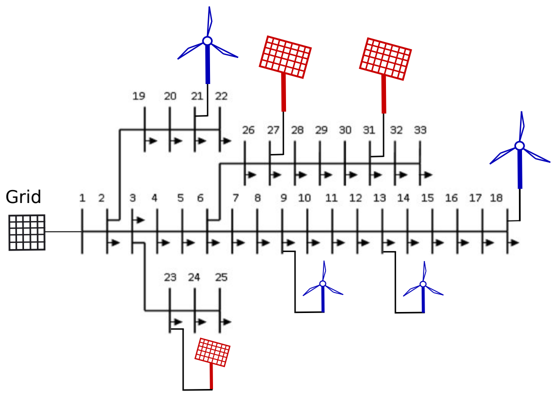

Abbildung 1: Power grid with renewable energies.

Aufgaben Entwurf und Implementierung eines neuronalen Netzes, das den Leistungsfluss nachahmt:

- Muss für Stromnetze unterschiedlicher Größen geeignet sein.
- Modellierung von Erzeugungs- und Lastleistung als Zufallsvariablen.
- Vergleich der Ergebnisse mit den Ergebnissen von pandapower oder dIgSILENT.

Imitation learning for power-flow study

Background The power-flow study calculates unobservable quantities in the power system (active and reactive power, voltage, angle) by an iterative approach. It is the foundation for more advanced optimizations such as the optimal power flow or for the planning of generators. However, the computational effort increases quadratically to the number of nodes in the power grid. As a result, solving large power grids become computationally expensive and consequently a simplified model is used in many cases. In this work a neural network should be developed that imitates the power-flow study for different power grids, while reducing the computation time. The power grids should have a variable number of renewable sources (wind turbines, solar panels) included.

Tasks Design and implementation of a neural network that imitates the power-flow study:

- Must be applicable for power systems of different sizes.
- Modeling generation and load power as random variables.
- Compare the results with those of pandapower or dIgSILENT.

Markus Miller, M1050, miller@iat.uni-bremen.de

Universität Bremen

Declaration of honour

I hereby confirm on my honor that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisor has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

Date: ________________ Signature: ___________________________

Declaration of publication

- ☐ I hereby agree, that my thesis will be available for third party review in purpose of academic research.
- ☐ I hereby agree, that my thesis will be available after 30 years (§7 Abs. 2 BremArchivG) in the university archive for third party review in purpose of academic research.
- ☐ I hereby do *not* agree, that my thesis will be available for third party review in purpose of academic research.

Datum: ________________ Unterschrift: ___________________________

Abstract

Power engineering relies heavily on power flow analysis as a primary method for arranging and controlling the operation of power systems. The issue of common power flow can be broken down into a series of nonlinear equations, typically solved through the use of numerical optimization. Methods of calculation such as the Newton-Raphson method. These methods, on the other hand, may become computationally expensive when applied to large systems, and convergence to the global optimum is not guaranteed. In recent years, a number of strategies Graph Neural Networks have been proposed with the intention of accelerating the computation of the power flow solutions without significantly compromising the accuracy of the results. This collection of models is able to learn properties independently of a larger graph's structure. Consequently, given that power systems are typically represented as graphs, these methods can, in theory, be generalized to systems of varying sizes and topologies. On the other hand, the majority of the approaches that are currently being used have only been tested on systems that have a single topology, and none of them have been trained simultaneously on systems with varying topologies.
The study presented in this project allowed to draw important insights about the applicability of GNN as power flow solvers.

Contents

List of Abbreviations

Abbreviation	**Definition**
ACPF	Alternating-Current Power Flow
ANN	Artificial Neural Network
ARMA	Auto-Regressive Moving Average
CNN	Convolutional Neural Network
ECC	Edge-Conditioned Convolution
DCPF	Direct-Current Power Flow
FC	Fully-Connected
GCN	Graph Convolutional Network
GCS	Graph Convolutional Skip
GNN	Graph Neural Network
GNS	Graph Neural Solver
GS	Gauss-Seidel
IEEE	Institute of Electrical and Electronics Engineers
MSE	Mean Squared Error
NR	Newton-Raphson
PF	Power Flow

Chapter 1

Introduction

The operation, planning, maintenance, and control of power systems all rely heavily on power flow analysis as a fundamental component. The flow of power must be distributed in such a way that the demand can be satisfied without exceeding the physical limits of the electrical components that make up the energy network in order to guarantee sufficient reliability for the day-to-day operation of the system. A power solver needs information about how the electrical nodes in the power grid are connected in order to perform the computations necessary to find this balanced state of operation. It is possible to determine the balanced power flow distribution of a system by solving a set of non-linear equations that are derived from Kirchhoff's current law. These equations are typically solved through the application of iterative methods such as the Newton-Raphson method. These iterative procedures have the ability to find the solution at a speed that is reasonable for networks of moderate size or smaller. However, the computational complexity of these methods can render them unviable for tasks such as quickly screening how different possible contingencies will affect a grid if they are applied to larger grids or to different grid configurations in rapid succession. This can happen when the methods are applied to larger grids. These numerical optimizers have a number of drawbacks, one of which is that convergence to the global optimum is not guaranteed in general, and another is that they are sometimes sensitive to initialization. Graph Neural Networks (GNNs), which are a subset of artificial neural networks (ANNs), are a set of models within relational deep learning that are capable of performing regression and classification tasks on graphs. GNNs are able to scale well to larger power grids and have the potential to be trained on smaller grids before being applied to larger ones. At this time, several models that make use of GNNs have been proposed for various variants of the power flow problem; however, the majority of these models use a configuration in which a model is trained and tested only on the same grid topology.In other words, they only receive training on a single grid topology before being tested on a variety of different topologies. Additionally, none of these GNNs have been trained or tested on variations of a predetermined topology in any way. The performance of various GNN models derived from well-established neural network methodologies has been evaluated for the purposes of this project, with regard to issues concerning grids of constant size, grids of completely different sizes, and grids that have been subjected to perturbations. In addition, the purpose of these tests is to investigate and talk about the following topics: i) the ability of GNNs to generalize to different topologies; ii) the most effective way to represent the power system as a graph; and iii) which system quantities should be used as input and how they should be incorporated into the graph representation.

Chapter 2

Theory

This part presents the theoretical background needed to understand the power flow problem for grids and the ANN models will be trained to solve. Section 2.1 presents the theory related to the power flow problem. Section 2.2 gives a brief description of graphs and GNNs.

2.1 The power flow problem

The objective and structure of the power flow problem are explained in this section along with some basic theory. A definition of a power system and the pertinent quantities are presented in Section 2.1.1. Fundamentals are covered in Section 2.1.2. Background theory for the issue with AC power flow. A brief explanation of a linear approximation the AC power flow problem is provided in Section 2.1.3. Equations for computing the transmission line injections in the power flow solution are provided in Section 2.1.4.

2.1.1 Definition of a power system

Before the power flow, it is necessary to establish some terminology and equations related to power systems. A collection of free, open-source Matlab-language M-files called Pandapower[9] is designed to help with the optimization and simulation of steady-state power system issues like the power flow problem. The majority of the definitions are consistent with those used by Pandapower. The authors Zimmerman, Murillo-S'anchez, and Thomas provide these definitions (2010). At its most basic level, a power system is made up of a grid or network of buses, also known as nodes, that are connected to one another by transmission lines, transformers, and phase shifters, among other things. Now, these various parts will be combined to create a unified branch model, adhering to the framework established by Pandapower. This branch model is made up of a couplet in series with an ideal phase shifting transformer, a standard transmission line model, and a regular transformer placed at the branch's terminus where it connects to the main line. It measures the dynamic effects of the component's material susceptibility to polarization and the ease with which current can flow through it. Each grid element has a corresponding admittance value, which expresses how easily current can pass through it. Therefore, to relate currents and voltages in a power system, component admittance of buses and branches is used. Since component admittance is used to link currents and voltages, it is crucial for determining the best power routing in power flow analysis. For each branch, we can create a 2 by 2 branch admittance matrix called $\mathrm{Y_{br}}$. This matrix establishes a connection between the complex voltages $\mathrm{v_f}$ and $\mathrm{v_t}$ and the complex current injections $\mathrm{i_f}$ and $\mathrm{i_t}$ at the beginning and end of the branch. This relationship mathematical expression is:

$$\begin{bmatrix} i_f \\ i_t \end{bmatrix} = Y_{br} \begin{bmatrix} v_f \\ v_t \end{bmatrix} \tag{2.1}$$

where f and t denote the beginning and end of the branch, respectively. provides the following representation of the branch admittance matrix.

$$y_{br} = \begin{bmatrix} y_{ff} & y_{ft} \\ y_{tf} & y_{tt} \end{bmatrix} = \begin{bmatrix} (y_s + j\frac{b_c}{2})\frac{1}{r^2} & -y_s\frac{1}{\tau e^{j\theta} shift} \\ -y_s\frac{1}{\tau e^{j\theta} shift} & y_s + j\frac{b_c}{2} \end{bmatrix} \tag{2.2}$$

where j represents the imaginary unit, $\mathrm{y_s}$ the series admittance in the model, and $\mathrm{b_c}$ represents the total charging susceptance. The tap ratio magnitude and the phase shift angle of the transformer are denoted by the variables $\tau$and Θ_{shift} in this equation. When there are $\mathrm{N_l}$ branches in a system, it is possible to construct four $\mathrm{N_l}\cdot 1$ vectors using the corresponding elements in y_{br} from each branch. These vectors are denoted as $y_{\mathrm{ff}}, y_{\mathrm{ft}}, y_{\mathrm{tf}}$ and y_{tt}[19]. Constant power loads in the grid are represented by specific amounts of real (or active) and reactive power consumption at buses. Reactive power is the energy that is only stored in the circuit and flows back and forth between the source and the load without actually transferring any energy to the load. Active power in a circuit refers to the actual power that is consumed by a load, also known as the net transfer of energy in one direction. The real power that is consumed is represented by active power, which sets it apart from reactive power.The complex power load for each bus is provided by

$$s_d = p_d + jq_d \tag{2.3}$$

where $\mathrm{p_d}$ is the real or active component and $\mathrm{g_d}$ is the reactive component (where d stands for demand). Similarly, generators are represented by complex power injections at specific buses, and for each generator bus the injection is

$$s_g = p_g + jq_g \tag{2.4}$$

where $\mathrm{p_g}$ and $\mathrm{q_g}$ are the active and reactive generator power injection, respectively. In addition to loads and generators, Pandapower includes shunt connected elements at buses such as capacitors and inductors that are modeled as fixed impedance to ground. The shunt admittance is given by

$$y_{sh} = g_{sh} + jb_{sh} \tag{2.5}$$

where $\mathrm{g_{sh}}$ is the shunt conductance value and $\mathrm{b_{sh}}$ is the shunt susceptance value. For a grid with $\mathrm{N_b}$ buses, all constant impedance elements of the network can be incorporated into a $Y_{bus} \in \mathbb{C}^{N_b \times N_b}$ nodal admittance matrix Ybus, formed from the elements of the admittance matrices given by (2.2) and the shunt admittances. The current injection $i_{bus,k}$ of a bus k is calculated with is given as

$$i_{bus,k} = \sum_{N_b}^{j=1} Y_{bus,K_j} v_j \tag{2.6}$$

where v_j is the complex voltage of bus j.
Similarly, for the branches, two $\mathrm{N_l} \times \mathrm{N_b}$ branch admittance matrices $\mathrm{Y_f}$ and $\mathrm{Y_t}$ can be formed from the vectors created from each row of (2.2), and these relate the bus voltages to the current injections at the from and to ends of the branches. For branch k the relations are given as

$$i_{f,k} = \sum_{N_b}^{j=1} Y_{f,K_j} v_j \tag{2.7}$$

$$i_{t,k} = \sum_{N_b}^{j=1} Y_{t,K_j} v_j \tag{2.8}$$

To construct these three system admittance matrices Y_f, Y_t and Y_{bus} , two sparse $\mathrm{N_l} \times \mathrm{N_b}$ connection matrices $\mathrm{C_f}$ and $\mathrm{C_t}$ used. Let the ij-th element of C_f and the ik-th element of C_t be 1 if branch i connects bus j to bus k in the grid, and zero otherwise. If we let diag(·) denote an operator which creates a corresponding diagonal matrix from a vector, the admittance matrices can be formed as

$$Y_f = diag(Y_{ff})C_f + diag(y_{ft})C_t \tag{2.9}$$

$$Y_t = diag(Y_{tf})C_f + diag(y_{tt})C_t \tag{2.10}$$

$$Y_{bus} = C_T^f Y_f + C_T^t Y_t + diag(Y_{sh}) \tag{2.11}$$

From the current injections, the corresponding power injections for each bus and branch k are found as

$$s_{bus,k} = v_k i^*_{bus,k} \tag{2.12}$$

$$s_{f,k} = \left(\sum_{j=1}^{N_b} C_{f,kj} v_j \right) i^*_{f,k} \tag{2.13}$$

$$s_{t,k} = \left(\sum_{j=1}^{N_b} C_{t,kj} v_j \right) i^*_{t,k} \tag{2.14}$$

where the asterisk denotes complex conjugate.

2.1.2 The AC power flow problem

In the traditional AC power flow (ACPF), four variables are of interest for each electrical bus k in a power system [16]. These are

p_k : Net active (or real) power injection
q_k : Net reactive power injection
$|v_k|$: Voltage magnitude
θ_k : Voltage phase angle

The net power injections are the difference between generation and load consumption at the buses, i.e.

$$p_k = p_{g,k} - p_{d,k} \tag{2.15}$$

$$q_k = q_{g,k} - q_{d,k} \tag{2.16}$$

With these variables known, along with relevant admittances, all currents and power flows in the system can be calculated from the equations in the previous section.

The power flow problem consists of finding the steady-state for a system given load and generation. The steady-state can be seen as a static case where power equilibrium is achieved. Of course, in reality the system is not truly static, but will always be subject to small load changes and other transients [1].

The steady state satisfy a set of equilibrium equations of the form

$$g(X) = 0 \tag{2.17}$$

where X is a set of variables. More specifically, for each bus we get two power balance equations derived from Kirchhoff's current law[7]

$$\sum_{j=1}^{N_b} |v_k| \, |v_j| \left(G_{bus,kj} cos(\theta_k - \theta_j) + B_{bus,kj} sin(\theta_k - \theta_j)\right) - p_k = 0 \tag{2.18}$$

$$\sum_{k=1}^{N_b} |v_k| \, |v_j| \left(G_{bus,kj} sin(\theta_k - \theta_j) - B_{bus,kj} cos(\theta_k - \theta_j)\right) - p_k = 0 \tag{2.19}$$

where $\mathrm{G}_{\mathrm{bus,k_j}}$ and $\mathrm{B}_{\mathrm{bus,k_j}}$ are the respective real and imaginary parts of the kj-th element in the bus admittance matrix $\mathrm{Y}_{\mathrm{bus}}$. From here on, equations (2.18) and (2.19) will be referred to as the active/real and reactive power equations[7].
Combined for all buses, the power balance equations yields a system of $2N_b$ non-linear equations, with a total of $4N_b$ unknowns. To have a unique solution that satisfies all of them two out of the four variables $\mathrm{p_k}$, $\mathrm{q_k}$, $|\mathrm{v_k}|$, Θ_k must be specified beforehand for each bus. There are naturally multiple ways to accomplish this, but in the most common methodology a single bus with a generator is chosen as a reference bus or slack bus for which voltage magnitude and phase angle are specified. The phase angle for the other buses are expressed as differences from the slack bus. Also, the reference effectively serve as an active power slack for the system, i.e. $\mathrm{p_k}$ is unknown for this bus. The other buses with generators are referred to as PV buses, where active power injection and voltage magnitude is known. Finally, the remaining buses are called PQ buses, and for these, real and reactive injections are specified, i.e. all load values are fixed beforehand. An overview of the different bus types is given in Table 1.

Bus type	Specified quantities
Reference/slack bus	$|\mathrm{v_k}|$, Θ_k
PQ bus	$\mathrm{p_k}$, $\mathrm{q_k}$
PV bus	$\mathrm{p_k}$, $|\mathrm{v_k}|$

Table 2.1: Overview of the different bus types.

The variables that are of main interest are the voltage magnitude at PQ buses and the voltage phase angle at all non-slack buses. This amounts to $\mathrm{N_{pv}} + 2\mathrm{N_{pq}}$ unknowns, where $\mathrm{N_{pv}}$ and $\mathrm{N_{pq}}$ are the number of PV and PQ buses, respectively. To solve for these variables, an equal number of equations can formed from the active and reactive power balance equations of the PQ buses and the active power equations of the PV buses. After finding all the voltage magnitudes and phase angles, the $\mathrm{N_{pv}} + 1$ unknown reactive power injections for generator buses can be found from the corresponding reactive power balance equations of these buses.
The non-linear nature of the power balance equations prevents the problem from being solved analytically. Instead the traditional approach is to use iterative procedures such as the Newton-Raphson (NR) method or the Gauss-Seidel (GS) method. The NR method is the most widely used of these

two, and has at least a quadratic rate of convergence if sufficiently close to the solution [6].

2.1.3 Compute branch injections

When studying the solution of the ACPF problem, one may be more interested in the power flows on the branches rather than the voltages on the buses. After applying a solver such as the Newton-Raphson method, Pandapower provides the power injections and the voltages at both ends of each branch. These are given as

$$s_f = p_f + jq_f \tag{2.20}$$

$$s_t = p_t + jq_t \tag{2.21}$$

$$v_f = |v_f|\, e^{j\theta_f} \tag{2.22}$$

$$v_t = |v_t|\, e^{j\theta_t} \tag{2.23}$$

The complex current injections at both ends then become

$$i_f = \left(\frac{s_f}{v_f}\right)^* \tag{2.24}$$

$$v_t = \left(\frac{s_t}{v_t}\right)^* \tag{2.25}$$

2.2 Graphs and GNNs

As mentioned in Section 2.1, it may be natural to model power systems as graphs. The admittance matrix can be seen as a form of graph adjacency matrix, and the different system components can be seen as graph nodes and edges, with the different system quantities as node and edge features. In this section, basic theory for graphs and GNNs will be presented. First, Section 2.2.1 will present a graph definition and relevant graph attributes. Section 2.2.2 will cover some general GNN theory. Section 2.2.3 presents one of the more widely used graph convolutional network methodologies. And finally, Section 2.2.4, 2.2.5 and 2.2.6 present, respectively, the ARMA, GCN and Linear layer, which will be used for the GNN models in the experiments.

2.2.1 Graph definition

A mathematical graph G = (V, ε) is defined by a set of nodes (or vertices) V and a set of edges (or arcs) ε that connect them. Graph edges can be either oriented or non-oriented, which depends on the ordering of connected node pairs. If the connected pairs are not ordered, i.e. if $\varepsilon \subseteq \{\{u, v\} \,|u, v\epsilon V\}$ [3], edges are non-oriented and the graph is said to be undirected. In opposition, if the pairs are ordered, meaning $\varepsilon \subseteq \{\{u, v\} \,|u, v\epsilon V\}$, the edges are oriented and the graph is directed. In other words, for a directed graph, a distinction is made between an edge going from node u to v and one going from v to u. An illustration of an undirected graph versus a directed graph is given in Figure 2,1.

The edges found in ε can be encoded into a binary adjacency matrix $A \,\epsilon\, \mathbb{R}^{|\nu|\times|\nu|}$, where element A_{uv} is equal to 1 if an edge goes from node u to node v, and 0 otherwise. The lack of ordering in undirected graphs means that A is always symmetric, while this is not necessarily the fact for directed graphs. For the graphs in Figure 2,1, the respective adjacency matrices are

$$(a)A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (b)A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In graphs, a path is defined as a sequence of unique edges that joins a sequence of nodes, and the path length is given by the number of edges in this sequence. By finding the length of the shortest path between nodes, one can get an idea of the complexity of the graph.
An undirected graph is said to be connected if there exists a path for every vertex pair, and this is the case for the example graph in Figure 2.1 a. A directed graph is, on the other hand, said to be weakly connected if replacing each edge with an undirected one yields a connected graph. If a directed path exists from u to v and from v to u for every node pair u, v, then the graph is strongly connected. The example graph in Figure 2.1b is weakly connected but not strongly, since e.g. there is no directed path going from node 5 to node 2.

The neighborhood of node u in an undirected graph, denoted as N_u, is defined as the set of nodes that have edges connecting to node u, i.e. N_u

$=\{v|\,\{u,v\}\,\epsilon\varepsilon\}$. The neighbors of each node can thus be seen from the adjacency matrix, where node v is a neighbor of u if $A_{uv} = A_{vu} = 1$. If the neigborhood also includes u itself (meaning the graph has self-connecting edges or self-loops), it is said to be closed. Otherwise, it is referred to as an open neighborhood. For a directed graph we can differentiate between out-neighbors and in-neighbors for each of the two possible edge orientations. The set of out-neighbors is then given by $N_u^{out} = \{v|\,\{u,v\}\,\epsilon\varepsilon\}$ and the set of in-neighbors as $N_u^{in}= \{v|\,\{u,v\}\,\epsilon\varepsilon\}$. As an example, for the undirected graph in Figure 2.1 the neigborhood of node 1 is $N_1 = \{\ 2,3,4,5\}$, while for the directed graph the out- and in-neigborhoods are $N_u^{out} = \{\ 4,5\}$ and $N_u^{in} = \{\ 3,2\}$.
In addition to the topological information in A, the graph representation can be extended by having additional feature information on the nodes and edges. Each node is then associated with a corresponding feature vector x, and an edge connecting node u to node v comes with a corresponding feature vector $\mathrm{e}_{\mathrm{u}\rightarrow\mathrm{v}}$ (or e_{uv}). Note that for non-oriented edges $\mathrm{e}_{\mathrm{uv}} = \mathrm{e}_{\mathrm{vu}}$. The features in these vectors can be discrete, continuous or possibly a combination of both.

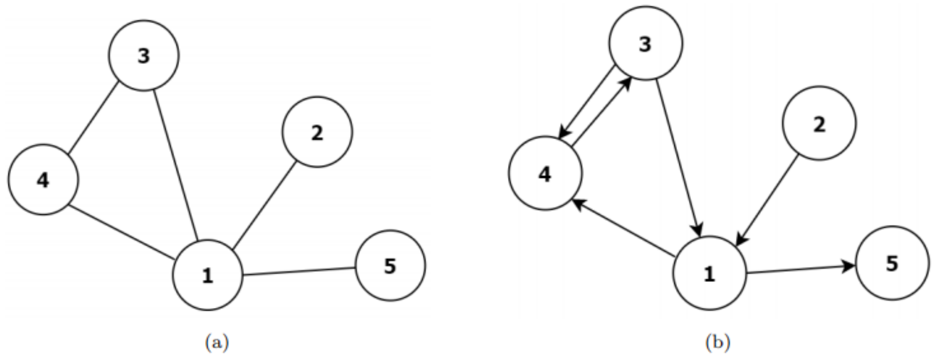

Figure 2.1: (a) An illustration of an undirected graph. By substituting two similar edges facing the opposite direction from each other for each edge in this graph, a directed version can be created. (b) An illustration of a directed graph with an asymmetric adjacency matrix.

2.2.2 Neural Networks using graphs

Since graphs are the best representation of many types of non-Euclidean real-world data, there is interest in developing and refining Artificial Neural Networks (ANN) for graph applications. Until now, graph-based neural

network approaches have been used to categorize protein functions for biological protein-protein interaction graphs[12], recommend content to social media users[17], predict side effects from drug-drug interactions [20], and categorize document topics [14].
However, it is challenging to define key mechanisms and terms that apply to all of the new techniques that combine ANNs and graphs because there are so many of them. There are numerous ways to complete this task, making it even more difficult.
The same words for similar techniques or different techniques using the same words. In the current literature, for instance, the phrase "graph convolution" is frequently used because many methods employ some sort of convolution operator (either in the spatial/node or spectral domain) and/or incorporate a local receptive field, similar to those produced by the filters in convolutional neural networks [11].Good examples include the methods used by Defferrard, Bresson, and Vandergheynst (2016)[8], Kipf and Welling (2017)[14], Atwood and Towsley (2016)[2], and Duvenaud et al. (2015)[9]. Additionally, any model in the larger field of artificial neural networks (ANNs) that makes use of graphs in some way is frequently referred to as a "Graph Neural Network" (GNN).
Even though different versions of GNNs have been derived, there are some basic ideas that apply to most of them. In the next part, we'll talk about some of these shared ideas, with a lot of help from Bacciu et al.'s (2020)[3] introduction to GNNs.

2.2.3 Common concepts for GNNs

In a Graph Neural Network (GNN), every node is typically linked to a state vector $\boldsymbol{h_i}$, which undergoes updates through multiple iterations or layers of the network. At the l-th iteration step, we can represent the state of node i as $h_i^{(l)}$. Usually, the initial state of each node at the input layer is assigned to a specific input feature vector, denoted as $h_i^{(0)} = x_i$.
To modify the state of individual nodes within a graph, a common approach involves disseminating information across the graph through message passing between nodes [10]. To compute these messages for each node, one typically utilizes its present state and, potentially, the feature information from edges connecting to that node. Subsequently, these messages are transmitted to neighboring nodes, and each node updates its state by incorporating the incoming messages alongside its current state. When this message passing occurs iteratively, nodes gather contextual information from a broader area than just their immediate neighbors. This means that the local receptive field of each node expands over successive message passing steps, as demonstrated

in Figure 2.2. By undergoing multiple message passing steps (or propagation steps), node i in the figure can effectively receive information from all the neighbors of node j including node j itself. Consequently, if nodes contain pertinent information regarding distant neighbors, this propagation of information can yield benefits in various prediction and classification tasks.
For many of the established GNN architectures, the neighborhood aggregation of messages for node i at step l can be collectively expressed as [4]

$$h_i^{(l)} = \phi^{(l)} \left(h_i^{(l-1)}, \Psi \left(\left\{ g^{(l)} \left(e_{j \to i} \right)^T \psi^{(l)} \left(h_j^{(l-1)} \right) \mid j \in \mathcal{N}_i \right\} \right) \right) \tag{2.26}$$

In this context, ϕ, ψ, and g are generic transformation functions, while Ψ represents a permutation invariant function, such as the sum or max operator. Additionally, $\mathcal{N}_i$ can either denote the open or closed neighborhood of node i. It is worth mentioning that this formulation allows for the incorporation of edge features, whether they are discrete or continuous in nature. In the absence of edge features, the expression simplifies to a different form.

$$h_i^{(l)} = \phi^{(l)} \left(h_i^{(l-1)}, \Psi \left(\left\{ \psi^{(l)} \left(h_j^{(l-1)} \right) \mid j \in \mathcal{N}_i \right\} \right) \right) \tag{2.27}$$

Please note that equations (2.20) and (2.21) assume that the graphs are non-positional, necessitating Ψ to be permutation invariant.
As mentioned earlier, employing successive message passing and neighborhood aggregation expands the local receptive field of each node, which can be advantageous for inferring individual nodes or the entire graph. However, some popular aggregation techniques in GNNs suffer from a phenomenon known as over-smoothing, which occurs when the process involves numerous steps or layers.
The issue of over-smoothing was initially observed by Li et al. (2018) when studying the effectiveness of the widely used Graph Convolutional Network (GCN) model by Kipf and Welling (2017)[13]. The problem arises as the number of aggregations (or convolutions) increases, leading to node embeddings becoming practically indistinguishable from one another. Consequently, this severe decay in performance and expressive power can hinder the GNN's capabilities.
The exact cause of over-smoothing is not always straightforward and likely depends on both the specific GNN model and the data being used. Chen

et al. (2020)[5] suggest that it could result from a non-ideal information-to-noise ratio, where each node in a node classification task receives excessive noisy information from nodes belonging to different classes, weakening the inference process instead of aiding it. Moreover, if the number of aggregations greatly exceeds the size of the graph, the aggregation may end up providing very similar node representations due to significant overlap in the nodes' total receptive fields.
Various methods have been proposed to address over-smoothing with existing aggregation functions/methods, such as introducing normalization layers or making adaptive changes to the topology. Despite these efforts, GNNs generally remain shallower compared to other Deep Learning models like CNNs used in image-related tasks. In many cases, a deep GNN is not required to achieve satisfactory performance.

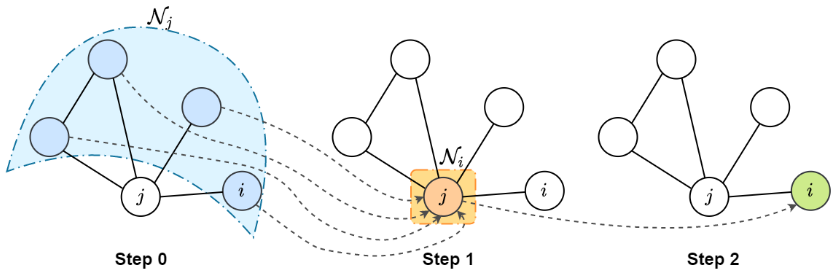

Figure 2.2: GNN propagation steps

In a typical GNN, successive propagation steps increase the receptive field for each node. As we progress from step 0 to step 1, the state of node j is updated by aggregating messages from neighboring nodes. At the transition from step 1 to step 2, node i is updated using information from node j's state at step 1, which is composed of information from j's neighbors (including i) at step 0. As a result, by step 2, node i has received information indirectly from all of the nodes in the graph.

2.2.4 Graph Convolutional Network by Kipf et al.

In order to perform aggregation in each layer, the Graph Convolutional Network (GCN) model, developed by Kipf and Welling (2017)[14], uses a feed-forward method. A localized first-order approximation of spectral graph convolution serves as the foundation of the procedure. The propagation rule can be explained as follows for the given layer 'l':

$$H^{(l)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l-1)}W^{(l)}\right) \tag{2.28}$$

In this equation, $\tilde{A}$ represents the adjacency matrix of an undirected graph, which includes self-connections. The $\tilde{D}$ matrix represents the degree matrix of $\tilde{A}$, and $H^{(l)}$ contains the node activations for the 'l-th' layer, with hidden state vectors $\boldsymbol{h}_i^{(l)}$ arranged as rows. Additionally, $W^{(l)}$ denotes the layer-specific weight matrix, and σ is the activation function used.

Upon closer examination, we can draw parallels between the propagation rule's aggregation process and that of a Convolutional Neural Network (CNN). Notably, the weights $W^{(l)}$ are shared across nodes and not node-specific, similar to how the filter weights operate in a CNN. The aggregation can be described for a single graph node as follows:

$$\boldsymbol{h}_i^{(l)} = \sigma\left(\left(W^{(l)}\right)^T \sum_{j \in \mathcal{N}_i} \tilde{L}_{ij} \boldsymbol{h}_j^{(l-1)}\right) \tag{2.29}$$

In this expression, $\boldsymbol{h}_i^{(l-1)}$ corresponds to the hidden state vector of the neighboring node j from the previous layer. The matrix $\tilde{L} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$. facilitates neighborhood aggregation.

Furthermore, the matrix $\tilde{L}$ is seen as a renormalization of the symmetrically normalized Laplacian matrix $L = I + D^{-1/2}AD^{-1/2}$. This renormalization technique mitigates the risk of numerical instabilities, as well as issues related to exploding or vanishing gradients when applying equation (2.20) repeatedly. This improvement over using L was noted by Zhou et al. in 2018[18].

2.2.5 GNN framework with convolutional ARMA filters

Bianchi et al. (2021) introduce a novel graph convolutional layer inspired by the auto-regressive moving average (ARMA) filter. Unlike traditional polynomial approaches such as those presented by Defferrard et al. (2016)[8] and Kipf and Welling (2017)[4], this ARMA GNN framework aims to offer a more adaptable frequency response, increased resilience to noise, and enhanced ability to capture the global graph structure.
Within this framework, the ARMA1 filter is represented as a recursive approximation using a series of Graph Convolutional Skip (GCS) layers, defined as follows:

$$\bar{X}^{(1)} = \sigma\left(\hat{L}XW^{(0)} + XV\right) \tag{2.30}$$

$$\bar{X}^{(t)} = \sigma\left(\hat{L}\bar{X}^{(t-1)}W + XV\right), \quad t = 2, 3, \ldots, T \tag{2.31}$$

In the given context, $X \in R^{N \times \bar{F}_{in}}$ represents the initial node feature matrix or graph signal. $\bar{X}^{(t)}$ denotes the filtered graph signal at the t-th iteration step (with a total of T iterations). σ represents a non-linear activation function, while $W^{(0)} \in R^{\bar{F}_{in} \times \bar{F}_{out}}$, $W \in R^{\bar{F}_{in} \times \bar{F}_{out}}$, and $V \in R^{\bar{F}_{in} \times \bar{F}_{out}}$ are weight parameters that can be trained.
Moreover, $\hat{L}$ is a rescaled version of the symmetrically normalized graph Laplacian, and it is defined as follows:

$$\hat{L} = \frac{1}{2}(\lambda_{max} - \lambda_{min})I - D^{-1/2}AD^{-1/2} \tag{2.32}$$

with $\lambda_{max} = 2$ and $\lambda_{min} = 0$

The output of K unique GCS stacks: information from all nodes in the graph is approximated by an $ARMA_K$ filter as an average.

$$\bar{X} = \frac{1}{K}\sum_{k=1}^{k} \bar{X}_T^k \tag{2.33}$$

The use of skip-connections in the GCS layers allows nodes to better preserve their own features, alleviating the problem of over-smoothing. The use of multiple GCS stacks allows the ARMA layer to focus on different types of node relations, and weight sharing within each stack allows for more propagation steps while keeping the number of trainable parameters at a manageable level.

2.2.6 Training and Loss function

Loss function is a function that tells us, how good our neural network model for the dataset. The intuitive way to do it is, take each training example, pass through the model to get the number, subtract it from the actual number we wanted to get and square it (because negative numbers are just as bad as positives).

$$MSE(y, \widehat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2 \tag{2.34}$$

If the Loss function is big then our network doesn't perform very well, we want as small number as possible.

2.2.7 Power system units

The units of the quantities that will be used as input features and output labels for the GNN models are shown in Table 2.2. The "per unit" (p.u.) unit of measurement denotes that system quantities are expressed as fractions of a defined base unit quantity. Each power grid in Pandapower has a single base apparent power value in MVA and several base voltage values in kV for the system's various voltage levels. Voltage magnitude for each bus is then converted from kilovolt to per unit basis using the corresponding base level in kV, and branch resistance, reactance, and charging susceptibility are converted to p.u. using both the base MVA and corresponding base Kvs.

Quantity	Unit
Active power, p	Megawatts, MW
Reactive power, q	Mega volt amps (reactive), MVAr
Apparent power, $\|s\|(s = p + jq)$	Mega volt amps, MVA
Voltage magnitude, $\|v\|$	Per unit, p.u.
Voltage phase angle, j	Radians
Current (real and imaginary), i	Kiloamps, kA
Resistance, r	Per unit, p.u.
Resistance, x	Per unit, p.u.
Total charging susceptance,b_c	Per unit, p.u.

Table 2.2: Units for relevant power system quantities.

Chapter 3

Experiment

This section presents the design of the experiments, with a focus on objective, grid configurations, and model architectures. Before presenting the design and model architectures for the different experiments (Section 3.2 and 3.3), Section 3.1 will explain how data is acquired.

3.1 Dataset construction

Similar to most of the GNN approaches mentioned in 2.2, data for the experiments are acquired by resampling values in reference grids. This thesis will consider the case grids found in Pandapower, which are based on either real-world power systems or IEEE systems. Therefore, by focusing on the data on these grids, some amount of realism is ensured. However, by randomly sampling values in the grids, the end result could end up being unrealistic. To ensure that model instances do not learn from possibly intractable variants, grids that fail to converge with Pandapower's Newton-Raphson implementation. Of course, this does not fully ensure realism, but ensure feasibility in operation. The sampling ranges are also conservative, due to difficulties in acquiring data that should satisfy the above requirement when ranges were wider. The actual sampling procedure goes as follows:

- Active and reactive load values on buses are sampled uniformly in a range going from 50 to 150 percent of the original values in the reference grid. For example, if a bus in the reference grid has an active load of 40 MW, the sampling range of this bus goes from 20 to 60 MW.
- The voltage magnitude of the generators are sampled uniformly in a range from 0.9 p.u. to 1.1 p.u.
- Generator Bus positions are sampled randomely in the grid

3.2 First Experiment

In the first task a GNN model will be trained and tested on datasets of four differently sized grids, called case39, caseIEEE30, Case57 and case89pegase in Pandapower, The names refer to the number of buses each grid has. The goal is to predict the power flow solution of these grids in terms of voltage magnitude and phase angle at buses. Subsequently, the input data will be structured as bus graphs, and GNN instances are only trained on grids of the same size. The complete dataset consists of 2000 samples of case39, caseIEEE30, case57 and case89pegase, generated with the procedure in the previous section. For each case grid, 70% of the samples is used for training and the rest is for testing, and 20% of the training set is used for validation. The aim of this experiment is to test the GNN model with different grid datasets separately. The GNN architecture is made up of two graph convolutional layers which process the input node features and their connectivity information. The activation function is applied after each convolution to introduce non-linearity. Following the convolutional layers, the output is flattened and passed through two linear layers with activation functions. The final linear layer reshapes the output to have a dimension of 2, which will provide the output in voltage magnitude and phase angle.
The model takes a data object containing node features and edge connectivity information as input. The input passes through the convolutional layers, activation functions, flattening operation, and linear layers, resulting in the final output.
For training, the model is equipped with a learning rate of 0.0001, set for 2000 epochs. Additionally, early stopping is implemented with a patience of 5, monitoring validation loss. The training loop prints training and validation losses for each epoch and halts training if no improvement is observed in the validation loss for 5 consecutive epochs.
To test the Model performance three of the grid datasets are used for training, validation and fourth grid is used for testing which will be a unseen dataset which had been never trained for the model.

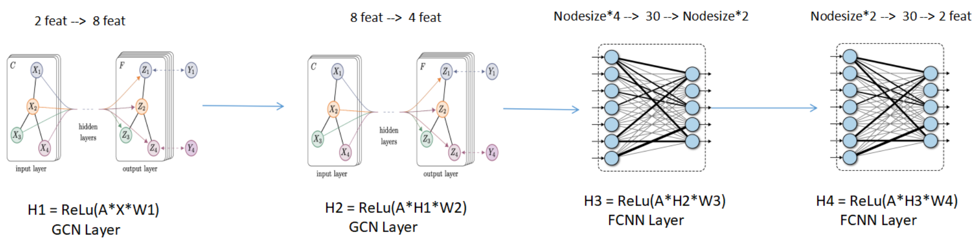

Figure 3.1: Overview of the model architecture used for the first experiment.

3.3 Second Experiment

The final task will involve samples of multiple case grids with completely different topology, where the goal is to predict the voltages and voltage angles. When training using grids with widely different topologies.

The data consists of samples based on Pandapower's case89, caseIEEE30, Case57 and case 39 Two thousand samples are made for each grid. three grids data is used for training, one of the grid is for validation and fourth one is used for testing purpose.

The GNN architecture is made up of two FCNN layers which process the input node features and their connectivity information. The activation function is applied after each FCNN to introduce non-linearity. Following the FCNN layers, the output is passed through four ARMA layers with activation functions. The final two FCNN layer reshapes the output to have a dimension of 2, which will provide the output in voltage magnitude and phase angle.

For training, the model is equipped with a learning rate of 0.0001, set for 2000 epochs. Additionally, early stopping is implemented with a patience of 5, monitoring validation loss. The training loop prints training and validation losses for each epoch and halts training if no improvement is observed in the validation loss for 5 consecutive epochs.

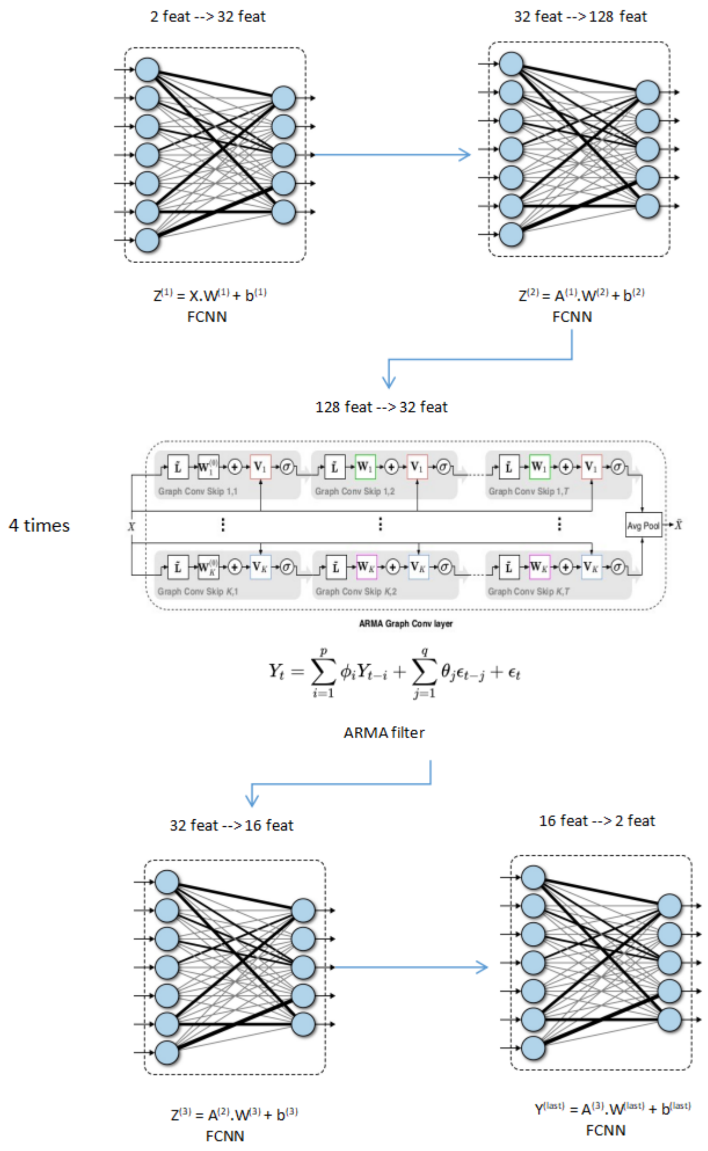

Figure 3.2: Overview of the model architecture used for the Second experiment.

Chapter 4

Results and Discussion

In this section the experimental results will be presented in order, along with a discussion for each experiment.

4.1 First Experiment

The GNN model given in Section 3.1 were trained for 2000 epochs with a learning rate of 0.0001 and a batch size of 50. For each instance, the model weights that yielded the lowest validation MSE loss during training were restored upon completion.
The MSE for single grid topology is tabulated below

Case IEEE30	Cas39	case 57	case 89pegase
0.0000515	0.0000997	0.0000400	0.0012014

Table 4.1: MSE of the different bus types

When training on case IEEE30 samples, the GNN fully converges towards the end of the training interval.The convergence is reached after 80 epochs in IEEE30 test case. The difference in values of ground truth and prediction from GNN are plotted in the figures 4.2 and 4.3 .
The loss of the model is lower on the validation dataset than the training dataset. There is some gap between the train and validation loss learning curves during the convergence. This gap is referred to as the “generalization gap”. which means model is a good fit for this case IEEE30 dataset.

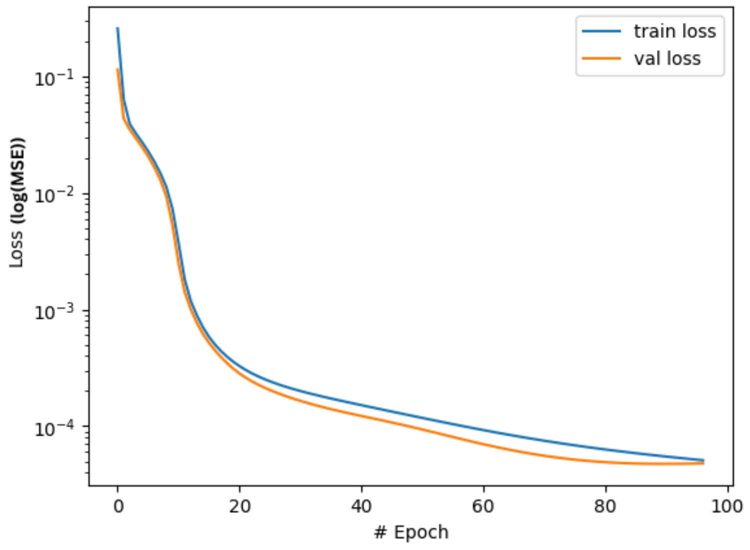

Figure 4.1: MSE plot for case IEEE30

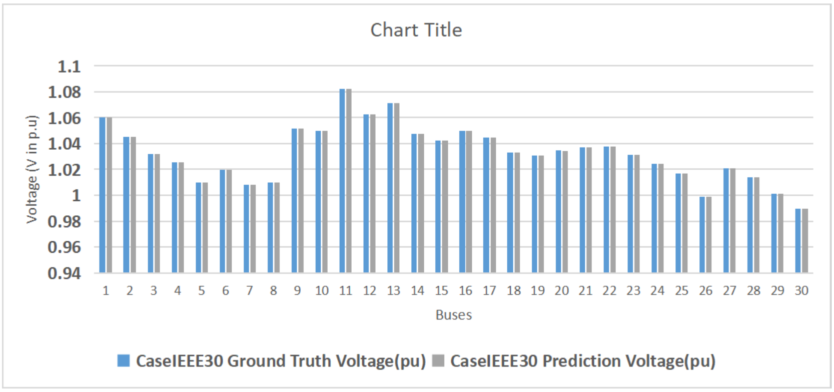

Figure 4.2: Case IEEE30 comparison of Ground truth value and prediction value of Voltage (V in p.u)

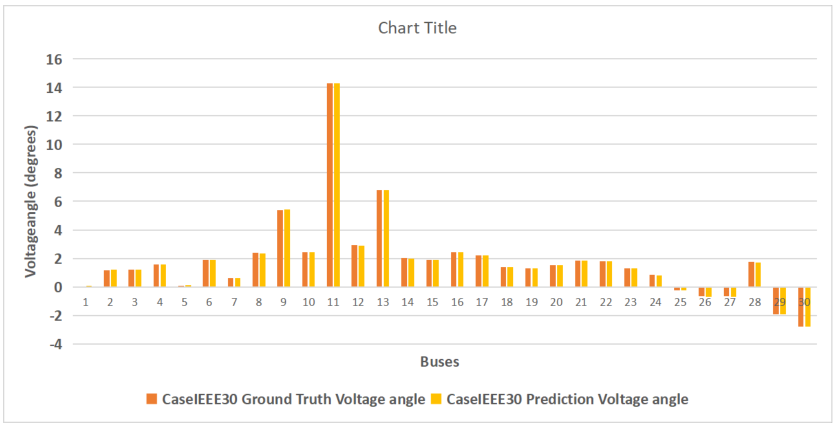

Figure 4.3: Case IEEE30 comparison of Ground truth value and prediction value of Voltage angle(degrees)

When trained on case 39 samples, the GNN more or less converges towards the end of the training interval and getting diverged and stopped due to early stopping.The convergence is reached after 25 epochs in case 39 test case. The difference in values of groundtruth and prediction from GNN are plotted in the figures 4.5 and 4.6 . Loss of the model is overfitting little bit which is stopped due to early stopping function in model.

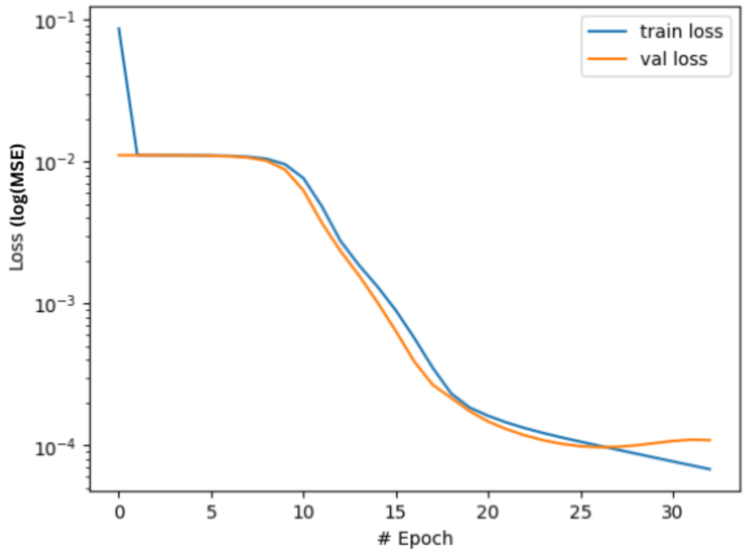

Figure 4.4: MSE plot for case 39

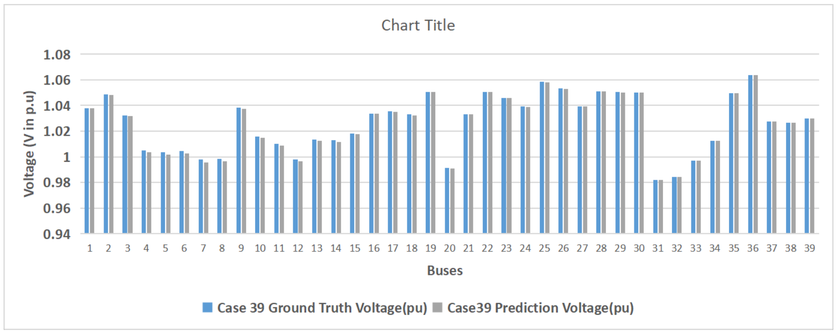

Figure 4.5: Case 39 comparison of Ground truth value and prediction value of Voltage (V in p.u)

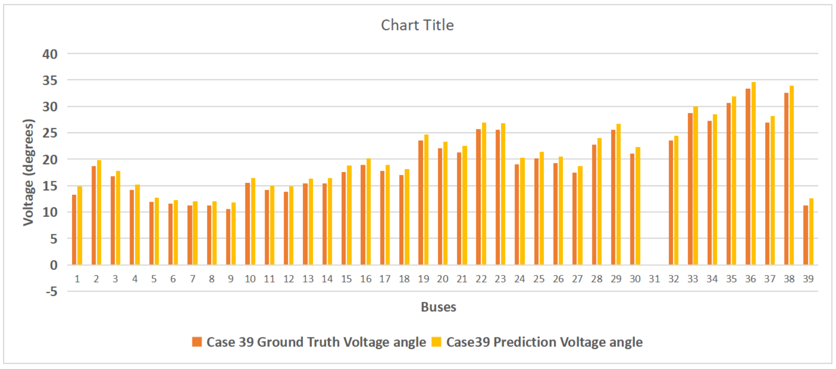

Figure 4.6: Case 39 comparison of Ground truth value and prediction value of Voltage angle(degrees)

When trained on case 57 samples, the GNN more or less converges towards the end of the training interval and getting diverged and stopped due to early stopping.The convergence is reached after 18 epochs in case 57 test case. The difference in values of groundtruth and prediction from GNN are plotted in the figures 4.11 and 4.12 . Loss of the model is overfitting which is stopped due to early stopping function in model.

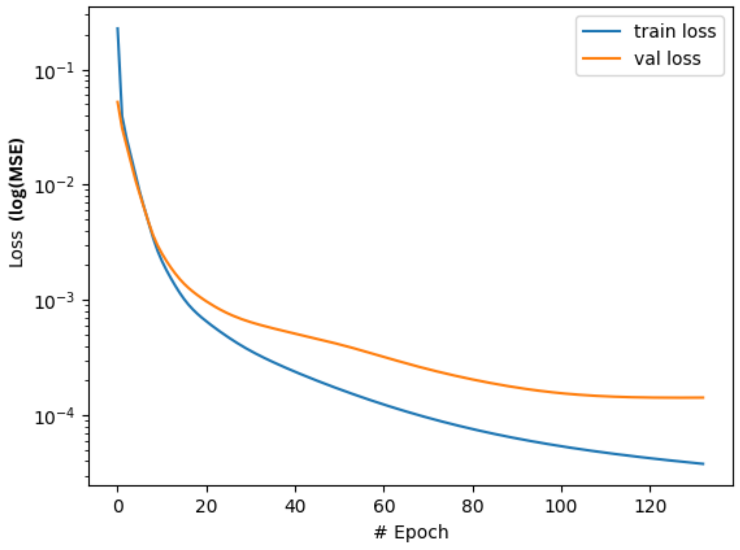

Figure 4.7: MSE plot for case 57

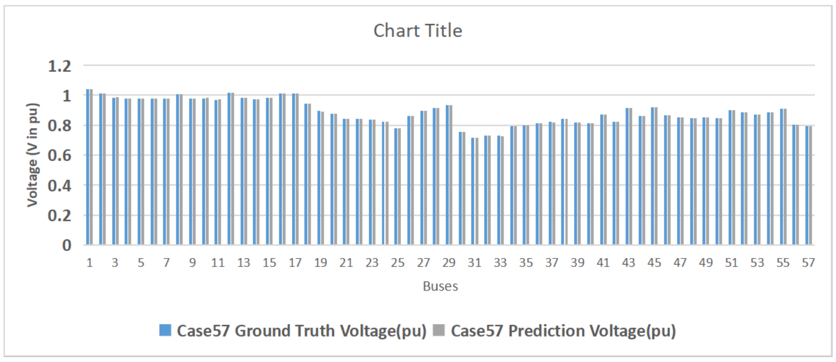

Figure 4.8: Case 57 comparison of Ground truth value and prediction value of Voltage (V in p.u)

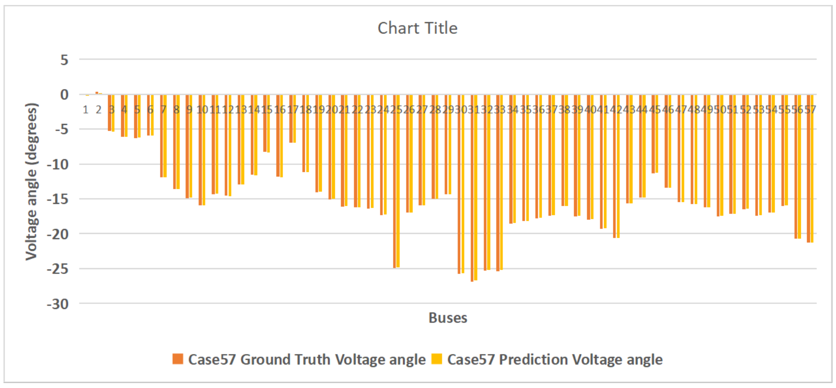

Figure 4.9: Case 57 comparison of Ground truth value and prediction value of Voltage angle(degrees)

When training on case 89 samples, the GNN converges towards the end of the training interval.The convergence is reached after 25 epochs in case 39 test case. The difference in values of groundtruth and prediction from GNN are plotted in the figures 4.11 and 4.12 . Loss of the model fully converges at 25 epoch and gap of the train loss and valid loss is small which shows that model is works perfectly for larger grids.

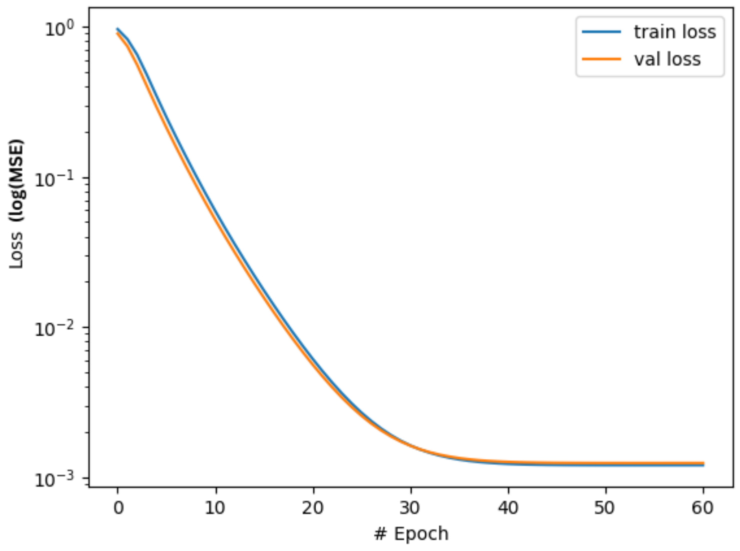

Figure 4.10: MSE plot for case 89Pegase

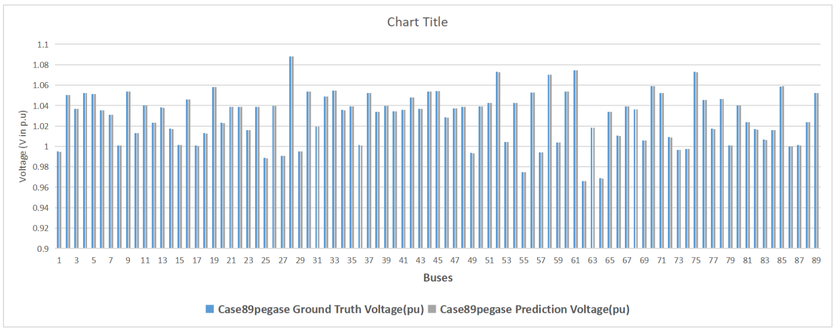

Figure 4.11: Case 89 comparison of Ground truth value and prediction value of Voltage (V in p.u)

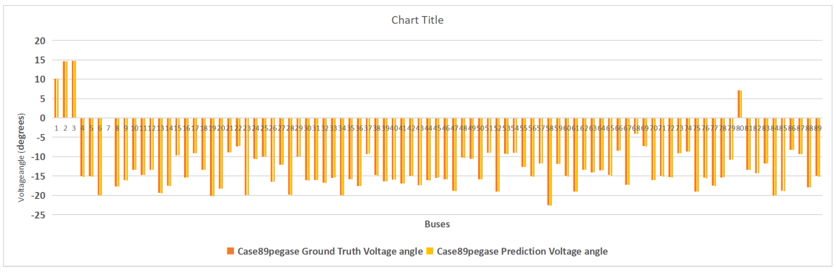

Figure 4.12: Case 89 comparison of Ground truth value and prediction value of Voltage angle(degrees)

4.2 Second experiment

The GNN model given in Section 3.2 was trained on the mixed dataset with caseIEEE30, case 39, case 57 and case89pegase testcases. Again, the model weights that yielded the lowest validation loss were restored after training to measure the performance on the test set. three grids data is used for training, one of the grid is for validation and fourth one is used for testing purpose. From MSE graph plot in figure 4.12. the MSE seems to be converging after

Mixed Dataset
1.8300723

Table 4.2: MSE of the Mixed dataset.

50 epochs, since roughly after 25 epochs the training loss continues a steady descent while validation loss slows down and almost reaches a complete halt at the end of the 50 epochs.

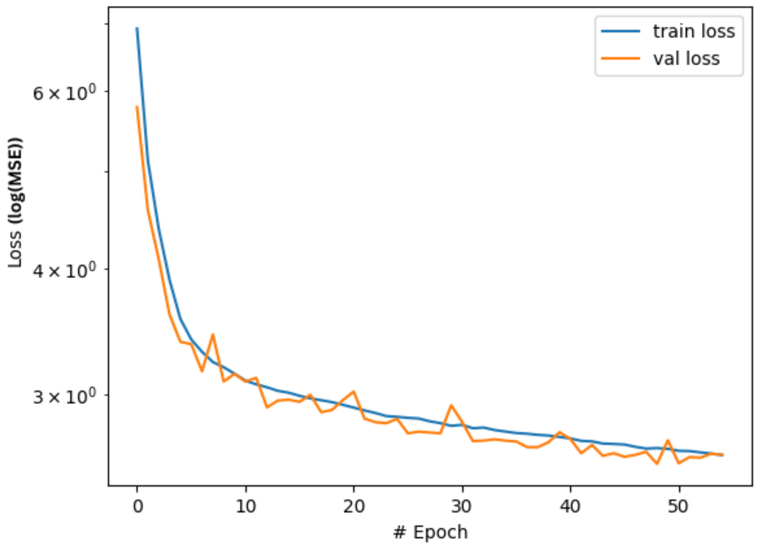

Figure 4.13: MSE plot for Mixed dataset

The Best model which is saved after the measuring the MSE is used to find the Test loss against each Test case. Give below the Test loss when the model is used against different test cases. When the model is introduced with unseen data which is not present in training data which showed how well model is likely performed. In the below scenarios the Test loss performance is not too big which shows that model performed well.

IEEE30	case39	case89pegase	case 57
0.6351079	0.5726835	0.6814936	0.325869

Table 4.3: Test loss against different Test cases

Chapter 5

Conclusion

As a whole, the experimental results show that in tasks involving either a datasets with fixed topolopy or multiple grid dataset, GNN seems very suitable for multiple grid dataset, as the usage of local operations allow these models to better learn features that are independent from the global structure and thus better generalize to different grid topologies. Still, the GNN model used in the second experiment was shown to be very successful when faced with the larger and similar size grid. Some of the trouble with generalizing to new grid topologies may stem from weaknesses in how input data is formed. In all of the experiments, system quantities such as active power injections and reactive load have been expressed in MW or MVA, while quantities such as voltage magnitude have been expressed in a per unit (p.u.) measure. For each grid, these p.u. measures are based on the base apparent power value and several base voltage levels. Thus even within the same grid, 1 p.u. voltage magnitude represents different amounts of volt or ohms in different parts of the grid, since these per unit values are computed from different base voltages. The potential problem with this is that GNNs work locally and applies the same set of operations on all graph nodes, and the models used here are not sufficiently equipped to handle such categorical differences. In case30, there is only one voltage level, which might be a part of the reason why the GNN model actually achieved better performance Another potential drawback with this approach is that the input features of the models only consisted of quantities that were resampled (with the exception of the slack bus and branch direction indicators). Since transformers on the branches are the cause of the different voltage levels, it could be beneficial to include the tap ratio value τ and the phase shift angle ϑshift as additional branch features. These values would be the same for all samples of the same grid, but their location in the grid is directly tied to the change in voltage levels and thus in the interpretation of the per unit basis. Since the GNNs work locally, these values could help in making distinctive aggregation computations for different kinds of neighborhoods in the grid, which in turn could help the generalization to different topologies. For similar reasons, other quantities that are not resampled such as shunt loads on buses, should maybe also be included in some way. It might also be beneficial to express either all or none of the quantities in p.u. The reason for this is that with the mix of units used for this project, further problems could arise if grids with different base MVA are considered, as this affects the p.u. values of resistance, reactance and charging susceptance. Also, the initial idea for the second experiment was to train the models with more than just the one topologies, but some of the other Pandapower grids such as case14 and case57 lacked base voltage values, which made it impossible to compute the branch currents in kA. Thus, it might be better to convert all quantities to p.u., including power and current. In this way, one does not need to explicitly include the base MVA and base kV values into the input features, and the values found in different grids would be constrained to more similar value distributions, which could greatly help the generalization capability. It is, however, likely that this would only be effective for the GNNs if the attributes of transformers are included in the input features, due to their connection to the different voltage levels

There are different options to further investigate the applicability of GNNs for the power flow problem. A natural extension of the work in this thesis would be to investigate whether the generalizability of a fully localized and supervised GNN can be improved by the inclusion of: a training dataset with a larger number of topologies, a full conversion to the p.u. basis, and input features representing transformers and shunt elements.

It would also be interesting to see if a GNN could be used to initialize a Newton-Raphson solver. If the GNN gives predictions that are fairly close to the solution but not fully satisfying in terms of accuracy, the total inference time of applying NR after a GNN might be lower than if NR is initialized randomly. Naturally, this is under the assumption that the GNN is able to give predictions in a reasonable time. Of course, if several grids are fed simultaneously through a disjoint union, as done in this thesis, GNN predictions can be made for many grids at a time, which should cut down the total inference time. In an ideal scenario, this procedure could also improve the overall convergence of NR, as long as the GNN does not mimic NR too closely.

Bibliography

[1] G Andersson. Modelling and analysis of electric power systems. eeh-power systems laboratory, swiss federal institute of technology (eth). 2008.

[2] Towsley D. Atwood, J. "Diffusion-convolutional neural networks. In Advances in neural information processing systems". pp. 1993–2001, 2016.

[3] Micheli A Podda M. A Bacciu D, Errica F. "Gentle introduction to deep learning for graphs.". https://pubmed.ncbi.nlm.nih.gov/32559609/, Sep, 2020.

[4] Grattarola D. Livi L. Alippi C. Bianchi, F. M. "Graph neural networks with convolutional arma filters. IEEE Transactions on Pattern Analysis and Machine Intelligence.", 2021.

[5] Lin Y. Li W. Li P. Zhou J. Sun X. Chen, D. " Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In Proceedings of the aaai conference on artificial intelligence", 2020.

[6] Sanchez-Gasca J. J Chow, J. H. Power system modeling, computation, and control. john wiley sons. 2020.

[7] Baringo-L Conejo, A. J. Power system operations. springer. 2018.

[8] Bresson X. Vandergheynst P. Defferrard, M. "Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in neural information processing systems". pp. 3844–3852, 2016.

[9] Maclaurin D. Iparraguirre J. Bombarell R. Hirzel T. AspuruGuzik A. Adams R. P. Duvenaud, D. K. "Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems". pp. 2224–2232, 2015.

[10] Schoenholz S. S. Riley P. F. Vinyals O. Dahl G. E Gilmer, J. " Neural message passing for quantum chemistry. In Proceedings of the 34th international conference on machine learning (icml)". pp. 1263–1272., 2017.

[11] Bengio Y. Courville A. Goodfellow, I. "Deep learning. MIT Press.". http://www.deeplearningbook.org, 2016.

[12] Ying Z. Leskovec J. Hamilton, W. "Inductive representation learning on large graphs. In Advances in neural information processing systems". (pp. 1024–1034), 2017.

[13] Welling Kipf, T. N. "Semi-supervised classification with graph convolutional networks. In Proceedings of the 5th international conference on learning representations (iclr).", 2017.

[14] Welling M. Kipf, T. N. "Semi-supervised classification with graph convolutional networks. In Proceedings of the 5th international conference on learning". ICLR, 2017.

[15] Gori M. Tsoi A. C. Hagenbuchner M. Monfardini G. Scarselli, F. "graph neural network model. IEEE Transactions on Neural Networks,". pp. 61–80., 2008.

[16] Seifi Sepasian. Electric power system planning: Issues, algorithms and solutions. 2011.

[17] He R. Chen K. Eksombatchai P. Hamilton W. L. Leskovec J. Ying, R. "Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th acm sigkdd international conference on knowledge discovery data mining". (pp. 974–983), 2018.

[18] Cui G. Zhang Z. Yang C. Liu Z. Wang L. . . . Sun M. Zhou, J. "Graph neural networks: A review of methods and applications.". arXiv preprint arXiv:1812.08434 ., 2018.

[19] Murillo-S´anchez C. E Zimmerman, R. D. "Matpower user's manual, version 7.1". `https://matpower.org/docs/MATPOWER-manual-7.1.pdf`, 2020.

[20] Agrawal M. Leskovec J. Zitnik, M. "Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics,". i457–i466, 2018.

List of Figures

List of Tables