UiT The Arctic University of Norway

Faculty of Science and Technology

Department of Physics and Technology

Power Flow Optimization with Graph Neural Networks

Jonas Berg Hansen

STA-3941 Master’s thesis in applied physics and mathematics 30 SP – June 2021

Abstract

Power flow analysis is an important tool in power engineering for planning and operating power systems. The standard power flow problem consists of a set of non-linear equations, which are traditionally solved using numerical optimization techniques, such as the Newton-Raphson method. However, these methods can become computationally expensive for larger systems, and convergence to the global optimum is usually not guaranteed. In recent years, several methods using Graph Neural Networks (GNNs) have been proposed to speed up the computation of the power flow solutions, without making large sacrifices in terms of accuracy. This class of models can learn localized features that are independent from a global graph structure. Therefore, by representing power systems as graphs these methods can, in principle, generalize to systems of different size and topology. However, most of the current approaches have only been applied to systems with a fixed topology and none of them were trained simultaneously on systems of different topology. Hence, these models are not fully shown to generalize to widely different systems or even to small perturbations of a given system.

In this thesis, several supervised GNN models are proposed to solve the power flow problem, using established GNN blocks from the literature. These GNNs are trained on a set of different tasks, where the goal is to study the generalizability to both perturbations and completely different systems, as well as comparing performance to standard Multi-Layered Perceptron (MLP) models.

The experimental results show that the GNNs are comparatively successful at generalizing to widely different topologies seen during training, but do not manage to generalize to unseen topologies and are not able to outperform an MLP on slight perturbations of the same energy system.

The study presented in this thesis allowed to draw important insights about the applicability of GNN as power flow solvers. In the conclusion, several possible ways for improving the GNN-based solvers are discussed.

Table of Contents

Part V / Results and Discussion 39

Part VI / Conclusions 53

Part VII / Appendix 57

Abbreviations

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
IPOPT	Interior Point Optimizer
MAPE	Mean Absolute Percentage Error
MLP	Multi-Layered Perceptron
MPNN	Message Passing Neural Network
MSE	Mean Squared Error
NR	Newton-Raphson
OPF	Optimal Power Flow

Part I / Introduction

1. Motivation

Power flow study is a fundamental part of the operation, planning, maintenance, and control of power systems (Salam, 2020). To ensure adequate system reliability for day-to-day operation, the power flow must be distributed such that the demand is met without violating the physical limits of the electrical components of the energy network. In addition, the power system should be reasonably resilient to outages and disconnections. To perform the computations necessary to find this balanced state of operation, a power solver needs information about how the electrical nodes in the power grid are connected, as well as the physical properties of the infrastructure (e.g. resistances on transmission lines).

The balanced power flow distribution of a system is found by solving a set of non-linear equations derived from Kirchhoff's current law, and these equations are most commonly solved using iterative methods such as the Newton-Raphson method (Murty, 2017). For moderately sized networks, these iterative procedures are able to find the solution at a reasonable speed. However, if applied to larger grids or to different grid configurations in quick succession, the computational complexity of these methods can render them unviable for tasks such as fast screenings of how possible contingencies will affect a grid. Another problem with these numerical optimizers is that convergence to the global optimum is in general not guaranteed and they can be sensitive to the initialization.

Some Artificial Neural Network (ANN) models have been proposed to solve the power flow problem, where presently models using Multi-Layered Perceptrons (MLPs) make up the majority. These MLP models are shown to yield decent accuracy and improved inference time compared to the traditional numerical methods, but they do not explicitly account for the topological structure of the power grids, which make them unable to generalize to differently structured grids. In particular, many of the MLP models found in literature work in a centralized/global setting, i.e. the perceptron units are tied to specific electrical buses and/or transmission lines. The features learned are then directly tied to the global structure of the grid, which means that the models become very sensitive to any perturbation of this structure.

The arbitrary and non-Euclidean structure of power grids is perhaps best described using mathematical graphs, where data is represented as a set of nodes with edges that connect them. Among ANNs, Graph Neural Networks (GNNs) is a set of models within relational deep learning that can perform regression and classification tasks on graphs and, in opposition to MLPs, these networks are able to perform localized operations and learn parameters that are independent from the global graph structure. This independence means that GNNs scale well to larger power grids and can possibly be trained on smaller grids and then deployed on larger ones.

Presently, some models using GNNs have been proposed for different variants of the power flow problem, but most use a setup where a model is trained and tested only on the same grid topology. And for the few models where trained instances are tested against differently sized grids, the training is still done using equally sized grids. In other words, they are trained on only one grid topology and then tested on other topologies. Also, none of these GNNs have been trained and tested on perturbations of a set topology. For instance, line outages can be simulated by removing lines from the power system, which is useful in contingency analysis (Idema, Lahaye, et al., 2014). Thus it would be interesting to see if a GNN can be directly trained to compute the power flow solutions for such contingency systems.

In this thesis, the performance of some GNN models made from established neural network methodologies is measured for problems involving: constantly sized grids, completely different grids, and perturbations of the same grid. In addition, within these tests, the aim is to study and discuss: i) the generalization capability of GNNs to different topologies, ii) the impact of the number of GNN layers, iii) how to best represent the power system as a graph, and iv) what system quantities should be used as input and how they should be incorporated into the graph representation.

2. Problem statement

Formally, the goals for this thesis are:

- Design supervised and fully data-driven GNN models for solving the power flow problem based on established graph neural network blocks found in literature.
- Apply GNN models to a set of problems including:
  - A problem where all data points are based on a fixed grid topology. This task only requires learning features that applies to a specific set of electrical components, and is therefore a good starting point for studying the applicability of GNNs to the power flow problem. The performance against an unseen topology will also be measured, which will reveal whether features learned from one grid are transferable to a grid with a different topology.
  - Training and testing on a grid where one or two randomly selected transmission lines have been disconnected. This problem will show whether the GNN models can yield good performance for slight perturbations of a topology, which could be useful for analysing the impact of component outages in a system.
  - Training and testing on grids with very different topologies. The aim of this task is to see if an applied GNN model can learn to solve the power flow problem for a selection of grids, and to see if learning from a selection of grid topologies can improve the applicability to new, unseen topologies.
- Discuss experimental results with a focus on comparative performance to MLP models and the generalizability to different topologies.

3. Paper structure

The main body of the thesis is divided into five parts:

- **Part II** – This part presents the necessary background theory for the power flow problem, and the ANN models that will be trained to solve it.

- **Part III** – In this part the methodology of applied MLP and GNN models is presented, as well as specifics regarding graph, data and training setup.

- **Part IV** – The generation of data used in this study is covered in this part, followed by the design of each experiment which includes the exact architecture of applied ANN models.

- **Part V** – This part presents the experimental results and a discussion for each experiment.

- **Part VI** – Conclusive remarks for experimental results is given in this part, in addition to weaknesses with the approach and possible improvements. This part will also highlight some differences between the approaches in this thesis from that of a GNN in the literature. At the end, some proposals for future work is given.

  In addition, a small appendix is given in **Part VII**.

Part II / Theory

This part presents the theoretical background needed to understand the power flow problem for energy grids and the ANN models that will be trained to solve it.

Section 4 presents the theory related to the power flow problem. Section 5 gives a brief description of MLPs, while graphs and GNNs are covered in Section 6. At the end, Section 7 presents some previous Deep Learning approaches that have been applied to tasks related to power flow optimization.

4. The power flow problem

This section introduces some basic theory needed to understand the objective and structure of the power flow problem.

Section 4.1 presents a definition of a power system and the relevant quantities. Section 4.2 covers fundamental background theory for the AC power flow problem. In Section 4.3, a brief presentation of a linear approximation to the AC power flow problem is given. Finally, Section 4.4 gives equations for computing the transmission line injections in the power flow solution.

4.1. Definition of a power system

Before defining the power flow problem, some terminology and equations regarding power systems need to be established. Most of the following definitions fall in line with the ones used by MATPOWER (Zimmerman, Murillo-Sánchez, & Thomas, 2010), which is a package of free, open-source Matlab-language M-files for solving steady-state power system simulation and optimization problems, such as the power flow problem.

The basic structure of a power system consists of a set of buses (or nodes) in a grid or network, connected by components such as transmission lines, transformers and phase shifters. Here, these components will be combined in a unified branch model, following the framework of MATPOWER. This branch model consists of a standard π transmission line model, couplet in series with an ideal phase shifting transformer, and with a regular transformer located at the *from* end of the branch.

Each grid component has some associated admittance value, which is a measure of how easily it allows current to flow through it, and it also measures the dynamic effects of the component's material susceptance to polarization. The component admittance of buses and branches is thus used to relate currents and voltages in a power system, and is subsequently important for finding the optimal power routing in power flow analysis.

For each branch, we can construct a 2×2 branch admittance matrix Y_{br} which relates the complex current injections i at the *from* and *to* end of the branch with the respective complex voltages v. Mathematically the relation is given by

$$\begin{bmatrix} i_f \\ i_t \end{bmatrix} = Y_{br} \begin{bmatrix} v_f \\ v_t \end{bmatrix}, \tag{1}$$

where the subscripts f and t denote the *from* and *to* ends of the branch, respectively. The branch admittance matrix is given as (Zimmerman & Murillo-Sánchez, 2020)

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

where j is the imaginary unit, y_s is the series admittance in the π model and b_c is the total charging susceptance. The variables τ and θ_{shift} are, respectively, the tap ratio magnitude and phase shift angle of the transformer. In a system with N_l branches, four $N_l \times 1$ vectors $\boldsymbol{y}_{ff}$, $\boldsymbol{y}_{ft}$, $\boldsymbol{y}_{tf}$ and $\boldsymbol{y}_{tt}$ can be constructed from the corresponding elements in Y_{br} from each branch.

Constant power loads in the grid are represented by specified quantities of real (or active) and reactive power consumption at buses. In a circuit, active power refers to the real power that is consumed by a load, i.e. the net transfer of energy in one direction, whereas reactive power refers to energy that is only stored in the circuit, meaning it flows back and forth between source and load without transferring any net energy to the load. For each bus, the complex power load is given by

$$s_d = p_d + jq_d, \tag{3}$$

where p_d is the real or active component and q_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

$$s_g = p_g + jq_g, \tag{4}$$

where p_g and q_g are the active and reactive generator power injection, respectively.

In addition to loads and generators, MATPOWER 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

$$y_{sh} = g_{sh} + jb_{sh}, \tag{5}$$

where g_{sh} is the shunt conductance value and b_{sh} is the shunt susceptance value.

For a grid with N_b buses, all constant impedance elements of the network can be incorporated into a complex $N_b \times N_b$ bus/nodal admittance matrix Y_{bus}, formed from the elements of the admittance matrices given by (2) and the shunt admittances. This matrix relates complex bus current injections i_{bus} to the complex voltages v, and for bus k the relation is given as

$$i_{\text{bus},k} = \sum_{j=1}^{N_b} Y_{\text{bus},kj}\, v_j, \tag{6}$$

where v_j is the complex voltage of bus j.

Similarly, for the branches, two $N_l \times N_b$ branch admittance matrices Y_f and Y_t can be formed from the vectors created from each row of (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

$$i_{f,k} = \sum_{j=1}^{N_b} Y_{f,kj}\, v_j \tag{7}$$

$$i_{t,k} = \sum_{j=1}^{N_b} Y_{t,kj}\, v_j \tag{8}$$

To construct these three system admittance matrices, two sparse $N_l \times N_b$ connection matrices C_f and C_t can be 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 $\text{diag}(\cdot)$ denote an operator which creates a corresponding diagonal matrix from a vector, the admittance matrices can be formed as

$$Y_f = \text{diag}(\boldsymbol{y}_{ff})C_f + \text{diag}(\boldsymbol{y}_{ft})C_t \tag{9}$$

$$Y_f = \text{diag}(\boldsymbol{y}_{tf})C_f + \text{diag}(\boldsymbol{y}_{tt})C_t \tag{10}$$

$$Y_{\text{bus}} = C_f^T Y_f + C_t^T Y_t + \text{diag}(Y_{sh}) \tag{11}$$

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

$$s_{\text{bus},k} = v_k i^*_{\text{bus},k} \tag{12}$$

$$s_{f,k} = \left(\sum_{j=1}^{N_b} C_{f,kj} v_j \right) i^*_{f,k} \tag{13}$$

$$s_{t,k} = \left(\sum_{j=1}^{N_b} C_{t,kj} v_j \right) i^*_{t,k} \tag{14}$$

where the asterisk denotes complex conjugate.

4.2. The AC power flow problem

In the traditional formulation of the AC power flow problem (ACPF), four variables are of interest for each electrical bus k in a power system (Seifi & Sepasian, 2011). These are

$$\begin{aligned} p_k &: \text{Net active (or real) power injection} \\ q_k &: \text{Net reactive power injection} \\ |v_k| &: \text{Voltage magnitude} \\ \theta_k &: \text{Voltage phase angle} \end{aligned}$$

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

$$p_k = p_{g,k} - p_{d,k}, \tag{15}$$

$$q_k = q_{g,k} - q_{d,k}, \tag{16}$$

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 actual power flow problem consists of finding the steady-state for a system given a pattern of 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 (Andersson, 2008). Larger fluctuating changes could also occur like e.g. load demand at a buses being considerably higher during the day compared to at night and admittance on transmission lines being affected by weather conditions. However, in a shorter time frame, these variations are often so small that a static steady-state model is a justified representation of the system in practice. But in a strict sense, the objective to the power flow problem is to analyze a "snap-shot" of the system, and not its dynamic evolution (Conejo & Baringo, 2018).

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

$$g(X) = 0, \tag{17}$$

where X is a set of variables. More specifically, for each bus we get two power balance equations derived from Kirchhoff's current law (Conejo & Baringo, 2018):

$$\sum_{j=1}^{N_b} |v_k||v_j| \left(G_{\text{bus},kj} \cos\left(\theta_k - \theta_j\right) + B_{\text{bus},kj} \sin\left(\theta_k - \theta_j\right)\right) - p_k = 0, \tag{18}$$

$$\sum_{k=1}^{N_b} |v_k||v_j| \left(G_{\text{bus},kj} \sin\left(\theta_k - \theta_j\right) - B_{\text{bus},kj} \cos\left(\theta_k - \theta_j\right)\right) - q_k = 0, \tag{19}$$

where $G_{\text{bus},kj}$ and $B_{\text{bus},kj}$ are the respective real and imaginary parts of the kj-th element in the bus admittance matrix Y_{bus}. From here on, equations (18) and (19) will be referred to as the active/real and reactive power equations.

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 satisfy all of them (on average) two out of the four variables $\{p_k, q_k, |v_k|, \theta_k\}$ 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. 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.

Table 1: *Overview of the different bus types*

Bus type	Specified quantities
Reference/slack bus	$\|v_k\|$ and θ_k
PQ bus	p_k and q_k
PV bus	p_k and $\|v_k\|$

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 $N_{\text{PV}} + 2N_{\text{PQ}}$ unknowns, where N_{PV} and 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 $N_{\text{PV}} + 1$ unknown reactive power injections for generator buses can be found from the corresponding reactive power balance equations of these buses. Lastly, the net real power injection of the reference bus, which ensures that production meets consumption, is found from the real power balance equation of this bus.

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 (Chow & Sanchez-Gasca, 2020). In comparison the GS method has a linear convergence rate.

Naturally, as the number of buses and branches increase, these iterative procedures become increasingly expensive computationally, and possibly not suitable for online use. In addition, convergence is not guaranteed. To reduce the computation time and help convergence, other faster approximations could be used to warm-start the procedure. One method that can be used for this purpose is the DC power flow approximation, which is briefly covered in the following section.

4.3. DC power flow approximation

In the DC formulation of the ACPF problem the power system is linearized, yielding a non-iterative and absolutely convergent procedure. As a result the DCPF formulation is easier to solve, but the deviations from the true AC power balance for the steady-state may be substantial. Thus, the DCPF is mostly useful in tasks where the need for short calculation speeds is higher than the need for high calculation precision, e.g. in real power dispatch or power market analysis (Zhu, 2015).

The derivation of the DCPF solution is not highly relevant for the later experimental part, but the assumptions that are made differ a bit in the literature, so a relatively short presentation of MATPOWER's formulation, which is used in the experiments, is given in Appendix A for transparency. However, for the experiments, it is important to know that the DCPF always set bus voltage magnitudes to 1 *per unit* (p.u.) and reactive power flows are ignored. This will be of importance when making comparisons with the DCPF approximations in the experiments. The per unit quantity for the voltage will be a bit more thoroughly explained in Section 10.

4.4. 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, MATPOWER provides the power injections and the voltages at both ends of each branch. These are given as

$$s_f = p_f + jq_f \tag{20}$$

$$s_t = p_t + jq_t \tag{21}$$

$$v_f = |v_f|e^{j\theta_f} \tag{22}$$

$$v_t = |v_t|e^{j\theta_t} \tag{23}$$

The complex current injections at both ends then become

$$i_f = \left(\frac{s_f}{v_f}\right)^* \tag{24}$$

$$i_t = \left(\frac{s_t}{v_t}\right)^* \tag{25}$$

5. Multi-Layered Perceptrons

Among Deep Learning models, the Multi-Layered Perceptron (MLP) is perhaps one of the most straightforward and simple. At its core, an MLP consist of at set of layers with a set number of neurons or processing units representing the feature space of each layer. An MLP always has an input layer and an output layer, and at least one layer in-between called the hidden layer(s). Mathematically, the processing of an MLP with L hidden layers is given as (Alpaydin, 2014)

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \boldsymbol{x}, \\
\boldsymbol{h}^{(l)} &= \sigma^{(l)}\left(W^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \qquad l = 1, 2, ..., L, \\
\boldsymbol{y} &= \sigma^{(L+1)}\left(W^{(L+1)}\boldsymbol{h}^{(L)} + \boldsymbol{b}^{(L+1)}\right),
\end{aligned}
$$

where $\boldsymbol{x} \in \mathbb{R}^{F_{\text{in}} \times 1}$ is the input, $\boldsymbol{h}^{(l)} \in \mathbb{R}^{F_l \times 1}$ is the hidden state at the l-th layer, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{F_l \times F_{l-1}}$ is a trainable weight matrix, $\boldsymbol{b}^{(l)} \in \mathbb{R}^{F_l \times 1}$ is a trainable bias vector, $\sigma^{(l)}$ is an activation function, and $\boldsymbol{y} \in \mathbb{R}^{F_{\text{out}} \times 1}$ is the processed output. F_l is equal to the number of processing units at the l-th layer.

6. Graphs and GNNs

As mentioned in Section 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 6.1 will present a graph definition and relevant graph attributes. Section 6.2 and 6.3 will cover some general GNN theory. Section 6.4 presents one of the more widely used graph convolutional network methodologies. And finally, Section 6.5 and 6.6 present, respectively, the ECC and ARMA layer, which will be used for the GNN models in the experiments.

6.1. Graph definition

A mathematical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes (or vertices) $\mathcal{V}$ and a set of edges (or arcs) $\mathcal{E}$ 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 $\mathcal{E} \subseteq \{\{u, v\} \mid u, v \in \mathcal{V}\}$ (Bacciu, Errica, Micheli, & Podda, 2020), edges are non-oriented and the graph is said to be undirected. In opposition, if the pairs are ordered, meaning $\mathcal{E} \subseteq \{(u, v) \mid u, v \in \mathcal{V}\}$, 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 1.

The edges found in $\mathcal{E}$ can be encoded into a binary adjacency matrix $A \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, 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 1, the respective adjacency matrices are

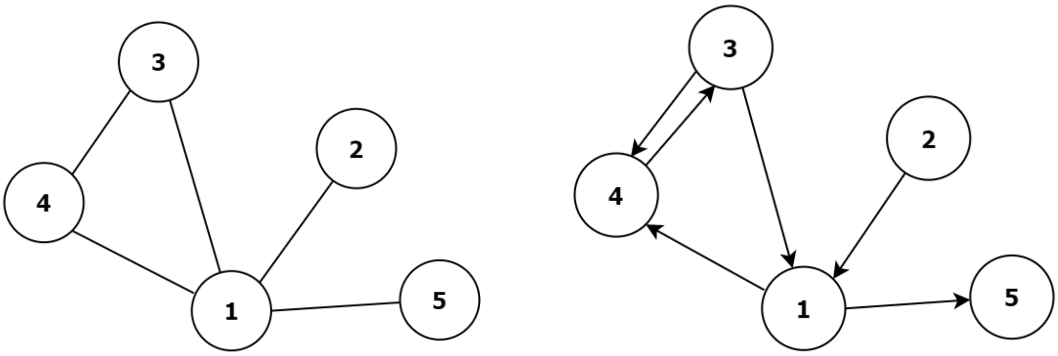
$$\text{(a)}: A = \begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 0 \end{array} \qquad \text{(b)}: A = \begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

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 1a. 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 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 $\mathcal{N}_u$, is defined as the set of nodes that have edges connecting to node u, i.e. $\mathcal{N}_u = \{v \mid \{u, v\} \in \mathcal{E}\}$. 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 $\mathcal{N}_u^{\text{out}} = \{v \mid (u, v) \in \mathcal{E}\}$ and the set of in-neighbors as $\mathcal{N}_u^{\text{in}} = \{v \mid (v, u) \in \mathcal{E}\}$. As an example, for the undirected graph in Figure 1 the neigborhood of node 1 is $\mathcal{N}_1 = \{2, 3, 4, 5\}$, while for the directed graph the out- and in-neigborhoods are $\mathcal{N}_1^{\text{out}} = \{4, 5\}$ and $\mathcal{N}_1^{\text{in}} = \{3, 2\}$.

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 $\boldsymbol{x}$, and an edge connecting node u to node v comes with a corresponding feature vector $\boldsymbol{e}_{u\to v}$ (or $\boldsymbol{e}_{uv}$). Note that for non-oriented edges $\boldsymbol{e}_{uv} = \boldsymbol{e}_{vu}$. The features in these vectors can be discrete, continuous or possibly a combination of both. Additional information about node relationships can also be included by using a weighted adjacency matrix, i.e. the 1s are replaced with continuous values which represent some "strength" measure of the edges.



(a) (b)

Figure 1: *(a) Example of an undirected graph. This graph can be transformed into a directed version by replacing each edge by two similar edges facing the opposite way of each other. (b) Example of a directed graph for which the adjacency matrix is asymmetric.*

6.2. Neural Networks using graphs

Many types of non-Euclidean real world data are most naturally represented in the form of graphs, and hence there is currently an interest in developing and improving techniques that apply Artificial Neural Networks (ANNs) to the graph domain. So far, neural network approaches using graphs have been applied to tasks such as classification of protein functions for biological protein-protein interaction graphs (Hamilton, Ying, & Leskovec, 2017), content recommendation for users on social platforms (Ying et al., 2018), prediction of side effects emerging from drug-drug interactions (Zitnik, Agrawal, & Leskovec, 2018) and document topic classification (Kipf & Welling, 2017), to name a few.

The large variety of recently developed techniques using ANNs in conjunction with graphs does, however, make it tricky to define key mechanisms and terminology that apply to all of them. This task is made even trickier by the fact that many approaches use either different terminology for similar techniques, or same terminology for differing techniques. For instance, the term graph convolution is repeated a number of times throughout the current literature, as several approaches utilize some form of convolution operator (either in the spatial/node or spectral domain) and/or incorporate some type of local receptive field similar to the ones that arise from the filters in Convolutional Neural Networks (CNNs) (Goodfellow, Bengio, & Courville, 2016). Examples include spectral approaches such as the ones seen in (Defferrard, Bresson, & Vandergheynst, 2016) and (Kipf & Welling, 2017) and spatial ones as in (Atwood & Towsley, 2016) and (Duvenaud et al., 2015). Also, the term Graph Neural Network (GNN) is often used as a generic term to refer to a model belonging to the general field of ANNs incorporating graphs in some way, while some texts reserve this term specifically for the pioneering model by Scarselli et al. (2008). In this thesis, the former definition is used.

Even though the techniques used in the developed GNNs vary, there are some main concepts which apply to many of them. The next section will present some of these shared concepts, and this presentation will rely heavily on the introduction to GNNs by Bacciu et al. (2020).

6.3. Common concepts for GNNs

In a GNN, each node is typically associated with a corresponding state vector $\boldsymbol{h}_i$, which is often updated through successive iteration steps or layers of the network. For the l-th iteration step, we can be denote the state of node i as $\boldsymbol{h}_i^{(l)}$. Commonly, the state of each node at the input layer is set equal to some input feature vector, i.e. $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i$.

To update the state of each node, information is typically spread across the graph with some form of message passing between nodes (Gilmer, Schoenholz, Riley, Vinyals, & Dahl, 2017). These messages can be computed for each node by using its current state and, possibly, the feature information on edges connected to the node. The messages are then sent to neighboring nodes, and the state of each node is updated using incoming messages and possibly its current state. When message passing is applied in successive steps, nodes receive contextual information from an area larger than their first-order neighborhood. In other words, the local receptive field of each node increases, and this process is illustrated in Figure 2. Here, through several message passing steps (or propagation steps), node i is able to receive information from all the neighbors of node j (including itself). Naturally, if nodes carry information that are relevant to far away neighbors, this propagation of information can be beneficial in both 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 (Bacciu et al., 2020)

$$\boldsymbol{h}_i^{(l)} = \phi^{(l)}\left(\boldsymbol{h}_i^{(l-1)}, \Psi\left(\{g^{(l)}(\boldsymbol{e}_{j\to i})^T\psi^{(l)}(\boldsymbol{h}_j^{(l-1)}) \mid j \in \mathcal{N}_i\}\right)\right) \tag{26}$$

Here ϕ, ψ and g represent arbitrary transformation functions, Ψ is a permutation invariant function (e.g. the sum or max operator) and $\mathcal{N}_i$ is either the open or closed neighborhood of node i. Note that this formulation allows for the inclusion of edge features (both discrete and continuous). If edge features are not included, the expression is reduced to

$$\boldsymbol{h}_i^{(l)} = \phi^{(l)}\left(\boldsymbol{h}_i^{(l-1)}, \Psi\left(\{\psi^{(l)}(\boldsymbol{h}_j^{(l-1)}) \mid j \in \mathcal{N}_i\}\right)\right) \tag{27}$$

Note that in both equation (26) and (27) the graphs are assumed to be non-positional, hence Ψ must be permutation invariant.

As recently mentioned, successive application of message passing and neighborhood aggregation expands the local receptive field of each node, which can prove beneficial when making inference of individual nodes or even for the entire graph. However, several of the most popular aggregation techniques for GNNs suffer from a so called over-smoothing of the node features when the process involves too many steps or layers. This drawback of deeper GNNs was first noted by Li et al. (2018) in a work studying the effectiveness of the popular *Graph Convolutional Network* (GCN) model by Kipf and Welling (2017). The potential issue is that as the number of aggregations (or convolutions) increase, node embeddings can end up being effectively indistinguishable from each other, which can lead to a severe decay in the performance and expressive power of the GNN. The actual reason for over-smoothing is not always crystal clear, and likely depends on both the model in use and the data. Chen et al. (2020) propose that it can be caused by non-ideal information-to-noise ratio, where in e.g. a node classification task each node can receive too much noisy information from nodes belonging to a different class that do not help inference, but rather weakens it. It is also easy to imagine that if the number of aggregations far exceeds the size of the graph, the aggregation could also end up giving very similar node representations, since the total receptive fields of the nodes greatly overlap. Several methods have been proposed to combat over-smoothing for existing aggregation functions/methods, such as a type of normalization layer (Zhao & Akoglu, 2019) and adaptive changes to the topology (Chen et al., 2020). Regardless, GNNs tend to be shallower than other Deep Learning models such as CNNs used in image related tasks, and in many cases a deep GNN is not necessary to achieve good performance.

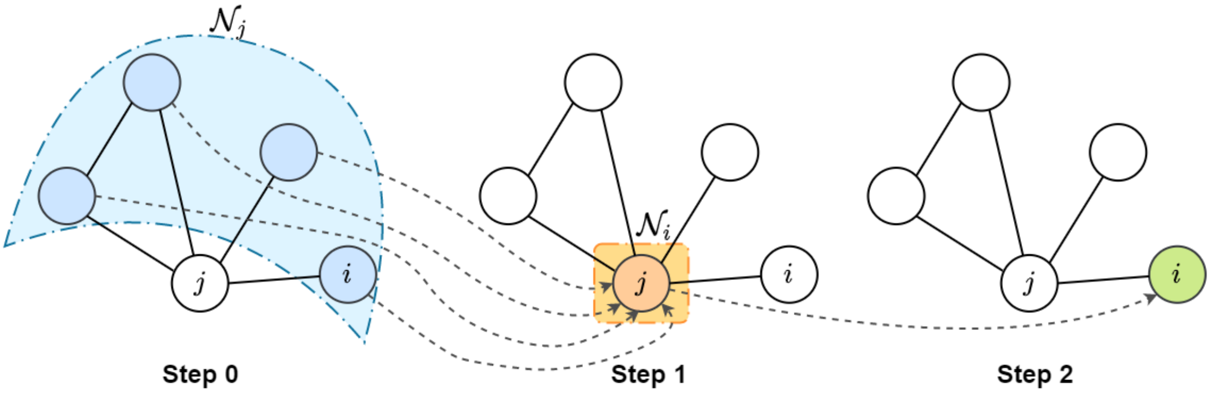


Figure 2: *Illustration of how successive propagation steps in a typical GNN increases the receptive field for each node. Going from step 0 to step 1, the state of node j is updated by aggregating messages coming from it's neighboring nodes. At the transition from step 1 to 2, node i is updated using information of the state of node j at step 1, which in turn consists of information of the state of j's neighbors (including i) at step 0. Thus by step 2, node i has indirectly received information from all the nodes in the graph.*

6.4. Graph Convolutional Network by Kipf et al.

In the Graph Convolutional Network (GCN) model by Kipf and Welling (2017) aggregation is done through a feedforward approach with a layer-wise propagation rule that is based on a localized first-order approximation of spectral graph convolution. For a given layer l, the propagation rule is given by

$$H^{(l)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l-1)}W^{(l)}\right) \tag{28}$$

Here, $\tilde{A} = A + I$ is the adjacency matrix (binary or weighted) of an undirected graph with added self-connections. I is the identity matrix with the same dimensions as A. $\tilde{D}$ is the degree matrix of $\tilde{A}$, where $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. $H^{(l)}$ is the matrix of node activations at the l^{th} layer, where the hidden state vectors $\boldsymbol{h}_i^{(l)}$ are placed as rows. Finally, $W^{(l)}$ is a layer-specific weight matrix and σ is an activation function. If one studies the arithmetic of the propagation rule a bit closer, there are clear parallels to the aggregation in a CNN. If

$\tilde{A} \in \mathbb{R}^{N \times N}$ and $H^{(l)} \in \mathbb{R}^{N \times F_l}$, then $W^{(l)} \in \mathbb{R}^{F_l \times F_{l+1}}$, i.e. the weights are not node-specific, and the aggregation is done by centering on each node and using the same set of weights, which is similar to how the filter weights work in a CNN. For a single graph node, the neighborhood aggregation in the propagation rule can be expressed as

$$\boldsymbol{h}_i^{(l)} = \sigma \left(\left(W^{(l)} \right)^T \sum_{j \in \mathcal{N}_i} \tilde{L}_{ij} \boldsymbol{h}_j^{(l-1)} \right), \tag{29}$$

where $\boldsymbol{h}_i^{(l-1)} \in \mathbb{R}^{F_{l-1} \times 1}$ and $\tilde{L} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$.

Moreover, the matrix $\tilde{L} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$ can be seen as a renormalization of the symmetrically normalized laplacian matrix $L = I + D^{-1/2} A D^{-1/2}$, and is therfore often called the renormalized graph laplacian. This renormalization trick reduces the risk of numerical instabilities and exploding/vanishing gradients for successive application of equation (28), compared to using L (Zhou et al., 2018).

6.5. Edge-Conditioned Convolution

The paper by Simonovsky and Komodakis (2017) introduces a method of performing convolution-like aggregation on local neighborhoods while exploiting labels on the edges. More specifically, the filter weights used to aggregate information from neighbors are conditioned on said edge labels. The operation itself is formalized as

$$\boldsymbol{h}_i^{(l)} = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} g^{(l)} \left(\boldsymbol{e}_{j \to i} ; \mathbf{W}^{(l)} \right) \boldsymbol{h}_j^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{30}$$

where $\boldsymbol{e}_{j \to i}$ is the feature vector of the edge going from node j to node i in the original input graph and $g^{(l)}$ is a differentiable function parametrized by a set of learnable weights $\mathbf{W}^{(l)}$.

If $\boldsymbol{h}_i^{(l)} \in \mathbb{R}^{F_l \times 1}$, $\boldsymbol{h}_i^{(l-1)} \in \mathbb{R}^{F_{l-1} \times 1}$ and $\boldsymbol{e}_{j \to i} \in \mathbb{R}^S$, then g must produce an $F_l \times F_{l-1}$ matrix given a vector of length S. So, if g is for instance based on MLPs, this can be achieved by e.g. using F_l MLPs that each output a vector of length F_{l-1} or by using a single MLP outputting a vector of length $F_l \cdot F_{l-1}$ that is then reshaped into a matrix.

In the above formulation, each node can only utilize its own features at the previous layer if the neighborhoods are closed (i.e. each node must have a corresponding self-loop). The function g might then need to treat these self-loops differently than the other edges, as it is not always obvious what edge labels should be assigned to these loops. For instance, in a power system, a graph representation could be constructed by letting buses act as graph nodes and branches as edges. Natural choices for edge features may then be quantities related to the branch admittance matrices such as resistance and reactance. However, in this case there are no physical component that can naturally represent a self-loop. Instead, edge features for a self-loop could for example be created from an average of the feature vectors of the other edges that connect to this node. The potential drawback with such an approach is that the node features that are retained after aggregation depend on these self-loop features (that are treated like the other edge features), rather than the node features themselves.

In the models used for the experiments in this thesis, an extended formulation that avoids this issue will instead be used , and it is given as (Grattarola & Alippi, 2020)

$$\boldsymbol{h}_i^{(l)} = \boldsymbol{h}_i^{(l-1)} W_{\text{root}}^{(l)} + \sum_{j \in \mathcal{N}_i} g^{(l)} \left(\boldsymbol{e}_{j \to i}; \mathbf{W}^{(l)} \right) \boldsymbol{h}_j^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{31}$$

i.e. the aggregation of each node can now also directly leverage the nodes' own features, done through a skip-connection with a trainable weight matrix $W_{\text{root}}^{(l)} \in \mathbb{R}^{F_l \times F_{l-1}}$. Technically, this formulation is more inline with the *Message Passing Neural Networks* (MPNN) framework of Gilmer et al. (2017), but henceforth it will here be referred to as an ECC layer, since the conditioning on edge features is its most distinctive attribute compared to other popular GNN frameworks that use convolution-like operators.

6.6. GNN framework with convolutional ARMA filters

Bianchi et al. (2021) present a graph convolutional layer inspired by the auto-regressive moving average (ARMA) filter. Compared to polynomial approaches such as the ones by Defferrard et al. (2016) and Kipf and Welling (2017), this ARMA GNN framework is meant to provide a more flexible frequency response, be more robust to noise, and better capture the global graph structure.

In this framework, an ARMA_1 filter is expressed as a recursive approximation using a stack of *Graph Convolutional Skip* (GCS) layers that are given as

$$\bar{X}^{(1)} = \sigma \left(\hat{L} X W^{(0)} + XV \right) \tag{32}$$

$$\bar{X}^{(t)} = \sigma \left(\hat{L} \bar{X}^{(t-1)} W + XV \right), \quad t = 2, 3, ..., T \tag{33}$$

where $X \in \mathbb{R}^{N \times F_{\text{in}}}$ is the initial node feature matrix (or graph signal), $\bar{X}^{(t)}$ is the filtered graph signal at iteration step t (of which there are T in total), σ is some non-linear activation function, and $W^{(0)} \in \mathbb{R}^{F_{\text{in}} \times F_{\text{out}}}$, $W \in \mathbb{R}^{F_{\text{in}} \times F_{\text{out}}}$ and $V \in \mathbb{R}^{F_{\text{in}} \times F_{\text{out}}}$ are trainable weight parameters. Here, $\hat{L}$ is a rescaled version of the symmetrically normalized graph laplacian, given as

$$\hat{L} = \frac{1}{2}(\lambda_{\text{max}} - \lambda_{\text{min}}) I - D^{-1/2} A D^{-1/2}, \tag{34}$$

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

An ARMA$_K$ filter is approximated as an average of the output of K unique GCS stacks:

$$\bar{X} = \frac{1}{K} \sum_{k=1}^{K} \bar{X}_k^{(T)} \tag{35}$$

The usage of skip-connections in the GCS layers allow nodes to better preserve their own features, and thus alleviate the over-smoothing problem. The usage of several GCS stacks can allow the ARMA layer to focus on different types of relations between nodes, and the weight sharing within each stack allow for more propagation steps while keeping the amount of trainable parameters at a moderate level.

7. Previous Deep Learning approaches for power flow optimization

The next sections will cover some of the Deep Learning approaches that have been proposed to solve power flow related tasks. With the exception of one approach, the below models are trained in a supervised learning setup, where some form of discrepancy is measured between predictions and given labels and stochastic gradient descent is used to iteratively update model weights in an attempt to reduce this discrepancy.

Section 7.1 present a couple of MLP approaches and Section 7.2 cover several GNN approaches.

7.1. MLP approaches

An early example of MLPs used for the power flow problem is given in (Paucar & Rider, 2002). Here, each unknown voltage magnitude and phase angle is computed using a corresponding shallow MLP. In other words, this model uses N_{PQ} MLPs to compute the unknown $|v|$ values and $N_{PQ} + N_{PV}$ MLPs to compute the unknown θ values. As input, these MLPs use the known voltage magnitudes and active power generation of PV buses, the diagonal values of the bus admittance matrix and values related to a contingency simulation. The added contingency values are meant to allow the model to take into account things like different loading scenarios, line outages and different generation voltage values. Despite this, the model as a whole can only learn to solve the problem for a specific power system, and it will not be able to scale to systems with a different number of buses.

Donnot et al. (2018) present a model based on MLPs that is capable of training on N-1 contingencies for transmission lines, i.e. power flows can be evaluated for every possible (single) line disconnection. This is done through the inclusion of conditional hidden units in the architecture, which are activated in the case of line disconnections. This technique was named "guided dropout" due to inspiration from the more standard dropout technique (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). As input, this model only uses the load and active power generation at the buses. Hence, physical characteristics on the transmission lines, such as resistance and reactance, are not included. The model is trained on N-1 contingency samples based on a reference topology, and it is shown to outperform the DCPF approximation in terms of mean-squared error for the power flows. The model is also shown to be able to generalize to N-2 situations (two disconnections) without being trained on such case samples. Despite being able to handle disconnections of one or two transmission lines, this model is not able to generalize to systems with topologies that differ more than this with respect to the reference topology, such as systems with a different number of buses or completely different line arrangement. This is due to the use of input units in the model that are connected to specific buses and lines in the reference topology.

MLPs can not explicitly make use of the topological information in power systems, and thus their scalability and generalization capability are severely limited. In theory, a single MLP used for this problem is scalable if it makes inference in a local fashion, i.e. it predicts values for each bus using variables only belonging to this bus. However, such a model would only be able to make inference on a grid with a fixed topology. In the case of a constant topology, an MLP working in a global fashion, i.e. one that takes in multiple bus variables and line variables at once, will perform better than a local one, since it can learn relations between buses. However, the complexity of the model will scale poorly with the dimension of the energy grid. Additionally, as seen in the approach proposed by (Donnot et al., 2018), global MLPs can only handle small changes in the topology if the architecture is carefully constructed. These changes in the grid are also primarily disconnections, so if components are added or more extensive reconfigurations are done, these models will likely have to be trained once again, which might squander the benefit gained from the reduced computation time compared to numerical approaches.

To overcome the limitations of MLPs, we thus desire models that more explicitly make use of the topological information found in power systems and can generalize to power systems with different topologies. This information would make scaling more straightforward, assuming the models work in a local fashion, and it could possibly lead to better generalization for fixed-size power grids, since the topology is closely related to the underlying physics of the problem. The approaches in the next section all try to make more explicit use of the grid topology to solve different power flow related problems, although this is done through quite different methodologies.

7.2. GNN approaches

Donon et al. (2019) present a GNN model that uses a so-called dual line graph representation to compute the ACPF solution of a power system with respect to the power flows on the transmission lines. In the model, an embedding step is used to map the problem from the bus-space to the line-space and aggregation is then performed on the lines using a GCN-like technique with two different line adjacency matrices, that each focus on one of the transmission line ends (origin/from end and extremity/to end). Load and generator injections on the buses are used as input and all the transmission lines are assumed to have the same properties, i.e. features like line resistance are not taken into account. The model is trained in a supervised fashion using the output of a Newton-Raphson solver as labels, and in the experiments detailed in the paper, the model is shown to outperform an MLP network and the DCPF approximation when trained and tested on randomly generated grids of constant size. It is also shown to outperform the DCPF approximation on grids with sizes ranging from 10 to 110 buses when trained on grids with 30 buses.

A follow-up to the above GNN model, dubbed the Graph Neural Solver (GNS), is given by Donon et al. (2020). For this model the goal is not to predict the line power flows, but instead the voltage magnitude and phase angle of the buses. The most unique aspect of this model is that it is trained to directly minimize the violation of Kirchhoff's law at each bus in the grid, i.e. it is trained unsupervised. To accomplish this, the model uses a combination of physical equations, neural network blocks and message passing between buses, and the architecture consists of a fixed number of layers which act like correction steps that iteratively tries to get closer to a power equilibrium. As input, the model uses a wide arrange of quantities related to load buses, shunt elements, generators and branches. This model is shown to be more accurate than the DCPF approximation when trained and tested on grids of constant size, which are generated by sampling bus and branch quantities in some selected IEEE grids. In addition, this GNS model is shown to generalize fairly well to smaller grids when trained on larger ones, but the opposite is not always the case. Also, the predictions are shown to be highly correlated to the solutions given by a Newton-Raphson solver. The principal drawback of this model may be that the reliance on physical equations makes it so that one needs a bit more familiarity with the related physics to be able to alter the model to fit a setup with for instance additional input parameters and/or a different form of grid modeling (e.g. one that does not use shunt elements to model capacitors and inductors).

Another supervised GNN approach for solving the ACPF problem is given by Bolz et al. (2019). In this approach, the goal is to output the loadings on each transmission line for the steady-state solution, and to compute predictions, a form of spatial graph convolution is used to aggregate information on the lines. The line loadings are defined as the percentage difference between the current on the line in the ACPF solution and the maximum current it can carry. The most unique feature of the aggregation methodology of this model, compared to the previous two approaches, is that it does not use pre-defined neighborhood structures through e.g. an adjacency matrix, but instead uses estimated correlations to determine which transmission lines are related to each other. This is motivated by the fact that one might not always have access to a complete adjacency matrix and the correlation estimates can uncover inherent dependencies between lines that are not directly inferred by the grid topology. Presented tests against resampled instances of an IEEE grid and two larger real world type grids show that this model is able to outperform an MLP model for the larger grids, but falls behind on the smaller IEEE grid. Although it is not explicitly mentioned, presumably, several instances of the same grid topology is needed to estimate the correlations. This means that this model cannot generalize to unseen grids with different topologies after training.

A supervised GNN model for solving the *Optimal Power Flow* (OPF or ACOPF) problem of a power system is given by Owerko et al. (2020). In short, compared to the regular PF problem, the OPF problem adds an objective function that is to be optimized for the system, and possibly some constraints on certain parameters such as power limits on generators and current limits on the transmission lines. In the paper by Owerko et al. (2020) the objective function is a sum of second order polynomial cost functions for the generators. The cost of each generator is computed from the active power injection, and the goal is to find the power injections that minimize the overall cost. In addition, there are box constraints on the bus voltage and power solutions, i.e. they have to be within a set range. The GNN that is set to find the active powers that accomplishes this, consists of polynomial aggregation filters that sort of resemble the framework by Defferrard et al. (2016). It is also worth noting that the adjacency matrices used for the aggregation are weigthed with the impedance between connected buses, i.e. the reciprocal of the corresponding element in the bus/nodal admittance matrix Y_{bus}. Two versions of the GNN is used for experimentation, where one is referred to as a local GNN that has a fully connected (FC) layer at the end which act on the representations/features of individual nodes at a time, and the second is called a global GNN, where the FC layer acts across all nodes at the same time. Naturally, the latter configuration can not be used on grids of different size as the number of processing units in the last FC layer depend on the number of nodes. As input, both models use solutions from the DCOPF approximation, which is the OPF equivalent to the approximation mentioned in Section 4.3. Experiments are performed on datasets constructed from resampled versions of the IEEE30 and IEEE118 reference grids, where the targets for supervised gradient descent learning are provided by an IPOPT solver (Wächter & Biegler, 2006). To benchmark the GNNs, a local and a global MLP is used, which have been given roughly the same structure as GNN counterparts in terms of number of layers and hidden units. Reported results show that the global GNN performs the best, with the global MLP in second, the local GNN in third and the local MLP in last.

When the load demand pushes the generation capacity of a power system to its limits, power companies can choose to reduce the consumption through deliberate shutdown of electric power in one or more parts of a power distribution system, which is generally done to prevent failures that can occur if the system becomes overloaded. This procedure is called load-shedding, and Kim et al. (2019) presents a GNN model for predicting the optimal load-shedding ratio that prevents transmission lines from being overloaded in case of a line contingency (e.g. a line outage). The GNN is based on node feature updates for the buses using the GCN framework by Kipf and Welling (2017). Training is done in a supervised setting, and to obtain targets to learn from, the task is framed as an OPF problem where the objective is to minimize the load-shedding ratio for a given load profile and contingency scenario. Note that the desired load-shedding ratio is a single number for the entire power grid, so after the GCN layers, the model uses fully connected layers to map the node states into a univariate space. The number of units in these final layers depend on the number of buses in the grid, so the same model can not be applied to differently sized grids. Like most of the above approaches, the authors behind this model have used standard IEEE systems with resampled values for training and testing. Presented results show that this GNN model is able to outperform both a global MLP and a linear regression model.

Part III / Method

Here, the methodology used to design and implement the GNN models will be presented. The applied models are implemented using Keras in Tensorflow (ver. 2.4.1, Abadi et al., 2015) and the GNNs rely on the functions and layer implementations from the Spektral library (ver. 1.0.5, Grattarola & Alippi, 2020).

Sections 8 and 9 will detail the structure of the two main set pieces of the GNNs: ECC and ARMA. Section 10 lists the units for the relevant power system quantities. In Section 11, two different graph construction methods will be presented, along with the GNN design that will be used for each of these two representations. Section 12 presents two different MLP configurations that will be used to benchmark the GNNs. Finally, in Section 13 some details regarding the training and data setup will be given.

8. ECC setup

The ECC layers will rely on the implementation in Spektral, which falls inline with Eq. (31) rather than that of Eq. (30). The function for processing edge features, denoted by g, will be an MLP network, which is the only function type that is natively supported in the Spektral implementation of the ECC layer. Moreover, this MLP network will always have trainable bias terms and ReLU activations at each layer. An activation function σ is also placed at the end of the ECC layer, so the updated node states are given by

$$\boldsymbol{h}_i^{(l)} = \sigma\left(\boldsymbol{h}_i^{(l-1)} W_{\text{root}}^{(l)} + \sum_{j \in \mathcal{N}_i} \text{MLP}^{(l)}\left(\boldsymbol{e}_{j\to i}; \mathbf{W}^{(l)}\right) \boldsymbol{h}_j^{(l-1)} + \boldsymbol{b}^{(l)}\right), \tag{36}$$

where $\boldsymbol{e}_{j\to i}$ is an edge vector containing the features of the corresponding branch. To get the correct output dimensions, the MLP has an output layer with $F_l \cdot F_{l-1}$ units, which is then reshaped into a $F_l \times F_{l-1}$ matrix before multiplication with $\boldsymbol{h}_j^{(l-1)}$.

9. ARMA setup

For the ARMA layers, the implementation will differ slightly from the one presented in the paper by Bianchi et al. (2021). The first difference is due to the way weight sharing is done in Spektral's implementation of the ARMA layer. To achieve weight sharing within each stack, a division is done after the first iteration step, where weights assigned to step $t = 2$ applies to all subsequent steps. As a consequence, the first iteration step has an unique weight matrix V for the skip connection. Formally, the implementation of the ARMA layer in Spektral is therefore

$$\bar{X}^{(1)} = \sigma\left(\tilde{L}XW^{(0)} + XV^{(0)}\right), \tag{37}$$

$$\bar{X}^{(t)} = \sigma\left(\tilde{L}\bar{X}^{(t-1)}W + XV\right), \quad t = 2, 3, ..., T, \tag{38}$$

where $V^{(0)}$ and V are two different weight matrices, rather than the same one as in Equation (32)-(33). Changing this implementation such that it matches the originally intended setup does not require much effort, but there should not be any disadvantage with Spektral's implementation, other than a small increment in the number of parameters Therefore, it is used as-is.

The second difference is that $\hat{L}$, the symmetrically normalized and rescaled graph laplacian, will be replaced by a normalized adjacency matrix, i.e.

$$\hat{L} \rightarrow D^{-1/2}AD^{-1/2}. \tag{39}$$

Tests showed that using either matrix yields similar performance after training. However, when using the laplacian the overall GNN setup becomes a bit more complicated. This is due to the ECC layer, which requires an adjacency matrix to track the location of the edges. Since the laplacian introduces self-loops, which will have no corresponding edge features, both the normalized laplacian for ARMA and another adjacency matrix without self-loops for ECC would have to be used in this case. Therefore, it is more memory efficient to only use the normalized adjacency. Also, the skip-connections present in both ARMA and ECC make the inclusion of self-loops somewhat redundant.

Each ARMA layer consist of K stacks that each have T iteration steps, which determines the number of propagation steps on the graph, and this type of layer will be denoted as ARMA_K^T. No activation function is applied after the average in Eq. (35), due to the presence of activations at the end of each stack. Also, when constructing the GNNs, several ARMA layers will be used in succession to create a receptive field around each node that covers either most of or the entire graph. The use of successive layers is mostly done to increase the capacity of the overall network with respect to the number of trainable parameters, since GNNs with only a single ARMA layer with a higher T value showed significantly worse performance in early tests. This could be due to difficulties in finding a single set of weights that manage to capture relevant features for a large number of propagation steps.

10. Power system units

Table 2 displays the units of the quantities that will be used as input features and output labels for the GNN models. The "per unit" (p.u.) measure mean that system quantities are expressed as fractions of a defined base unit quantity. In MATPOWER, each power grid has a single base apparent power value in MVA and several base voltage values in kV for the different voltage levels of the system. Voltage magnitude for each bus is then converted from kilovolt to the per unit basis through the corresponding base level in kV, and branch resistance, reactance and charging susceptance use both the base MVA and corresponding base kVs to convert to p.u.

Table 2: *Units for relevant power system quantities*

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, θ	Radians
Current (real and imaginary), i	Kiloamps, kA
Resistance, r	Per unit, p.u.
Reactance, x	Per unit, p.u.
Total charging susceptance, b_c	Per unit, p.u.

11. Graph and GNN construction

In the experiments that will be detailed in Section 4, two different graph representations of the power grids will be used. In the first representation, buses will essentially take the role of graph nodes, and branches will act as graph edges. This type of representation will be referred to as *bus graph*. In the second representation, branches will instead act as graph nodes, and buses will sort of act as edges. This representation will be referred to as *branch graph*, and can be seen as a type of line graph (Harary & Norman, 1960) form of the bus graphs. Each representation will be detailed in Sections 11.1 and 11.2, and Section 11.3 details the basic structure or design of the GNNs used for the experiments.

11.1. Bus graph representation

In the experiments presented in Sections 15 and 16, the input data to the GNN models are expressed in terms of graphs were buses are represented as nodes and branches are represented as edges. From the corresponding bus, each node is assigned the following input features: active and reactive load, active power injection and voltage magnitude set point for connected generator, and a one-hot label which indicates whether the node represents the slack bus (1 if true, 0 otherwise). If a bus does not have load values or a generator, then the corresponding node features are zero.

The topology of the power systems is represented as an undirected graph, i.e. the topology of each system is incorporated into a symmetric adjacency matrix. In reality, the branches have a form of directionality, since they have designated *to* and *from* ends. However, early tests showed that when propagating information through a directed graph, represented by an asymmetric adjacency matrix, the GNN solvers achieved worse performance. This is perhaps due to the directed versions not being strongly connected, which causes the diffusion process on the graph to become hindered. Predictions have to be made for every bus, and for semi-remote buses at the peripheral regions of the grid, the information gathering can become difficult with directed edges. Going back to the example graph in Figure 1b, if message passing is done by following the direction of the edges, then node 2 has no way of receiving information from the rest of the graph, which is not ideal for a power system where the power flow solution of all buses depend on each other. Therefore an undirected graph representation is chosen, where each branch is represented by two opposite facing edges. Each edge is associated with a vector containing the following attributes: branch resistance, reactance and charging susceptance, and a 2-dimensional one-hot label indicating the direction of the edge. In this last one-hot vector, the first element is 1 if the direction matches that of the corresponding branch, and the second element is instead 1 if the opposite is true. The branch directionality is with this encoded into the edge features.

The bullet list below summarizes the feature assignment of the bus graphs

- Node feature vector $\boldsymbol{x}$: active load p_d, reactive load q_d, active power injection from generator p_g, voltage magnitude of connected generator $|v_g|$ and slack bus indicator I_{slack}.
- Edge feature vector $\boldsymbol{e}$: branch resistance r, reactance x, total charging susceptance b_c and direction indicator I_{dir}.

All of the above (non-indicator) quantities are directly involved in the equations of Section 4, with the exception of branch resistance and reactance which are connected through the series admittance y_s:

$$y_s = \frac{1}{z_s}, \tag{40}$$

$$z_s = r + jx, \tag{41}$$

where z_s is the series impedance.

11.2. Branch graph representation

As seen in the power flow theory of Section 4.2, all buses are not treated equally in the ACPF setup. The voltage magnitude of generator buses are fixed beforehand and the slack bus introduces some ambiguity since the voltage phase angle of each bus is expressed as a difference from the reference. This causes some fundamental differences between graph nodes in the bus graphs, which a GNN will have to somehow model or take into account to make good inference for all nodes. When only training on grids with either roughly or the exact same topology and bus assignment, this does not matter that much, especially if the GNN model implements enough propagation steps to allow all nodes to communicate and/or use "graph-spanning" fully connected layers at the end, like the so-called global GNN by Owerko et al. (2020). However, if a GNN is meant to be trained on widely different grids, it is not possible to use such FC layers and the variations in local neighborhoods might become so large that the GNN is not able to accurately model the categorical differences between buses with only localized operations, even if there are features on the nodes that indicate if they represent a generator bus or the slack bus. The branches, on the other hand, are not as categorically different from each other, even across grids with heterogeneous topologies. The assignment of PQ, PV and slack bus does to some degree affect the branches too, such as the voltage level at the ends being fixed if a branch is connected to a bus with a generator, but the power and current injections on the branches are not pre-determined. These injections also represent the full power flow solution, since given these values one can find all the complex bus voltages and vice versa. Thus for a task involving a GNN solver meant to work with grids that are quite different in terms of size and topology, it might make more sense to directly make predictions for the branch power and current. This can be accomplished by using a branch graph representation, where branches act as nodes.

In this alternate representation, buses will define connections between branches, but do not represent edges in the same sense branches do in the bus graphs. Each bus can in theory be connected to an unlimited number of branches. So in these branch graphs, buses can represent multiple edges. More specifically, for branch k that connects bus i to bus j, any other branch that is connected to either bus i or j will be considered as a neighbor of branch k. Note that this results in a symmetric adjacency matrix for the branches, which in early tests proved to be more beneficial than an asymmetric one that also use the direction of the branches to define neighborhoods. An illustration of the move from a bus focused graph to a branch focused one is given in Figure 3.

Despite the change in the representation of graph nodes, the same GNNs used for bus focused graphs can be used with this alternate graph formulation, where bus features can be placed on the edges and branch features on the nodes. However, every branch is connected to two buses, so there is no need for a framework that than can handle an arbitrary number of edges with attributes such as the ECC layer. Instead, values related to the buses can be added directly to the node features. Thus for the experiment that involves this branch graph representation, each graph node will be assigned the following features:

- From corresponding branch: resistance r, reactance x and charging susceptance b_c
- From bus connected at *from* end of corresponding branch: active load p_d, reactive load q_d, active power injection from connected generator p_g, voltage magnitude of connected generator $|v_g|$ and slack bus indicator I_{slack}
- From bus connected at *to* end of corresponding branch: Same as for the *from* end bus.

11.3. GNN design

To utilize all of the most important quantities related to buses and branches, a GNN must be able to aggregate information based on node features, as well as utilizing edge features if they are included. Also, in contrast to many other tasks where GNNs have been used, information from distant neighbors may be useful for achieving accurate predictions. In a power system, it is clear from the physical equations that the ACPF solutions of all buses and branches depend on each other, since the goal is to achieve a full system power

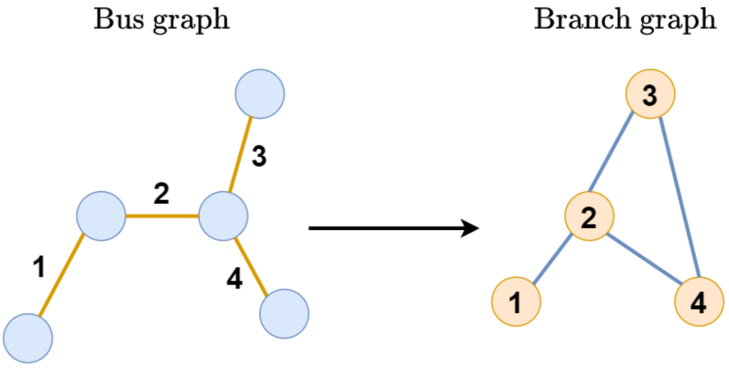


Figure 3: *Simple illustration of the transition from bus graph to branch graph.*

equilibrium. It is therefore beneficial to use a GNN that can aggregate information from a large neighborhood around each graph node, while preventing, at the same time, to over-smooth the node features.

In the models used in this thesis, the aggregation of node features will primarily be done with the use of ARMA layers. The skip connection in the GCS stacks help to prevent over-smoothing by allowing the GNN to better model sharp frequencies in the data and the sharing of weights within each stack allow for many propagation steps without a massive increase in trainable parameters which could easily lead to overfitting.

To incorporate edge features, a single ECC layer will be used to embed information from the connecting edges into the nodes. Further aggregation of node features is then intended to be done using ARMA layers. A GNN comprised only of ECC layers would utilize both node and edge features, but each ECC layer would bring a substantial number of parameters and only cover a single propagation step. Also, unlike the node features, the edge features do not change between stacked ECC layers. Therefore, not much more should be gained from repeatedly relying on the same edge features for the aggregation, compared to what is gained from the edge feature information that is embedded into the node features after the first ECC layer. Hence both of ECC and ARMA are used to build the core GNN architecture for use on graphs with edge features.

Before and after these core GNN layers, pre- and post-processing layers will be used, which are fully connected layers that act like the local MLPs described in Section 12. The intention of the pre-processing layers is to allow the model to map node features into a more meaningful representation before aggregation starts, and the post-processing layers will map the aggregated node states into the output space. Such a configuration is aligned with the study on the architectural design space of GNNs by You et al. (2020), which showed that pre- and post-processing layers could improve performance when combined with different aggregation techniques.

Because of the difference in structure between the two graph representations, the GNN design will also be slightly different. The main difference is that the ECC layer is not used when dealing with branch graphs, since both branch and bus attributes are included in the node features, i.e. there are no edge features to process in this case. Illustrations of the basic GNN structure for use with bus graphs and branch graphs are shown in Figure 4 and 5, respectively.

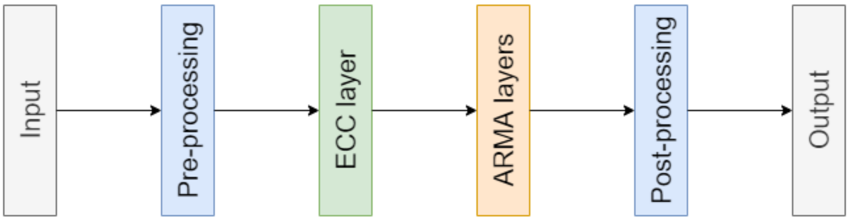


Figure 4: *Basic overview of the structure of the GNNs that are applied to bus graphs.*

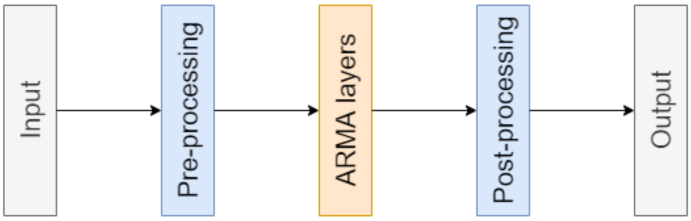


Figure 5: *Basic overview of the structure of the GNNs that are applied to branch graphs.*

12. MLP configurations

To benchmark the GNNs, comparisons will be made to the performance of basic MLP models. Each of these models will follow one of two main configurations, where the first takes in the input features of a single bus and the second takes the features from the entire grid as input. These two architectures will be referred to as *local MLP* and *global MLP*, respectively. Illustrations of the architecture of both local and global MLPs are given in Figure 6.

Since a local MLP predicts the ACPF output of a bus based only on the features of that bus, it is expected that this narrow perspective will not be enough to make accurate predictions, and it will likely struggle even more if it is trained on grids with differing topology due to a higher number of unique buses. Therefore, this configuration will only be used in an experiment involving grids of constant topology. The performance of the Local MLPs can then give a pin-point of how much underlying information in the grid that can be modeled from the one-to-one relation between a bus' features and its ACPF solution. As input, these models will use the same bus/node features as the bus graphs.

The global MLPs will, on the other hand, use features for all buses and branches as input. These features will include all of the ones used in the bus graphs, with the exception of the indicator variables. This omission comes from the fact that each input unit and output unit in the global MLPs will correspond to a specific bus or branch, so the information in these indicators would be redundant. To achieve centralized computations that utilize the entire grid in the global MLP layers, the input features of all buses are concatenated into a single one-dimensional array. The same is done for the input features of all branches, and to reduce the number of weight parameters related to the input layer, the resulting bus and branch arrays are processed by separate perceptron layers, which is illustrated by the purple units in Figure 6b. The output of these two "pre-processing" layers are then joined together and fed as input to the hidden layers of the network, given by the orange units in Figure 6b. At the end, the network outputs the full set of features for either all buses or branches in the form of a single vector, e.g. all relevant voltage magnitudes and voltage phase angles are found in this vector. With this setup, the number of input and output units depend on the grid size, so if the global GNN is meant to be applied to differently sized grids some form of padding must be used, where the input and output dimensions matches the largest size the grids are allowed to take.

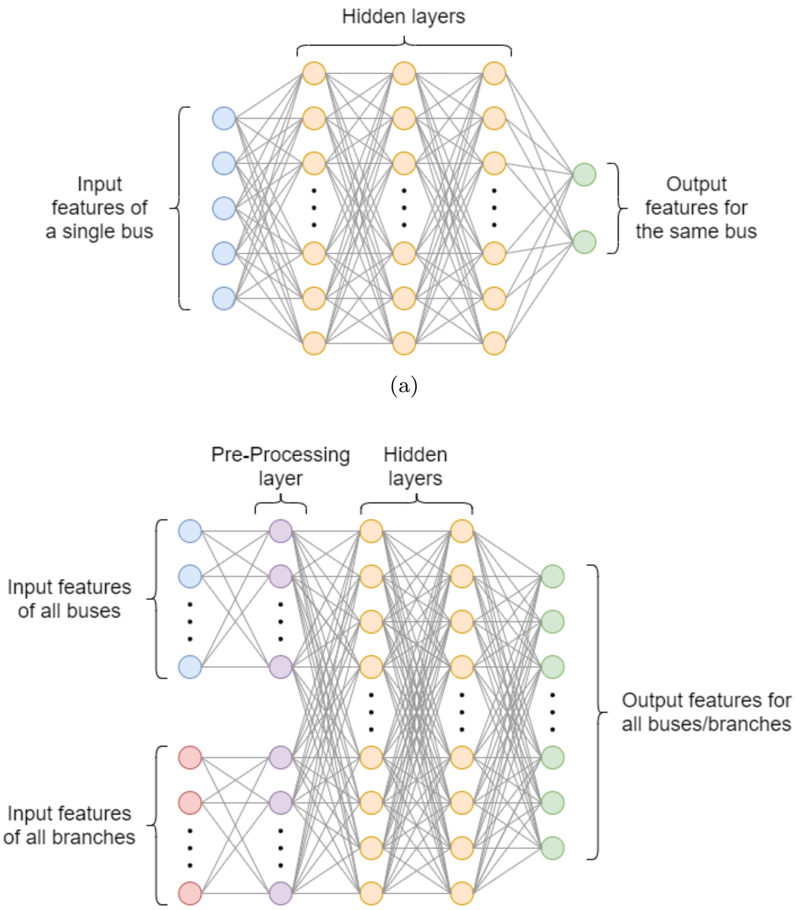
In general, the MLPs used for the experiments will not be as deep as the GNN counterparts they are meant to be compared to. This is due to the differences in how much of the grid is included when computing the output. For the GNNs, an increase in number of layers or propagation steps means that each graph node gets a larger receptive field for the computation of the predictions, and thus it makes sense to make them deeper to encompass a large part of the graph for all nodes. The local MLPs, however, only consider the features of a single node at a time, so adding more layers does not have the same impact. And for the global MLPs, a centralized perspective is used already from the first hidden layer, which means that meaningful hidden features for the entire grid should be obtainable with only a few layers. For the experiments, the number of hidden units in the MLPs are chosen such that the amount of trainable parameters is either roughly the same as or at least not several orders of magnitude larger than the amount for the corresponding GNN model. The global MLPs will deviate the most from the GNNs in this regard, since the number of parameters depend on the number of buses/branches.

13. Training and data structuring

All of the models in the experiments will be trained to minimize a supervised loss with gradient descent, where targets (or labels) are provided by MATPOWER's Newton Raphson solver. For the experiments involving bus graphs, the targets will be the voltage magnitudes of PQ buses and the voltage phase angles of all non-slack buses. In practice, the GNNs predict values for all buses, due to the localized operations. So to ignore the output associated with the fixed voltage quantities in the ACPF problem, they are masked out before computing the loss. This is also done for the local MLPs, while the global MLPs are set to only output the desired quantities.

For the task involving branch graphs, the targets will instead be the power and current injections on the branches, which are computed from the bus voltages using Eq. (20)-(25). Thus for each branch the goal will be to predict the real and imaginary components of the injections at both the *from* and *to* end, which means that eight quantities are to be predicted per branch. This applies to all branches, so there is no need for a masking operation for the GNN in this case.

The actual dimensions and structure of the input data vary between the model types. The MLPs only use 1D vectors as input, and thus the data is fed in batches as 2D matrices where the vectors are placed as rows. For the GNNs it is a bit more complicated. Tensorflow and Spektral can not do sparse operations for tensors with more than 2 dimensions, so to utilize the sparsity of the adjacency matrices, the input must be given as two-dimensional arrays. To accomplish this, a batch of graphs that are fed to a GNN is put together in a “disjoint union”, i.e. they are represented by a single graph. For the node features this is done by first placing the node feature vectors of each graph


Hidden layers
Input features of a single bus
Output features for the same bus
Pre-Processing layer
Hidden layers
Input features of all buses
Input features of all branches
Output features for all buses/branches

(a)

(b)

Figure 6: *Examples of the structure of a local MLP (a) and a global MLP (b). In (a), only a single bus' features are used as input and output. In (b), features from all buses and branches are used as input, and output features are computed for either all buses or branches depending on the use case. The number of layer units and hidden layers shown here are only for illustration purposes and do not accurately reflect the MLP models used in the experiments.*

as rows in a matrix. The resulting matrices for the graphs are then stacked on top of one another to form a single matrix. The same applies to the edge features, where the number of rows correspond to the total number of edges for all the graphs combined. Sparse operations can then be achieved by placing the adjacency matrices (or the normalized counterparts) as blocks in a sparse block diagonal matrix, where the placement corresponds with the ordering in the node and edge feature matrices. Utilizing sparse operations is naturally more efficient both computation- and memory-wise, and this disjoint union gives a straightforward way of training GNNs with a dataset containing graphs of different size. An illustration of the resulting disjoint union is shown in Figure 7.

As for training configuration, all models are trained with the ADAM optimizer using Tensorflow's default parameters and the loss is computed as mean squared error (MSE):

$$\mathrm{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{42}$$

where $\boldsymbol{y}$ are the actual labels and $\hat{\boldsymbol{y}}$ are the predictions made by the model.

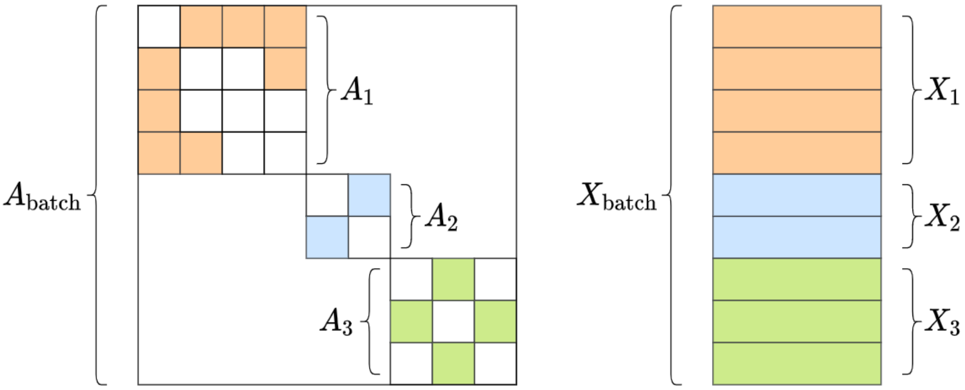


Figure 7: *[Based on a figure in the documentation of the Spektral library] Illustration of how the disjoint union between graphs is constructed. On the left: an adjacency matrix is formed from the graph batch by placing the individual adjacency matrices as blocks in a sparse block diagonal matrix. On the right: Node features from each graph are stacked on top of each other, following the same order as in* A_{batch}.

Part IV / Experimental Design

This section presents the design of the experiments, with focus on objective, grid configurations and model architectures, which rely on the methods of Part III. Before presenting the design for the different experiments, Section 14 will explain how data is acquired.

14. Dataset construction

Similar to most of the GNN approaches mentioned in 7.2, data for the experiments are acquired by resampling values in reference grids. This thesis will consider the case grids found in MATPOWER, 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 MATPOWER's Newton-Raphson implementation get discarded. Of course, this does not fully ensure realism, but the grids are at least solvable. The sampling ranges are also somewhat conservative, due to difficulties acquiring data that satisfied 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 active power on generators are resampled uniformly from 75 to 125 percent of the corresponding values in the reference grid. After sampling, a check is performed to see if the active power on the PV generators exceeds the total active load demand. If this is true, the slack bus generator will be assigned a negative active power to achieve system equilibrium, i.e. it will act as a dispatchable load. While having such a dispatchable load is not necessarily unrealistic, this can happen rather often for some of the case grids, so in these cases the sampled grids are discarded to avoid creating some sort of bias in the data.

- The voltage magnitude of the generators are sampled uniformly in a range from 0.9 p.u. to 1.1 p.u.

- Resistance, reactance and charging susceptance values on the branches are resampled uniformly from 90 to 110 percent of the corresponding values in the reference grid.

For the second experiment, presented in Section 16, perturbations in the topology of a specific grid will be generated by randomly disconnecting one or two branches before the above resampling is performed. However, in this selection process, the branches connected to the slack bus are never disconnected, since this bus is vital for the ACPF solution and should not be isolated from the rest of the system. After disconnecting the branches, any buses that no longer have a connected path to the slack bus get discarded. Both ARMA and ECC can work with disconnected graphs and MATPOWER can also be used on disconnected systems, but additional considerations must be made for each of the disconnected parts in the grid. For example, each of the separated "islands" need to have generators that can meet the load consumption of the buses within that island. Keeping track of aspects such as this and implementing additional consistency checks is out of the scope of this thesis, and hence the study in the second experiment will be limited to fully connected grids obtained from perturbations of the topology of a specific system.

15. First experiment

In this first task a GNN model will be trained and tested on resampled variants of two differently sized reference grids, called case30 and case300 in MATPOWER, where 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 the 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 15 000 samples of both case30 and case300, 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 bullet points below show some of the structural differences between case30 and case300 that is not directly tied to the difference in number of buses.

Case30

- Has 6 generators.
- Has no transformers on branches and subsequently has only one voltage level.
- Between bus graph nodes there is an average shortest path length of 3.20 with a standard deviation of 1.30.
- The largest shortest path length is 6.
- All load values are non-negative

Case300

- Has 69 generators
- Has 107 transformers on branches and 13 different voltage levels.
- Between bus graph nodes there is an average shortest path length of 9.90 with a standard deviation of 3.68.
- The largest shortest path length is 24.
- Has some negative load values which could be power output from intermittent sources such as wind mills or solar panels.

The aim of this experiment is to test the GNN model on unseen data with the same topology it is trained on, as well as the performance against an unseen grid with a different topology. Also, to benchmark the GNN, comparisons are made with i) a localized MLP that makes predictions for each bus only based on the input features of that bus, ii) a global MLP model that acts in a centralized way, using at once features from all buses and branches. In theory, this setting with fixed topology is optimal for a global MLP, since all buses (and branches) are always assigned to the same input and output processing units and there is no need for padding.

The GNN architecture consists of two pre-processing layers with respective 16 and 32 hidden units, a single ECC layer with 32 hidden units and a three layered MLP with 8 hidden units in each hidden layer, two ARMA_2^5 layers with 32 hidden units each, and two post-processing layers with 16 units and 2 units, respectively. The use of two GCS stacks in the ARMA layers should allow the model to leverage different types of relations within the graphs, and in total, the GNN implements 11 propagation steps on the graph (1 from ECC and $5 \cdot 2$ from ARMA). Also, dropout is applied to the units related to the skip connections in ARMA, with a dropout rate of 0.5. This should encourage the model to learn a different set of features in each stack. Leaky ReLU activations are applied after each layer in the pre-processing stage, at the end of the ECC layer, within the ARMA stacks, and after the first post-processing layer. The negative slope coefficient α in the leaky ReLUs is set to 0.3. The last post-processing layer acts as the output layer, and thus has no activation since this is a regression task where (in theory) the output can take any real value.

The local MLP model consists of: two hidden layers with 128 units, one hidden layer with 64 units and an output layer with 2 units. The global MLP model consists of: bus and branch pre-processing layers with 64 units each, a hidden layer with 64 units and an output layer where the number of output units match the number of voltage magnitudes and voltage phase angles which are to be predicted. Leaky ReLU activation with $\alpha = 0.3$ are placed after all layers except for the last one in both MLP configurations.

Comparisons will also be made with the output of the DCPF approximations, which is provided by MATPOWER.

An overview of the model architectures is given in Figure 8.

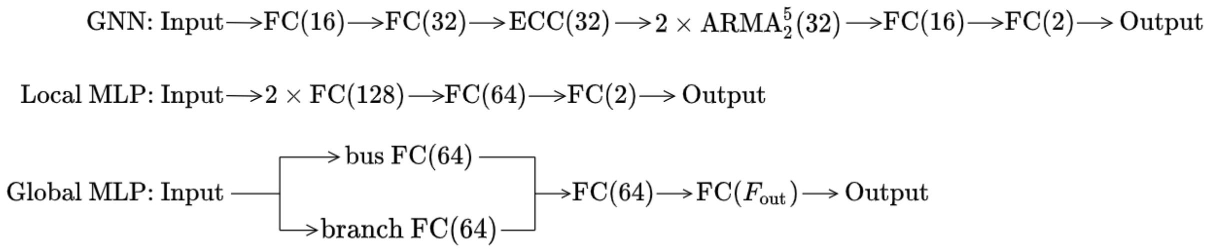

GNN: Input ⟶ FC(16) ⟶ FC(32) ⟶ ECC(32) ⟶ 2 × ARMA_2^5(32) ⟶ FC(16) ⟶ FC(2) ⟶ Output
Local MLP: Input ⟶ 2 × FC(128) ⟶ FC(64) ⟶ FC(2) ⟶ Output
Global MLP: Input
bus FC(64)
branch FC(64)
FC(64) ⟶ FC(F_{out}) ⟶ Output

Figure 8: *Overview of the model architectures used for the first experiment. F_{out} is equal to the number of voltage magnitudes and phase angles that are to be predicted, i.e. $F_{out} = N_{PQ} + N_{non\text{-}slack}$.*

16. Second experiment

In the second experiment, the goal will be to train the neural network models to compute ACPF bus voltage predictions for perturbations of a grid topology obtained by disconnecting some branches. This experimental setting emulates realistic scenarios where a power line is disconnected from the grid due to faults or maintenance operations. The case300 grid is chosen to be the reference for this task due to its moderately large size and overall complexity, and disconnections and resampling follow the procedure described in Section 14. The dataset consists of 30 000 samples, the training/testing split is 70/30, and 20% of the training set is used for validation.

Compared to the architecture in the previous experiment, the GNN model will now have four ARMA$_2^5$ layers instead of two. The total number of propagation steps on the graph is thus increased to 21, which is not enough to create a receptive field (i.e., the size of the graph neighborhood accounted by the GNN layers) that covers the whole grid when centered on each possible bus, but it is enough to cover most of them. In particular, the receptive field is more than twice the size of the average shortest path length of 9.90 for the reference grid. The applied disconnections can increase the shortest path lengths, but on average they only increase the path length approximately by 0.5%.

As for the MLP models, the local MLP is not included since it is very unlikely that it will be able to make good inference with a perspective that only cover a single bus at a time when the underlying topology changes. The global MLP model, on the other hand, will still be applied and it is given the same architecture as in the previous experiment. Also, comparisons will again be made with the DCPF approximations from MATPOWER.

An overview of the model architectures is given in Figure 9

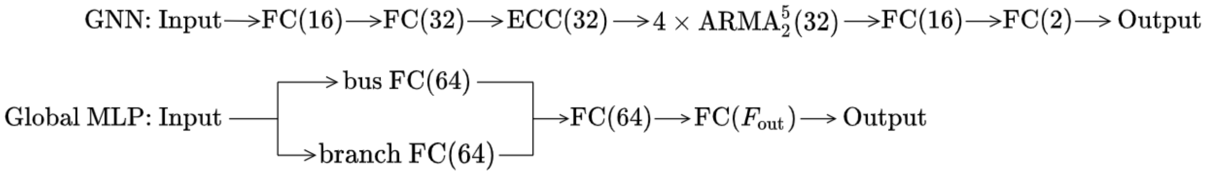

GNN: Input→FC(16)→FC(32)→ECC(32)→4 × ARMA$_2^5$(32)→FC(16)→FC(2)→ Output
Global MLP: Input
bus FC(64)
branch FC(64)
→FC(64)→FC(F_{out})→ Output

Figure 9: *Overview of the model architectures used for the second experiment. F_{out} is equal to the number of voltage magnitudes and phase angles that are to be predicted, i.e. $F_{out} = N_{PQ} + N_{non\text{-}slack}$.*

17. Third experiment

The final task will involve samples of multiple case grids with completely different topology, where the goal is to predict the power and current injections on the branches. The shift in perspective from buses to branches is done due to the more invariant nature of the branches. When training using grids with widely different topologies, the different set of PV buses and slack bus could cause problems for a GNN if the bus graphs are used. Especially for prediction of θ values, which are sensitive to the choice of slack bus, since the angles are given as offsets to this bus. Representing data as branch graphs should alleviate this issue, and the power and current injections is perhaps more dependent on the immediate neighborhood of each branch. Still, some dependence on the voltage level of the slack bus will remain and its power output isn't known beforehand, so its relative location will still be of some importance for each branch.

The data consists of samples based on MATPOWER's case9, case30, case39 and case118. Ten thousand samples are made for each grid (no disconnections, only resampled values). Each grid case sample set is split into training, validation and testing sets following a 56/14/30 percent division. The respective sets for each case are then combined, giving complete train/valid/testing sets with equal proportions of each case grid. Also, a similar dataset of ten thousand instances of case300 is made, but this one is meant for testing purposes only.

As input, the applied GNN model now uses the aforementioned branch graphs of the grids, and as output it is meant to give the real and imaginary power and current injections at both ends of each branch. Thus the GNN model is given mostly the same architecture as in the second experiment, where the only difference is that the ECC layer is removed since the branch graphs do not have edge features and the final FC layer has 8 output units instead of 2.

Also the global MLP model mostly retains its structure from the previous experiment. The only difference here is related to the number of input and output units, which must be padded. It is, in this case, technically still feasible to have units corresponding to specific buses or branches, where units also corresponds to specific grids. However, this would yield an enormous amount of units, and the purpose of this experiment is to see if a neural network can generalize to grids with multiple topologies without relying on such one-to-one relationships between processing units and buses/branches. Instead, the input and output units are partitioned into padded segments, each corresponding to a specific feature. So, for instance, the bus part of the input layer has a segment of units reserved for active load values at buses and another for active power injections of connected generators. Similarly the branch part of the input layer has segments for each of the different branch input features and the output layer has a segment for each of the eight branch output features. The size of each padded segment is equal to whatever is needed to fit any of the grid samples. To avoid having more units than necessary for this experiment, the dimensions of case118 is used to find the maximum number of units needed, since this grid has the highest number of load buses, generators and branches. When a smaller grid is given, unused parts of the segments are filled with zeros. During training, a masking operation is applied to the output units before computing the loss to prevent the inclusion of predictions for more branches than the grid has.

Although only small changes are made compared to the previous experiment, an overview of the model architectures is given in Figure 10.

The power and current injections in MW/MVA and kA used here have a much wider range than the voltage values used in the two previous experiments. In addition, the distributions for the inputs and branch labels differ in both shape and width between the different case grids. Thus z-score normalization (subtract mean and scale to unit variance) is applied to both input features and output labels, where the mean and standard deviation are computed across the training samples from all of the grids. This is a common operation in Machine Learning that encourages weights to be smaller (in absolute value) and balances out the rate at which weights learn, which can speed up convergence when training models (LeCun, Bottou, Orr, & Müller, 1998).

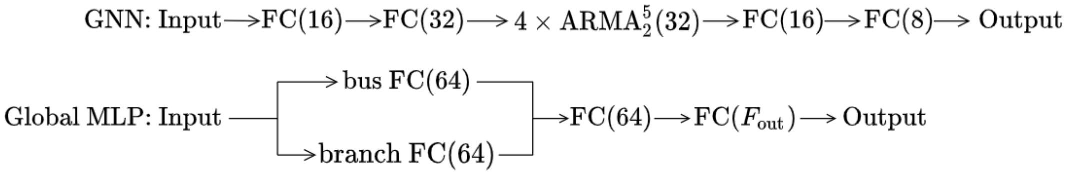

GNN: Input → FC(16) → FC(32) → 4 × $\text{ARMA}_2^5(32)$ → FC(16) → FC(8) → Output
Global MLP: Input
bus FC(64)
branch FC(64)
FC(64) → FC(F_{out}) → Output

Figure 10: *Overview of the model architectures used for the third experiment. Here,* $F_{out} = 8N_{br}$.

Part V / Results and Discussion

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

18. First Experiment

Ten randomly initialized instances of the GNN, local MLP and the global MLP model from Section 15 were trained for 100 epochs with a learning rate of 0.001 and a batch size of 16. For each instance, the model weights that yielded the lowest validation MSE loss during training were restored upon completion.

Besides the MSE defined in (42), which is also used as the loss to train the models, the mean absolute percentage error (MAPE) will also be considered as an additional metric to compare the performance of the models. The MAPE is defined as

$$\mathrm{MAPE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|, \tag{43}$$

where y_i and $\hat{y}_i$ are the true value and prediction of sample i, respectively.

Table 3 and 4 compare the performance of the different model types in terms of the total MSE, total MAPE, as well as the separate MAPE values for the predicted voltage magnitudes and voltage phase angles. The names *case30 GNN* and *case300 GNN* in the tables refer to which grid topology the GNN instances have been trained on. The MLPs have also been trained only on a single grid topology and the performance is reported in Table 3 (case30) and Table 4 (case300).

Table 3: *Performance achieved on the test set of case30. Values are given as mean plus/minus standard deviation.*

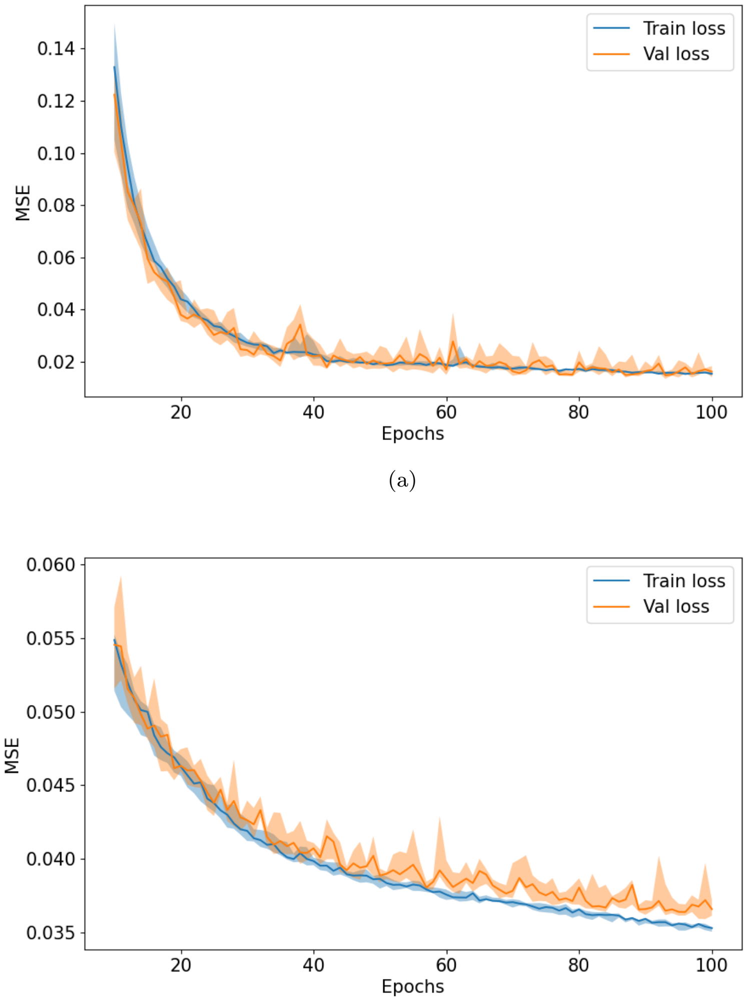
Model/Metric	Total MSE	Total MAPE	MAPE $\|v\|$	MAPE θ
DCPF	0.001275	75.25	3.98	134.23
Local MLP	0.001287 ± 0.000013	167.02 ± 15.79	3.41 ± 0.01	302.43 ± 28.84
Global MLP	0.000062 ± 0.000022	29.31 ± 6.81	0.65 ± 0.21	53.04 ± 12.45
Case30 GNN	**0.000013 ± 0.000001**	**15.32 ± 1.66**	**0.25 ± 0.02**	**27.79 ± 3.02**
Case300 GNN	0.176011 ± 0.058714	2519.50 ± 528.99	4.40 ± 0.97	4600.96 ± 967.15

Before going over the results, it should be noted that in terms of loss progression there are some differences among the models. The local MLP converges very quickly with the number of epochs, i.e. the loss settles quite early. The MSE losses of the global MLP, on the other hand, keep descending throughout all of the training epochs, although the descent is very slow at the end. For the GNNs, the convergence differ between the training datasets. When training on case30 samples, the GNN more or less fully converges towards the end of the training interval. For case300, on the other hand, the descent is noticeably

Table 4: *Performance achieved on the test set of case300. Values are given as mean plus/minus standard deviation.*

Model/Metric	Total MSE	Total MAPE	MAPE $\|v\|$	MAPE θ
DCPF	0.238885	142.13	5.65	247.57
Local MLP	0.075323 ± 0.000834	133.67 ± 3.93	4.32 ± 0.05	233.61 ± 6.99
Global MLP	**0.016763 ± 0.004707**	**33.71 ± 10.02**	4.12 ± 0.27	**56.57 ± 17.66**
Case30 GNN	2.9326 ± 1.01207	387.41 ± 96.30	138.56 ± 44.02	579.66 ± 171.09
Case300 GNN	0.038304 ± 0.000461	71.33 ± 2.86	**3.62 ± 0.04**	123.64 ± 5.06

slower overall, and the model never fully converge during the 100 training epochs. The loss progression for the global MLP and the GNN with the case300 set is shown in Figure 11. This last difference regarding the loss descent of the GNN could be due to the larger size of case300, as well as an increase in the complexity of the topology, compared to case30.



(a)

(b)

Figure 11: *Plot of the loss progression for case300 in the first experiment, starting from the 10th epoch. (a): Global MLP and (b): GNN. The darker lines show the sample median of the 10 model instances, while the lighter colored bands cover the range between the 25% and the 75% quantiles.*

As for the performance metrics, Table 3 shows clear differences between the models for the case30 testing dataset. Firstly, if we exclude the case300 GNN, the local MLP has the highest mean value in all metrics. Since the local MLP has access only to the information of a single bus to compute its output, the best it can do is to compute an output close to the mean of the training set. As an exception, the local MLP manages to outperform the DCPF approximation in terms of MAPE for voltage magnitude. This is likely due to the fact that the voltage magnitude stays mostly within the range 0.90 to 1.1 p.u. in the data. So while DCPF always gives 1 p.u., the local MLP can achieve better performance by staying near the average of the training data. The voltage phase angles in radians, on the other hand, span a much wider range, so the prediction for each bus must be more distinct to achieve a low MAPE score. From the sample standard deviation, it seems that there is a chance that a trained local MLP instance can achieve a better total MSE score than the DCPF, but not to any substantial degree.

As opposed to the local MLP, the global MLP manages fairly well in all metrics, and achieves an error well below the DCPF approximation with a performance uplift of about 95% in terms of total MSE. The contrast between the local and global MLP show the advantage of capturing the dependence between buses and branches even if the topology is always the same.

The case30 GNN also achieves good comparative performance according to each metric, even surpassing the global MLP. Interestingly, even if this model only use localized operations, it manages to capture the dependencies in the global structure in a better way than the global MLP.

However, the case300 GNN in Table 3, which is a GNN trained on case300 and then applied to case30, performs markedly worse than any of the other models and the DCPF. This indicates that the features learned from the case300 grids do not translate well to the case30 grids. In general, to achieve good performance in a transfer-learning application such as this one, it is very important that the new "unseen" data comes from a similar distribution as the training data. This is probably not the case here, since case30 and case300 are very different.

Results obtained on the case300 testing dataset are given in Table 4 and the trend here is fairly similar to that of case30. Here, the local MLP outperforms the DCPF with respect to all metrics, which could indicate that the linear DCPF approximation leads to higher inaccuracy for larger systems. The global MLP is again a clear improvement from the local one, but does not introduce the same level of improvement in terms of voltage magnitude MAPE compared to the case30 grid results. The case300 GNN only surpasses the global MLP in terms of voltage magnitude MAPE, and for the other metrics the global MLP achieves roughly 50% better performance. Once again, the GNN model trained on the other grid topology does worse than all other ANN models and the DCPF.

Some of the performance difference of the GNN model between the case30 and case300 when testing on the same grid topology used for training may be due to the different "coverage" of the grids. For the case30 samples, the GNN model manages to create a receptive field that reaches each part of the graph from each node, since the total number of propagation steps exceed the largest shortest path length of 6. In fact, for several nodes the receptive field will span the entire graph several times, which should be more than enough to capture all underlying dependencies between buses. For the case300 samples, the GNN model has enough propagation steps to cover the average shortest length, but for many buses it does not manage to reach every part of the graph.

As stated before, all buses depend on each other in the ACPF solution, so if e.g. the load value of one bus changes, this change affects the solution of all buses. Thus, compared to other types of applications on graphs, it is crucial that GNNs have large receptive fields to make the best possible inference. Also, since the phase angle is expressed as differences from the slack bus angle, it is likely very vital that the graph nodes receive (indirect) information from this bus to yield accurate predictions. This might be a major reason as to why the GNN model falls behind the global MLP in terms of the phase angle predictions for the case300 samples. In addition, the slack bus will also affect all voltage magnitudes through its role as a power slack, but this quantity also heavily depend on the other generators that are spread throughout the grid. Thus, for voltage magnitude, a decent enough accuracy can likely be achieved through exchanges of information from the more immediate surrounding. This might explain why the case300 GNN achieves the lowest MAPE value for voltage magnitude in Table 4.

As seen in Figure 11, the GNN did not fully converge when training with the case300 samples and could hence see some improvement if trained longer. However, this somewhat applies to the global MLP model as well, and it should also be noted that, compared to the GNN model, each epoch took considerably less time when training the global MLP, especially on a GPU, as the fewer and larger matrix multiplications in the MLP model are more suited for paralleled computations.

Moving back to the transfer learning aspect of the experiment, the most interesting result may be the mismatch in the metrics when comparing performance of GNNs trained on a different topology than the ones in the testing set. By looking at the total MSE only, one might get the impression that the case300 GNN learns features that better generalize to the case30 grid than vice versa. On the other hand, the total MAPE suggests that the situation is the opposite. This is likely due to the difference in magnitude for values of $|v|$ and θ, since $|v|$ is on average larger by an order or two. Therefore, when the case30 GNN does badly in terms of $|v|$ on case300 grids, it has a much more negative impact on MSE than when the case300 GNN does badly in terms of θ on the case30 grids. The difference in voltage magnitude MAPE for these cross-GNN tests is, however, difficult to explain. It could be that the GNNs trained on case300 encounter inputs that take values in wider range and, thus, manage better to give predictions closer to the average $|v|$-value when faced with case30 grids. Finally, the large difference in terms of voltage phase angle MAPE could be due to weaknesses of the metric itself. The angles of case30 are, on average, roughly ten times smaller than that of case300 and MAPE has a tendency to “blow up” for small values since even relatively minor absolute differences could result in large percentage deviations.

Overall, the failure to generalize to the other grid in these transfer learning tests is likely due to differences in the distributions for input features and output labels. Underlying differences tied to grid attributes such as voltage levels, which are not explicitly expressed in the input features of the models, may also contribute.

19. Second experiment

Ten instances of the global MLP model and the GNN model detailed in Section 16 were trained for 250 epochs with a learning rate of 0.001 and a batch size of 16. As in the previous experiment, the model weights that yielded the lowest validation loss during training were recovered upon completion.

The performance metric values obtained on the testing set are given in Table 5. At first glance, the metric values are not too different from the ones in Table 4, which is based on samples based on the same reference grid, case300. This suggests that the added branch disconnections may not have caused that much more complexity, compared to the setup of the previous experiment. However, the DCPF experiences a drop in performance that suggests some increase in complexity, but this likely varies depending on which branches were disconnected. The global MLP, on the other hand, achieves improved performance compared to the previous experiment. The same also goes for the GNN model, which falls shortly behind the global MLP.

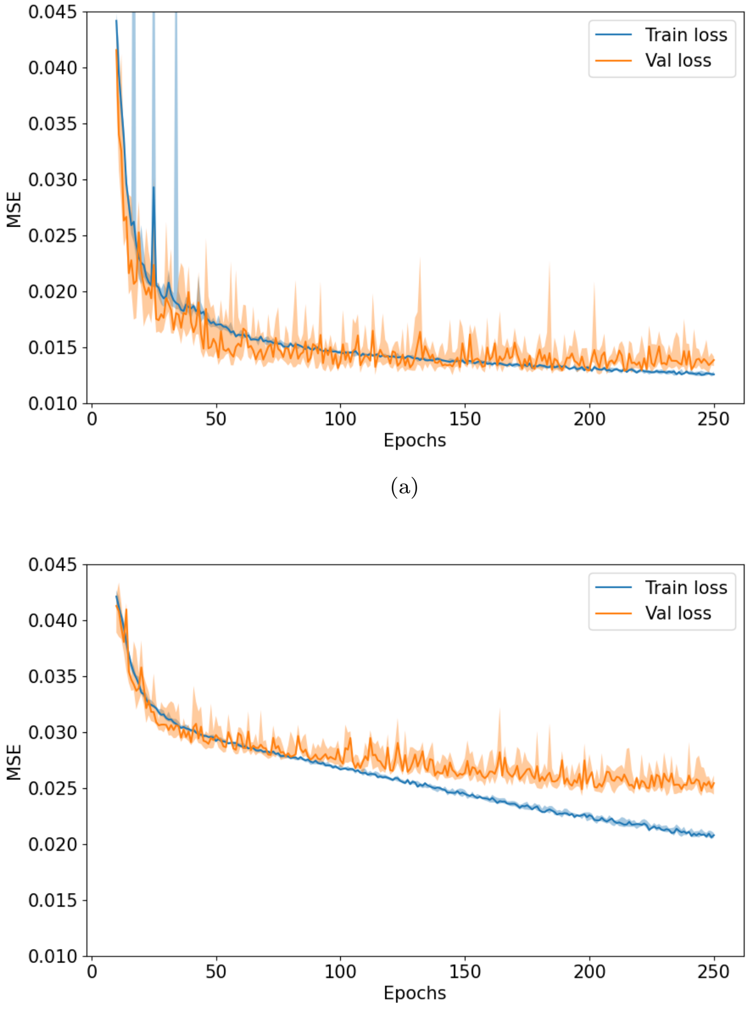
Table 5: *Performance against the testing set of the second experiment. Values are given as mean plus/minus standard deviation.*

Model/Metric	Total MSE	Total MAPE	MAPE $\lvert v\rvert$	MAPE θ
DCPF	0.2454283	165.14	5.70	288.34
Global MLP	$\mathbf{0.014861 \pm 0.002824}$	$\mathbf{32.23 \pm 5.39}$	3.90 ± 0.19	$\mathbf{54.12 \pm 9.53}$
GNN	0.027452 ± 0.000706	59.04 ± 3.66	$\mathbf{3.49 \pm 0.03}$	101.96 ± 6.49

The loss progression of the two models is shown in Figure 12. In terms of training convergence, the loss of the global MLP seems to almost fully settle within the 250 training epochs and manage to get below that of the previous experiment. Curiously, there is not the same improvement in the progression of the GNN loss. Instead, the GNN seems to be on the brink of overfitting, since roughly after 100 epochs the training loss continues a steady descent while validation slows down and almost reaches a complete halt at the end of the 250 epochs.

The improvement in the metrics and loss progression for the global MLP is probably mostly due to the doubled size of the train set and the higher number of training epochs. These two things is likely also why the GNN loss improves, in addition to the increase in propagation steps from having two additional ARMA layers. The difference in training progression between the global MLP and the GNN is, however, harder to explain. Both models are trained from the same data following the same training/validation split, so judging by the global MLP loss, there does not seem to be a bias between the training and validation set. Hence, the behavior of the GNN must be caused by something else.

It is also doubtful that the added disconnections is the cause, since the increasing gap between validation and training loss should then have manifested in both models. If anything, the disconnections could have slightly aided the generalization. For instance, in the global MLP model, each bus and branch has corresponding processing units in the input layer, so when branches (and possibly buses) are disconnected, these units receive



(a)

(b)

Figure 12: *Plot of the loss progression in the second experiment, starting from the 10^{th} epoch. (a): Global MLP and (b): GNN. The darker lines show the sample median of the 10 model instances, while the lighter colored bands cover the range between the 25% and the 75% quantiles.*

zeros as input. This process is sort of similar to a dropout procedure (Srivastava et al., 2014), where the model in this case should learn to not fully rely on all branches and buses being present when performing the global computations, which could translate into a higher generalization capability when faced with new configurations of the topology. The same should also somewhat apply to the GNN, as the disconnections should lead to more diverse neighborhood configurations, which in turn should encourage the model to learn more versatile weights for aggregation on nodes. Thus, an extended study with different reference topologies is probably needed to reveal the reason behind the overfitting trend of the GNN.

The maintained success of the global MLP is because all of the configurations caused by disconnections involve the same buses and branches. If instead additions of one or two new buses and/or branches are made, the global MLP would likely struggle, even if padded since it would be unfeasible to equip the model with input units that correspond to any conceivable expanded topology. One could alternatively add optional input and output units that do not correspond to specific configurations but rather to some arbitrary extra buses or branches. Still, the input units would then no longer be associated with the buses and branches of a single fixed topology, so the model would not be as powerful anymore. It might also be that if more disconnections are made, the permutations of which buses and branches remain become so high that the GNN model instead performs better, since it is better equipped to learn features that are independent of the global structure.

The performance metrics displayed in the Tables 3, 4 and 5 are good for comparing the performance between models, but do not reveal whether there is a substantial difference in the error obtained at each bus. To perform such an analysis, Figure 13 shows the distribution of the nodal MSE averaged over the 10 trained GNN instances. The plot shows error measures only for the buses the model instances have been trained to predict, where the MSE of buses that don't fall into this group is set to zero. To highlight the different bus types, the PV buses are marked with a yellow outline and the slack bus is given a red outline.

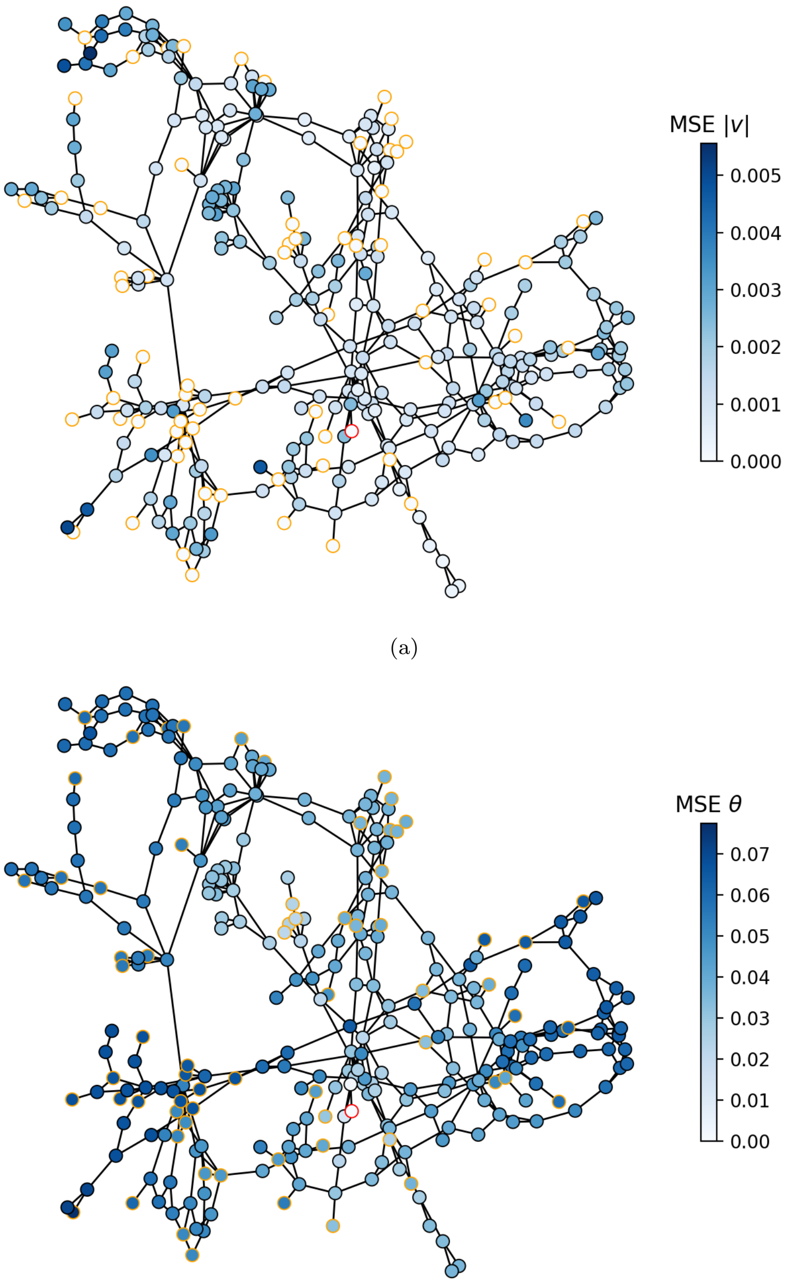
From the MSE graph plot in Figure 13, it seems that the MSE values for voltage magnitude is somewhat more uniform than for voltage phase angle. Again, it should be noted that the value range for the voltage magnitude $|v|$ labels is a lot narrower than for the phase angle θ. So, the GNN model likely struggles more with lowering the overall error for the θ predictions. In general, it seems that the MSE increases the further away one goes from the slack bus, but this trend seems stronger for θ than for $|v|$. This is visualized in Figure 14, which shows the average MSE as a function of the shortest path length from the slack bus.

In addition to the path length from the slack bus, the amount of variation in the magnitude of the labels could explain why the MSE varies among the buses. Figure 15 shows graph plots of the measured standard deviation for the output labels, and there seems to be a strong correlation between these plots and the ones in Figure 13.

To analyze whether the disconnections also contribute to the MSE differences, Figure 16 shows the number of times each bus is disconnected in the complete dataset. Here, there is some correlation to the MSE values of voltage magnitude in Figure 13a. However, there are some regions that do not match, for instance, a chain of nodes in the lower part of the grid are disconnected fairly often but have relatively low MSE values in Fig. 13a. Hence there is reason to believe that the disconnections have not negatively affected the predictive power of the GNN.

20. Third experiment

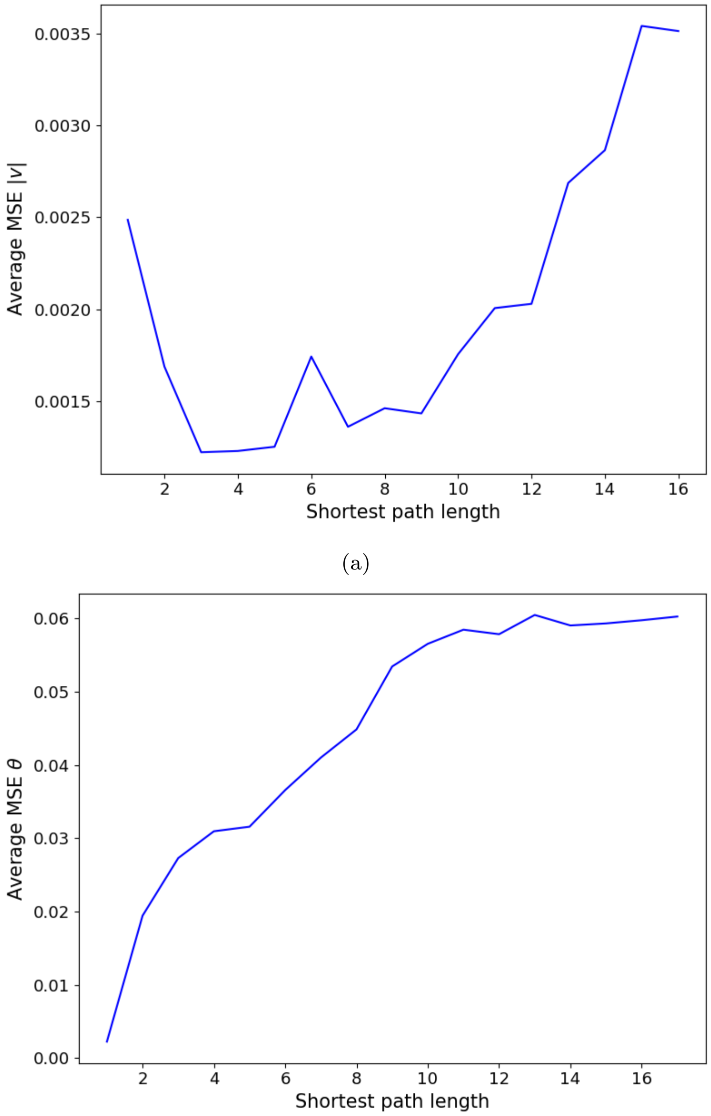
Ten instances of the global MLP model and the GNN model given in Section 17 were trained on the mixed dataset with resampled variations of case9, case30, case39 and case118. The training parameters were the same as for the previous experiments, with the exception of batch size which was increased to 64 to speed up training with this larger dataset. Again, the model weights that yielded the lowest validation loss were restored after training to measure the performance on the test set.



(a)

(b)

Figure 13: *Graph plots for the GNN performance showing the node/bus distribution of MSE values for (a): voltage magnitude and (b): voltage phase angle. In addition, PV buses are marked with a yellow outline and the slack bus with a red outline.*



(a)

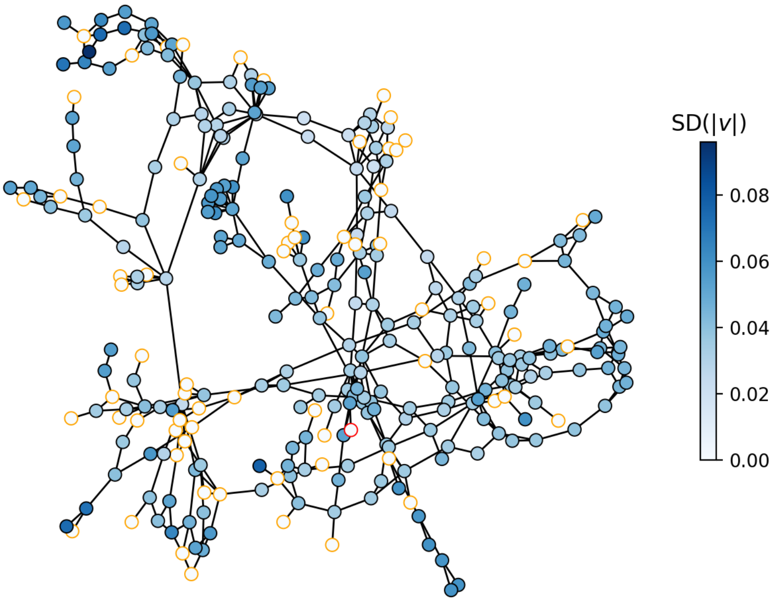
(b)

Figure 14: *Average (nodal) MSE values as functions of the shortest path length from the slack bus. (a): MSE values for voltage magnitude and (b): MSE values for voltage phase angle.*

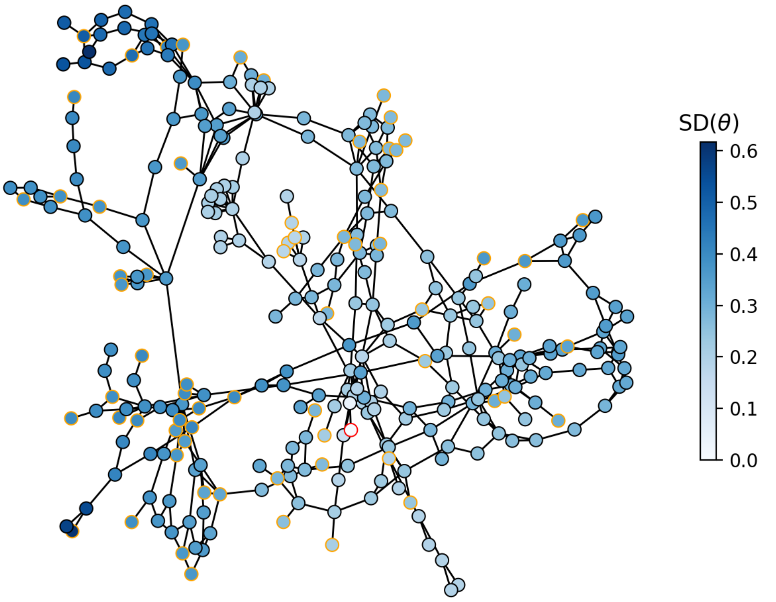
Table 6 lists the MSE values for the branch predictions in the original (unnormalized) scale. Unlike in the second experiment, the GNN now outperforms the global MLP in terms of MSE. In fact, the global MLP is even outmatched by the DCPF approximation which only makes predictions for active injections.

Plots of the loss progression for the two ANN models are given in Figure 17. This time, the global MLP instances start to overfit after roughly the 25th epoch, and the GNN instances maintain a steady descent for both training and validation loss during the entire training interval.

The drop in performance for the global MLP is likely due to the difference in the problem setting, since the MLP now does not have processing units that correspond to a specific topology. Instead, the model tries to find a global set of operations for widely different topologies using padded input and output layers where the units are only feature-specific.



(a)



(b)

Figure 15: *Graph plots showing the nodal sample standard deviation for the output labels in the complete dataset. (a): voltage magnitude and (b) voltage phase angle.*

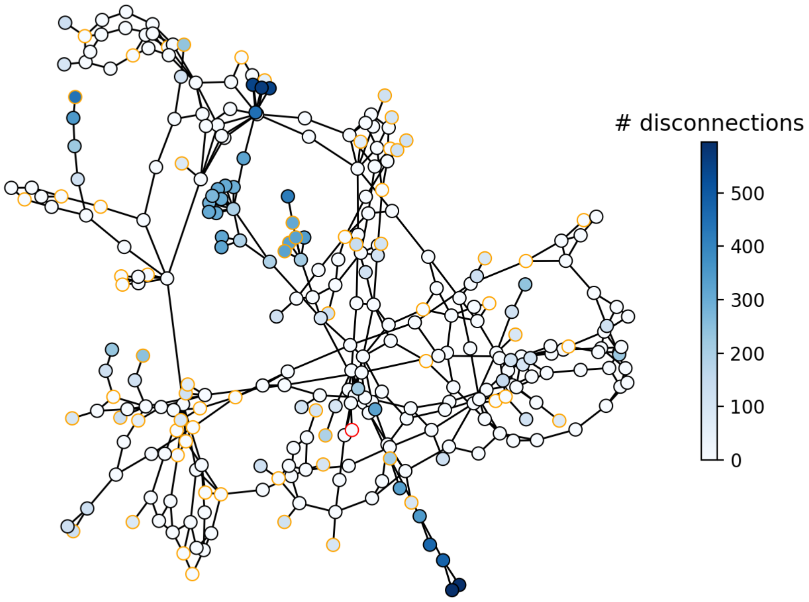


Figure 16: *Graph plot showing the number of times each bus is disconnected in the complete dataset.*

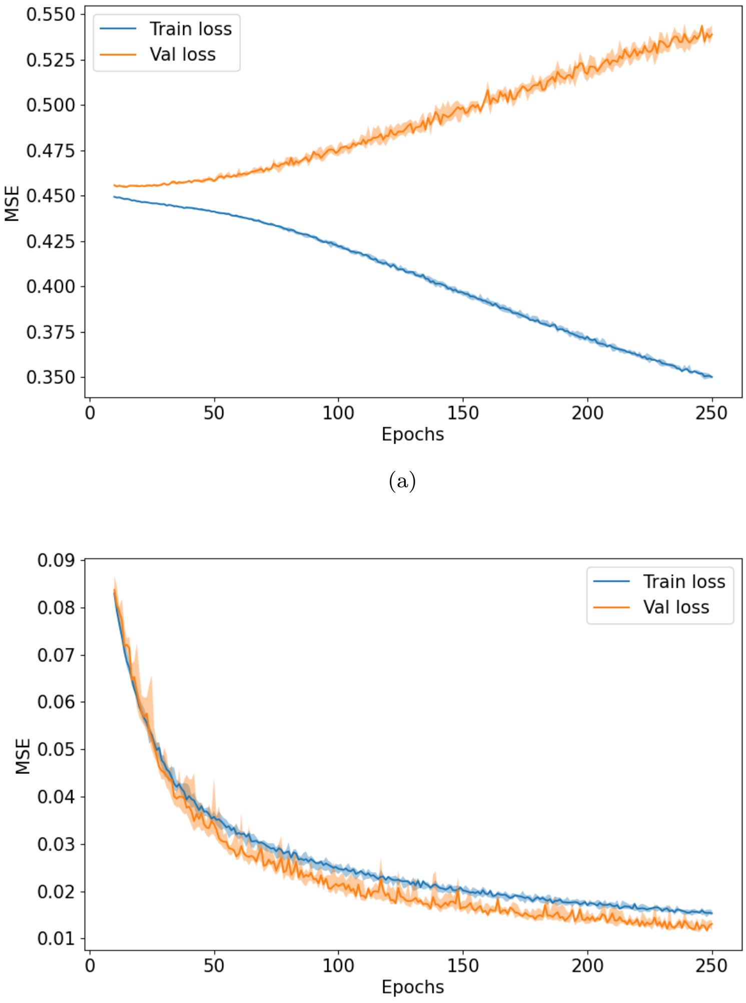
Table 6: *MSE values for the predictions of the testing dataset given as mean ± standard deviation. The metrics are computed in the original scale where power is given in MW/MVA and current is in kA.*

Data/Model	DCPF	Global MLP	GNN
case9	262.13	2617.71 ± 1001.89	$\mathbf{52.03 \pm 5.43}$
case30	167.30	3309.75 ± 440.94	$\mathbf{13.30 \pm 1.25}$
case39	6001.99	13461.70 ± 4614.44	$\mathbf{272.56 \pm 23.02}$
case118	4174.21	7261.94 ± 963.00	$\mathbf{100.67 \pm 6.84}$
Total	3764.94	7565.36 ± 1017.39	$\mathbf{114.46 \pm 6.68}$

This makes it much more difficult to capture dependencies between buses and branches, and as seen in Figure 17a, the model starts to overfit quite quickly.

For both the ANN models and the DCPF, the magnitude order of MSE for the different grid topologies is the same. The most interesting result may be that the MSE for case39 is the highest in all cases, instead of case118 which is larger and possibly has a more complex topology. To explain this result, Figure 18 and 19 show histograms for the normalized input features and output labels for the different case grids (including case300). The histograms are computed from arrays where all of the different features have been concatenated, and the input features include both bus and branch quantities. These plots give a rough idea of the input and output distributions of the different case grids. The greater width of the case39 histograms might explain the higher MSE achieved on this topology. In fact, the width of the label histograms matches well with the differences in magnitude of the MSE.

Overall, the results show that a GNN using localized operations is more suitable than an MLP for predicting the ACPF solutions on branches for grids with topologies that differ more than just slight variations of a given topology, as in the second experiment. However, in a practical setting one might want to be able to train on one set of grids and then deploy the model on a new grid with a previously unseen topology. Table 7 show



(a)

(b)

Figure 17: *Plot of the loss progression in the third experiment, starting from the 10th epoch. (a): Global MLP and (b): GNN. The darker lines show the sample median of the 10 model instances, while the lighter colored bands cover the range between the 25% and the 75% quantiles. MSE values are computed from the normalized labels.*

the MSE values for the predictions of the case300 test set, which is a grid topology that the GNN model instances have not encountered during training. Predictions were not computed for the global MLP, since the case300 grid does not fit inside the input layer. The min and max MSE values are given for the GNN to showcase just how different the errors are for the model instances. Even the smallest value is considerable larger than the DCPF MSE, and the large span from smallest to largest suggest that the GNN model in essence does little more than guess-work in its predictions. Going back to Figure 18 and 19, the distribution of values does not seem that different for the case300 compared to the grids that the model has been trained on. The incapability to generalize to this unseen case grid may therefore instead be tied to the topology and density of components such as transformers and shunt elements, which are not accounted for as input features.

Table 7: *MSE values for the predictions of the unseen case300 dataset. The metrics are computed in the original scale for power and current, and minimum and maximum MSE values for the GNN instances are displayed instead of mean and standard deviation.*

DCPF	GNN
6897.71	min: 38514.19, max: 581948.13

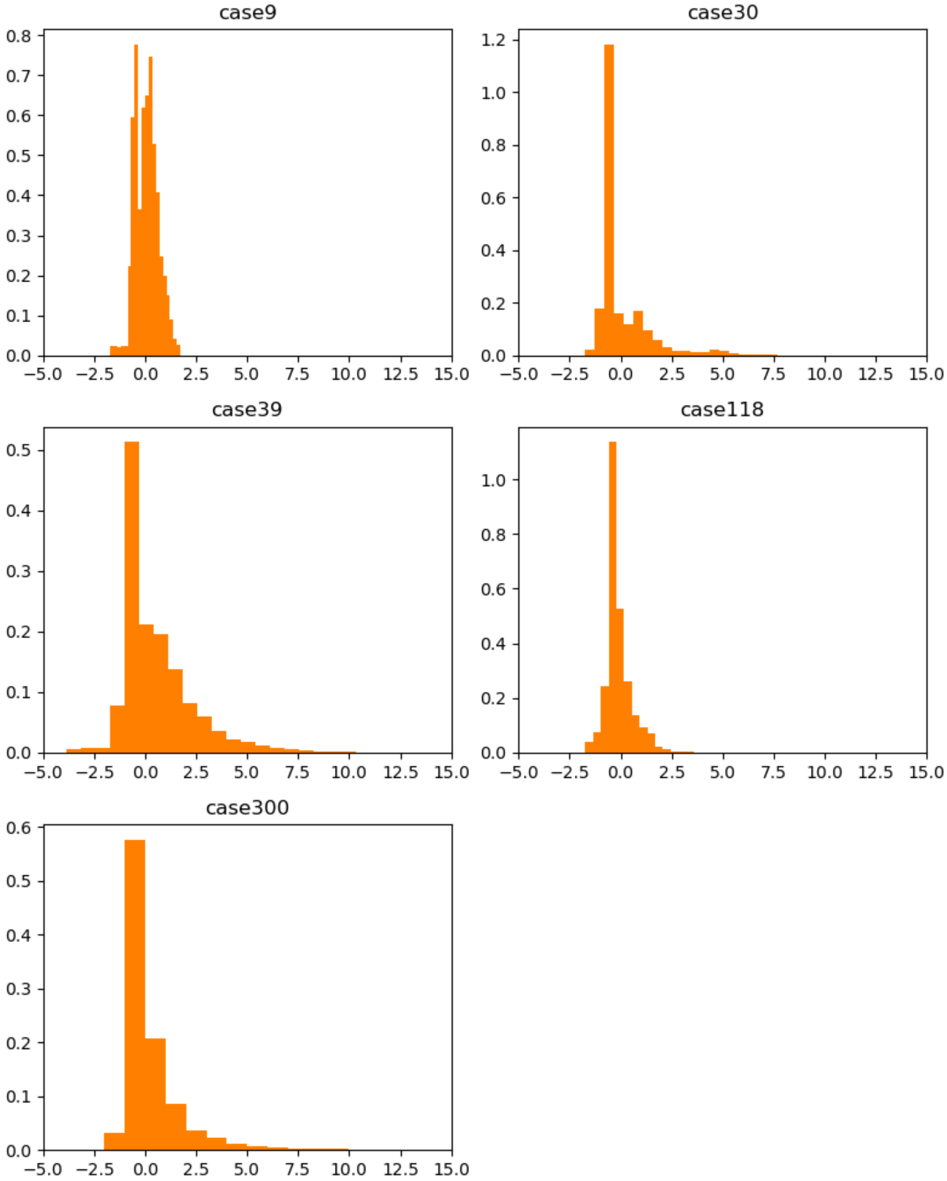


Figure 18: *Histograms of the input features present in the training dataset. The features include both bus and branch quantities. The histograms for each case grid is computed by concatenating all the relevant features from all corresponding training samples into a single vector.*

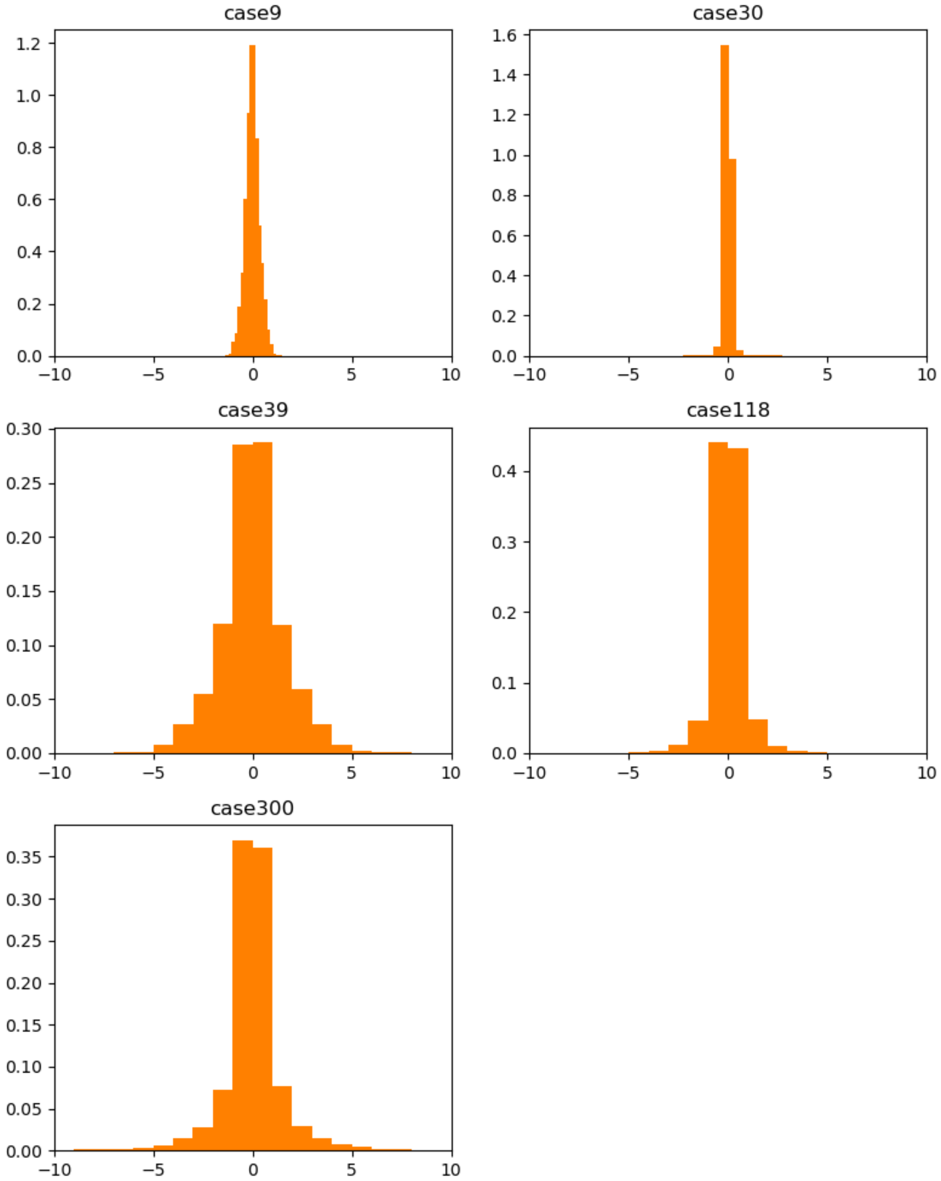


Figure 19: *Histograms of the output labels present in the training dataset. The features include the aforementioned eight branch power and current injection values. The histograms for each case grid is computed by concatenating all the relevant features from all corresponding training samples into a single vector.*

Part VI / Conclusions

21. Conclusive remarks for the experiments and possible improvements

As a whole, the experimental results show that in tasks involving either a grid with fixed topolopy or grids with limited permutations of a set topology, a global MLP might be more suitable for solving the ACPF problem than a GNN model such as the ones developed here. If the topology differs more, a GNN seems more suitable, 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 third experiment was shown to not be very successful when faced with the larger and unseen case300 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 and branch resistance 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 or resistance 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 than the global MLP for this grid in the first experiment. The branch resistance, reactance and charging susceptance p.u. amounts also depend on the base apparent power value, but for all the grids used here this base amount was 100 MVA, so this base quantity did probably not contribute to the issues with generalizing to new unseen case grids.

Another potential drawback with the used 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 θ_{shift} 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. For the global MLPs with procession units that correspond to a specific topology, the inclusion of values from these extra components would probably not lead to significant improvements, since these MLPs work in a centralized setting. Any information these transformers and shunts hold is therefore indirectly leveraged when modeling the dependencies between all of the buses and branches.

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 thesis, 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 third experiment was to train the models with more than just the four topologies, but some of the other MATPOWER 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.

22. Purely data driven versus partly equation based

Of the current GNN approaches to the ACPF problem found in literature, the so-called Graph Neural Solver (GNS) of Donon et al. (2020) is perhaps the one shown to achieve the highest level of generalization to unseen grid topologies, especially if trained on a larger grid. As described in Section 7.2, this GNS model is unsupervised and learns to directly minimize the violation to Kirchhoff's law through physical equations. These physical equations involve global operations, which might give this model a more complete vision in how the magnitude of each of the system quantities matter for finding the solution, at least compared to fully localized and purely data-driven approaches, such as the one used in this thesis.

The principal drawback with the GNS model is its complexity and the fact that it is specifically tuned to the ACPF problem and the way MATPOWER handles grid modeling. On the other hand, purely data driven and supervised approaches are more general and can more easily be tuned to solve different tasks. For instance, a supervised bus graph GNN could easily be converted to a task where the goal is to predict the power output on generators for an optimal power flow problem. The only things needed to change are: i) the output layer such that predictions are only made for generator buses and ii) to provide labels from e.g. an IPOPT solver during training. On the other hand, one of the main drawbacks of the purely data driven models is the need for labels, which could be computationally expensive to acquire. Also, these models likely need to train on a wider set of topologies to be able to generalize to unseen ones.

Aside from these differences, there is one key aspect both the GNS and the supervised and fully localized GNN models share: a reliance on propagation steps to diffuse the information through message passing. As highlighted several times in this thesis, all buses and branches depend on each other in the ACPF solution. Therefore, the models ideally should be able to aggregate information from the entire grid when they are processing every specific node. As the models grow deeper to achieve this for larger grids, the training and inference time will increase and gradient problems could arise during training. Naturally, the necessary number of propagation steps does not grow linearly with the number of buses, but if the grids have more chain-like structures rather than radial ones it could become quite large.

23. Future Work

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.

Another possible extension of the second experiment would be to see if by increasing the number of disconnections there might eventually be a turning point where a GNN surpasses the performance of a global MLP.

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.

Part VII / Appendix

A. DCPF formulation and solution

In the DCPF formulation, all reactive power flows of the power system are ignored, which means that Q_i on the buses is neglected. In addition, MATPOWER includes the following assumptions:

- All voltage magnitudes are assumed to be constant and equal to 1.0 p.u. (one per unit).
- Branches are considered to be lossless, and as a consequence branch resistances r_s and charging capacitances b_c are negligible:

$$y_s = \frac{1}{r_s + jx_s} \approx \frac{1}{jx_s}, \;\; b_c \approx 0 \tag{44}$$

- Voltage angle differences across branches are assumed to be small enough such that

$$\sin(\theta_f - \theta_t - \theta_{\text{shift}}) \approx \theta_f - \theta_t - \theta_{\text{shift}} \tag{45}$$

  This also applies to the differences between buses.

From the second of the above assumptions, the branch admittance matrix in (2) approximates to

$$Y_{br} \approx \frac{1}{jx_s} \begin{bmatrix} \frac{1}{\tau^2} & -\frac{1}{\tau e^{-j\theta_{\text{shift}}}} \\ -\frac{1}{\tau e^{-j\theta_{\text{shift}}}} & 1 \end{bmatrix} \tag{46}$$

This matrix, combined with the first and third assumption, yield the following approximation of the real power flow of each branch

$$p_f = \Re\{s_f\} \approx \frac{1}{x_s \tau}(\theta_f - \theta_t - \theta_{\text{shift}}) \tag{47}$$

And due to the lossless assumption we have that $p_t = -p_f$.

The relationship between active power flows and voltage angles for each branch is now given by

$$\begin{bmatrix} p_f \\ p_t \end{bmatrix} = B_{br} \begin{bmatrix} \theta_f \\ \theta_t \end{bmatrix} + \boldsymbol{p}_{\text{shift}} \tag{48}$$

where

$$B_{br} = \frac{1}{x_s\tau}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{p}_{\text{shift}} = \frac{\theta_{\text{shift}}}{x_s\tau}\begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

For a shunt element the amount of consumed complex power is

$$s_{sh} = v(y_{sh}v)^* \approx g_{sh} - jb_{sh}, \tag{49}$$

which follows from the first assumption. Consequently, the amount of active power consumed is $p_{sh} \approx g_{sh}$.

Next, let $\boldsymbol{b}_{ff}$ be an $N_l \times 1$ vector where the k-th element is b_k, where

$$b_k = \frac{1}{x_{s,k}\tau_k} \tag{50}$$

Also, let $\boldsymbol{p}_{f,\text{shift}}$ be an $N_l \times 1$ vector where the k-th element is equal to $-\theta_{\text{shift},k}b_k$. The relationship between all nodal active power injections and voltage angles can now be expressed in matrix form as

$$\boldsymbol{p}_{\text{bus}} = B_{\text{bus}}\boldsymbol{\theta} + \boldsymbol{p}_{\text{bus,shift}}, \tag{51}$$

where

$$\boldsymbol{p}_{\text{bus,shift}} = (C_f - C_t)^T\boldsymbol{p}_{f,\text{shift}} \tag{52}$$

$$B_{\text{bus}} = (C_f - C_t)^T\text{diag}(\boldsymbol{b}_{ff})(C_f - C_t) \tag{53}$$

Here, the $N_l \times N_l$ matrix B_{bus} is analogous to the nodal admittance matrix Y_{bus} in ACPF. Finally, the active power balance equations for the system can be compactly expressed as

$$B_{\text{bus}}\boldsymbol{\theta} + \boldsymbol{p}_{\text{bus,shift}} + \boldsymbol{g}_{sh} - \boldsymbol{p}_{\text{net}} = \boldsymbol{0}, \tag{54}$$

where all vectors are $N_b \times 1$ and the k-th element is equal to the value of the corresponding bus. The vector $\boldsymbol{p}_{\text{net}}$ contain the net active power injections given by (15). The unknowns in $\boldsymbol{\theta}$ can now be solved analytically since all of the remaining balance equations are linear.

Other formulations of the DC power flow problem include additional assumptions that further simplify the problem, such as ignoring tap settings on branch transformers (Seifi & Sepasian, 2011; Zhu, 2015). Many of the power system formulations found in the literature also do not include shunt elements on buses, which further simplifies the derivation.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., . . . Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Alpaydin, E. (2014). *Introduction to machine learning.* MIT Press.

Andersson, G. (2008). Modelling and analysis of electric power systems. *EEH-Power Systems Laboratory, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland*.

Atwood, J., & Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in neural information processing systems* (pp. 1993–2001).

Bacciu, D., Errica, F., Micheli, A., & Podda, M. (2020). A gentle introduction to deep learning for graphs. *Neural Networks*.

Bianchi, F. M., Grattarola, D., Livi, L., & Alippi, C. (2021). Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Bolz, V., Rueß, J., & Zell, A. (2019). Power flow approximation based on graph convolutional networks. In *2019 18th ieee international conference on machine learning and applications (icmla)* (pp. 1679–1686).

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., & Sun, X. (2020). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 3438–3445).

Chow, J. H., & Sanchez-Gasca, J. J. (2020). *Power system modeling, computation, and control.* John Wiley & Sons.

Conejo, A. J., & Baringo, L. (2018). *Power system operations.* Springer.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems* (pp. 3844–3852).

Donnot, B., Guyon, I., Schoenauer, M., Marot, A., & Panciatici, P. (2018). Fast power system security analysis with guided dropout. *arXiv preprint arXiv:1801.09870*.

Donon, B., Clément, R., Donnot, B., Marot, A., Guyon, I., & Schoenauer, M. (2020). Neural networks for power flow: Graph neural solver. *Electric Power Systems Research*, *189*, 106547.

Donon, B., Donnot, B., Guyon, I., & Marot, A. (2019). Graph neural solver for power systems. In *2019 international joint conference on neural networks (ijcnn)* (pp. 1–8).

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems* (pp. 2224–2232).

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

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. (`http://www.deeplearningbook.org`)

Grattarola, D., & Alippi, C. (2020). Graph neural networks in tensorflow and keras with spektral. *arXiv preprint arXiv:2006.12138*.

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

Harary, F., & Norman, R. Z. (1960). Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, *9*(2), 161–168.

Idema, R., Lahaye, D. J., et al. (2014). *Computational methods in power system analysis.* Springer.

Kim, C., Kim, K., Balaprakash, P., & Anitescu, M. (2019). Graph convolutional neural networks for optimal load shedding under line contingency. In *2019 ieee power & energy society general meeting (pesgm)* (pp. 1–5).

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

LeCun, Y., Bottou, L., Orr, G., & Müller, K. (1998). Efficient backprop. *Neural Networks: Tricks of the Trade*, 546–546.

Li, Q., Han, Z., & Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 32).

Murty, P. (2017). *Power systems analysis.* Butterworth-Heinemann.

Owerko, D., Gama, F., & Ribeiro, A. (2020). Optimal power flow using graph neural networks. In *Icassp 2020-2020 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5930–5934).

Paucar, V. L., & Rider, M. J. (2002). Artificial neural networks for solving the power flow problem in electric power systems. *Electric Power Systems Research*, *62*(2), 139–144.

Salam, M. A. (2020). *Fundamentals of electrical power systems analysis.* Springer.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Seifi, H., & Sepasian, M. S. (2011). *Electric power system planning: Issues, algorithms and solutions.* Springer-Verlag Berlin Heidelberg.

Simonovsky, M., & Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3693–3702).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). 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).

You, J., Ying, Z., & Leskovec, J. (2020). Design space for graph neural networks. *Advances in Neural Information Processing Systems*, *33*.

Zhao, L., & Akoglu, L. (2019). Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*.

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

Zhu, J. (2015). *Optimization of power system operation.* John Wiley & Sons.

Zimmerman, R. D., & Murillo-Sánchez, C. E. (2020). *Matpower user's manual, version 7.1.* Retrieved from `https://matpower.org/docs/MATPOWER-manual-7.1.pdf`

Zimmerman, R. D., Murillo-Sánchez, C. E., & Thomas, R. J. (2010). Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on power systems*, *26*(1), 12–19.

Zitnik, M., Agrawal, M., & Leskovec, J. (2018). Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, *34*(13), i457–i466.