Master's Thesis

Graph Neural Networks for Electrical Grid State Estimation

Saur, Armin

Examiner: Prof. Dr. Hannah Bast
Advisers: B. Rückauer, S. Walter und W. Biener

University of Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Algorithms and Data Structures

March 21 th, 2024

Writing Period

24. 10. 2023 – 21. 03. 2024

Examiner

Prof. Dr. Hannah Bast

Second Examiner

Prof. Dr. Christof Wittwer

Advisers

B. Rückauer, S. Walter und W. Biener

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg i. Br. 21.03.2024

Place, Date

Signature

Abstract

Increasing loads on the Low-voltage network (LVN) does require constant status checks. The health status of a LVN can be determined using automated Electrical grid power system state estimation (SE). The reformulation of State estimation task (SE-task) uses heterogeneous graph concepts. Using Graph Neural Networks (GNNs), applied complex voltage can be gained in spare measured girds. A date fitting concept is presented for parsing LVN into a proper data structure. The author presents GNN GSETR, capable of performing SE. A great performance of 99 % is reached by GSETR, knowing all power measurements. Moreover, this work demonstrates the robustness of the model. For this purpose, the measured power data is only sparsely available to the model. A 91 % accuracy is achieved, in this second scenario.

Contents

List of Figures

List of Tables

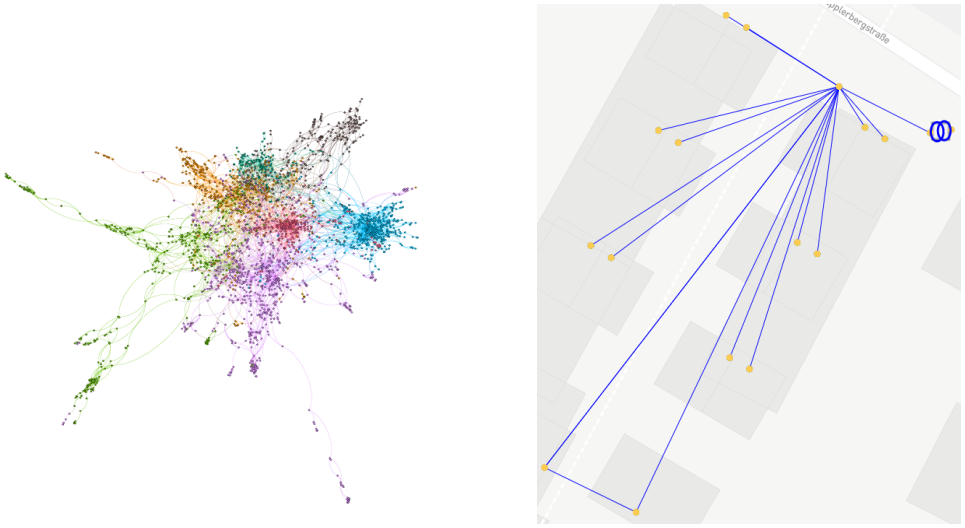
List of Algorithms

1. Introduction

The electric power grid, arguably the largest complex system on Earth, is recognized as the greatest engineering achievement of the 20th century [25]. Former power concepts of the grid, with separated industrial parks, centralized *electricity generators*, and neighborhoods, are vanishing. Decentralized power injections by households with photovoltaic systems and upcoming electrical energy-intensive interactors endanger the health state of Low-voltage networks (LVNs). The problem of gaining the grid state is named State estimation task (SE-task). Since 2022 is the Fraunhofer Institute for Solar Energy Systems (ISE) researching with Graph Neural Network (GNN) to analyze LVNs.

1.1. Motivation

A GNN learns data together with topological structures. Besides Electrical grid power system state estimation (SE), many tasks have been formulated in a graph fashion, e. g. teaching a GNN model finding a spelling error in a sentence [24]. The *figure 1.1* provides an impression of the special graph topologies in LVN. A heterogeneous citation graph is compared with the LVN *Allensbach*. In comparison, the LVN topology is more flat. The natural representation of a LVN as a graph and in the past years increasing performances for GNN architectures for temporal graphs, [9] and [8], motivate to choose such an architecture. With the determined energy transition, LVN becomes a more active part of the electrical system. [13]



(a) Cora citation graph [17]

(b) Allensbach LVN graph

Figure 1.1.: Two different graph sets. Plotted with different settings.

Therefore, it is required to gain knowledge about the state of all LVNs. Each SE must be computed frequently, to well monitor a system's state. Generally, the SE is automatically determined by a SE-solver. The advancements of Artificial intelligence (AI) techniques can omit the limitations of these physical laws, when applied to the SE-task. A GNN can be more robust against noisy measurements and lack of information. Further, it may be used for SE forecasts.

1.2. State estimation task

An adapted treatment for LVNs, solving the SE-task is presented in this work. The knowledge about the applied complex power in a whole LVN is the state of the grid. This work re-formulates the SE-task for a GNN. Power measurements within a LVN are transformed into a temporal heterogeneous graph dataset. The dataset contains different node and edge types. Node types are used to hide features from a model. Two scenarios are defined: (1) with all features present for a GNN and (2) one, where features are only sparsely available. This mechanism (2) represents a more realistic

case and is testing the robustness of a GNN model. Two GNN models, GSETR and GSENR, are proposed by the author. Each is learning the state of a LVN. GSETR performs in (1) nearly perfect, with an accuracy of 99 %, and in (2) well, with 91 %. However, GSENR is introduced for future research.

1.3. Structure

An overview of related research and the SE project at the ISE provides chapter 2. The third chapter 3 presents the LVN as a heterogeneous graph and defines the SE-task. Moreover, basic concepts of Machine Learning (ML) and the processing of heterogeneous data by GNN model are stated. The approach of the SE-task in GNN fashion is defined chapter 4. Based on an on-top definition, the data fetching is also sketched. The chapter 5 is showing the reproduction experiments and analysis of the former project. These results flow into the models of chapter 6. Applying GSETR and GSENR on two scenarios is contained in chapter 7. Moreover, a benchmark and baseline for future research are made. A conclusion, with a discussion and additional refinements, summarizes this work in chapter 8. This includes questions about the reliability of this work.

2. Related work

Different approaches for handling temporal graph problems and solving the SE-task are done by other researchers. Selected works are presented below, as well as in section 2.1 the project state on ISE for solving the SE-task.

Evaluating on temporal taxi data to forecast the traffic speed on each node in the graph is done by [27] . The author presents scenarios for learning from all node features and from spatial node features. The proposed GNN architectures use a Gated Recurrent Unit (GRU)-Graph Convolution Network (GCN) and Long-Short-Tearm memory (LSTM)-GCN attempt. The authors of [8] developed this model further to *CTGCN*.

Using a GNN model design to solve the SE-task with supervised learning is done before by [7] . The electrical grid is extended. For each node, four labeled nodes were added for the features and the targets. Two feature nodes provide measured complex current and complex voltage to the GNN model. These nodes form a bipartite graph. Target nodes are used for learning the real and imaginary parts of the node state. The authors suggest a heterogeneous model using Graph Attention (GAT) techniques.

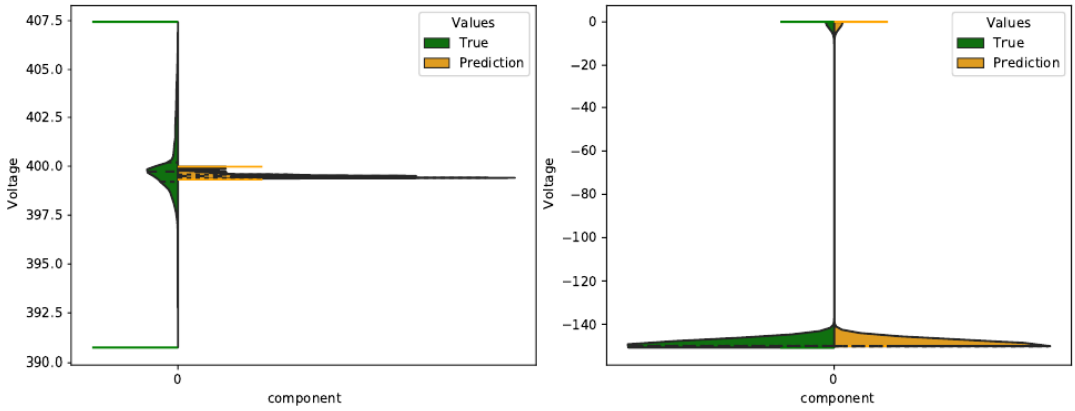
[17] also performs SE, but in a LSTM fashion. Complex current, complex power, active power, and reactive power are defined as input features. For selected nodes, these node attributes are available to the model. The authors also describe noticed differences, when applying GNN learning to electrical grid data.

2.1. GNNSE 2023

Since 2022 is the ISE researching AI tools for SE. Below is a summary of the project status until March 2023.

The project state until March 2023 is referenced as [20] . Reliable published results are referenced as [11] (September 2022). This differentiation is required, since code from [20] is used to reproduce [11] .

Up to twenty GCN layer uses [11] to solve the SE-task in an LVN. A LVN is represented as a heterogeneous graph. Each node is associated with features and targets. For the features, only active power and reactive power are used. The node target vector contains voltage magnitude and voltage angle. A heterogeneous GNN model learns the targets.



(a) Comparing voltage magnitude **(b)** Comparing voltage angle

Figure 2.1.: Violin plots, comparing targets (green) with estimated targets (orange). [11]

The project is adapted by [20] . The author reduced the number of GCN layer to five and the number of samples for training and validation. Training samples are filtered based on the standard deviation, and validation samples are dropped randomly. A normalization was applied to some data. Further, the bias weight of the last linear

layer is pre-defined, for all node types. However, [20] also concludes, that the model is poor in variance.

3. Basics

This chapter provides all the basic concepts required for understanding this work. First, in section 3.1, the basic electrical grid and the graph data structures gets defined. Further, is the SE-task formal defined. After introducing ML concepts in the section 3.2, the GNN model architecture is explained in section 3.3.

3.1. Electrical grid as heterogeneous graph

The fundamental basics of electrical grids are covered in the following subsections: Begin and end of a LVN are defined in section 3.1.1. This falls together with section 3.1.2, the representation of a LVN as a heterogeneous graph. A qualified graph data structure is covered in section 3.1.3. In the last section 3.1.4, is the SE-task for LVNs covered.

3.1.1. Low-voltage network

Nominal voltage V_{nom} is the defined voltage in an electrical network at normal operation. Standardized German electrical grid classes relevant for this work are listed below:

network level 5 A medium-voltage network provides a fixed nominal voltage or supply voltage of $V_{\text{nom,nl5}} = 10$ kV or up to $V_{\text{nom,nl5}} = 30$ kV. For simplicity, this network is represented as a single bus called *slack* $i^{(\text{s})}$.

Network level 6 The transforming network consists of a *transformer*, often located at the city border, with $V_{\text{nom,nl5}}$ input from the *slack bus* and output $V_{\text{nom,nl7}}$.

network level 7 The LVN is by a *transformer* connected with the network level 5 and provides $V_{\text{nom}} = V_{\text{nom,nl7}} = 0.4$ kV to the city with 50 Hz.

[6] Such a structure is in *figure 3.1*, with tree topology for the LVN. For simplicity, notes LVN in the following also the transformer and slack bus. This may represent a

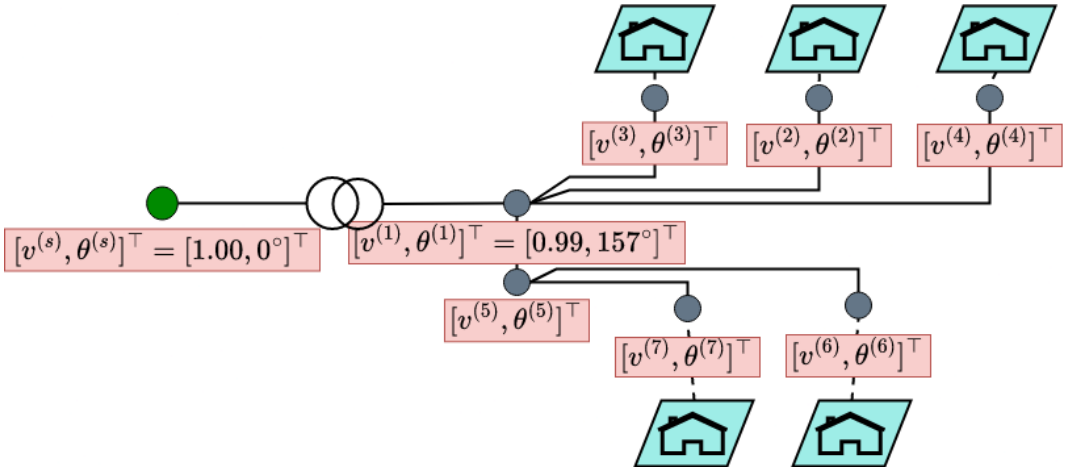


Figure 3.1.: Abstract illustration of *slack* bus (green) via *transformation* (overlapped circles) connected to LVN, consists out of seven buses (dark gray) linked by wires. Each node holds a vector (red) with a complex voltage. Interactors (cyan) are linked to nodes.

street with five households. A filled circle (node) represents an electrical grid bus. The *slack node* is green-notated, and the LVN dark gray. Each node with a house represents a bus with participants in the electrical grid. As presented, also buses with no participants are also possible. On each node in the grid is a complex voltage applied (red), where the plotted values are in relative representation, as in section 3.1.4 defined. Further, there exists by construction only one *slack node* and may be considered the tree root. For this research, only LVNs with tree topology are respected.

3.1.2. Heterogeneous graph

A graph $\boldsymbol{G}_t = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{E}}, \boldsymbol{R}, \boldsymbol{\Sigma}_t^{(\mathcal{N})}, \boldsymbol{\Sigma}_t^{(\mathcal{E})})$ represents an LVN at a time stamp $t \in \boldsymbol{T}$, with

- nodes $\boldsymbol{\mathcal{N}} := \{i | i = 1 \dots \mathcal{N}\}$ representing the buses,
- the branches $\boldsymbol{\mathcal{E}} \subset \boldsymbol{\mathcal{N}} \times \boldsymbol{\mathcal{N}}$ connected by the $\mathcal{N} := |\boldsymbol{\mathcal{N}}|$ different nodes,
- the edge attributes $\boldsymbol{\Sigma}_t^{(\mathcal{E})}$ and
- node attributes $\boldsymbol{\Sigma}_t^{(\mathcal{N})}$.

The graph time series $\boldsymbol{G} := [\boldsymbol{G}_1, \dots, \boldsymbol{G}_T]^\top$ contains independent graphs $\boldsymbol{G}_t$, in order by increasing timestamps $t = 1 \dots T$. In $\boldsymbol{G}$ are collected timestamps notated as $\boldsymbol{T} = \{t | t = 1 \dots T\}$. The set of all edges is bidirectional,

$$\forall (i,j) \in \boldsymbol{\mathcal{E}} \implies (j,i) \in \boldsymbol{\mathcal{E}}$$

, and without self-loops, i. e. $(i,i) \notin \boldsymbol{\mathcal{E}}$. In a heterogeneous graph $\boldsymbol{G}_t$, each node $i \in \boldsymbol{\mathcal{N}}$ is associated with a node type

$$\forall i \in \boldsymbol{\mathcal{N}} \implies \exists r_i^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})}.$$

Further, for each edge, there is an edge type

$$\forall (i,j) \in \boldsymbol{\mathcal{E}} \implies \exists r_{(i,j)}^{(\mathcal{E})} \in \boldsymbol{R}^{(\mathcal{E})}.$$

Such a graph contains Node-edge-node-type links (*nent-link*s) $\boldsymbol{r}_{(i,j)} \in \boldsymbol{R}$

$$(r_i^{(\mathcal{N})}, r_{(i,j)}^{(\mathcal{E})}, r_j^{(\mathcal{N})}) = \boldsymbol{r}_{(i,j)},$$

with $\boldsymbol{R} \subset \boldsymbol{R}^{(\mathcal{N})} \times \boldsymbol{R}^{(\mathcal{E})} \times \boldsymbol{R}^{(\mathcal{N})}$ representing all *nent-link* in $\boldsymbol{G}$. The argument $r_i^{(\mathcal{N})}$ is mapping $r_i^{(\mathcal{N})} : \boldsymbol{\mathcal{N}} \longrightarrow \boldsymbol{R}^{(\mathcal{N})}$ and $r_{(i,j)}^{(\mathcal{E})}$ maps $r_{(i,j)}^{(\mathcal{E})} : \boldsymbol{\mathcal{E}} \longrightarrow \boldsymbol{R}^{(\mathcal{E})}$.

The binary adjacency matrix $\boldsymbol{A}_{\mathcal{N} \times \mathcal{N}}$ describes the neighborhood of nodes for a defined node order, where $A_{i,j} := 1$ only if $(i,j) \in \boldsymbol{\mathcal{E}}$. The neighborhood set $\boldsymbol{A}(i)$ for node $i \in \boldsymbol{\mathcal{N}}$ contains all reachable nodes $i \in \boldsymbol{\mathcal{N}}$ within one node-hop $\boldsymbol{A}(i) := \{j | (i,j) \in \boldsymbol{\mathcal{E}}\}$. The number of neighbors $A(i) = |\boldsymbol{A}(i)|$ of i is called the node degree of i.

Permutation invariant

For a binary permutation matrix $\boldsymbol{P}_{\mathcal{N}\times\mathcal{N}}$, the order of nodes (row and column) in the adjacency matrix $\boldsymbol{A}$ can be adapted $\boldsymbol{PAP}^\top$. Two graphs $\boldsymbol{G}_t$ and $\boldsymbol{G}_t^{'}$ have the same topology if they are isomorphic $\boldsymbol{G}_t \cong \boldsymbol{G}_{t'}^{'}$. This is the case if $\boldsymbol{P}$ exists, s. t.

$$\boldsymbol{A}^{'} = \boldsymbol{PAP}^\top$$
$$\boldsymbol{\Sigma}_{t',k}^{'(\mathcal{N})} = \boldsymbol{P}\boldsymbol{\Sigma}_{t,k}^{(\mathcal{N})},$$

for all node attributes $\boldsymbol{\Sigma}_{t,k}^{(\mathcal{N})} \in \boldsymbol{\Sigma}_t^{(\mathcal{N})}$ [4]. In the case of a homogeneous graph, the edge and node types lists are updated. The graph nodes are not represented in a Euclidean space, even if the node attributes can be used for that.

Example

The *figure 3.2* shows a heterogeneous graph $\boldsymbol{G}_t$ representing an LVN. An interactor

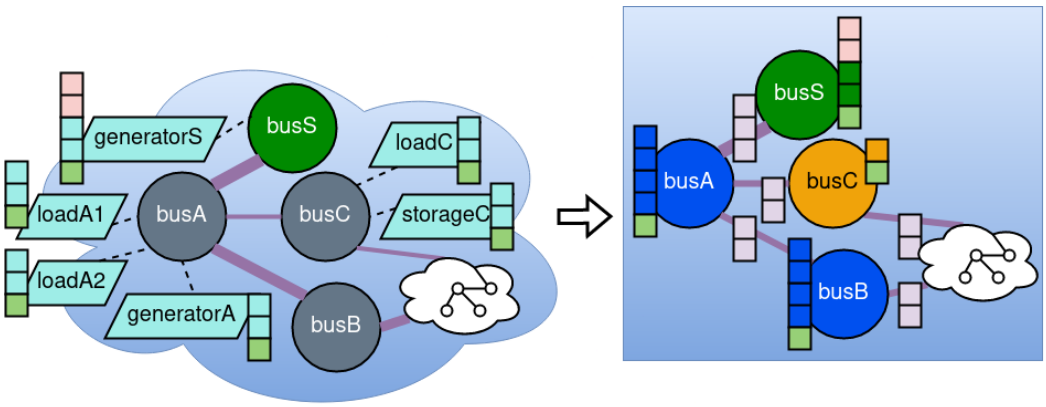


Figure 3.2.: (left) LVN with interactors (cyan). (right) A heterogeneous graph $\boldsymbol{G}_t$, with colored nodes.

(cyan) with the LVN e. g. *generator*, *loader* or *storage* is connected to one node. Nodes are labeled with *bus**. Nodes (dark green, orange, dark blue) are of different node type. Interactors hold attributes as a vertical vector. Different edge types are marked by the line width (small, wide).

In $\boldsymbol{G}_t$ are the interactors aggregated to node attributes. Each interactor, node $\boldsymbol{\Sigma}_t^{(\mathcal{N})}$ and edge $\boldsymbol{\Sigma}_t^{(\mathcal{E})}$ attributes are different regarding their type. In $\boldsymbol{G}_t$ are nodes of different types (color) $\boldsymbol{R}^{(\mathcal{N})}$ linked by edges of different types (width) $\boldsymbol{R}^{(\mathcal{E})}$.

3.1.3. Data structure

An alternative representation of the graph $\boldsymbol{G}_t$ uses an adjacency list $\boldsymbol{A}_{\text{list}} \in \mathbb{R}^{2\times\mathcal{E}}$ for representing the edges,

$$\forall(i,j) \in \boldsymbol{\mathcal{E}} \implies \exists k, (A_{\text{list}}^{1,k}, A_{\text{list}}^{2,k}) = (i,j)$$

, with $0 < k \leq \mathcal{E}$. An adjacency list $\boldsymbol{A}_{\text{list}}^{\boldsymbol{r}} \in \mathbb{N}^{2\times\mathcal{E}^{(r)}}$ contains all *nent-link* in $\boldsymbol{G}_t$ matching to the respective type triple $\boldsymbol{r} \in \boldsymbol{R}$. The set $\boldsymbol{R}$ contains all *nent-link* existing in $\boldsymbol{G}$. Further contains a set $\boldsymbol{\mathcal{N}}^{(\mathrm{r}^{(\mathcal{N})})}$ of all nodes of the type $r^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})}$.

Let be $\boldsymbol{G}_t$ a heterogeneous graph, e. g. like $\boldsymbol{G}_6$ in *figure 3.3*. The graph $\boldsymbol{G}_t$ is stored

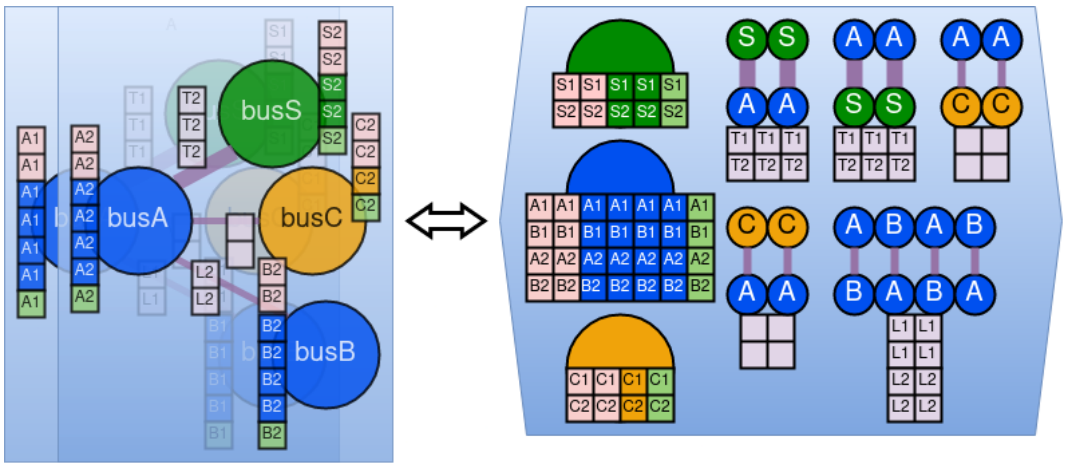


Figure 3.3.: (left) LVN time sequence $\boldsymbol{T} = \{5, 6\}$.
(right) The equivalent graphs $\boldsymbol{G} = [\boldsymbol{G}_5, \boldsymbol{G}_6]^\top$, with stacked list representation.

in the following data structure:

- an adjacency lists $\boldsymbol{A}_{\text{list}}^{\boldsymbol{r}}$ for each *nent-link* $\boldsymbol{r} \in \boldsymbol{R}$,

- an edge feature matrix $\boldsymbol{X}_t^{(\boldsymbol{r})} \in \mathbb{R}^{\mathcal{E}^{(\boldsymbol{r})} \times \mathcal{F}^{(\boldsymbol{r})}}$ for each *nent-link* $\boldsymbol{r} \in \boldsymbol{R}$, with $\mathcal{F}^{(\boldsymbol{r})}$ the feature dimension,

- a node feature matrix $\boldsymbol{X}_t^{(\mathrm{r}(\mathcal{N}))} \in \mathbb{R}^{\mathcal{N}^{(\mathrm{r}(\mathcal{N}))} \times \mathcal{F}^{(\mathrm{r}(\mathcal{N}))}}$ for each node type $r^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})}$, with feature dimension $\mathcal{F}^{(\mathrm{r}(\mathcal{N}))}$,

- a node target matrix $\boldsymbol{Y}_t^{(\mathrm{r}(\mathcal{N}))} \in \mathbb{R}^{\mathcal{N}^{(\mathrm{r}(\mathcal{N}))} \times \mathcal{T}^{(\mathrm{r}(\mathcal{N}))}}$ for each node type $\boldsymbol{R}^{(\mathcal{N})}$, with target dimension $\mathcal{T}^{(\mathrm{r}(\mathcal{N}))}$ and

- less structured metadata $\boldsymbol{M}$.

It is to point out that this list-based data structure can store multiple isomorphic graphs $\boldsymbol{G}_t \cong \boldsymbol{G}_{t'}^{'}$. The lists of $\boldsymbol{G}_t$ and $\boldsymbol{G}_{t'}^{'}$ are stacked in this case.

Example

The heterogeneous graph $\boldsymbol{G}$ in *figure 3.3* knows three node types (green, blue, organ) and two edge types (small, wide). It is stored in the above-defined graph data structure. $\boldsymbol{G}$ represents the LVN during the time window $\boldsymbol{T}$. Therefore, the duplicated edges, the edge attributes, and the time-dependent node attributes are stacked for all time stamps $t \in \boldsymbol{T}$. $\boldsymbol{G}$ may be stored as a collection of

- five adjacency lists, like $\boldsymbol{A}_{\text{list}}^{\text{green,wide,blue}} \in \mathbb{N}^{2\times 2}$ or $\boldsymbol{A}_{\text{list}}^{\text{blue,small,blue}} \in \mathbb{N}^{2\times 4}$,

- five edge feature matrices, like $\boldsymbol{X}_t^{(\text{green,wide,blue})} \in \mathbb{R}^{2\times 3}$ or $\boldsymbol{X}_t^{(\text{blue,small,blue})} \in \mathbb{R}^{4\times 2}$,

- three feature matrices, $\boldsymbol{X}_t^{(\text{green})} \in \mathbb{R}^{2\times 2}$, $\boldsymbol{X}_t^{(\text{blue})} \in \mathbb{R}^{4\times 4}$ and $\boldsymbol{X}_t^{(\text{orange})} \in \mathbb{R}^{2\times 1}$,

- three target matrices, $\boldsymbol{Y}_t^{(\text{green})}, \boldsymbol{Y}_t^{(\text{orange})} \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{Y}_t^{(\text{blue})} \in \mathbb{R}^{4\times 2}$, and

- some metadata (light green).

3.1.4. State estimation

This section describes the SE-task and the approach to solving this task by numerical methods. This is mainly done by (1) aggregating the interactors on the nodes and (2) applying a SE solver.

In a nutshell, the SE aims to determine in graph $\boldsymbol{G}_t$ the grid state $\boldsymbol{Y}_t \in \mathbb{R}^{\mathcal{N}\times 2}$, defined as a list of all node states,

$$[\boldsymbol{y}_{t,1}, \ldots, \boldsymbol{y}_{t,\mathcal{N}}]^\top = \boldsymbol{Y}_t.$$

A node state is the complex voltage $\boldsymbol{y}_{t,i} = [y_{t,i}^{(|\mathrm{V}|)}, y_{t,i}^{(\theta)}]^\top \in \mathbb{R}^2$ at a node $i \in \boldsymbol{\mathcal{N}}$. [25] The active power P_i and reactive power Q_i at a node $i \in \boldsymbol{\mathcal{N}}$ is gained by (1) aggregating the power of K_i different interactors attached to this node

$$P_i = \sum_{k}^{K_i} P_{i,k} \tag{3.1}$$

$$Q_i = \sum_{k}^{K_i} Q_{i,k}. \tag{3.2}$$

An interactor k may be a power consumer, power generator, or power storage unit, with $P_{i,k}, Q_{i,k} \in \mathbb{R}$. The power is flowing and balancing in the LVN between all nodes. The *figure 3.2* shows different interactors (cyan) attached to some nodes.

Let j be the imaginary number. As in *appendix* A.1 showed, is the complex power $\underline{S}_n$ for a node $n \in \boldsymbol{\mathcal{N}}$ computable by

$$\underline{S}_n = P_n + jQ_n \tag{3.3}$$

$$= \underline{V}_n \cdot \underline{I}_n{}^* \tag{3.4}$$

$$= |\underline{V}_n| e^{j\theta_n} \cdot (\sum_{m\in A(n)} \underline{Y}_{n,m} \cdot \underline{V}_m)^*. \tag{3.5}$$

Node state $\boldsymbol{y}_{t,i}$ is affected by its neighbors $j, \forall j \in \boldsymbol{A}(i)$, hence at most by $\mathcal{N} - 1$ nodes. Due to [25] follows from (3.5) for active power,

$$0 = -P_i + \sum_{j=1}^{\mathcal{N}} |V_i||V_j| \cdot (G_{i,j} \cos(\triangle\theta_{i,j}) + B_{i,j} \sin(\triangle\theta_{i,j}))$$

with $\triangle\theta_{i,j} = \theta_i - \theta_j$. Respectively, it holds for reactive power,

$$0 = -Q_i + \sum_{j=1}^{\mathcal{N}} |V_i||V_j| \cdot (G_{i,j}\cos(\triangle\theta_{i,j}) + B_{i,j}\sin(\triangle\theta_{i,j})).$$

The number of unknown variables is at most $2(\mathcal{N}-1)$ [25].

Sate estimation solver

To solve the SE-task in $\boldsymbol{G}_t$, the following assumptions are made:

- The graph topology is known and constant.
- The admittance on all edges is known.
- For *slack node* $i^{(\mathrm{s})}$, all information is known.
- For all nodes $i, \forall i \in \boldsymbol{\mathcal{N}}$, all interactor powers $P_{i,k}$ and $Q_{i,k}$ are known.

[16], [6] Under these conditions, the SE-task at a time stamp t can be solved independent of all other time stamps $t \neq t^{'}$, with $t, t^{'} \in \boldsymbol{T}$. Defining the complex power $\underline{V}_s$ at the *slack node* $i^{(\mathrm{s})} \in \boldsymbol{\mathcal{N}}$ in the convention,

$$|V_{t,s}| := V_{\mathrm{nom,nl5}} \tag{3.6}$$

$$\theta_{t,s} := 0^{\circ} \tag{3.7}$$

enables to solve the power flow at all nodes $i, \forall i \in \boldsymbol{\mathcal{N}}$. [15] As stated before and due to convention, the node state $\boldsymbol{y}_{t,i}$ is represented in relative form,

$$y_{t,i}^{(|\mathrm{V}|)} := \delta^{(|\mathrm{V}|)}|V_i|$$
$$y_{t,i}^{(\theta)} := \delta^{(\theta)}\theta_i,$$

with $\boldsymbol{\psi} = \{|V|, \theta\}$ and

$$\boldsymbol{\delta} := [\delta^{(|\mathrm{V}|)}, \delta^{(\theta)}]^{\top} = [\frac{1}{V_{\mathrm{nom}}}, \frac{2\pi}{360}]^{\top}.$$

Let be trns$(\cdot)$ the function applying this transformation

$$y_{t,i}^{(\psi)} = \text{trns}(\psi_i, \delta^{(\psi)}). \tag{3.8}$$

The authors of [25] provide an overview of the developed methods for successfully solving the SE-task. However, these methods are not covered in this work. Interested readers are also directed to [6] , which introduces the *Newton-Raphson method*.

Example

The *figure 3.1* shows a *slack node* (green) with complex voltage stated in polar representation, $\boldsymbol{y}_{t,i} = [y_{t,S}^{(|\mathrm{V}|)}, y_{t,S}^{(\theta)}]^\top \sim [|V_S|, \theta_S]^\top$. The generator edge $(S, 1) \in \boldsymbol{\mathcal{E}}$ shifts the voltage angle and reduces the voltage magnitude, e. g. due to admittance $\underline{Y}_{S,1}$.

3.2. Machine learning

The general ML concept is introduced in the following subsection. For this concept, the graph topology is not respected. Nevertheless, GNN techniques follow these exactly. However, section 3.2.1 tells about the training and section 3.2.2 about the validation of the model.

3.2.1. Training

Training is one main part of ML and is covered in the following.

In ML, a model is a mathematic abstraction of a parameterized architecture, learning the relation between an input signal $\boldsymbol{H}^{(0)} = \boldsymbol{X}_t \in \mathbb{R}^{\mathcal{N} \times \mathcal{F}^{(\mathcal{N})}}$ and target signal $\boldsymbol{Y}_t \in \mathbb{R}^{\mathcal{N} \times \mathcal{T}}$. Let be MODEL such a model. Moreover, let be $\boldsymbol{G} = [\boldsymbol{G}_t | t = 1 \dots T]^\top$ a dataset containing the time stamps $\times$. These samples $\boldsymbol{G}_t = (\boldsymbol{X}_t, \boldsymbol{Y}_t), \forall t \in \boldsymbol{T}$ are

used for training the model. $\mathcal{F}^{(\mathcal{N})}$ is the number of node features and $\mathcal{T}$ of targets for each node $i, \forall i \in \boldsymbol{\mathcal{N}}$. MODEL contains L layer,

$$\boldsymbol{H}^{(\mathrm{l})} = f^{(\mathrm{l})}(\boldsymbol{H}^{(\mathrm{l}-1)}, \cdot),$$

producing a new hidden value $\boldsymbol{H}^{(\mathrm{l})}$, with $l := 1 \dots L$. The output $\hat{\boldsymbol{Y}}_t = \boldsymbol{H}^{(\mathrm{L})} \in \mathbb{R}^{\mathcal{N} \times \mathcal{T}}$ is called the prediction or estimation. A basic layer type is the *linear*-layer,

$$\boldsymbol{H}^{(\mathrm{l})} = \boldsymbol{W}^{\top(\mathrm{l})} \boldsymbol{H}^{(\mathrm{l}-1)} + \boldsymbol{w}_{\mathrm{b}}^{(\mathrm{l})},$$

where the weight $\boldsymbol{W}^{\top(\mathrm{l})} \in \mathbb{R}^{\mathcal{F}^{(\mathcal{N})} \times \mathcal{F}^{(\mathrm{H,l})}}$ and bias $\boldsymbol{w}_{\mathrm{b}}^{(\mathrm{l})} \in \mathbb{R}^d$ are learnable parameters [1]. Optionally, $\boldsymbol{w}_{\mathrm{b}}^{(\mathrm{l})}$ can be a non-learnable constant. Additional to the features, $\boldsymbol{X}_t$ is also $\boldsymbol{Y}_t$ accessible, hence performs MODEL *supervised-learning* [18]. In the task of performing SE, the target domain is not a set of discrete labels, i. e. $\boldsymbol{y}_{t,i} \in \mathbb{R}^{\mathcal{T}}$.

Batch set

Model MODEL processes dataset $\boldsymbol{G}$ sample by sample. Generally, the dataset size T too large. Processing it at once leads to increased time consumption or reaches computational limitations. Therefore, is the set $\boldsymbol{T}$ gapless divided into $B \in \mathbb{N}$ sub-sequences,

$$[\boldsymbol{T}_{1|}^{\top} \parallel \boldsymbol{T}_{2|}^{\top} \parallel \dots \parallel \boldsymbol{T}_{B|}^{\top}]^{\top} = [t | t = 1 \dots T]^{\top},$$

with concatenation operator $\parallel$. Each batch sequence list $\boldsymbol{T}_{i|} \in \mathbb{N}^b$ contains the ordered samples,

$$\boldsymbol{T}_{i|} = [t | (i-1) \cdot b < t \leq i \cdot b, t \in \boldsymbol{T}]^{\top}.$$

$b \in \mathbb{N}$ is the batch size, and for simplicity, it is assumed $T = B \cdot b$. The batch set $\boldsymbol{G}_t$ for $\boldsymbol{T}_{i|}$ contains b graphs,

$$\boldsymbol{G}_{i|} = [\boldsymbol{G}_t | t = T_{i|,0} \dots T_{i|,b}]^{\top}.$$

A sub-feature Multidimensional array (*md-array*) $\boldsymbol{X}_{i|} \in \mathbb{R}^{b \times \mathcal{N} \times \mathcal{F}^{(\mathcal{N})}}$ contains all feature values from the samples in $\boldsymbol{G}_{i|}$. Respectively, holds the same for target and prediction *md-arrays* $\boldsymbol{Y}_{i|}, \hat{\boldsymbol{Y}}_{i|} \in \mathbb{R}^{b \times \mathcal{N} \times \mathcal{T}}$. Let be batch$(\cdot)$ a function performing the above-defined split into batch sets,

$$(\boldsymbol{X}_{i|}, \boldsymbol{Y}_{i|}) = \text{batch}(\boldsymbol{G}, \boldsymbol{T}_{i|}).$$

Loss function

Let the MODEL predicts $\hat{\boldsymbol{Y}}_{i|}$ for $\boldsymbol{Y}_{i|}$. The loss $L = \text{loss}(\boldsymbol{Y}_{i|}, \hat{\boldsymbol{Y}}_{i|})$ is computed by a loss function,

$$\text{loss} : (\mathbb{R}^{T \times \mathcal{N} \times \mathcal{T}}, \mathbb{R}^{T \times \mathcal{N} \times \mathcal{T}}) \longrightarrow \mathbb{R}.$$

The *L1-loss* is a common loss function, as it is defined as *L1-norm*,

$$L = \frac{1}{\mathcal{T}T\mathcal{N}} \sum_{t=1}^{T} \sum_{i=1}^{\mathcal{N}} \sum_{\psi=1}^{\mathcal{T}} |y_{t,i}^{(\psi)} - \hat{y}_{t,i}^{(\psi)}|. \tag{3.9}$$

As for *L1-Loss* is the loss node-vise computed during all timestamps.

Process of training

Minimizing L, regarding the performance metric loss$(\cdot)$,ç is the objective of the weight parameter optimization $\text{MODEL}' = \text{OPT}(\text{MODEL}, L)$.

As in *algorithm* 1 sketched, during learning, batch set after batch set is processed. Optionally, a transformation function trns used to modify values,

$$\text{trns} : \mathbb{R}^{T \times \mathcal{N} \times \mathcal{T}} \longrightarrow \mathbb{R}^{T \times \mathcal{N} \times \mathcal{T}}.$$

Algorithm 1: Phase of MODEL training by supervised learning. Learn the model to fit the data.

Function `model_training()`:
  Input: MODEL, $\boldsymbol{G}$, b, trns
  Output: MODEL

  `// Iterate over batch samples`
  for $i := 1 \dots B$ **do**
    $(\boldsymbol{X}_{i|}, \boldsymbol{Y}_{i|}) = \text{batch}(\boldsymbol{G}, \boldsymbol{T}_{i|})$

    $\hat{\boldsymbol{Y}}_{i|} = \text{MODEL}(\boldsymbol{X}_{i|})$ `// predict output`
    if trns **then**
      `// If defined, apply transformation`
      $\boldsymbol{Y}_{i|} = \text{trns}(\boldsymbol{Y}_{i|})$
      $\hat{\boldsymbol{Y}}_{i|} = \text{trns}(\hat{\boldsymbol{Y}}_{i|})$

    $L = \text{loss}(\hat{\boldsymbol{Y}}_{i|}, \boldsymbol{Y}_{i|})$ `// Compute loss`
    $\text{MODEL} = \text{OPT}(\text{MODEL}, L)$ `// Optimize weight parameter`

  return MODEL

3.2.2. Validation

The task of the validation is to control the learning target. The concept for this is described below.

Let be $\boldsymbol{G}$ defined as for the training, but containing samples not used for training. During the validation phase are no changes to the model applied, no parameter learning happens. Let be $\text{acc}(\cdot)$ a well-defined accuracy function,

$$\text{acc} : (\mathbb{R}^{T \times \mathcal{N} \times \mathcal{T}}, \mathbb{R}^{T \times \mathcal{N} \times \mathcal{T}}) \longrightarrow \mathbb{R}.$$

The metric $\text{acc}(\cdot)$ is used to measure how well the model learned the desired target.

Process of validation

The validation process of the model MODEL is stated in *algorithm* 2. After processing

Algorithm 2: Phase of MODEL validation. Control the behavior of the model.

Function `validation()`:
  Input: MODEL, $\boldsymbol{G}$, $\mathrm{ACC}_{\mathrm{best}}$, denorm

  `// Iterate over samples`
  for $t := 1 \dots T$ **do**
    $\hat{\boldsymbol{Y}}_t = \mathrm{MODEL}(\boldsymbol{X}_t)$ `// predict output`
    `// Stack new targets and prediction to previous`
    $\boldsymbol{Y} = [\boldsymbol{Y}^\top \parallel \boldsymbol{Y}_t^\top]^\top$
    $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{Y}}^\top \parallel \hat{\boldsymbol{Y}}_t^\top]^\top$
  if denorm **then**
    `// If defined, apply denormalization`
    $\boldsymbol{Y} = \mathrm{denorm}(\boldsymbol{Y})$
    $\hat{\boldsymbol{Y}} = \mathrm{denorm}(\hat{\boldsymbol{Y}})$
  $\mathrm{ACC} = \mathrm{acc}(\boldsymbol{Y}, \hat{\boldsymbol{Y}})$

  `// Check if this model is the best yet seen`
  if $\mathrm{ACC} > \mathrm{ACC}_{\mathrm{best}}$ **then**
    `// Save` MODEL `and new best accuracy`
    SAVE(MODEL, ACC)

all samples, the accuracy ACC computed and compared with the best before-seen accuracy $\mathrm{ACC}_{\mathrm{best}}$. The best accuracy and model are stored for later use. If the target values are normalized, they need to be denormalized before being provided to the accuracy function.

3.2.3. Process of evaluation

Different models are compared by the evaluation, as shown in the following sketched.

Dataset $\boldsymbol{G}$ with before not respected sample is used. Let's assume the set $\{\mathrm{ACC}_k \,|k = 1 \dots K\}$ contains the best accuracy on $\boldsymbol{G}$ from K different model. The mean benchmark result is $\mathrm{ACC}^{(\mathrm{avg})}$ and the standard deviation is $\mathrm{ACC}^{(\sigma)}$. Let be $\mathrm{acc}_k(\cdot)$ a

function computing this benchmark

$$(\mathrm{ACC}^{(\mathrm{avg})}, \mathrm{ACC}^{(\sigma)}) = \mathrm{acc}_k(\{\mathrm{ACC}_k \,|k = 1 \dots K\}).$$

3.3. Graph Neural Network

In the following subsection, a GNN model architecture is introduced that is compatible with the previously described concept of ML. A homogeneous GNN model is in section 3.3.1 first presented and further in section 3.3.2 generalized into a heterogeneous GNN model.

The overall motivation of GNN models is to combine two kinds of information:

- The embedded values, that are provided by the node attributes of the graph. *What information is embedded in the node features?*
- The graph structured, defined by nodes and their neighbors. *How do linked nodes influence each other?*

[4] A model respecting the graph topology is called GNN model.

3.3.1. Homogeneous graphs

Within this section, the Message-passing framework (MPF) for homogeneous graphs is defined.

Let be the dataset $\boldsymbol{G} = [\boldsymbol{G}_1, \dots, \boldsymbol{G}_T]^\top$ containing homogeneous graph samples. Further, let be MODEL a GNN model learning supervised. Different from for general ML is the input for a GNN model a graph sample $\boldsymbol{G}_t$,

$$\hat{\boldsymbol{Y}}_t = \mathrm{MODEL}(\boldsymbol{H}^{(0)}, \boldsymbol{A}_{\mathrm{list}}, \boldsymbol{X}_t^{\mathcal{E}}).$$

Therefore, the input is composed of (1) the node features $\boldsymbol{H}^{(0)} := \boldsymbol{X}_t \in \mathbb{R}^{\mathcal{N} \times \mathcal{F}^{(\mathcal{N})}}$, (2) the graph topology $\boldsymbol{A}_{\mathrm{list}}$ and (3) the edge features $\boldsymbol{X}_t^{\mathcal{E}} \in \mathbb{R}^{\mathcal{E} \times \mathcal{F}^{(\mathcal{E})}}$.

Message-passing framework

A GNN layer learning the graph structure is called MPF. Let be $\boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})}$ a neural message receiving at node i. $\boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})}$ consists out of, into it passed node embeddings from all neighbors $\boldsymbol{h}_j^{(\mathrm{l}-1)}, \forall j \in \boldsymbol{A}(i)$. A single message sent from a neighbor j to i is aggregated by $\phi(\cdot)$. These messages are merged by an aggregation function into the neural message $\boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})} := \underset{j \in \boldsymbol{A}(i)}{\oplus}(\cdot)$.

From observing graph properties, two qualities should be respected for processing a graph $\boldsymbol{G}_t$:

- The set of nodes $\boldsymbol{\mathcal{N}}$, edges $\boldsymbol{\mathcal{E}}$ and node neighbors $\boldsymbol{A}(i)$ for $i \in \boldsymbol{\mathcal{N}}$ can be various large.
- The nodes $i \in \boldsymbol{\mathcal{N}}$ are not ordered, hence are multiple adjacency matrices $\boldsymbol{A}^{'}$ possible.

[4] The message aggregation function $\underset{j \in \boldsymbol{A}(i)}{\oplus}(\cdot)$ should therefore fulfill one of the following properties:

- Is invariant to the node order in the adjacency matrix $\boldsymbol{A}$,

$$f(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{\top}) = f(\boldsymbol{A}). \tag{3.10}$$

- Is consistent with node order,

$$f(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{\top}) = \boldsymbol{P}f(\boldsymbol{A}). \tag{3.11}$$

Defined as a permutation matrix $\boldsymbol{P}$. [4]

Let be $f_{\mathrm{msg}}^{(\mathrm{l})}$ a MPF defined as

$$\begin{aligned}
\boldsymbol{h}_i^{(\mathrm{l})} &= f_{\mathrm{msg}}^{(\mathrm{l})}(\boldsymbol{h}_i^{(\mathrm{l}-1)}, \boldsymbol{A}_{\mathrm{list}}, \boldsymbol{H}^{(\mathrm{l}-1)}, \boldsymbol{X}_t^{\mathcal{E}}) && (3.12)\\
&= \lambda(\boldsymbol{h}_i^{(\mathrm{l}-1)}, \underset{j \in \boldsymbol{A}(i)}{\oplus}(\phi(\boldsymbol{h}_i^{(\mathrm{l}-1)}, \boldsymbol{h}_j^{(\mathrm{l}-1)}, \boldsymbol{x}_{t,(i,j)}^{\mathcal{E}}))) && (3.13)\\
&= \lambda(\boldsymbol{h}_i^{(\mathrm{l}-1)}, \boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})}). && (3.14)
\end{aligned}$$

With an update function, $\lambda(\cdot)$ regulates the influence between the past layer $l-1$ and the neural message. [4] Moreover, notice is that the edge features are not to be updated.

Graph Convolution

A simple MPF for a GCN architecture generates the message

$$\boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})} = \boldsymbol{W}_{\mathrm{self}}^{(\mathrm{l})} \boldsymbol{h}_i^{(\mathrm{l}-1)} + \boldsymbol{W}_{\mathrm{neigh}}^{(\mathrm{l})} \sum_{j \in \boldsymbol{A}(i)} \frac{\boldsymbol{x}_{t,(i,j)}^{\mathcal{E}} \cdot \boldsymbol{h}_j^{(\mathrm{l}-1)}}{\sqrt{A(i)A(j)}} \tag{3.15}$$

and is update by

$$\begin{aligned} \boldsymbol{h}_i^{(\mathrm{l})} &= \lambda(\boldsymbol{h}_i^{(\mathrm{l}-1)}, \boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})}) \\ &= \boldsymbol{m}_{\mathrm{i}}^{(\mathrm{l})} \end{aligned}$$

[22]. For the equation (3.15) is property (3.10) fulfilled. In this kind of graph convolution, the number of nodes $\mathcal{N}$ is not changing, and the graph structure remains. Due to this property, iterating over many GCN layer $0 << l < L$ can lead to the known issues below:

Vanashing signals Assuming $\boldsymbol{W}_{\mathrm{self}}^{(\mathrm{l})} << \boldsymbol{W}_{\mathrm{neigh}}^{(\mathrm{l})}$ and focusing on node $i \in \boldsymbol{\mathcal{N}}$. The node features $\boldsymbol{h}_i^{(0)}$ and direct neighbor signals $\boldsymbol{h}_j^{(0)}, \forall j \in \boldsymbol{A}(i)$ are vanishing.

Over-smooting Signals in hidden values become too similar, $\boldsymbol{h}_i^{(\mathrm{l})} \simeq \boldsymbol{h}_j^{(\mathrm{l})}$.

[4]

Gatv2 Convolution

An attempt to be stable against the node degree is to assign attention weights to each neighbor $j, \forall j \in \boldsymbol{A}(i)$. The Graph Attention 2 layer (GATv2) uses these weights to

define the importance of neighbor j for node i. Further, it uses GATv2 a Multilayer perceptron (MLP) to learn edge scores,

$$s_{i,k} = \boldsymbol{a}^{\top} \sigma_{\text{att}}(\boldsymbol{W}_{\text{att}}^{\top(\text{l})}[\boldsymbol{h}_i^{(\text{l})} \parallel \boldsymbol{h}_k^{(\text{l})}] + \boldsymbol{W}_{\mathcal{E}\text{att}}^{(\text{l})} \boldsymbol{x}_{t,(i,k)}^{\mathcal{E}}), \tag{3.16}$$

for all edges $(i,k), \forall k \in \boldsymbol{A}(i)$. Further, are $\boldsymbol{a} \in \mathbb{R}^{2\mathcal{F}^{(\mathcal{N})}}$, $\boldsymbol{W}_{\text{att}}^{(\text{l})} \in \mathbb{R}^{\mathcal{F}^{(\mathcal{N})} \times \mathcal{F}^{(\text{H,l})}}$ and $\boldsymbol{W}_{\mathcal{E}\text{att}}^{(\text{l})} \in \mathbb{R}^{\mathcal{F}^{(\mathcal{E})} \times \mathcal{F}^{(\text{H,l})}}$ learnable weight parameters and $\sigma_{\text{att}}(\cdot)$ a non-linear function. Knowing the edge scores, a *softmax* based attempt produces a normalized attention weight $0 < \alpha_{i,j} < 1$ for each neighbor j of the node i,

$$\alpha_{i,j} = \frac{\exp(s_{i,j})}{\sum_{k \in \boldsymbol{A}(i)} \exp(s_{i,k})}. \tag{3.17}$$

Using (3.17) the message aggregation and update of the hidden value may be

$$\boldsymbol{h}_i^{(\text{l}+1)} = \alpha_{i,i} \boldsymbol{W}_{\text{att}}^{(\text{l})} \boldsymbol{h}_i^{(\text{l})} \sum_{j \in \boldsymbol{A}(i)} \alpha_{i,j} \boldsymbol{W}_{\text{att}}^{(\text{l})} \boldsymbol{h}_j^{(\text{l})},$$

for all $i, \forall i \in \boldsymbol{\mathcal{N}}$. [23], [2], [21]

Example

The example in *appendix* A.2 provides an intuition for the MPF. Node-degree instability and the problem of vanishing signals are also illustrated.

3.3.2. Heterogeneous graphs

In this subsection is the MPF from (3.13) generalized to process a heterogeneous graph $\boldsymbol{G}_t$.

$\boldsymbol{G}_t$ contains different *nent-link*s $\boldsymbol{r} \in \boldsymbol{R}$, with different node feature dimension $\mathcal{F}^{(\text{r}^{(\mathcal{N})})}$ and different edge feature dimension $\mathcal{F}^{(\boldsymbol{r})}$. Each *nent-link* requires therefore its own MPF $f_{\text{msg}}^{(\text{l},\boldsymbol{r})}$. This has the effect of learning each *nent-link* more distinct. However, this requires additional weight parameters for each *nent-link*.

Massage-passing framework map

Let be MODEL a heterogeneous model with layer l a GCN layer. This layer is a MPF map. The input for the first layer of MODEL consists of features for all node types,

$$\boldsymbol{H}^{(0,\mathrm{r}^{(\mathcal{N})})} = \boldsymbol{X}_t^{(\mathrm{r}^{(\mathcal{N})})} \in \mathbb{R}^{\mathcal{N}^{(\mathrm{r}^{(\mathcal{N})})} \times \mathcal{F}^{(\mathrm{r}^{(\mathcal{N})})}}, \forall r_i^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})},$$

and edge types,

$$\boldsymbol{X}_t^{(\boldsymbol{r})} \in \mathbb{R}^{\mathcal{E}^{(\boldsymbol{r})} \times \mathcal{F}^{(\boldsymbol{r})}}, \forall \boldsymbol{r} \in \boldsymbol{R}.$$

The last layer output and target value are defined as

$$\hat{\boldsymbol{Y}}_t^{(\mathrm{r}^{(\mathcal{N})})}, \boldsymbol{Y}_t^{(\mathrm{r}^{(\mathcal{N})})} \in \mathbb{R}^{\mathcal{N}^{(\mathrm{r}^{(\mathcal{N})})} \times \mathcal{T}^{(\mathrm{r}^{(\mathcal{N})})}}, \forall r^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})}.$$

The MPF map l in MODEL contains a *nent-link* map,

$$\{f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})} : \forall \boldsymbol{r} \in \boldsymbol{R}\},$$

with MPF $f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})}(\cdot)$. Focusing a node $i \in \boldsymbol{\mathcal{N}}$, with neighbors $j, k \in \boldsymbol{A}(i)$ of different type $r_j^{(\mathcal{N})} \neq r_k^{(\mathcal{N})}$. This heterogeneous sub-graph $\boldsymbol{G}_t'$ is processed:

(1) The respective *nent-link*s $\boldsymbol{r}_{(i,j)}$ and $\boldsymbol{r}_{(i,k)}$ are mapped into different MPFs.

(2) Each MPF processes its input.

(3) The outputs of these MPFs are aggregated to update the hidden value of node i.

For each *nent-link* are the adjacency list $\boldsymbol{A}_{\mathrm{list}}^{\boldsymbol{r}}$, required node features $\boldsymbol{H}^{(\mathrm{l},\mathrm{r}^{(\mathcal{N})})}$ and edge features $\boldsymbol{X}_t^{(\boldsymbol{r})}$ mapped to its respective framework $f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})}$, i. e.

$$\boldsymbol{h}_i^{(\mathrm{l})} = \bigoplus_{\boldsymbol{r} \in \boldsymbol{R}} (f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})}(\boldsymbol{h}_i^{(\mathrm{l})}, \boldsymbol{A}_{\mathrm{list}}^{\boldsymbol{r}}, \boldsymbol{H}^{(\mathrm{l},\mathrm{r}^{(\mathcal{N})})}, \boldsymbol{X}_t^{(\boldsymbol{r})})).$$

And further for each node $i, \forall i \in \boldsymbol{\mathcal{N}}$.

The *nent-link* aggregation $\bigoplus_{\boldsymbol{r} \in \boldsymbol{R}}(\cdot)$ should fulfill either (3.10) or (3.11), to be consistent with the neural message aggregation $\bigoplus_{j \in \boldsymbol{A}(i)}(\cdot)$. [12]

Node type map

Assuming the model MODEL has a layer l of other kind than a MPF. This layer is a Node type map (NT map),

$$\{f^{(\mathrm{l,r}^{(\mathcal{N})})} : \forall r^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})}\},$$

processing for each node type $r^{(\mathcal{N})}$, the hidden values

$$\boldsymbol{H}^{(\mathrm{l,r}^{(\mathcal{N})})} = f^{(\mathrm{l,r}^{(\mathcal{N})})}(\boldsymbol{H}^{(\mathrm{l-1,r}^{(\mathcal{N})})}).$$

Example

The *figure 3.4* shows a heterogeneous batch set $\boldsymbol{G}_{i|}$ and two mapping layers. For each

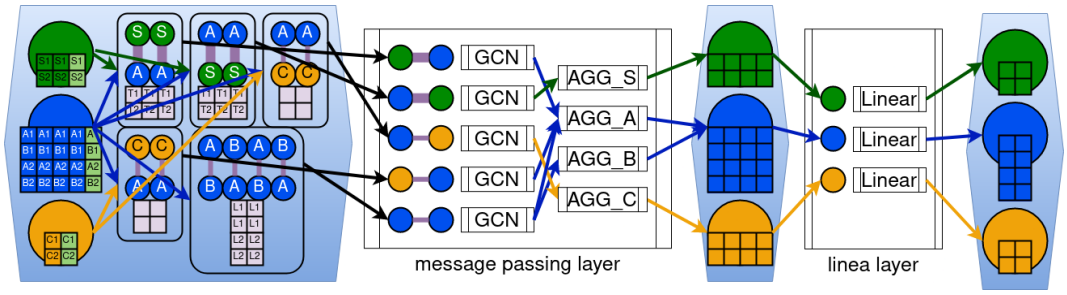


Figure 3.4.: (left) A batch set $\boldsymbol{G}_{i|}$ with batch size $b = 2$ as input for a MPF map. The following NT map processes the node features for each node type $r^{(\mathcal{N})}$ (right).

nent-link in $\boldsymbol{G}_{i|}$ contains the MPF map a parameterized MPF $f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})}(\cdot)$. Each $f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})}$ gets as input the edges, node features, and edge features from $\boldsymbol{G}_{i|}$, regarding $\boldsymbol{r}$. Each $f_{\mathrm{msg}}^{(\mathrm{l},\boldsymbol{r})}$ applies an inner update as defined in (3.14). For the finale update of $i^{(\mathrm{A})}$ is the *nent-link* aggregation of three outputs is required. The update to the other nodes is trivial. The next layer is a NT map $f^{(\mathrm{l,r}^{(\mathcal{N})})}$. These linear layers are learning distinct, the hidden values for each node type.

4. Approach

Taking the SE-task, this chapter presents the adaptation to the GNN approach. Formulating the SE-task for heterogeneous graph learning is defined in section 4.1. Based on this, it provides the second section 4.2, an overview of the data fetching. In the third section 4.3 is the validation metric presented. The last section 4.4 provides information about the project code.

4.1. Concept

Learning a GNN model to solve the SE-task is based on the concept, presented in this section. First are the concepts for the different node and edge types defined in section 4.1.1. In the second section 4.1.2 follows the association with attributes.

This work is only focusing on the topology of the *Allensbach* LVN. Since [20] only focused on learning one topology, this work continues on this setup. It is assumed that a LVN contains metadata $\boldsymbol{M}$, as required below.

4.1.1. Node types and edge types

This work uses the in [11] proposed node types

$$\boldsymbol{R}^{(\mathcal{N})} := \{\text{slack},\ \text{measured},\ \text{unmeasured}\}$$

and edge types

$$\boldsymbol{R}^{(\mathcal{E})} := \{\text{transformer},\ \text{line}\}.$$

The node types represent:

slack node This node $i^{(\mathrm{s})} \in \boldsymbol{\mathcal{N}}^{(\mathrm{slack})}$ determines the complex voltage $|V|^{(s)}$ and $\theta^{(s)}$ provided for the LVN [6]. The slack node is always determined via the metadata $\boldsymbol{M}$ and it holds $\mathcal{N}^{(\mathrm{slack})} = 1$.

measured node For these nodes $i, \forall i \in \boldsymbol{\mathcal{N}}^{(\mathrm{meas})}$ are power measurements available.

unmeasured node Due to physical limitations, reduction of monetary costs, or other technical difficulties, there are no measurements available on nodes $i \in \boldsymbol{\mathcal{N}}^{(\mathrm{unmeas})}$. Such a node may be an apartment building, just an underground link, or a supermarket.

So stand the edge types for:

transformer edge The *transformator* connection *slack* node $i^{(\mathrm{s})}$ and node $i, i \neq i^{(\mathrm{s})}$. The network level 6 *transformer* is defined as a $0.4kV$-*transformer*.

line edge Collection of standardized electrical wires used in the German LVN.

4.1.2. Features and targets

It is assumed that at most the information is available, as required for solving the general SE-task. In general SE:

- the active power and reactive power must be available for all nodes, i. e. $\boldsymbol{X}_t^{(\mathrm{slack})}$ and $\boldsymbol{X}_t^{(\mathrm{meas})}$, and
- the wire properties are known, i. .e. $\boldsymbol{X}_t^{(*,\mathrm{trafo},*)}$ and $\boldsymbol{X}_t^{(*,\mathrm{line},*)}$.

These features are provided to a heterogeneous model MODEL. MODEL then estimates $\hat{\boldsymbol{Y}}_t$ for the system state $\boldsymbol{Y}_t$.

This work uses the feature dimension as defined by [11] . The node and edge attributes available for the different types are defined in *table 4.1.*

Table 4.1.: Attributes and targets are available for different node and edge types.

Type	**Attribute/ Target**	**Dimension**
slack node	$\boldsymbol{x}_{t,i}^{(\text{slack})} := [P_i^{\text{gener}}, Q_i^{\text{gener}}]^\top$	$\mathcal{F}^{(\text{slack})} := 2$
measured node	$\boldsymbol{x}_{t,i}^{(\text{meas})} := [P_i^{\text{load}}, P_i^{\text{gener}}, Q_i^{\text{load}}, Q_i^{\text{gener}}]^\top$	$\mathcal{F}^{(\text{meas})} := 4$
unmeasured node	$\boldsymbol{x}_{t,i}^{(\text{unmeas})} := [0]^\top$	$\mathcal{F}^{(\text{unmeas})} := 1$
slack node	$\boldsymbol{y}_{t,i}^{\text{slack}} := [\lvert V_i\rvert, \theta_i]^\top$	$\mathcal{T}^{(\text{slack})} := 2$
measured node	$\boldsymbol{y}_{t,i}^{\text{meas}} := [\lvert V_i\rvert, \theta_i]^\top$	$\mathcal{T}^{(\text{meas})} := 2$
unmeasured node	$\boldsymbol{y}_{t,i}^{\text{unmeas}} := [\lvert V_i\rvert, \theta_i]^\top$	$\mathcal{T}^{(\text{unmeas})} := 2$
transformer edge	$\boldsymbol{x}_{t,(i,j)}^{(*,\text{trafo},*)} := [X_{i,j}, R_{i,j}, G_{i,j}, B_{i,j}, \tau_{i,j}, \theta_{i,j}^{(T)}]^\top$	$\mathcal{F}^{(*,\text{trafo},*)} := 6$
line edge	$\boldsymbol{x}_{t,(i,j)}^{(*,\text{line},*)} := [X_{i,j}, R_{i,j}, G_{i,j}, B_{i,j}]^\top$	$\mathcal{F}^{(*,\text{line},*)} := 4$

A positive signed feature value always represents injection and negative consumption of power in the LVN. The different node features represent

generator Provided power P_i^{gener} or Q_i^{gener} at node i.

load The power P_i^{load} or Q_i^{load} is either consumed or provided. Possible injected power by a *storage unit* is (1) used to demand the *load* on the attached node i and (2) afterward provided to neighbor nodes $j \in \boldsymbol{A}(i)$.

For the node types, this means:

slack node By definition, it only provides power to its neighbors.

measured node Provided or consumed power is known.

unmeasured node No data is available, as defined. However, target values are known for supervised learning.

The two edge types holding the similar properties:

line edge The physical properties: series reactance, series resistance, shunt conductivity, and shunt susceptance are used. These edge features are relative to the V_{nom}.

transformer edge All *line edge* features are available. The transformer also provides information about its tab ratio and phase shifting to the model.

On all nodes, the target feature $\boldsymbol{y}_{t,i}$ always the node state. This allows a combined processing via

$$\begin{aligned}\boldsymbol{Y}_t &= [\boldsymbol{Y}_t^{(\text{slack})\top} \parallel \boldsymbol{Y}_t^{(\text{meas})\top} \parallel \boldsymbol{Y}_t^{(\text{nnmeas})\top}]^\top \\ \hat{\boldsymbol{Y}}_t &= [\hat{\boldsymbol{Y}}_t^{(\text{slack})\top} \parallel \hat{\boldsymbol{Y}}_t^{(\text{meas})\top} \parallel \hat{\boldsymbol{Y}}_t^{(\text{unmeas})\top}]^\top.\end{aligned}$$

Hence, MODEL is learning the grid state $\boldsymbol{Y}_t \in \mathbb{R}^{\mathcal{N}\times\mathcal{T}}$. The voltage magnitude target $y_{t,i}^{(|\text{V}|)}$ and the voltage angle target $y_{t,i}^{(\theta)}$ are the results from (3.8). The *table A.1* provides the units for all physical properties.

4.2. Data fetching

By the ISE created and provided, LVN sets need to be parsed and stored in a graph data structure compatible for GNN models. This section presents an overview of the parsing, creating out of LVNs a data structure $\boldsymbol{G}$. In the first section 4.2.1, the actual parser concept is presented. Using the section above, section 4.2.2 describes the attribute aggregation. Further, in section 4.2.3 are policies for assigning node types defined.

4.2.1. Parser pipeline

In practice, different LVN $\{\boldsymbol{G}'^{(1)}, \ldots, \boldsymbol{G}'^{(K)}\}$ are used. The parser merges these into one dataset (graph time series) $\boldsymbol{G}$ as sketched in *appendix* C by *algorithm* 4 and below in *figure 4.1.* The basic idea is:

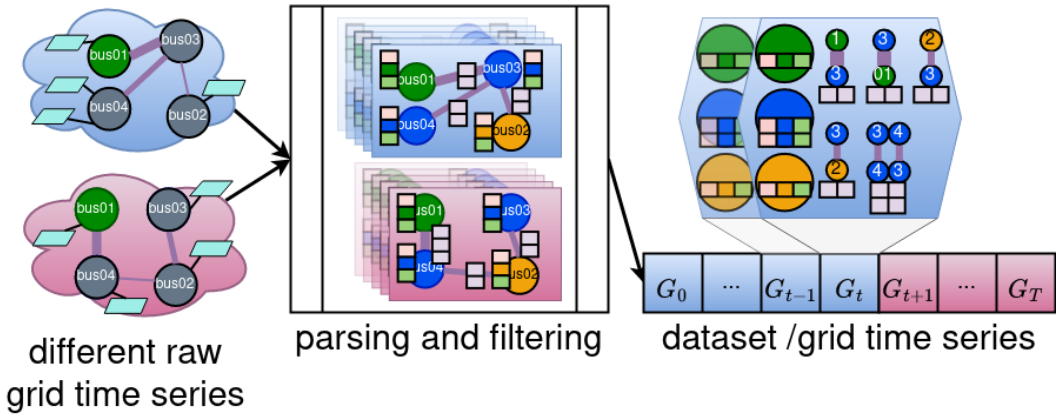


Figure 4.1.: LVN as input for the filter and parser and a dataset $\boldsymbol{G} = [\boldsymbol{G}_0^\top \parallel \ldots \parallel \boldsymbol{G}_T^\top]^\top$ as output.

(1) to parse and aggregate a LVN $\boldsymbol{G}^{(\mathrm{k})}$ into independent graph samples $\boldsymbol{G}_t^{(\mathrm{k})}$,

(2) repeat this for all K different LVN and

(3) concatenate all samples together into a single dataset

$$\boldsymbol{G} = \Big\|_{k=1}^{K} \Big\|_{t=1}^{T^{(k)}} \boldsymbol{G}_t^{(\mathrm{k})\top}.$$

(1) is done:

(a) by aggregating interactors,

(b) assigning node types and edge types,

(c) collecting node attributes, and

(d) storing the data in the graph data structure.

The *algorithm* 3 provides a more detailed view of the data fetching. Each sample t is stored in the list and matrix representation, but with support for all possible *nent-link* combinations

$$\forall \boldsymbol{r} \in \boldsymbol{R}_{\text{ds}} = \boldsymbol{R}^{(\mathcal{N})} \times \boldsymbol{R}^{(\mathcal{E})} \times \boldsymbol{R}^{(\mathcal{N})}.$$

In *figure 4.2* is this $\boldsymbol{R}_{\text{ds}}$ presented. For a *nent-link* not contained in a LVN $\boldsymbol{G}^{'}$, an

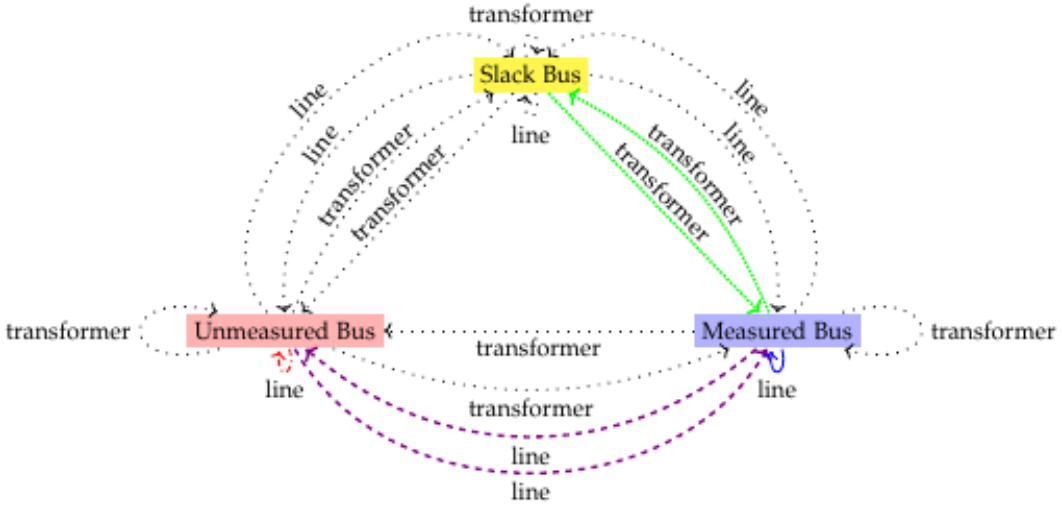


Figure 4.2.: All possible combinations of *nent-link* $\boldsymbol{r} \in \boldsymbol{R}_{\text{ds}}$. [11]

empty frame is added. If the dataset always contains the structure for each possible *nent-link* $\boldsymbol{r}$, a model must also support all *nent-link*s. As in *figure 4.1* illustrate, can $\boldsymbol{G}$ contain LVNs with different topologies. However, the in (b) applied assignment should cover the same $\boldsymbol{R}$.

4.2.2. Attribute aggregation

The in LVN value parsing and attribute aggregation from (a) are done for each sample t. The SE-task is solved via a solver (section 3.1.4). The output gains all node states, $[|V_i|, \theta_i]^\top, \forall i \in \boldsymbol{\mathcal{N}}$. The interactors $k = 1 \dots K_i$ on each node $i \in \boldsymbol{\mathcal{N}}$ in sample $\boldsymbol{G}_t$ are

processed twice. First generated by the solver are P_i and Q_i, as in (3.1) and (3.2). A second aggregation loop is generating, regarding the interactor types, the features below:

$$P_i^{\text{load}} = \sum_k^{K_i} \begin{cases} -P_{i,k} & \boldsymbol{M} \text{ tells about } \textit{load} \text{ interactor} \\ P_{i,k} & \boldsymbol{M} \text{ tells about } \textit{storage} \text{ interactor} \\ 0 & \text{else} \end{cases}$$

$$P_i^{\text{gener}} = \sum_k^{K_i} \begin{cases} P_{i,k} & \boldsymbol{M} \text{ tells about } \textit{generator} \text{ interactor} \\ 0 & \text{else} \end{cases}$$

$$Q_i^{\text{load}} = \sum_k^{K_i} \begin{cases} -Q_{i,k} & \boldsymbol{M} \text{ tells about } \textit{load} \text{ interactor} \\ Q_{i,k} & \boldsymbol{M} \text{ tells about } \textit{storage} \text{ interactor} \\ 0 & \text{else} \end{cases}$$

$$Q_i^{\text{gener}} = \sum_k^{K_i} \begin{cases} Q_{i,k} & \boldsymbol{M} \text{ tells about } \textit{generator} \text{ interactor} \\ 0 & \text{else} \end{cases}$$

The power values and complex voltages are node attributes. Values for the edge attributes are directly collected from the LVN. The solver also provides the edge attributes in the representation relative to V_{nom}.

4.2.3. Assignment of types

A type assignment is for all samples $t \in \boldsymbol{T}$ constant. These types are used for feature and target selection. The assignment of follows Edge-type policy (ETP) and Node-type policy (NTP). A chosen NTP assigns node types and an ETP edge types.

GNNSE2 policies

[11] proposes the assignment of the types defined below. One node $i \in \boldsymbol{\mathcal{N}}$, well tagged in the graph metadata $\boldsymbol{M}$, is assigned as *slack node* $i^{(\mathrm{s})} \in \boldsymbol{\mathcal{N}}^{(\mathrm{slack})}$. A node $i \in \boldsymbol{\mathcal{N}}$ with over time constant zero active power is defined as *unmeasured node* else as *measured node*:

$$r_i^{(\mathcal{N})} = \begin{cases} \textit{slack node} & \boldsymbol{M} \text{ tells } \textit{slack node} \text{ functionality} \\ \textit{unmeasured node} & P_{t,i} == 0, \forall t \in \boldsymbol{T} \\ \textit{measured node} & \text{otherwise} \end{cases}$$

The information about the edge type $\boldsymbol{R}^{(\mathcal{E})}$ for edge $(i,j) \in \boldsymbol{\mathcal{E}}$ is given by the metadata $\boldsymbol{M}$:

$$\forall (i,j) \in \boldsymbol{\mathcal{E}}, r_{(i,j)}^{(\mathcal{E})} = \begin{cases} \textit{transformer edge} & \boldsymbol{M} \text{ tells } \textit{transformer edge} \text{ functionality} \\ \textit{line edge} & \text{otherwise} \end{cases}$$

UnmeasuredPQZero node-type policy

A slightly redefined NTP is *UnmeasuredPQZero*. In the case of $Q_{t,i} \neq 0$, with this adaptation, no information is erased. A node $i \in \boldsymbol{\mathcal{N}}$ with over time constant zero active power and also reactive power is defined as *unmeasured node* else, as *measured node*,

$$\forall i \in \boldsymbol{\mathcal{N}}, r_i^{(\mathcal{N})} = \begin{cases} \textit{slack node} & \boldsymbol{M} \text{ for } \textit{slack node} \text{ functionality} \\ \textit{unmeasured node} & P_{t,i} == 0 \wedge Q_{t,i} == 0, \forall t \in \boldsymbol{T} \\ \textit{measured node} & \text{otherwise} \end{cases} .$$

This policy removes no information from the electrical grid relevant for SE. When assigning type *unmeasured node* instead of *measured node* aggregated information can be lost,

$$\begin{aligned} \boldsymbol{x}_{t,i}^{(\mathrm{unmeas})} &\neq \boldsymbol{x}_{t,i}^{(\mathrm{meas})} \\ [0]^\top &\neq [P_i^{\mathrm{load}}, P_i^{\mathrm{gener}}, Q_i^{\mathrm{load}}, Q_i^{\mathrm{gener}}]^\top . \end{aligned}$$

But for the power flow, only the information about the power at a node i is relevant,

$$\begin{aligned}[0]^\top &\approx [P_i^{\text{load}} + P_i^{\text{gener}}, Q_i^{\text{load}} + Q_i^{\text{gener}}]^\top \\ &\approx [P_i, Q_i]^\top \\ &\approx [0, 0]^\top.\end{aligned}$$

Hence, the model can access all relevant active power and reactive power measurements. However, the *unmeasured node* type is required, due to parser limitations.

RandomUnmeasured node-type policy

Related works defined graphs containing sparse information to test how robust a GNN model is. Therefore, a more realistic second NTP is defined. k randomly chosen *non-slack node*s are defined as *unmeasured node*s. Node type *unmeasured node* allows removing information from $\boldsymbol{X}_t^{(\text{meas})}$. This assignment stated formally is:

$$\forall i \in \boldsymbol{\mathcal{N}}, r_i^{(\mathcal{N})} = \begin{cases} \textit{slack node} & \boldsymbol{M} \text{ tells about } \textit{slack node} \text{ functionality} \\ \textit{unmeasured node} & i \in \boldsymbol{\mathcal{N}}^{(\text{unmeas})} \\ \textit{measured node} & \text{otherwise} \end{cases}.$$

4.2.4. Data filter

Given a dataset $\boldsymbol{G}$, there are additional operations possible to prepare the data for a model. One option is to standardize all node attributes. The opposite thinking, is to transform the node targets from a relative representation into the actual one. These filters are defined in this section.

Value standardization

As per convention, the target values are in different relative representations, and active power and reactive power are without modification by default. As from *figure*

2.1 to extract, also the target values are distributed in different ranges. The commonly used standard score is used to standardize the values

$$z_{t,i}^{(\mathrm{r}(\mathcal{N}))} = \frac{z_{t,i}^{'(\mathrm{r}(\mathcal{N}))} - \bar{z}^{(\mathrm{r}(\mathcal{N}))}}{\sigma_z^{(\mathrm{r}(\mathcal{N}))}}.$$

All training and validation samples are pre-computed to gain the mean $\bar{z}$ and standard deviation σ_z,

$$\bar{z}^{(\mathrm{r}(\mathcal{N}))} = \frac{1}{T\mathcal{N}^{(\mathrm{r}(\mathcal{N}))}} \sum_{t \in \boldsymbol{T}} \sum_{i \in \boldsymbol{\mathcal{N}}^{(\mathrm{r}(\mathcal{N}))}} z_{t,i}^{(\mathrm{r}(\mathcal{N}))}$$

$$\sigma_z^{(\mathrm{r}(\mathcal{N}))} = \sqrt{\frac{1}{T\mathcal{N}^{(\mathrm{r}(\mathcal{N}))}} \sum_{t \in \boldsymbol{T}} \sum_{i \in \boldsymbol{\mathcal{N}}^{(\mathrm{r}(\mathcal{N}))}} (z_{t,i}^{(\mathrm{r}(\mathcal{N}))} - \bar{z}^{\mathrm{r}(\mathcal{N}))})^2}.$$

As notated, this is done for all node features $\forall z \in \{P^{\mathrm{load}}, P^{\mathrm{gener}}, Q^{\mathrm{load}}, Q^{\mathrm{gener}}\}$ and each node type $\forall r^{(\mathcal{N})} \in \boldsymbol{R}^{(\mathcal{N})}$. The features of *unmeasured node* are constant in time,

$$\boldsymbol{x}_{t,i}^{'(\mathrm{unmeas})} = [0]^\top, \forall t \in \boldsymbol{T} \implies \boldsymbol{x}_{t,i}^{(\mathrm{unmeas})} = [0]^\top, \forall t \in \boldsymbol{T}.$$

By always taking special care of dividing by zero, i. e. $\sigma_z^{(\mathrm{r}(\mathcal{N}))} = 0$, some attributes are not touched. Also, the targets are normalized. The statistic values $\bar{z}^{(\mathrm{y})}$ and $\sigma_z^{(\mathrm{y})}$ are defined over all nodes $i \in \boldsymbol{\mathcal{N}}$. This is done, since the attributes for the targets $\boldsymbol{Y}$ are the same $z \in \{|V|, \theta\}$ for all node types. Hence, the standard score for a target attribute is

$$z_{t,i}^{(\mathrm{y})} = \frac{z_{t,i}^{'(\mathrm{y})} - \bar{z}^{(\mathrm{y})}}{\sigma_z^{(\mathrm{y})}}.$$

4.2.5. Voltage transformation

The general complex voltage as the target value is represented via (3.8). This transformation is reversed:

$$y_{t,i}^{(|\mathrm{V}|,+)} = V_{\mathrm{nom}} \cdot y_{t,i}^{(|\mathrm{V}|)}$$

$$y_{t,i}^{(\theta,+)} = \frac{180}{\pi} \cdot y_{t,i}^{(\theta)}.$$

Let be $\text{trns}^{-1}(\cdot)$ a function applying this reverse transformation,

$$\boldsymbol{Y}^{(\psi,+)} = \text{trns}^{-1}(\boldsymbol{Y}^{(\psi)}, \delta^{(\psi)}). \tag{4.1}$$

4.3. Validation metric

This section defines the metric to control the learning of the model.

Let be $\boldsymbol{y}_{t,i} = [y_{t,i}^{(|\text{V}|)}, y_{t,i}^{(\theta)}]^\top \in \mathbb{R}^2$ the state of node i for sample t and $\hat{\boldsymbol{y}}_{t,i} = [\hat{y}_{t,i}^{(|\text{V}|)}, \hat{y}_{t,i}^{(\theta)}]^\top \in \mathbb{R}^2$ the estimation by MODEL. The basic threshold

$$\boldsymbol{\varepsilon}^{(\text{b})} = [\varepsilon^{(|\text{V}|,\text{b})}, \varepsilon^{(\theta,\text{b})}]^\top \tag{4.2}$$

is defined as:

$$\varepsilon^{(|\text{V}|,\text{b})} = 10^-3 \text{ pu}$$
$$\varepsilon^{(\theta,\text{b})} = 10^-2 \text{ rad.}$$

These values are defined for a potential application in the field. Applying the transformation (4.1) means an actual threshold of:

$$\varepsilon^{(|\text{V}|,+)} = 0.4V$$
$$\varepsilon^{(\theta,+)} = 0.6°.$$

Let be the targets values $\boldsymbol{\psi} \in \{|V|, \Theta\}$. Starting with the definition of the accuracy at a single node $i \in \boldsymbol{\mathcal{N}}$ of a sample $\boldsymbol{G}_t$. A predicted node target $\hat{y}_{t,i}^{(\psi)}$ is defined as correct, if it holds

$$\hat{y}_{t,i}^{(\psi)} \in \boldsymbol{\mathcal{Y}}_{t,i}^{\psi}, \tag{4.3}$$

with

$$\boldsymbol{\mathcal{Y}}_{t,i}^{\psi} = \{\hat{y}_{t,i}^{(\psi)} | y_{t,i}^{(\psi)} - \varepsilon^{(\psi)} \leq \hat{y}_{t,i}^{(\psi)} \leq y_{t,i}^{(\psi)} + \varepsilon^{(\psi)}\}. \tag{4.4}$$

The family of sets $\boldsymbol{\mathcal{Y}}_{t,i}^{\psi}$ contains all correct predictions for $y_{t,i}^{(\psi)}$. A prediction node state $\hat{\boldsymbol{y}}_{t,i}$ is correct, if all targets within are correct

$$\hat{\boldsymbol{y}}_{t,i} \in \boldsymbol{\mathcal{Y}}_{t,i},$$

with

$$\boldsymbol{\mathcal{Y}}_{t,i} = \{\hat{\boldsymbol{y}}_{t,i} | \bigvee_{\psi \in \boldsymbol{\psi}} \hat{y}_{t,i}^{(\psi)} \in \boldsymbol{\mathcal{Y}}_{t,i}^{\psi}\}.$$

For accuracy, all nodes $i = 1 \dots \mathcal{N}$ and all samples $\boldsymbol{G}_t, t = 1 \dots T$ are respected

$$\text{ACC} = \frac{|\{\hat{\boldsymbol{y}}_{t,i} | \hat{\boldsymbol{y}}_{t,i} \in \boldsymbol{\mathcal{Y}}_{t,i}, \forall i \in \boldsymbol{\mathcal{N}}, \forall t \in \boldsymbol{T}\}|}{\mathcal{N} \cdot T} \tag{4.5}$$

$$= \text{acc}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}, \boldsymbol{\varepsilon}). \tag{4.6}$$

The same is true for $\text{ACC}^{|\text{V}|}$ and ACC^{θ}, focusing only on one target, ψ.

4.4. Code standard

In this section, code improvements and design choices are discussed.

The field of AI plays a significant and increasing role in our modern world. Therefore, AI research should fulfill the highest standards for reproducibility. Still, absolute transparency results can be a challenge, due to well-placed random parameters and complex model architectures. Therefore, around fifty unit tests are defined, and a straight-code style is used. More information about unit tests and code standards is provided in *appendix* C.2.

5. Reproduction experiment

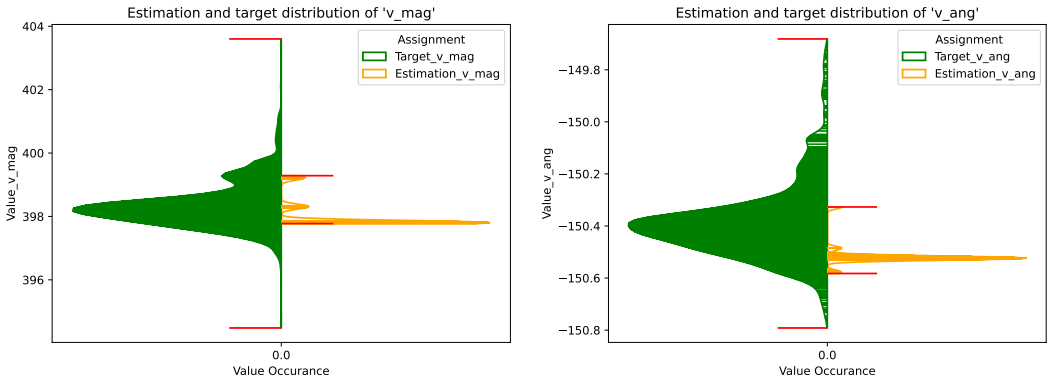
This section contains an analysis for the GNNSE 2023 model (*GNNSE3 model*). The first section 5.1 analyses the known variance problem from [20] . A second section 5.2 analyzation determines a design problem for the validation of voltage angle. In section 5.3 are the results collected.

The validation accuracy for the estimated node states is up to 50 %, with an voltage magnitude accuracy of 50 % and for voltage angle 90 %. The author of [11] stated two reasons: (1) the data distribution range for the target voltage magnitude is wider than for voltage angle, and (2) the value accumulation is more steep for voltage angle.

5.1. Poor variance

The section presents the analysis for the variance problem. The issue has been known since [11] and in the following reproduced.

GNNSE3 model and closely related models never reached an accuracy better than 60 %. The benchmarks for the evaluation are listed in *table 5.2*. Comparing the value distribution from [11] in *figure 2.1* with the one for *GNNSE3 model* in *figure 5.1*, the poor variance of the models is verified. The *figure 2.1* contrasts the ground truth (green) and estimated value (orange). In (a) is voltage magnitude and in (b) is the voltage angle compared. On the y-axes, for each occurred value, there is a horizontal bar with the amount of occurrence as its length. An ideal estimation would reflect



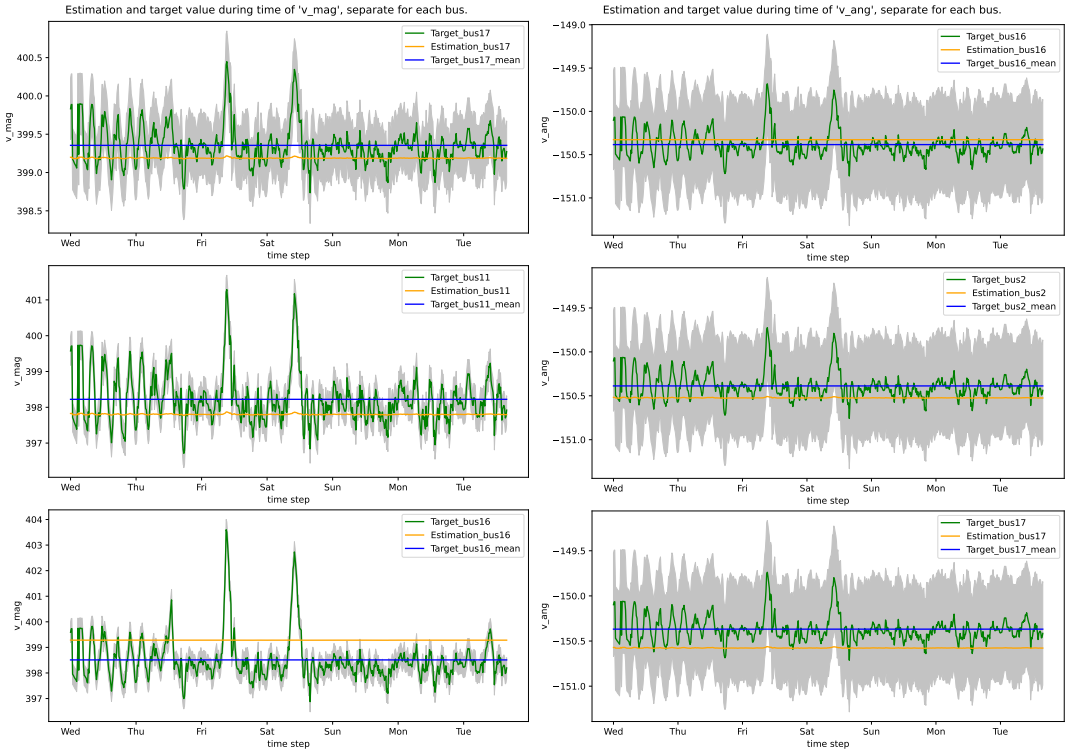
(a) Comparing voltage magnitude **(b)** Comparing voltage angle

Figure 5.1.: Violin plots comparing targets (green) with estimated target (orange) for *GNNSE3 model*.

all target nodes on the y-axes. The first difference in the setup for the plots is the by $\text{trns}^{-1}(\cdot)$ transformed value range. By definition of the SE-task, full knowledge about the *slack node* state is given. The model still estimates $\hat{\boldsymbol{y}}_{t,s}$, but the results for this node state are trivial. Since this is given, the results for *slack node* are not represented in *figure 5.1* and in the following plots. Further, the model is evaluated on *dataset* ds1 and for the shape, no filling was applied.

5.1.1. Model learns only bias

The results in *figure 5.1* can be taken into relation with the sample/time stamp t and interpreted per node. Such a setup is illustrated in *figure 5.2* for voltage magnitude in (a) and voltage angle in (b). Each plot shows the values for a different node, with the target values (green) and the value estimation of *GNNSE3 model* (orange). The total target node mean is blue, and further is the basic threshold (4.2) range (gray) in the background. The shape is forming the set of correct estimations $\boldsymbol{\mathcal{Y}}^{\psi}_{t,i}$. Again, the poor variance of *GNNSE3 model* appears. It seems the trained *GNNSE3 model* estimation is converged to the target mean value, with a small gap.



(a) voltage magnitude **(b)** voltage angle

Figure 5.2.: By *GNNSE3 model* estimated time series (orange) of target values (green) with one plot per node. The target mean is blue, and the threshold range is gray.

The nodes are selected based on the over-sampling estimation *L1-loss*. Selected is (top) the smallest loss, (middle) the closest to the average loss, and (bottom) the largest loss. This nearly constant estimation appears as the mean without outlier influence. The model generalization is motivated to cross low-variance samples with high density ranges, i. e. the model learns only the bias.

5.2. Poor validation

This section shows the missing control functionality for the voltage magnitude estimation.

In the following is an approximation of *GNNSE3 model* defined. Let be the estimated value for node i be

$$\begin{aligned}\hat{y}_{t,i}^{(\psi)} &= \beta^{(\psi)} + \frac{1}{T}\sum_{t=1}^{T} y_{t,i}^{(\psi)} + \mathcal{N}_{t,i}^{(\text{noise})} \\ &= \beta^{(\psi)} + \bar{y}_i^{(\psi)} + \mathcal{N}_{t,i}^{(\text{noise})}\end{aligned}$$

$\forall i \in \boldsymbol{\mathcal{N}}$ and with $\boldsymbol{\psi} \in \{|V|, \theta\}$ the targets and $\beta^{(\psi)}$ a respective offset. Using this observation, Constant Mean Estimation model (*CME model*) can be learned by

$$\begin{aligned}\hat{\boldsymbol{Y}}^{(\psi)} &= \text{CME}(\boldsymbol{X}^{(\psi)}) \\ &= \text{CME}(\boldsymbol{Y}^{(\psi)}) \\ &= [\bar{y}_i^{(\psi)}, \ldots, \bar{y}_{\mathcal{N}}^{(\psi)}]^\top.\end{aligned}$$

The behavior of the model is approximated by $\mathcal{N}_{t,i}^{(\text{noise})} := 0$ and the graph state is the input feature $\boldsymbol{X}^{(\psi)} = \boldsymbol{Y}^{(\psi)}$. That is, learning on a *dataset A* the $\hat{\boldsymbol{Y}}^{(\psi)}$, results in estimating on all *datasets B* the total node means of *dataset A*. The behavior of *CME1 model* and *CME1 mode2*, defined in *table 5.1*, is related to the *GNNSE3 model*. Therefore, *CME1 model* and *CME1 mode2* are used for a technical proof. For *CME1*

Table 5.1.: Different training datasets and settings for *CME model*s.

name	**training dataset**	$\boldsymbol{\beta^{(\lvert\mathbf{V}\rvert)}}$	$\boldsymbol{\beta^{(\Theta)}}$
CME1 model	ds1	0.0	0.0
CME1 mode2	ds2, ds3	−0.002	0.0

mode2 is a small mean shifting $\beta^{(|\text{V}|)} = -0.002$ (motivated by *figure 5.2*) defined. The benchmark results are presented in *table 5.2*. *CME1 model* is trained and evaluated on the same dataset, so the estimation is the actual total node mean. An accuracy of

Table 5.2.: Accuracies for evaluation on *dataset* ds1, with the different thresholds.

model	**Threshold**	**ACC**	**ACC$^{\|\mathbf{V}\|}$**	**ACC$^{\boldsymbol{\theta}}$**
GNNSE3 model	$[10^{-3}, 10^{-2}]^\top$	**0.517**	0.517	0.983
CME1 model	$[10^{-3}, 10^{-2}]^\top$	0.608	0.608	0.993
CME1 model	$[10^{-3}, 10^{-3}]^\top$	0.450	0.608	0.453
CME1 mode2	$[10^{-3}, 10^{-2}]^\top$	0.506	0.506	1.000

$\text{ACC}_{\textit{CME1 model}} > 60$ % and $\text{ACC}^{\boldsymbol{\theta}}_{\textit{CME1 model}} \approx 100$ %, for an estimated mean, makes the training trivial. A model validation cannot check the regression learning target. It is also to be noticed that *CME1 mode2* nearly reaches the performance of *GNNSE3 model*. Due to the accuracy metric, $\text{ACC}^{|\mathrm{V}|}$ is always the limiting factor for ACC.

5.3. Conclusion

A short conclusion of the results from the past sections is contained in the lines below.

For deep GNNs, the hidden values can become very similar. When applying GCN layer, the model should therefore be more flat. The limitation of ACC < 60 % comes from missing variance. Since for *GNNSE3 model* learning only applies to the bias, it is required to choose a layer $l, \forall l = 1 \dots L-1$ with non-trainable bias parameters. Furthermore, should the last layer $l = L$ be a linear layer with learnable bias parameter to make sure the model still orients to total node means. The threshold $\boldsymbol{\varepsilon}^{(\mathrm{b})}$ prevents controlling the learning target. However, since the complex voltage angle θ is in practice not that relevant, this setup remains. The threshold $\varepsilon^{(\theta,\mathrm{b})}$ still provides information about leaned biases.

6. Model approach

Taking respect to the analysis of the *GNNSE3 model*, different attempts are required for future GNN model. This chapter presents two different models along with the motivation for them. First in section 6.1 a model using the transformer filter, and a second one in section 6.2 using normalization.

6.1. GSETR

In this section, a naive motivated model is presented, called GSETR.

6.1.1. Transformation

The motivation for this model is based on (A.6). In the following is j the imaginary number. Using (A.6) and (3.8) gains for a node $i \in \boldsymbol{\mathcal{N}}$

$$P_{t,i} + Q_{t,i} = \delta^{(|\mathrm{V}|)} y_{t,i}^{(|\mathrm{V}|)} \exp(j\delta^{(\theta)} y_{t,i}^{(\theta)}) \cdot \underline{I}_i^*.$$

The node state arguments $\boldsymbol{y}_{t,i}$, in this equation, are the linear regression target. Applying (4.1) on $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ before passing these into the loss function $\mathrm{loss}(\boldsymbol{Y}^{(+)}, \hat{\boldsymbol{Y}}^{(+)})$, slightly changes the regression target. This changes $\hat{\boldsymbol{y}}_{t,i} = [\hat{y}_{t,i}^{(|\mathrm{V}|)}, \hat{y}_{t,i}^{(\theta)}]^\top$ into $\hat{\boldsymbol{y}}_{t,i}^{(+)} = [\delta^{(|\mathrm{V}|)} \hat{y}_{t,i}^{(|\mathrm{V}|,+)}, \delta^{(\theta)} \hat{y}_{t,i}^{(\theta,+)}]^\top$. All features are used as defined in section 4.1.2. However, the model still estimates $\hat{\boldsymbol{Y}}_t$. This linear transformer can be seen, as a weight for the influence, of the attribute losses on the total loss.

6.1.2. Relative loss function

The *GNNSE3 model* uses *L1-loss* as in (3.9). Since the value ranges of voltage magnitude $y_{t,i}^{(|\mathrm{V}|,+)}$ and voltage angle $y_{t,i}^{(\theta,+)}$ are not equally large, a relative loss function is chosen. From the benchmark test in [5] is the Relative Root Mean Squared Error (RRMSE) loss function selected,

$$L = \sqrt{\frac{\frac{1}{\mathcal{T}\mathcal{N}T}\sum_{t=1}^{T}\sum_{i=1}^{\mathcal{N}}\sum_{\psi=1}^{\mathcal{T}}(y_{t,i}^{(\psi)} - \hat{y}_{t,i}^{(\psi)})^2}{\sum_{t=1}^{T}\sum_{i=1}^{\mathcal{N}}\sum_{\psi=1}^{\mathcal{T}}(y_{t,i}^{(\psi)})^2}}.$$

6.1.3. Convolution layer

Bringing the example in *figure 1.1* together with the information in the *table B.1.* LVN contain a relatively low number of edges, and the average node degree is less than three. The node states $\boldsymbol{y}_{t,i}, \boldsymbol{y}_{t,j}$ of two nodes $i, j \in \boldsymbol{\mathcal{N}}$ with $j \in \boldsymbol{A}(i)$ are highly related. Important information for the general SE-task is provided to the model by edge features, i. e. the complex admittance values. Therefore, the GATv2 layer is chosen for the message aggregation. Its MLP based edge attention also learns edge features. Moreover, multiple attention weights can be learned, called heads. Techniques like sub-graph sampling brings an information loss into the message flow [4] . Therefore, these are likely not sufficient.

6.2. Standardized model

A second model, GSENR is defined, using normalized data as input.

A numeric stable model is required since, (1) the number of different LVNs is large, (2) each contains a unique power flow, and (3) not-seen loads can occur. The node features are node-vise normalized, and the targets are node-type independent normalized, as defined in section 4.2.4. The *appendix* D provides more detailed information about the models GSETR and GSENR.

7. Experiments

In this chapter, first in section 7.1 the two model approaches are analyzed for the *UnmeasuredPQZero* NTP. For the NTP *RandomUnmeasured*, a benchmark result is presented in the second section 7.2.

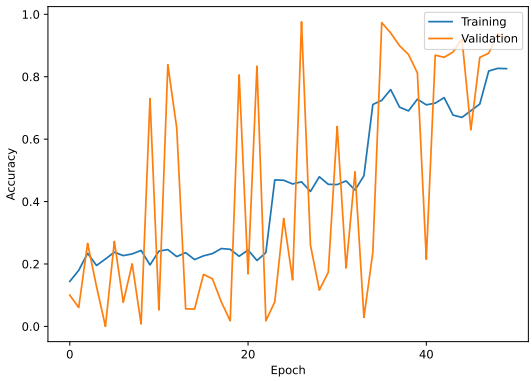
The validation scenario is a cross-scenario. The different models are (1) trained and validated on *dataset* $\{2, 4, 5\}$ and (2) evaluated on *dataset* ds1. The training-validate split is 75 % for training, without mixing the samples. This represents the case when a model learns from the past and applies it to future measurements. All statistics are for the basic threshold $\varepsilon^{(b)}$.

7.1. UnmeasuredPQZero policy

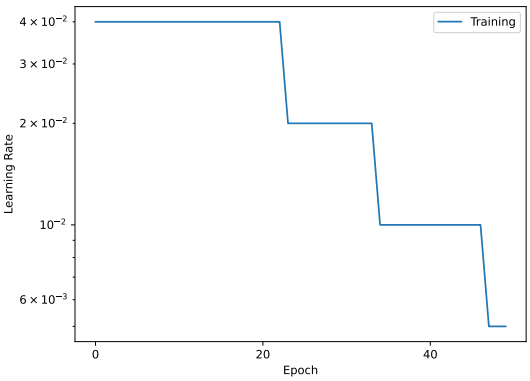
For this first experiment, the NTP *UnmeasuredPQZero* is used. Results and an analysis are presented in section 7.1.1 and in section 7.1.2. The *UnmeasuredPQZero* NTP leads to an unrealistic setup. Gained results, from this setup, are still relevant for a first model selection.

7.1.1. GSETR

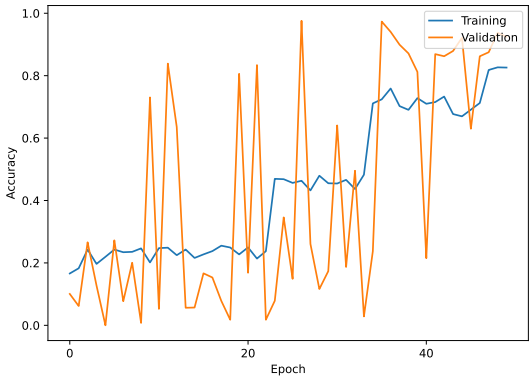
During the research, the GSETR model performed best regarding the accuracy metric. Selected statistic values are presented in *figure 7.1*. The statistical values are collected during 50 epochs, once for training (blue) and once for validation (orange). As for



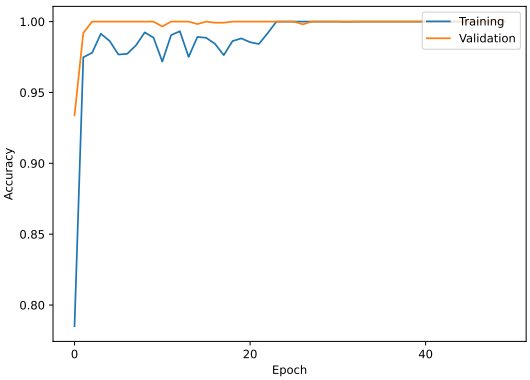
(a) Accuracy per epoch



(b) Learning rate per epoch



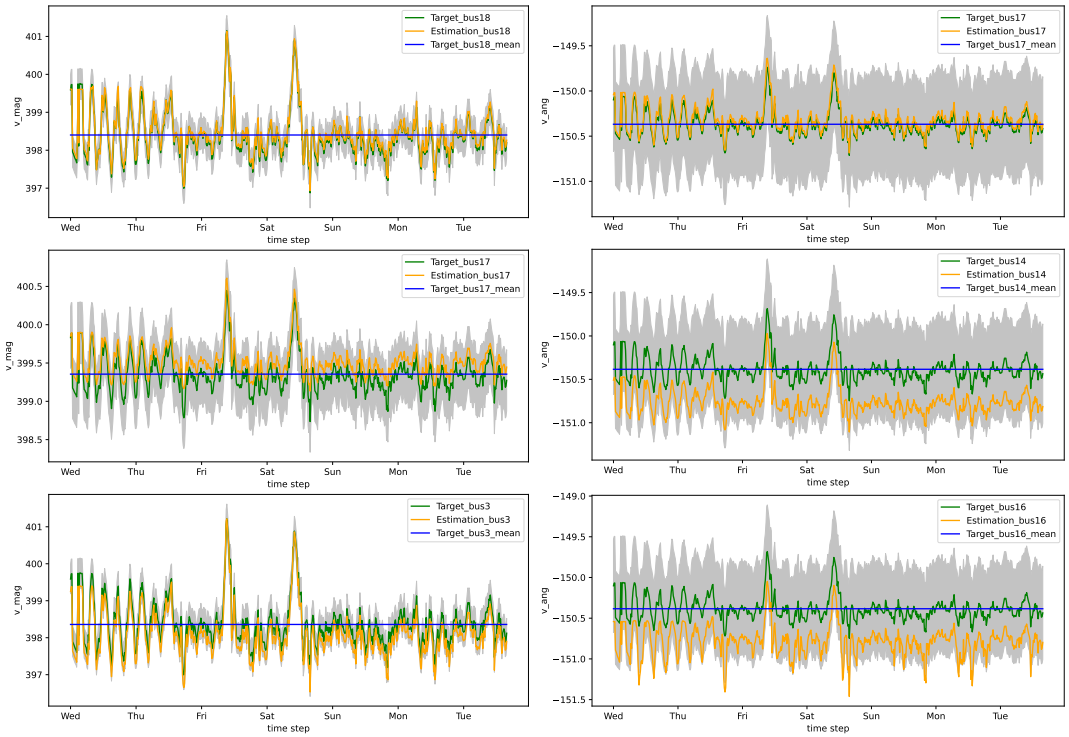
(c) Voltage magnitude accuracy $\mathrm{ACC}^{|\mathrm{V}|}$ per epoch



(d) Voltage angle accuracy ACC^{θ} per epoch

Figure 7.1.: Statistic values for GSETR during training and validation.

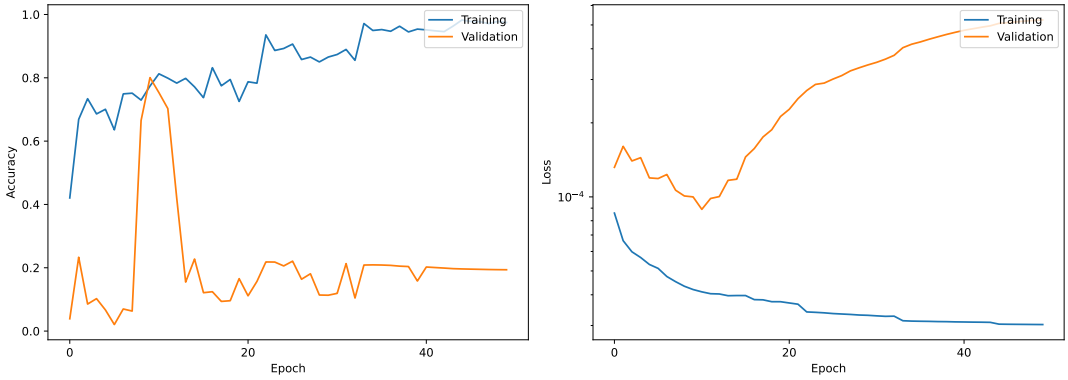
[11] and *GNNSE3 model* remains, the estimation of the voltage magnitude in (c) is the limiting factor. The GSETR converges instable, to a validation accuracy of ACC $\approx$ 97 %. The subfigure (a) shows a validation frequency with a large amplitude. On the other side, during training, the accuracy for voltage magnitude converges as the learning rate drops. Evaluated on *dataset* ds1 the model reaches an accuracy of ACC $\approx$ 99 %. Nearly all estimated targets in *figure 7.2* are within the threshold boundary. The voltage magnitude of the node 3 in (b) is often underestimated. For



(a) Voltage magnitude on three nodes **(b)** Voltage angle on three nodes

Figure 7.2.: Evaluation results of GSETR on *dataset* ds1. In the plots, there are targets (blue) and estimated targets (orange). A target threshold (gray) is in the background.

runs with a larger number of epochs, the validation accuracy becomes more stable as the learning rate decreases more.



(a) Accuracy per epoch **(b)** Loss per epoch

Figure 7.3.: Evaluation results for GSENR variant one.

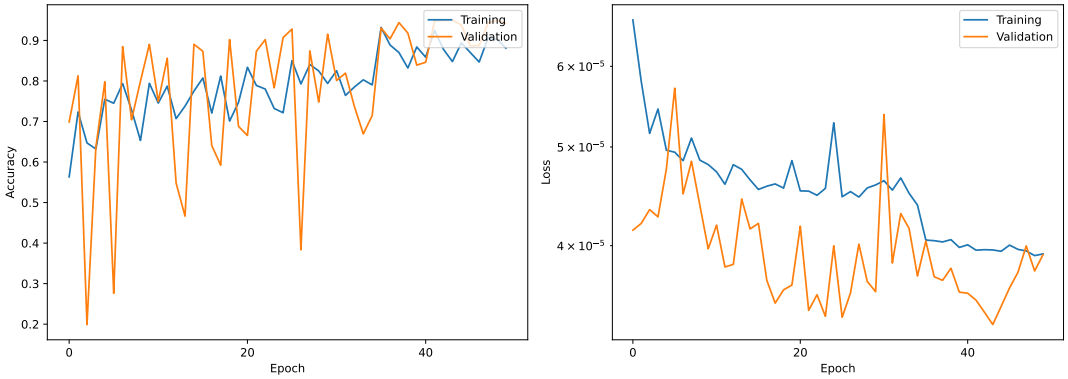
7.1.2. GSENR

Two architectures of the GSENR model are used for the experiment. For the variant in section 7.1.2, after each GCN layer follows a batch normalization and a Rectified Linear Unit (ReLU) activation function. In the second variant in section 7.1.2, batch normalization is not applied after the first layer.

In both setups, all mean and standard deviation values are pre-computed on *dataset* $\{2, 4, 5\}$. Pre-computed values are also preserved for the evaluation.

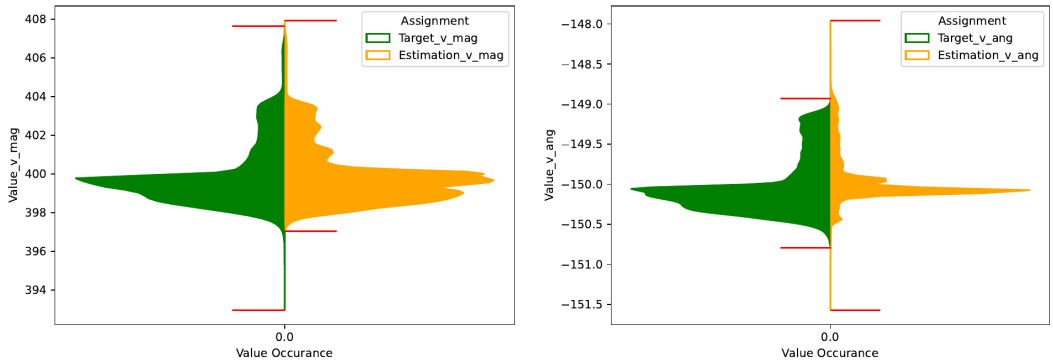
Full batch-normalization

An early overfitting of this model architecture can be concluded from *figure 7.3*. The model learns the training set, but does not generalize to the validation data. As good to see, in (a) diverges the training and validation accuracy. This is conformed by (b). As the model is overfitting, the losses diverge too.



(a) Accuracy per epoch

(b) Loss per epoch



(c) Value distribution of voltage magnitude target and estimation.

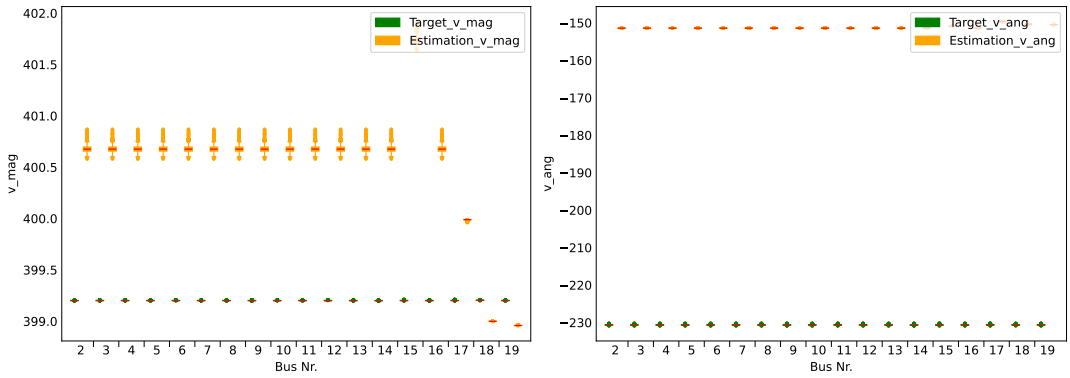
(d) Value distribution of voltage angle target and estimation.

Figure 7.4.: Training and validation results from GSENR variant two.

Late batch-normalization

When applying less batch normalization, the model is overfitting less. Results for this architecture are in figure *figure 7.4*. The inference from (a) and (b) is also that the model becomes more stable. The loss decreases over the different training and validation epochs. The *figure 7.4* also provides plots for the distribution of voltage magnitude (c) and voltage angle (d). Plotted estimations are from the epoch with the best validation accuracy. While the voltage magnitude target distribution is well covered, the estimated voltage angle values are spread too wide. Evaluating this model on *dataset* ds1 demon straits again, a low generalization, as the box plots

collection in *figure 7.5* is showing. On the y-axis is the value range. Each plot shows



(a) Value range of voltage magnitude target and estimation per node.

(b) Value range of voltage angle target and estimation per node.

Figure 7.5.: Evaluation results in box plots from a GSENR with less batch normalization.

18 pairs of box plots, representing the nodes. Each pair represents, in green, the target values and, in orange, the estimated values. An ideal estimation of a node would appear as two identical box plots. For the targets voltage magnitude in (a) and voltage angle in (b), the model applies a wrong offset. This offset is equal for most of the nodes. Therefore, the estimation for the most nodes is outside the threshold boundary.

7.2. RandomUnmeasured policy

As stated in section 3.1, not all interactors within the LVN are known or monitored. Robustness in this sense is required. A simple approach for such a benchmark setup is presented in this section. The NTP *RandomUnmeasured* is applied to simulate this behavior. Further, this setup is used to demonstrate that the model learns the node states of $i \in \boldsymbol{\mathcal{N}}^{(\text{unmeas})}$ from neighbor nodes $j \in \boldsymbol{A}(\boldsymbol{\mathcal{N}}^{(\text{unmeas})}), j \in \boldsymbol{\mathcal{N}}^{(\text{meas})}$.

Since the *gsenr* model provides poor benchmark results on NTP *UnmeasuredPQZero*,

it is left out. Nine GSETR models with different initialization seeds are defined. Furthermore, three seeds are used to define three different node assignment profiles of *RandomUnmeasured*. With $k = 9$ nodes declared as *unmeasured node*, half of the node features are suppressed. Each three of the three models is trained, validated, and evaluated on each node assignment profile. In *appendix* D are additional benchmark results listed.

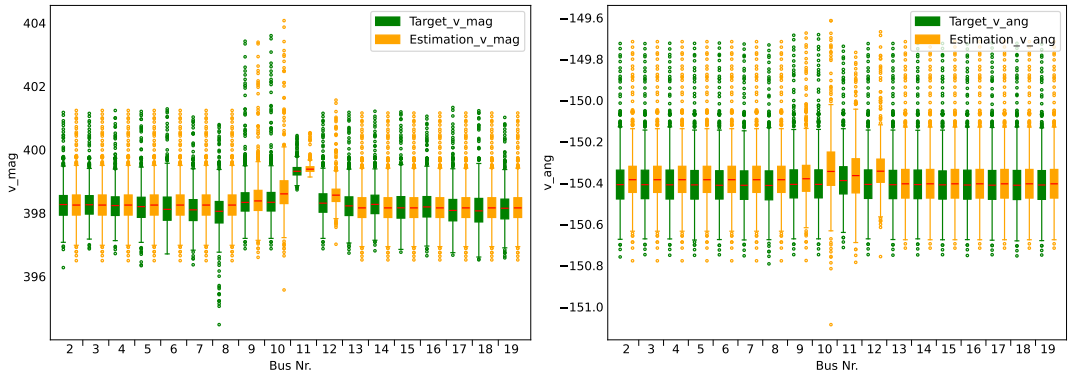
7.2.1. GSETR

The assignment of the node types influences the accuracy. If the distribution node $i \in \boldsymbol{\mathcal{N}}$ in the *Allensbach* gird (*figure 1.1*) is an *unmeasured node* instead of a *measured node* drops the accuracy $\triangle\,\mathrm{ACC} \approx 8\ \%$. However, the mean accuracy is for validation and evaluation around 91 %. The evaluation results for one model are plotted in

Table 7.1.: Benchmark results from GSETR models.

Phase	$\mathrm{ACC}^{(\mathrm{avg})}$	$\mathrm{ACC}^{(\sigma)}$
Validation	91.0 %	0.032 %
Evaluation	91.6 %	0.036 %

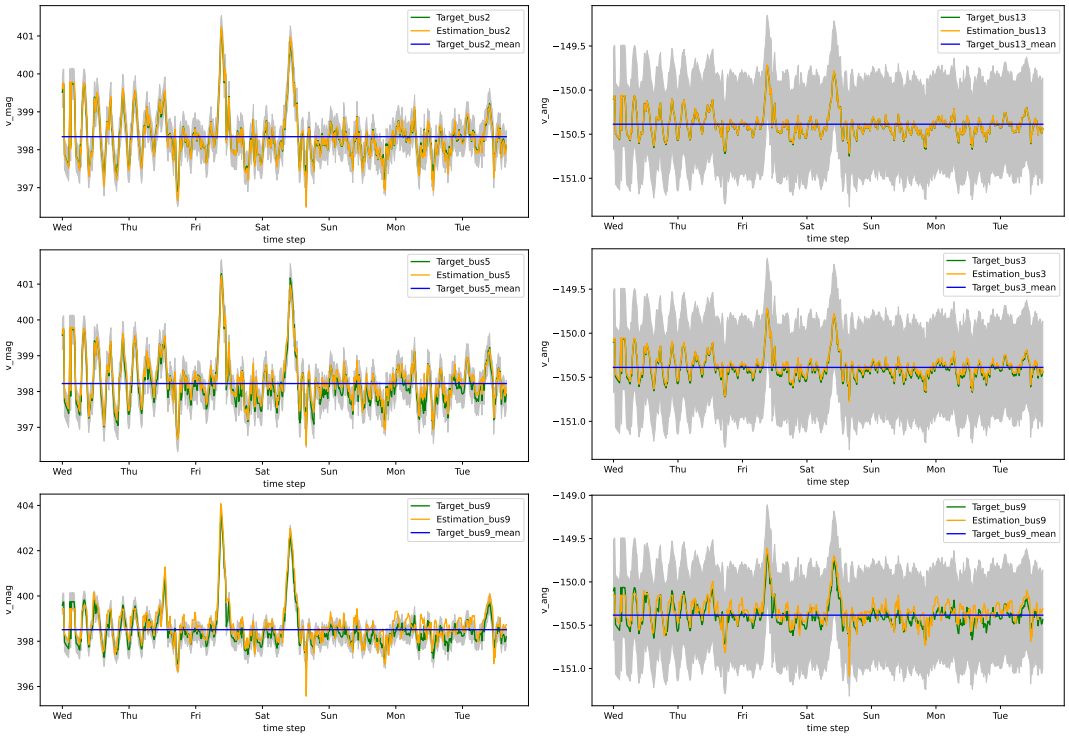
figure 7.6 and *figure 7.7*. For half of the nodes, no features had been accessible for the GSETR model. It can therefore be confirmed that the model learns the node state $i \in \boldsymbol{\mathcal{N}}$ from neighbor node $j \in \boldsymbol{A}(i)$. However, focusing on the last voltage magnitude plot in (a), the one with the largest loss. There is, e. g. node eight, the model is incapable of covering some outliers. The results for both targets adapting well to the input sequence.



(a) Value range of voltage magnitude target and estimation per node.

(b) Value range of voltage angle target and estimation per node.

Figure 7.6.: Evaluation results in box plots from a GSETR model.



(a) Voltage magnitude on three nodes

(b) Voltage angle on three nodes

Figure 7.7.: Evaluation results of a GSETR. In the plots, there are targets (blue) and estimated targets (orange). A target threshold (gray) is in the background.

8. Discussion

This chapter summarizes the results of the experiments. In section 8.1 the experiment-results and open questions are discussed. Further is in section 8.2 the validation metric criticized, used for this work. An additional research topic is suggested in section 8.3.

8.1. Models

The section 8.1.1 and section 8.1.2 covering the results of the introduced models GSETR and GSENR. Results from analyses are discussed in section 8.1.3.

8.1.1. GSETR

The accuracy of ACC 99 % with the *UnmeasuredPQZero* NTP outperformed the *GNNSE3 model* with ACC = 51 %. Problems stated in [11] are over come. Moreover, the learning respects the graph structure. This is demonstrated with the *RandomUnmeasured* NTP. Only for half of the nodes had information been accessible for GSETR. The model GSETR and the mean accuracy of $\mathrm{ACC}^{(\mathrm{avg})} = 91$ % are the first baseline. Additional attention heads and their aggregation could improve this model. Following research can adapt this. However, it is likely that this model does not perform well with different LVNs. Still, this is to show.

8.1.2. GSENR

A second architecture uses standardized features and targets. Due to the applied batch norm layers, the model is overfitting fast. Interested readers are linked to the *PairNorm* presented in [26] . This norm aims to overcome the over-smooting of hidden values by preserving a pairwise difference between hidden values. The author didn't apply countermeasures against this. Further researchers are pointed to this issue. All datasets used in this work are of a synthetic nature. In the field, measurements are mixed with some noise. It is likely that from a noise filter for the feature data, the GSENR model would benefit. This also allows, the models to be tested in a more realistic setting. Further allows the use of GATv2 dropouts, forcing parameter weights to generalize. Edge normalization is a yet-to-be-tried option and should also be considered.

Temporal models

The author of related work, used LSTM and GRU architecture. It can be interesting to reproduce that work and compare the GSENR with it. Also, the adaptation of these techniques may improve the model.

8.1.3. Evaluation data

More realistic scenarios can be created with more advanced NTP. Further presents this work only benchmark results for one LVN. It is possible that the presented models GSENR and GSETR performing only on the *Allensbach* LVN well. These models have to be trained, validated, and evaluated on different topologies. Further, there is a test on LVN with no synthetic nature to perform.

8.2. Metrics

This section provides in section 8.2.1 a list of changes for the model validation. The limitations of the current metric are addressed in section 8.2.3. Improving the performance of the models based on ACC will not be sufficient. Limitations of the models are to detect for further improvement. Furthermore, additional learning targets can be defined.

8.2.1. Thresholds

The limitations of the defined accuracy was demonstrated. The control mechanism for the voltage angle is not given. However, with an adapted threshold $\boldsymbol{\varepsilon} = [10^{-3}, 10^{-3}]^\top$, the regression learning target can be controlled again. The author proposes a threshold $\varepsilon^{(\theta)}$ that is relative large against measured outliers, i. e.

$$\varepsilon^{(\psi)} = \alpha \cdot \max_{t \in \boldsymbol{T}, i \in \boldsymbol{\mathcal{N}}} \{|y_{t,i}^{(\psi)} - \bar{y}^{(\psi)}|\}.$$

For $0 < \alpha << 1$, not all mean values $\bar{y}^{(\psi)}$ are correct by default

$$\bar{y}^{(\psi)} \notin \boldsymbol{\mathcal{Y}}_{t,i}^{\psi}, \exists t \in \boldsymbol{T}, \exists i \in \boldsymbol{\mathcal{N}}.$$

8.2.2. Node metric

Assuming a graph $\boldsymbol{G}_t$ with $\mathcal{N} = 19$ nodes. Let only the node $i \in \boldsymbol{\mathcal{N}}$ be sometimes wrongly estimated. An accuracy for samples $t = 1 \dots T$ is at least

$$\text{ACC} \geq \frac{1}{T\mathcal{N}} T(\mathcal{N} - 1) = \frac{18}{19} \geq 94\ \%.$$

This issue, should be detected. Therefore, the accuracy for each node i could be defined like

$$\text{ACC}_i = \frac{|\{\hat{\boldsymbol{y}}_{t,i} | \hat{\boldsymbol{y}}_{t,i} \in \boldsymbol{\mathcal{Y}}_{t,i}, \forall t \in \boldsymbol{T}\}|}{T}.$$

8.2.3. Extreme case metric

Applying SE gains knowledge about unusual behavior in the LVN. The main reason why the SE is important, is to detect theses. However, the general accuracy treats all samples equally. The accuracy *acc* should be applied again to samples containing extreme cases. A filter for such samples $\boldsymbol{G}_t$ is to be defined.

8.3. Node-type policies

Using the *RandomUnmeasured* node type demonstrated that the distribution node is essential for the Graph Neural Networks for state estimation (GNNSE). This setup should be further developed. A strategic use of such a NTP can be used to determine the most essential nodes. Monetary-costly measurement tools can then be placed in a more strategic fashion.

A. Basics

The first section A.1 provides the derivation of the complex voltage. The second section A.2 shows an example of an instable message function.

A.1. Complex power in graphs

The derivation of the complex voltage is shown below.

Keeping the known *Ohm's law* for Direct current (DC) in the mind $V = R \cdot I$. Comparable to DC, for Alternating current (AC) it holds in the complex representation

$$\underline{V} = \underline{Z} \cdot \underline{I} \tag{A.1}$$

$$\iff \frac{1}{\underline{Z}} = \frac{\underline{I}}{V} \tag{A.2}$$

$$\iff \underline{Y} = \frac{\underline{I}}{\underline{V}} \tag{A.3}$$

with the complex impedance

$$\underline{Z} = R + jX$$

and complex admittance

$$\underline{Y} = G + jB.$$

The symbol j is the complex number. The *table A.1* provides an overview of the physically arguments. In AC also holds *Kirchhoff's circuit law* for parallel circuits. It

states, that the current at all nodes $n, \forall n \in \mathcal{V}$ is the total in- and out-going current at any time zero

$$\underline{I}_n = \sum_{m \in \boldsymbol{A}(n)} \underline{I}_m. \tag{A.4}$$

[10]

Again, similar to DC in AC exists a complex power as the product of complex voltage and complex conjugate current

$$\underline{S} = \underline{V} \cdot \underline{I}^*.$$

But in AC, this power has no physical meaning. The complex voltage $\underline{V}$ represented in polar form from the known *Euler's formula*

$$\begin{aligned} & \underline{V} = |\underline{V}|(\cos(\theta) + j\sin(\theta)) \\ \iff & \underline{V} = |\underline{V}|e^{j\theta} \end{aligned}$$

where $|\underline{V}|$ is the absolute value and θ the argument of the pointer $\underline{V}$. The complex power $\underline{S}$ can be used for the relationship to active power P and reactive power Q

$$\underline{S} = P + jQ \tag{A.5}$$

$$\iff |\underline{V}|e^{j\theta} \cdot \underline{I}^* = P + jQ. \tag{A.6}$$

With (A.3) and (A.4) to compute the complex power $\underline{S}_n$ for node $n \in \mathcal{V}$ is

$$\underline{S}_n = |\underline{V}_n|e^{j\theta_n} \cdot \underline{I}_n{}^* \tag{A.7}$$

$$\iff \underline{S}_n = |\underline{V}_n|e^{j\theta_n} \cdot (\sum_{m \in \boldsymbol{A}(i)} \underline{Y}_{n,m} \cdot \underline{V}_m)^* \tag{A.8}$$

$\underline{Y}_{n,m}$ is the edge admittance between the nodes n and m. [10]

A.2. Concatenation message aggregation

A short example of a MPF.

Table A.1.: Physical components and their units. Fitted date defines the unit for non-normalized data used for training and validation.

Symbol	**Note**	**Unit**	**Fitted Data**
V	Voltage	V (Volt)	
$\|V\|$	Voltage magnitude	V (Volt)	pu (per unit V_{nom})
θ	Voltage angle	$^\circ$ (Degrees)	rad (Radiant)
I	Current	A (Amper)	
R	Resistance	Ω (Ohm)	
P	Active power	W (Watt)	10^6 W
Q	Reactive power	Var (Volt-ampere reactive)	10^6 Var
S	Power	VA (Volt-ampere)	
Z	Impedance	Ω (Ohm)	
X	Series reactance	Ω (Ohm)	pu (per unit V_{nom})
R	Series resistance	Ω (Ohm)	pu (per unit V_{nom})
Y	Admittance	S $= \Omega^{-1}$ (Siemens)	
G	Shunt conductivity	S (Siemens)	pu (per unit V_{nom})
B	Shunt Susceptance	S (Siemens)	pu (per unit V_{nom})
τ	Tab ratio		pu (per unit V_{nom})
$\theta^{(T)}$	Phase shift		$^\circ$ (Degrees)

Let be MODEL a model with two layers. And let be $\boldsymbol{G}_t$ a graph, as *figure A.1* shows. Each layer is a MPF $f_{\text{msg}}^{(\text{l},\boldsymbol{r})}(\cdot)$. The hidden values $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_{t,i}$ from the node $i \in \boldsymbol{\mathcal{N}}$ are the input for the update function $\lambda(\cdot)$ The single messages are simple; the neighbor features

$$\boldsymbol{h}_j^{(\text{l}-\text{l})} = \phi(\boldsymbol{h}_j^{(\text{l}-1)}, \cdot).$$

A *concatenation* message aggregation function is defined as

$$\begin{aligned}\boldsymbol{m}_{\text{i}}^{(\text{l})} &= \underset{j \in \boldsymbol{A}(i)}{\oplus^{(\text{cc})}} (\boldsymbol{h}_j^{(\text{l})}) \\ &= \|_{j \in \boldsymbol{A}(i)}\, \boldsymbol{h}_j^{(\text{l})}.\end{aligned}$$

At least is the update function

$$\begin{aligned}\boldsymbol{h}_i^{(\text{l})} &= \lambda(\boldsymbol{h}_i^{(\text{l}-1)}, \boldsymbol{m}_{\text{i}}^{(\text{l})}) \\ &= [\boldsymbol{h}_i^{(\text{l}-1)} \parallel \boldsymbol{m}_{\text{i}}^{(\text{l})}]^\top.\end{aligned}$$

Both layers of MODEL are applied to sample $\boldsymbol{G}_t$ to generate two stages of hidden values, $\boldsymbol{H}^{(1)}$ and $\boldsymbol{H}^{(2)}$. (1) $f_{\text{msg}}^{(\text{l},\boldsymbol{r})}$ is applied on all nodes $i, \forall i \in \boldsymbol{\mathcal{N}}$. (2) Then message passing is applied again for all nodes, with the input $\boldsymbol{h}_i^{(1)}$ to gain $\boldsymbol{h}_i^{(2)}$. The instability that comes due to $\underset{j \in \boldsymbol{A}(i)}{\oplus^{(\text{cc})}}(\cdot)$ is illustrated in *figure A.1.* After two iterations, the hidden values $\boldsymbol{h}_1^{(2)}$ for node *bus01* contains the signal from *bus02* (orange). The ideal property (3.10) is fulfilled for $\underset{j \in \boldsymbol{A}(i)}{\oplus^{(\text{cc})}}$. However, the feature number for *bus03* is increasing fast. The message aggregations are instable against the node degree. In $\boldsymbol{h}_3^{(2)}$ are the features $\boldsymbol{h}_1^{(0)} = \boldsymbol{x}_{t,1}$ (green) and $\boldsymbol{h}_2^{(0)} = \boldsymbol{x}_{t,2}$ (orange) two times represented and will be in $\boldsymbol{h}_3^{(3)}$ six times, an over-smoothing of the signals appears.

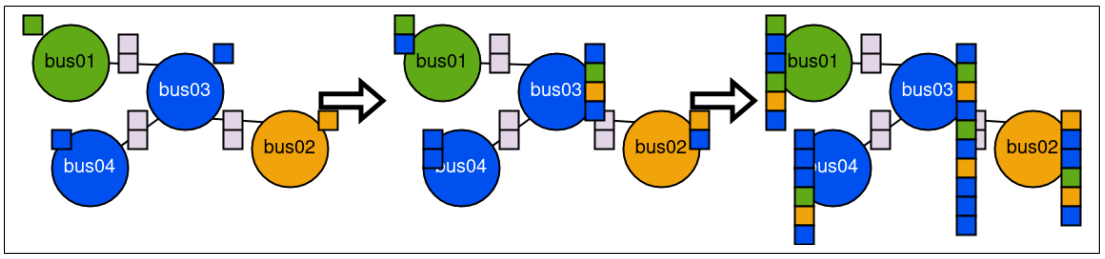


Figure A.1.: An example of a concatenation MPF with two layers (arrows). The input graph (most left) is forwarded through the first (middle) and the second layer (left).

B. Datasets

All LVN used for this work are of synthetic nature. This chapter provides an overview of these data. Moreover, in the section B.1 is the creation of the used data sketched.

The table *table B.1* contains all used datasets provided by the ISE.

Table B.1.: Overview of different datasets, containing LVN of Allensbach.

ID	**Title**	**Samples**	**Buses**	**Lines**	**Trafo.**	**Gener.**	**Load**	**Storage**
ds1	Allensbach 2020 first week	672	19	17	1	15	34	9
ds2	Allensbach 2019	35 040	19	17	1	15	34	9
ds3	Allensbach 2020	35 040	19	17	1	15	34	9
ds4	Allensbach 2021	35 040	19	17	1	15	34	9
ds5	Allensbach 2022	35 040	19	17	1	15	34	9
ds6	Allensbach 2022 typhoon	35 040	19	17	1	15	34	9
ds7	Allensbach 2022 typhoon 1	35 040	19	17	1	15	34	9

B.1. Synthetic grids

The following sketches the workflow for creating the synthetic data.

Synthetic grids are created by the following steps:

(1) A geographical area (e. g. a German city) is selected.

(2) Geo-reference data are mapped to this area. This gains the position of buildings and wires in the electrical grid. These node and edge data are converted into a graph topology.

(3) Interactors are added and become attached to the nodes, depending on a chosen scenario. A scenario mainly tells about the composition of interactor types. Interactor types may be:

Storage units provides or consumes power at a node, e. g. a E-vehicle.

Generator provides power to the grid, e. g. a photovoltaic system.

Load consumes power at a node, e. g. a heat pump.

(4) Interactor profiles are added by the SynPRO tool. This tool is developed byt the ISE. [3] A profile imitates interactions with the electrical grid over time. This may be

- a electrical car loading each work day at night,
- a heat pump is running only on cold days or
- a photovoltaic system injects power into the electrical grid depending on the brightness of the days.

The time interval $t = 1 \dots T$ of the LVN is over one year, with a sample every 15 minutes.

C. Code approach

This chapter provides in section C.1 additional information about the data pipeline. The second section C.2 provides motivations regarding the project code.

C.1. Passing pipeline

How the a LVN is processed is explained in more detail below.

The *algorithm* 4 shows the data parsing pipeline. The input LVNs is parsed into the output dataset $\boldsymbol{G}$. An inner calling routing is presented in *algorithm* 3. The reimplemented pipeline was chopped and extended into *DataCacher*, *SEGridFilter*, *HeteroGridParser* and *SEDataFilter*. The input electrical grid time series is implemented in *pyPSA* and the output is a list with *HeteroData* items.

C.2. Further code adaptions

Additional relevant code adaptions are mentioned.

One bottleneck in [20] is the not-testable grid parser, with insufficient code and behavior documentation. Due to its hard-coded behavior, this parser is not extendable to suit different training settings. Therefore, nearly the entire parsing pipeline has been redesigned.

C.2.1. Missing unit tests

The code provided by [20] is historical, growing, not well maintained, and sparsely documented. Not a single unit test was defined. A wide set of hard-coded settings and a lack of missing configuration options inhibit the reproducibility of experiments with different settings. Different data preparation, metrics, and use case implementation are problems for reliability as well.

However, some ground truth is defined ahead of previously coded behavior. Configurations are redefined and placed for all settings. These are associated with the ground truth. This makes sure that, after further research, the former models still work. During the process of result reproducing and code rewriting procedure, around 50 deterministic tests were created. If required, a test provides the option to set a seed. Each used function is part of at least one test. Redefining atomic tests for all basic functions exceeds the resources available for this work.

C.2.2. Code

The author chooses the Google style [19] for docstrings. This style is applied to all functions in the project. Further, is all code PEP8 [14] compatible. The former project is also separated into two projects. One provides utils for all GNN projects at the ISE and one is the actual GNNSE project. Therefore, many classes are divided into base classes and SE related classes.

Algorithm 3: Reimplemented parser applyes SE to gain targets, defines features and merges all data into a dataset.

Function `grid_parsing()`:

  Input: $\boldsymbol{G}^{'(k)} = [\boldsymbol{G}_1^{'(k)}, \ldots, \boldsymbol{G}_{T^{(k)}}^{'(k)}]^\top$: LVN time series

  Output: $\boldsymbol{G}^{(k)}$: Graph time series

  `// Gain time invariant data, some from fist timestamp`

  $\boldsymbol{\mathcal{N}} = \text{HeteroGridParser.get\_nodes}(\boldsymbol{G}_0^{'})$

  $\boldsymbol{\mathcal{E}} = \text{HeteroGridParser.get\_edges}(\boldsymbol{G}_0^{'})$

  $\boldsymbol{M} = \text{HeteroGridParser.get\_metadata}(\boldsymbol{G}^{'(k)})$

  `// Resolve node type and edge types, also gain edge feature`

  for $i \in \boldsymbol{\mathcal{N}}$ **do**

    $r_i^{(\mathcal{N})} = \text{HeteroGridParser.resolve\_node\_type}(\boldsymbol{G}^{'(k)}, i)$

  $\boldsymbol{R}^{(\mathcal{N})} = [r_1^{(\mathcal{N})}, \ldots, r_{\mathcal{N}}^{(\mathcal{N})}]^\top$

  for $(i, j) \in \boldsymbol{\mathcal{E}}$ **do**

    $r_{(i,j)}^{(\mathcal{E})} = \text{HeteroGridParser.resolve\_link\_type}(\boldsymbol{G}_0^{'}, (i, j))$

  $\boldsymbol{R}^{(\mathcal{E})} = [r_{(1,1)}^{(\mathcal{E})}, \ldots, r_{(\mathcal{E},\mathcal{E})}^{(\mathcal{E})}]^\top$

  `// Iterate over time stamps in graph time series`

  for $\boldsymbol{G}_t^{'} \in \boldsymbol{G}^{'(k)}$ **do**

    `// Apply state estimation for ground truth data`

    $\boldsymbol{Y}_t^{'} = \text{HeteroGridParser.newton\_raphson}(\boldsymbol{G}_t^{'})$

    `// Collect target and feature values for each node`

    `// Respect the node type`

    for $i \in \boldsymbol{\mathcal{N}}$ **do**

      $\boldsymbol{y}_{t,i} = \text{HeteroGridParser.get\_complex\_power}(\boldsymbol{G}_t^{'}, i, r_i^{(\mathcal{N})})$

      $\boldsymbol{x}_{t,i} = \text{HeteroGridParser.get\_interactor\_power}(\boldsymbol{G}_t^{'}, i, r_i^{(\mathcal{N})})$

    $\boldsymbol{X}_t = [\boldsymbol{x}_{t,1}, \ldots, \boldsymbol{x}_{t,\mathcal{N}}]^\top$

    $\boldsymbol{Y}_t = [\boldsymbol{y}_{t,1}, \ldots, \boldsymbol{y}_{t,\mathcal{N}}]^\top$

    `// Build graph`

    $\boldsymbol{G}_t^{(k)} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{E}}, \boldsymbol{R}, \boldsymbol{X}_t, \boldsymbol{Y}_t)$

  $\boldsymbol{G}^{(k)} = [\boldsymbol{G}_1^{(k)}, \ldots \boldsymbol{G}_{T^{(k)}}^{(k)}]^\top$

  return $\boldsymbol{G}^{(k)}$

Algorithm 4: Parsering pipeline from LVN to dataset $\boldsymbol{G}$.

Function `data_parsing()`:

Input: $[\boldsymbol{G}'^{(1)}, \ldots, \boldsymbol{G}'^{(K)}]^\top$: Electrical grid time series

Output: $\boldsymbol{G}$: Graph time series

`// Resolve cache name`

cache_name = DataCacher.resolve($\boldsymbol{G}$, SEGridFilter, SEDataFilter, HeteroDataParser)

`// Try loading and returning cache name`

if cache_name *exists* **then**

  return cache_name $\longrightarrow \boldsymbol{G}$

`// Filter on graph time series list, e.g. select time stamps.`

for $\boldsymbol{G}'^{(k)} \in [\boldsymbol{G}'^{(1)}, \ldots, \boldsymbol{G}'^{(K)}]^\top$ **do**

  $\boldsymbol{G}'^{(k)} = \text{SEGridFilter.apply}(\boldsymbol{G}'^{(k)})$

`// Parse and merge list of time series`

for $\boldsymbol{G}'^{(k)} \in [\boldsymbol{G}'^{(1)}, \ldots, \boldsymbol{G}'^{(K)}]^\top$ **do**

  $\boldsymbol{G}^{(k)} = \text{HeteroDataParser.parse}(\boldsymbol{G}'^{(k)})$

$\boldsymbol{G} = [\boldsymbol{G}_1^{(1)}, \ldots \boldsymbol{G}_{T^{(1)}}^{(1)}, \boldsymbol{G}_1^{(2)}, \ldots \boldsymbol{G}_{T^{(K)}}^{(K)}]^\top$

`// Filter on graph dataset e.g. normalize data.`

$\boldsymbol{G} = \text{SEDataFilter.apply}(\boldsymbol{G})$

`// Save under cache name`

$\boldsymbol{G} \longrightarrow$ cache_name

return $\boldsymbol{G}$

D. Models

This chapter provides additional information to reproduce the results presented in this work. In section D.1 the setup of the GSETR model is explained. The setup of section D.2 is provides information for the GSENR model.

D.1. GSETR Architecture

Due to the general parser *HeteroGridParser* is the set $\boldsymbol{R}$ maximal. Therefore, the finale model architecture is large. As sketched in section 3.3.2 exist a transformation from a homogeneous GNN model MODEL into a heterogeneous GNN model. This transformation is also used. Due to this setup, the model is defined as homogeneous. The model GSETR uses four layers:

(1) GATv2 with $\boldsymbol{H}^{(1)} \in \mathbb{R}^{16\times *}$, EdgeDim $= *$, no bias

(2) GATv2 with $\boldsymbol{H}^{(2)} \in \mathbb{R}^{16\times 16}$, EdgeDim $= *$, no bias

(3) GATv2 with $\boldsymbol{H}^{(3)} \in \mathbb{R}^{16\times 16}$, EdgeDim $= *$, Heads $= 2$, no bias

(4) general layer of a linear layer with $\boldsymbol{H}^{(4)} \in \mathbb{R}^{16\times 2}$

Only after the first layer is not a batch norm applied. Each layer output is passed through a ReLU activation function.

D.2. GSENR Architecture

The model GSENR uses four layers:

(1) GATv2 with $\boldsymbol{H}^{(1)} \in \mathbb{R}^{16\times *}$, EdgeDim $= *$, Heads $= 4$, no bias

(2) GATv2 with $\boldsymbol{H}^{(2)} \in \mathbb{R}^{16\times 16}$, EdgeDim $= *$, Heads $= 4$, no bias

(3) GATv2 with $\boldsymbol{H}^{(3)} \in \mathbb{R}^{16\times 16}$, EdgeDim $= *$, Heads $= 4$, no bias

(4) general layer of a linear layer with $\boldsymbol{H}^{(4)} \in \mathbb{R}^{16\times 2}$

After each layer is a batch norm applied. Each layer output is passed through a ReLU activation function. For this model, the mean square error loss function is used.

D.3. Benchmark models

Below are the differences between the models listed. They were used in the benchmark experiment. Further, are all the statistics are presented.

The GSETR models are identified by their initialization seed. For the three different node-type policies, their seed is also used for identification. All models are trained in parallel on the same host for 200 epochs. The host provides 32 CPU and 8 GB graphic card storage. Each model uses at most 20 GB RAM. However, someone ignored the reservation of the node and ran jobs too. In *table D.1* are relevant training dates for each model.

Table D.1.: Benchmark stats for the models GSETR.

Model	**NTP**	**Val ACC**	**Eval ACC**	**Runtime** (h)
1001	1001	0.960	0.957	9.8
1002	1001	0.956	0.954	9.8
1003	1001	0.955	0.965	9.8
1004	1002	0.867	0.882	9.8
1005	1002	0.870	0.881	9.8
1006	1002	0.873	0.895	9.3
1007	1003	0.903	0.903	9.3
1008	1003	0.903	0.911	9.3
1009	1003	0.901	0.894	9.3

Acronyms

CME model	Constant Mean Estimation model
GNNSE3 model	GNNSE 2023 model
md-array	Multidimensional array
nent-link	Node-edge-node-type link
AC	Alternating current
AI	Artificial intelligence
DC	Direct current
ETP	Edge-type policy
GAT	Graph Attention
GATv2	Graph Attention 2 layer
GCN	Graph Convolution Network
GNN	Graph Neural Network
GNNSE	Graph Neural Networks for state estimation
GRU	Gated Recurrent Unit
ISE	Fraunhofer Institute for Solar Energy Systems

LSTM	Long-Short-Tearm memory
LVN	Low-voltage network
ML	Machine Learning
MLP	Multilayer perceptron
MPF	Message-passing framework
NT map	Node type map
NTP	Node-type policy
ReLU	Rectified Linear Unit
RRMSE	Relative Root Mean Squared Error
SE	Electrical grid power system state estimation
SE-task	State estimation task

Bibliography

[1] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. New York: Springer, Jan. 2006. ISBN: 0-387-31073-8.

[2] Shaked Brody, Uri Alon, and Eran Yahav. "How Attentive are Graph Attention Networks?" In: (2021). Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.2105.14491. URL: https://arxiv.org/abs/2105.14491 (visited on 02/19/2024).

[3] David Fischer, Andreas Härtl, and Bernhard Wille-Haussmann. "Model for electric load profiles with high time resolution for German households". en. In: *Energy and Buildings* 92 (Apr. 2015), pp. 170–179. ISSN: 03787788. DOI: 10.1016/j.enbuild.2015.01.058. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378778815000845 (visited on 02/26/2024).

[4] William Hamilton. *Graph Representation Learning*. 2020th ed. McGill University. URL: https://www.cs.mcgill.ca/~wlh/grl_book/.

[5] Aryan Jadon, Avinash Patil, and Shruti Jadon. *A Comprehensive Survey of Regression Based Loss Functions for Time Series Forecasting*. arXiv:2211.02989 [cs]. Nov. 2022. URL: http://arxiv.org/abs/2211.02989 (visited on 02/23/2024).

[6] Ismail Kasikci. *Planung von Elektroanlagen: Theorie, Vorschriften, Praxis*. ger. 3., vollständig überarbeitete und erweiterte Auflage. Berlin Heidelberg: Springer Vieweg, 2018. ISBN: 978-3-662-56427-1 978-3-662-56426-4.

[7] Ognjen Kundacina, Mirsad Cosovic, and Dejan Vukobratovic. "State Estimation in Electric Power Systems Leveraging Graph Neural Networks". In: *2022 17th International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*. Manchester, United Kingdom: IEEE, June 2022, pp. 1–6. ISBN: 978-1-66541-211-7. DOI: 10.1109/PMAPS53380.2022.9810559. URL: https://ieeexplore.ieee.org/document/9810559/ (visited on 07/04/2023).

[8] Abigail Langbridge et al. "Causal Temporal Graph Convolutional Neural Networks (CTGCN)". In: (2023). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2303.09634. URL: https://arxiv.org/abs/2303.09634 (visited on 06/30/2023).

[9] Antonio Longa et al. *Graph Neural Networks for temporal graphs: State of the art, open challenges, and opportunities*. arXiv:2302.01018 [cs]. July 2023. URL: http://arxiv.org/abs/2302.01018 (visited on 02/25/2024).

[10] Ja Rgen Winter Marlene Marinescu. *Grundlagenwissen elektrotechnik*. ger. 3., bearbeitete und. Place of publication not identified: Vieweg+Teubner Verlag S, 2011. ISBN: 978-3-8348-9957-6.

[11] Erik Mauß. "Distribution System State Estimation using Graph Neural Networks". Jan. 2022.

[12] *Model Transformations*. URL: https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#model-transformations.

[13] *Monitoringbericht gemäß § 63 Abs. 3 i. V. m. § 35 EnWG und § 48 Abs. 3 i. V. m. § 53 Abs. 3 GWB*. Tech. rep. Bundesnetzagentur, Bundeskartellamt.

[14] *PEP 8: The Style Guide for Python Code*. URL: https://pep8.org/ (visited on 03/16/2024).

[15] *Power Flow — PyPSA: Python for Power System Analysis*. URL: https://pypsa.readthedocs.io/en/latest/power_flow.html (visited on 03/12/2024).

[16] *Power-flow study*. en. Page Version ID: 1161387199. June 2023. URL: https://en.wikipedia.org/w/index.php?title=Power-flow_study&oldid=1161387199 (visited on 07/19/2023).

[17] Martin Ringsquandl et al. "Power to the Relational Inductive Bias: Graph Neural Networks in Electrical Power Grids". In: (2021). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2109.03604. URL: https://arxiv.org/abs/2109.03604 (visited on 02/25/2024).

[18] Fathi M. Salem. *Recurrent neural networks: from simple to gated architectures*. eng. OCLC: 1291317305. Cham: Springer, 2022. ISBN: 978-3-030-89929-5.

[19] *styleguide*. en-US. URL: https://google.github.io/styleguide/pyguide.html (visited on 03/16/2024).

[20] Simon Sütterlin. *Results of GNN state estimation until 31.01.2023*. Jan. 2023.

[21] *torch_geometric.nn.conv.GATv2Conv — pytorch_geometric documentation*. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATv2Conv.html (visited on 02/19/2024).

[22] *torch_geometric.nn.conv.GCNConv — pytorch_geometric documentation*. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GCNConv.html#torch_geometric.nn.conv.GCNConv (visited on 02/17/2024).

[23] Petar Veličković et al. "Graph Attention Networks". In: (2017). Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.1710.10903. URL: https://arxiv.org/abs/1710.10903 (visited on 02/19/2024).

[24] Sebastian Walter. "Transformers and Graph Neural Networks for Spell Checking". University of Freiburg, 2022. URL: http://ad-publications.informatik.uni-freiburg.de/theses/Master_Sebastian_Walter_2022.pdf.

[25] Gang Wang et al. “Distribution system state estimation: an overview of recent developments”. en. In: *Frontiers of Information Technology & Electronic Engineering* 20.1 (Jan. 2019), pp. 4–17. ISSN: 2095-9184, 2095-9230. DOI: 10.1631/FITEE.1800590. URL: http://link.springer.com/10.1631/FITEE.1800590 (visited on 07/04/2023).

[26] Lingfei Wu et al., eds. *Graph Neural Networks: Foundations, Frontiers, and Applications*. en. Singapore: Springer Nature Singapore, 2022. ISBN: 9789811660535 9789811660542. DOI: 10.1007/978-981-16-6054-2. URL: https://link.springer.com/10.1007/978-981-16-6054-2 (visited on 02/25/2024).

[27] Ling Zhao et al. “T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.9 (Sept. 2020), pp. 3848–3858. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2019.2935152. URL: https://ieeexplore.ieee.org/document/8809901/ (visited on 06/30/2023).