Master Thesis

Deep Statistical Solver for Distribution System State Estimation

ET4300: Master Thesis EE/CE

Benjamin Habib - 5357241

Delft University of Technology

TU Delft

Master Thesis

Deep Statistical Solver for Distribution System State Estimation

by

Benjamin Habib - 5357241

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday the 26th of August, 2022 at 12:30.

Student number:	5357241	
Project duration:	December 15, 2021 – August 26, 2022	
Thesis committee:	Prof. dr. ir. M. Popov,	TU Delft, thesis advisor
	Dr. J. L. Cremer,	TU Delft, supervisor
	Dr. E. Isufi,	TU Delft, co-supervisor

This thesis is confidential and cannot be made public until August 31, 2023.

Cover:	Power pylons at sunset, photo by Matthew Henry on Unsplash
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Each of us has a talent. Each of us can do our part.

We just have to figure out how.

Cyril Dion, 2017

Acknowledgments

I would like first to thank my supervisor Jochen Cremer for his invaluable guidance and availability throughout this thesis. I would also like to thank my thesis committee, Elvin Isufi and Marjan Popov, for their expert advice and help during my work. Also, I am grateful to my company supervisors, Arjen Jongepier and Ward van Breda, for their support throughout the year and the opportunity to collaborate with Stedin on this project.

Special thanks to the Data Science team for assisting me during my time at Stedin and providing me with valuable tools for my work. Thanks should also go to the Delft AI Energy lab and the IEPG department for giving me practical tips and valuable feedback.

Finally, I want to thank my girlfriend, family, and friends for encouraging me and keeping me motivated during my Master's journey.

Abstract

Whereas in the past, Distribution Systems played a passive role in connecting customers to electricity, Distribution System Operators (DSOs) will have to take in the future a more active role in monitoring and regulating the network to deal with the new behaviors and dynamics of the system brought by the energy transition. State Estimation, a task traditionally reserved for Transmission System Operators (TSOs), is, therefore, a needed tool for DSOs to properly monitor the distribution grid in the future. However, the implementation of Distribution System State Estimation (DSSE) faces several challenges. The distribution system lacks observability to get satisfying estimation accuracy, the denser network increases the complexity of the estimation process, and the lack of labeled data makes training Machine Learning alternatives difficult. To tackle these issues, we propose the Deep Statistical Solver for Distribution System State Estimation (DSS^2), a Deep Learning model based on the Graph Neural Network (GNN) architecture and the Physic-Informed Machine Learning (PIML) framework.

The DSS^2 model is based on the Deep Statistical Solver (DSS) framework, which seamlessly models power systems into GNN using Hyper-Heterogeneous Multi Graphs (H2MG), and emphasizes semi-supervised learning by *learning to optimize*, using optimization problem as a loss function. This thesis extends the DSS framework to the DSSE problem, using the traditional State Estimation algorithm as an optimization problem to learn, and incorporating the power flow equations in the loss function. This model is trained through a semi-supervised approach to learn the physics of the problem and alleviate the need for labels and uses the Deep Learning tools to improve accuracy and robustness in the DSSE task.

Case studies on 14-bus, 70-bus, and 179-bus networks show promising results, with the model outperforming the traditional WLS algorithm while showing better robustness. The model also competed in performance against supervised models and showed to be more suitable for the semi-supervised learning approach than simpler GNN architectures.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
SE	State Estimation
PF	Power Flow
DSSE	Distribution System State Estimation
MV	Medium Voltage
LV	Low Voltage
DSO	Distribution System Operator
TSO	Transmission System Operator
DER	Distributed Energy Resources
WLS	Weighted Least Squares
KF	Kalman Filter
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
NN	Neural Network
ANN	Artificial Neural Network
DNN	Deep Neural Network
FFNN	Feed-Forward Neural Network
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
GNN	Graph Neural Network
DSS	Deep Statistical Solver

Abbreviation	Definition
DSS^2	Deep Statistical Solver for Distribution System State Estimation
H2MG	Hyper Heterogeneous Multi Graph
H2MGNN	Hyper Heterogeneous Multi Graph Neural Network
SSP	Statistical Solver Problem

1 Introduction

1.1. Background and motivation

State Estimation (SE), since its original implementation for Power Systems (PS) by Schweppe et al. in 1970 [1], has been widely developed and used by Transmission System Operators (TSO). TSOs have to deal with the dynamic behavior of generation, transmission, distribution, and energy consumption while maintaining the high reliability of the network through security control and economic dispatch [2]. Therefore, operators developed SE in the transmission grid to get an accurate estimation of the current state of the network with the sole use of the noisy measurements given from the field. The technique aims at minimizing the error between theoretical observations of the state variables and the given measurements using a Weighted Least Squares (WLS) error calculation, using the redundancy in measurements to optimize the estimation process. The output of the methodology is the estimated state variables: the voltage magnitude and angle on each grid bus. This vital monitoring tool is used to compensate for the imprecise measurements of the grid, as it aims for a minimal error on the estimation to proceed with crucial operations tasks such as contingency analysis and asset management.

Despite its high usage by TSOs and benefits, SE has been disregarded and poorly developed in distribution systems. Indeed, too few benefits were seen for the distribution network to compensate for the technical challenges of implementation. With the past's passive nature of the distribution system, there was only little need for Distribution System State Estimation (DSSE). Unidirectional power flows and the lack of active generation in the grid made the distribution system relatively stable and reliable with little uncertainty. Nowadays, the distribution system is taking a more active role. The integration of renewable energy resources (DER) and power electronics in the medium voltage (MV) and low voltage (LV) levels creates new dynamics in the grid. Also, the shift towards smart grids with demand response and management develops bi-directional flows in the network [3]. The integration of DER also leads to a decrease in inertia and an increase in voltage volatility which weakens the distribution grid, especially at the MV level. A weak MV grid can dramatically impact the network's reliability, especially when no awareness is provided on the related issues. Figure 1.1 shows an example of such issue. Due to the

dynamic load increase, voltage level recovery will be delayed after a fault. This extended recovery decreases the grid's stability, enhancing outage risk. Next, the increased penetration of DER will develop limitations in the distribution grid such as stead-state over-voltages, thermal limitations, short-circuits, protection miscoordination, power quality issues, and increased islanding risks. Closer to the interface with the transmission grid, other issues can arise, such as congestion due to reverse flows, voltage control problems, and stability issues. The effects of DER and dynamic loads are currently not observable nor controllable and happen in a "positive-feedback" loop which increases the risks of high-impact events even further [4].

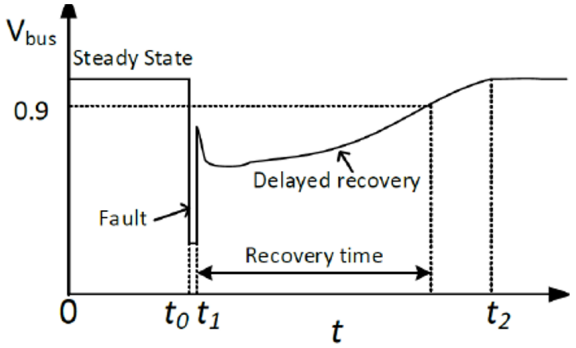

Figure 1.1: Delayed recovery behavior of bus voltage due to fault [4]

This new nature of the distribution system brings a need for more monitoring and control, which is possible by developing SE in the distribution system. Nevertheless, many challenges remain to be tackled for the successful development of DSSE. The distribution grid carries multiple unbalanced three-phase branches with high $\frac{r}{x}$ ratios which are against the assumptions taken in the traditional SE algorithms. Moreover, the lack of real-time measurements in the distribution grid makes DSSE hard to perform as it requires full observability of the grid (and preferably, redundancy in measurements) to estimate the state variables [5]. Distribution System Operators use pseudomeasurements, forecasted values based on historical data, to compensate for the lack of measurement. These pseudomeasurements are often inaccurate and can impact the accuracy of the SE algorithm. On top of that, the traditional method for SE using WLS uses an iterative process that is time-consuming and infeasible in real-time for a distribution system that counts more than hundreds of nodes. Therefore, new techniques are needed to perform accurate, robust, and fast SE in a poorly observable and high-dimensional environment such as the distribution system.

1.2. Literature review on Distribution System State Estimation

Multiple approaches have been proposed in the literature to tackle the challenges of DSSE. [6] proposes a survey reviewing conventional, data-driven, and probabilistic techniques. It first pinpoints the main challenges of DSSE: Limited observability, high $\frac{r}{x}$ value, imbalance, communication issues, network configuration, integration of DER, and cyber-security. Then, it describes the conventional WLS approach and shows some of its limitations, such as noise sensitivity, divergence issues, and the assumption of Gaussian distribution, which decrease the accuracy of the technique. Moreover, the WLS algorithm uses an iterative process that asks for a high computational cost, especially for a dense distribution grid. The survey presents other mathematical formulations to perform SE while increasing

robustness (Least Median of Squares, Least Trimmed Squares, Least Absolute Value, and Generalized Maximum Likelihood). However, these alternative structures, similarly to the WLS approach, require high computational costs or are sensitive to parameter selection.

1.2.1. Forecasting-Aided State Estimation and Kalman Filters

In order to speed up the SE task, Dynamic SE (DSE) has been investigated in the literature. The need to develop DSE is not only to perform SE in real-time but also to process Phasor Measurement Units (PMUs) data fast enough, which can improve the reliability of the SE. The primary approach for DSE is usually the Forecasting-Aided SE (FASE), which "consider the time evolution of state over time and can track system changes during its normal operation"[3]. FASE is based on Kalman Filters (KF) and uses iterations of state prediction and state filtering to update the belief on the state variables. Kalman Filters lie in the Bayesian Filter framework, which consists in updating our belief on variables' distribution. Different types of filters have been investigated to perform FASE in the literature, such as standard KF, Extended KF, Unscented KF, Cubature KF, Particle Filters, and Ensemble KF [3], [7], [8], [9]. Basic KF, although very effective in linear systems, cannot be used in DSSE due to the high nonlinearity of the system. To apply KF to nonlinear systems, Extended KF (EKF) uses Taylor series to linearize systems. This approach is very efficient in applying Bayesian Filters to nonlinear systems, but it is not suitable for large or highly nonlinear systems due to the linearization approximation. Another method to apply KF to nonlinear systems is deterministic sampling. Unscented KF (UKF) and Cubature KF (CKF) are examples of deterministic sampling methods which use carefully designed "sigma points"/"cubature points" to model distributions. These points are mathematically chosen to reflect the overall probability distribution and preserve that distribution through nonlinear transformations. These approaches can model nonlinearities accurately but are too computationally expensive for large systems. Also, these KF-based methods rely on the assumption of gaussian distribution, which is inaccurate for some variables in the power grid.

While UKF and CKF use deterministic sampling, Particle Filters (PF) use probabilistic approaches such as Monte Carlo sampling to approximate the distribution. This approach allows bypassing the common assumption of gaussian distribution while working on nonlinear systems. Although, these filters need even higher computation effort and are unsuitable for real-time DSSE. Finally, Ensemble KF (EnKF) combines multiple Gaussian distributions, namely an ensemble, to fit the output distribution of the nonlinear system. As the PF, it provides an accurate estimation but requests a high computational cost.

Therefore, these model-based solutions using Kalman Filters are limited by approximating assumptions (linearization, Gaussian distribution) or/and demand high computational cost. We previously stated how high computational costs are unwanted to reach real-time DSSE, and the assumptions taken can also lead to unwanted algorithm behaviors when other types of distributions are present in the system or when high nonlinearities arise.

1.2.2. Mean Squared Error-based and data-driven approaches

In[10], it is mentioned how inaccurate the Gaussian assumption can be for power systems. Instead, it proposes a Bayesian alternative to WLS using Mean Squared Estimator (MSE), which does not depend

on Gaussian uncertainty assumption and performs better in non-Gaussian uncertainty. This Bayesian approach aims at estimating states as conditional averaging operations:

$$\hat{x} = E\{x \mid z\} = \int \alpha f_{\alpha|z}(\boldsymbol{\alpha} \mid z) d\boldsymbol{\alpha} \tag{1.1}$$

The MSE-based method can be very effective as it does not need to assume Gaussian distribution or linearity and uses the Bayes rule to update the state variables' distribution following the measurements flow without the need for complete observability at each snapshot. Although, this approach depends on prior knowledge of the different distributions and the statistical properties of the system. The availability of accurate statistical knowledge can be difficult; hence MSE-based methods are complex to implement.

In an attempt to use an MSE-based approach for DSSE with non-Gaussian uncertainty, [10] develops a statistical analysis of the different variables (states, measurements, pseudomeasurements) of the system to provide a Bayes' rule-based DSSE. Even though it provides an efficient and accurate solution, this approach demands a thorough investigation to find the prior statistical knowledge to describe a specific system, meaning knowing the underlying probability distribution of each type of variable and measurement in the system. This investigation can be tedious, and the statistical behavior of the system might evolve during long-term operation.

If a sufficient amount of labeled data is provided to learn the system's statistics, it is possible to use an entirely data-driven technique inspired by the Bayesian framework to update our estimate of the state variables. [11] proposes a data-driven approach leveraging grid' data by feeding a Deep Neural Network (DNN) that will learn the underlying probability distribution itself. As the computational burden is sent to an offline training of the DNN, the technique is fast, efficient, and accurate. Moreover, it does not need prior knowledge of the system, linear/Gaussian assumptions, or full observability. However, such a data-driven technique needs a high amount of correctly labeled data that can fully describe our system to train the DNN. This requirement is a significant drawback, as in practice database contains raw data of power injection and absorption, and measurements are very sparse in the network. It is therefore impossible to train a DNN with a high amount of labeled data. In the same paper, power flow simulations are used to gather data with different scenarios in a Monte Carlo fashion. This can train a DNN efficiently; however, one can state as a drawback that the DNN will fit a specific power flow solver, which does not represent a real distribution system accurately. Also, the author assumes the existence of smart meter data and derives the probability distribution of the power injections in the grid from it. Although it is common practice to use generic historical data to get pseudomeasurements, relying solely on smart meter data is impractical. In reality, most nodes are unobserved, and only poor accuracy is reached with these pseudomeasurements.

The data-driven approach of training a DNN model has been investigated multiple times in the literature. [12] extends the approach with physical awareness by pruning links in the DNN if no connection exists between related nodes in the power system. The authors also use PMUs with high accuracy and synchronized values to partition the overall power system (and thus, the related DNN) into sub-areas around the PMUs. These added contributions are promising as they improve the training process and attempt to tackle the challenges of computational costs and communication issues. Similarly, [13] uses PMUs as references to partition the grid and contributes to the DNN approach by proposing a model with multiple parallel DNNs that are trained separately for each of the 3 phases of the power system, and

each partition of the grid. The objective of parallel DNNs is to reach faster training, improved accuracy, fewer communication issues, and fewer phase imbalance issues. These methods offer significant improvement and promising ideas for the data-driven DSSE, but all assume training with accurately labeled data from power flow solvers.

In [14], in an attempt to leverage the Bayesian SE approach where belief about the states is updated without full observability (optimizing the MSE), the authors apply Bayesian inference using Bayesian Neural Networks (BNN). First of all, as stated before, the data-driven MSE approach enables to work with limited observability and does not need any assumption or analysis about the statistical behavior of the system. Then, using BNN allows not only to compute the point estimate but also to provide uncertainty intervals about the output. The uncertainty intervals can provide more information and guarantees about outputs to help decision-making tasks. BNN are an exciting field of study to get more confidence in output in operation task such as DSSE; however, validation of the model and comparison must be investigated further. Notably, the added cost in computation and required data for using BNN to get uncertainty intervals should be mentioned. Finally, no prior knowledge about any physical or statistical properties is used to improve the architecture, and data is once again labeled using power flow simulators.

1.2.3. Model-based and data-driven hybrid models

The review proposed by [7] highlights the efficiency of Kalman Filters when the used models are entirely accurate and consistent with the practical system. The review also describes the use of Kalman Filters as simple and easy to perform for real-time estimation. However, it shows that KF techniques do not provide accurate results for complex systems, especially if non-gaussian noises characterize the system, which is in line with what we stated previously. The authors describe data-driven techniques as efficient tools to estimate complex systems, but only if high amounts of labeled data are available. These techniques are also prone to overfitting noise. Hence, it highlights that combining model-based estimation methods and data-driven techniques can alleviate the drawbacks and offer new possibilities to SE techniques.

To combine the efficiency and flexibility of model-based techniques with the speed and robustness of data-driven models, [15] proposes a hybrid approach called KalmanNet. The proposed idea is to enhance a basic, efficient KF with a DNN to correct inaccuracies in the model. Specifically, the data-driven block is trained to compensate for the model-based approximation's error by tweaking the Kalman Filter's gain. In this way, using the efficiency of KF for linear gaussian systems, a smaller amount of labeled data is sufficient for the model to show promising results with nonlinear, non-gaussian systems. This approach of combining model-based and data-driven techniques is promising; however, more validation is needed in the high-dimensional, low observability DSSE task. The model also needs supervised learning to be trained correctly, which is a setback. Moreover, KalmanNet has not been validated for use in large nonlinear systems such as the distribution network.

1.3. Research direction and contribution

The literature about DSSE is vast and active; multiple approaches are investigated to improve the estimation task. The review presented is far from exhaustive but grasps the overall state of research:

- Conventional SE, WLS, and derivatives are not fast and accurate enough to stay relevant with the energy transition and the new dynamics of the distribution network.
- Kalman Filters and derivatives are well-studied for the DSSE task but suffer from the trade-off between speed using simplification and accuracy without assumptions.
- Bayesian State Estimation (BSE) using Mean Squared Error (MSE) is interesting as it does not need full observability, is fast, and can handle heterogeneous input data. However, model-based BSE requires thorough statistical knowledge of the system to work accurately.
- Data-driven techniques are promising for DSSE. Most of the approaches use data to fit the statistical behavior of the system to perform BSE (MSE optimization) without statistical analysis. Even though some approaches try to improve this technique by introducing some inductive bias and physic-awareness, they all require extensive supervised learning using labeled data from simulations. They do not fit the real scarcity of labeled data.
- Combining model-based and data-driven is a promising direction for further research. However, more work is needed to validate models and compare them to the state-of-the-art on real systems. Also, semi-supervised and unsupervised learning should be investigated in these approaches to alleviate the need for power flow simulations' output and its impact.

With these conclusions, multiple promising approaches can be investigated further to improve research on DSSE. These include:

- Improve model-based data-driven hybrid approaches such as KalmanNet for larger, nonlinear systems and train them with semi-supervised learning methods.
- Investigate physic-driven DNN models for semi-supervised learning on DSSE.
- Add physic-based regularizers to data-driven techniques to improve the accuracy of BSE on real systems.

This thesis proposes investigating physic-driven DNN models for semi-supervised learning on DSSE using a Graph Neural Network-based architecture. The idea is to use the strength of data-driven technique (learn correlations from data, speed of computation, stability) while also including knowledge of physics to alleviate the need for labeled data and provide more robustness and flexibility.

The contribution of this thesis is as follows:

We propose the *Deep Statistical Solver for Distribution System State Estimation (DSS^2)*[1], a custom model based on the Deep Statistical Solver architecture [17] specialized for optimization tasks on Power Systems; to perform the estimation task in the Distribution System. This model is trained in a semi-supervised fashion with a custom loss function modeling the power flow equations to learn upon and aims at providing fast, stable and reliable estimations without the need for high system observability

[1] Code available on GitHub in the private TUDelft AI Energy lab repository [16]

or high amounts of labeled data. We validate the model using various case studies and compare it to the WLS algorithm baseline and other Deep Learning architectures, the Feed-Forward Neural Network and the Graph Convolutional Network, for further validation.

1.4. Research questions and thesis outline

Following this research direction, we can outline the main research question of this thesis with several sub-questions:

How well the DSS architecture can perform State Estimation in the poorly observable Distribution System ?

- How well can the Deep Statistical Solver architecture perform the Distribution System State Estimation task compared to the traditional Weighted Least Square approach?
- How accurate is the estimation provided by the semi-supervised DSS^2, compared to supervised models?
- What are the advantages provided by the DSS architecture?

These questions outline this thesis and will be answered along its development.

We summarize the necessary theory for this thesis in chapter 2, where we provide an introduction to Power System State Estimation, followed by an introduction to Machine Learning, Deep Learning, and Graph Neural Networks. We also introduce in this chapter the Deep Statistical Solver framework. We then present the methodology developed in this thesis in chapter 3, where we introduced the use of DSS in the context of the Distribution System State Estimation problem. In chapter 4, we proceed with case set-up, case studies, and results, with an analysis of the performance compared to the baseline algorithm and other models. Finally, we provide a discussion and a conclusion in chapter 5 to answer the research questions.

2

Theory

2.1. Power System State Estimation

The State Estimation (SE) in Power Systems problem formulation is used to determine the network's state variables using mathematical models of the power system and real-time observations from field meters. Calculating state variables by direct uses of measurement is not possible unless involving exact synchronized phasor measurements of every bus in the network, which is not the case. Usually, measurements in the grid are neither ideal nor synchronized and rarely involve phasors' measurement. As an alternative, the original idea of State Estimation for Power Systems was introduced by Schweppe et al. in 1970 [1] as a methodology to estimate the state of an over-determined power system. The technique aims to minimize the error between theoretical observations and measurements using a Weighted Least Squares calculation, using the redundancy in measurements. The output of the methodology is the estimated state variables, which consist of the steady-state voltage magnitude and angle at each grid bus. This methodology requires that the topology and parameters of the network are completely known [18].

In order to simplify the problem, it is common practice to assume that the power system operates in steady-state and balanced conditions. This assumption means that all three-phased bus loads and branch power flows are balanced and that all three-phased series or shunt devices are symmetrical. With these assumptions, we can model the complete power system using a single-phased positive sequence equivalent circuit which significantly simplifies the problem.

The assumption of a balanced system is more likely to be erroneous in the distribution system, and more complex algorithms involving all three phases of the system should be used instead. In this thesis, we do not consider the problem of an unbalanced system to ease the problem formulation and focus on other challenges of the Distribution System State Estimation (DSSE). Therefore, we will stick to the single-phase modeling of the grid's components. The use of Deep Learning models for DSSE in unbalanced three-phase systems has been investigated in the literature, for instance, with the parallel

training of specific models for each phase, and shows interesting results. The use of a three-phase representation of the distribution system, as detailed in [19], should be considered in future work.

2.1.1. The Weighted Least Squares estimator

State Estimation (SE) aims to find the most likely system state given the non-ideal measurements in the grid and the system model. In statistics, a method suited for such a task is the Maximum Likelihood Estimation (MLE), which assumes that the measurement errors have a known probability distribution of unknown parameters. We can derive the likelihood function L as the joint probability density function of the measurement probability distributions. The maximum values of L are found when the unknown parameters are the closest to their actual values. We can therefore set the optimization problem by maximizing L with these unknown parameters, and the solution provides the best estimates for the parameters. The Weighted Least Squares approach is a derivation of this problem [18].

In this approach, we assume that the m measurements in the system are independent and follow a gaussian distribution. The joint probability density function, the likelihood function, can therefore be expressed as:

$$f_m(\boldsymbol{z}) = f\left(z_1\right) f\left(z_2\right) \cdots f\left(z_m\right) \tag{2.1}$$

Where z_i is the i-th measurement and $\boldsymbol{z}$ represents the vector of m measurements. The objective of the estimation problem is then to maximize this function, which is done by varying the parameters of the density functions of the measurements. As we assume gaussian distributions, our parameters are the mean μ and the standard deviation σ of each density function. We can simplify the optimization by using the Log-Likelihood function $\mathcal{L}$ instead:

$$\begin{aligned} \mathcal{L} = \log f_m(\boldsymbol{z}) &= \sum_{i=1}^{m} \log f\left(z_i\right) \\ &= \sum_{i=1}^{m} \left[\log f(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left\{\frac{z-\mu}{\sigma}\right\}^2} \right] \\ &= -\frac{1}{2} \sum_{i=1}^{m} \left(\frac{z_i - \mu_i}{\sigma_i} \right)^2 - \frac{m}{2} \log 2\pi - \sum_{i=1}^{m} \log \sigma_i \end{aligned} \tag{2.2}$$

With this derivation, we notice that maximizing the likelihood function is equivalent to minimizing the first term of the last line of equation (the other terms being constant).

We now consider the residual r_i of a measurement i, and express the mean μ_i as the expected value $E\left(z_i\right)$ of measurement i, which can be defined as $h_i(\boldsymbol{x})$, the nonlinear function between the system state vector $\boldsymbol{x}$ and the measurement i:

$$r_i = z_i - \mu_i = z_i - E\left(z_i\right) \tag{2.3}$$

In the equation to minimize, we also consider the square of each residual r_i^2 being weighted by $W_{ii} =$

σ_i^{-2}, the inverse of the defined variance of the error of measurement i. Finally, we get an equivalent of the minimization problem as the minimization of the weighted sum of the squares of the residuals [18]:

$$\text{minimize} \sum_{i=1}^{m} W_{ii} r_i^2 \tag{2.4}$$

$$\text{subject to} \quad z_i = h_i(\boldsymbol{x}) + r_i, \quad i = 1, \ldots, m. \tag{2.5}$$

And these equations define the optimization problem of the Weighted Least Squares approach.

For the Power System, considering $\boldsymbol{x}$ in equations 2.5 as the vector of the actual state of the system, we can define the residual r_i of a measurement i as the noise or the error e_i of that measurement. For the whole measurement vector, we then get:

$$\boldsymbol{z} = h(\boldsymbol{x}) + \boldsymbol{e} \tag{2.6}$$

With $\boldsymbol{e}$ the measurement noise vector and h the measurement function. As $\boldsymbol{x}$ and h are deterministic, the probabilistic part of the measurement vector $\boldsymbol{z}$ is $\boldsymbol{e}$, with the assumption that it follows a gaussian distribution:

$$p(\boldsymbol{e}) \sim \mathsf{N}(0, \boldsymbol{R}) \tag{2.7}$$

With $\boldsymbol{R}$ being the measurement error covariance matrix, which is assumed diagonal as we assume independent random variables for the measurements. We therefore have $W_{ii} = R_{ii}$ and:

$$\boldsymbol{R} = \text{diag}\left\{\sigma_1^2, \sigma_2^2, \cdots, \sigma_m^2\right\} \tag{2.8}$$

And our objective function to minimize for the power state estimation, defined now as $J(\boldsymbol{x})$, becomes [18]:

$$J(\boldsymbol{x}) = [\boldsymbol{z} - h(\boldsymbol{x})]^T \boldsymbol{R}^{-1} [\boldsymbol{z} - h(\boldsymbol{x})] = \sum_{i=1}^{m} \frac{[z_i - h_i(\boldsymbol{x})]^2}{R_{ii}} \tag{2.9}$$

The minimum of the objective function is found using the first-order optimality conditions, which is expressed as :

$$g(\boldsymbol{x}) = \frac{\partial J(\boldsymbol{x})}{\partial \boldsymbol{x}} = -\boldsymbol{H}^T(\boldsymbol{x}) R^{-1} [\boldsymbol{z} - h(\boldsymbol{x})] = 0 \tag{2.10}$$

Where $H(x)$ is the first partial derivatives of h with respect to the state vector $\boldsymbol{x}$:

$$\boldsymbol{H}(\boldsymbol{x}) = \left[\frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}}\right] \tag{2.11}$$

Except with the measurement of phasors in every bus of the power grid, the measurement function $h(\boldsymbol{x})$ is non-linear. In such case, $g(x)$ is non-linear and is linearized by expanding it into its Taylor series around its operating point $\boldsymbol{x}^k$ at the k-th iteration:

$$g(\boldsymbol{x}) = g\left(\boldsymbol{x}^k\right) + \boldsymbol{G}\left(\boldsymbol{x}^k\right)\left(\boldsymbol{x} - \boldsymbol{x}^k\right) + \cdots = 0 \tag{2.12}$$

Where $\boldsymbol{G}\left(\boldsymbol{x}^k\right)$ is define as the gain matrix:

$$\boldsymbol{G}\left(\boldsymbol{x}^k\right) = \boldsymbol{H}^T\left(\boldsymbol{x}^k\right)\boldsymbol{R}^{-1}\boldsymbol{H}\left(\boldsymbol{x}^k\right) \tag{2.13}$$

Given that $\boldsymbol{R}$ is the covariance matrix of the measurements, the gain matrix $\boldsymbol{G}$ represents the inverse of the covariance matrix of the estimated states. If we neglect higher orders, a solution of this equation can be found using the iterative Gauss-Newton method [18]:

$$\begin{aligned} \boldsymbol{G}\Delta\boldsymbol{x} &= \boldsymbol{H}^T\boldsymbol{R}^{-1}\Delta\boldsymbol{z} \\ \Delta\boldsymbol{x} &= \boldsymbol{G}^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}\Delta\boldsymbol{z} \\ \Delta\boldsymbol{z} &= \boldsymbol{z} - h\left(\boldsymbol{x}^k\right) \\ \Rightarrow \boldsymbol{x}^{k+1} &= \boldsymbol{x}^j + \Delta\boldsymbol{x} \end{aligned} \tag{2.14}$$

This iterative process stops when it converges toward a solution, meaning that $\Delta\boldsymbol{x}$ reaches a maximum threshold, usually around 10^{-3}. This process requires the matrix $\boldsymbol{H}$ to be fully ranked, which means that the network must be observable. The observability requirement is met when we have at least $2n-1$ independent measurements in the network, with n being the number of buses in the grid.

2.1.2. Distribution System State Estimation

The Power system State Estimation (SE) is defined as “a data processing algorithm for converting redundant meter readings and other available information into an estimate of the state of an electric power system”[5]. Today, the measurement redundancy in the transmission system is used to get the system’s observability and to process bad data. The SE is only effective in the transmission system, as it carries enough measurements, and branches are characterized by a low $\frac{r}{x}$ ratio.

Improvements in the field of SE make it possible to use the WLS algorithm for DSSE. However, the specific characteristics of the Distribution System compared to the Transmission System implies some non-negligible differences of parameters for the algorithm, such as([3]):

- **High R/X ratios** Distribution Networks’ (DN) cables are usually smaller, which results in higher R/X ratios compared to the Transmission Network (TN). The usual algorithm to solve an SE cannot be used in that case as this ratio cannot be neglected anymore.
- **Low measurements availability** The DN is usually way denser than the TN, and carries fewer measurements. As a result, the DN is under-determined and common algorithms in such cases cannot converge towards a solution of SE. Using other sources of measurements such as PMUs,

μPMUs, and smart-meters combined with pseudomeasurements and virtual measurements such as load and generation estimation/forecasting is needed to provide more observability.

- **Scalability and Complexity** DN can be very dense in urban areas, although very sparse in more rural areas. The complexity of the DN makes it harder to develop an efficient algorithm for SE.
- **Unbalanced phases** The DS is often prone to imbalances between phases, a characteristic that does not share the TS and that makes the SE calculation methods inapplicable for DSSE.

Phasor Measurement Units (PMU) and other measurement tools like smart meters have increased the number of measurements in the distribution network. New techniques use the data from these measurement devices to develop an efficient DSSE through a mixed input of measurements and pseudomeasurements. Also, new algorithms are developed to perform the DSSE and tackle the related challenges. [3] enumerates further research, which includes, apart from improving the DSSE algorithm: load forecasting for pseudo measurement generation, event-triggered SE for computation efficiency, efficient incorporation of PMUs, and Automatic Demand Management System using DSSE.

2.2. Deep Learning

This section provides the necessary knowledge that brings the foundation for the methodology proposed in the present thesis. Our methodology is based on the Deep Statistical Solver approach developed by B.Donon in [17], and some mathematical definitions and derivations introduced in this section are also based on this work to follow the same conventions and notations.

2.2.1. Learning from data

Empirical risk

Deep Learning, as an extension of Machine Learning, aims to find patterns between sets of inputs and outputs by learning from a given dataset, and this in order to perform certain tasks. Given the sets of input space $\mathcal{X}$ and output space $\mathcal{Y}$, and the joint probability distribution $p(x, y)$ associated to the product $\mathcal{X} \times \mathcal{Y}$, there is a dependency $\boldsymbol{f}$ linking a sampled input x to its related output y:

$$(x, y) \sim p(x, y) \Rightarrow y = \boldsymbol{f}(x; \epsilon) \tag{2.15}$$

Where the random variable ϵ accounts for noise independent from x. The objective of a Machine Learning algorithm is to approximate this dependency between the variables x and y, which will result in the ability to predict the value of y given any input x. To approximate this function $\boldsymbol{f}$, the class of parametric methods in Machine Learning – which includes Deep Learning – searches in a set of functions $\boldsymbol{f}_\theta : \mathcal{X} \to \mathcal{Y}$ that are parameterized by some trainable parameters $\theta \in \Theta$. The objective becomes therefore to find the parameterized function f_θ that best approximates the dependency $\boldsymbol{f}$, which is equivalent to minimizing the underlying *risk* [17]:

$$\mathbb{E}_{x,y\sim p(x,y)} \left[L(\theta; x, y) \right] \tag{2.16}$$

with $L(\theta; x, y)$ defined as the *loss function* that estimates the quality of the approximation f_θ for a set of samples (x, y). Having access only to a finite set of values x and y sampled from the distribution $p(x, y)$, the learning task in Machine Learning is done through the minimization of the risk on a finite *train set* $D_{train} = \{(x_m, y_m)\}_{m \in \mathcal{M}_{train}}$, which means that the estimation is made through the minimization of the *empirical risk* over the train set [17]:

$$\frac{1}{|M|} \sum_{m \in \mathcal{M}_{train}} L(\theta; x_m, y_m) \tag{2.17}$$

Data splitting

We defined in the previous section the train set D_{train} as the dataset available for training our model. A common Machine Learning practice to ensure the generalization of our model to unknown distributions is to split the overall available dataset into three subsets: the previously mentioned train set, the validation set, and the test set.

The validation set $D_{val} = \{(x_m, y_m)\}_{m \in \mathcal{M}_{val}}$ is used in the hyperparameter tuning process. Indeed, an ML architecture is designed using several parameters that impact the model's performance. A common ML practice is to compare the performance of an ML model for different sets of hyperparameters by evaluating its performance on the validation set. This step is called the hyperparameter tuning step and occurs after the training. The model that performs best on the validation set is selected as the fine-tuned one, maximizing the performance for the given task. Comparing the model's performance on the train set and the validation set also helps to detect any underfitting or overfitting behavior, as shown in Figure 2.1, which may be countered with proper hyperparameter tuning. We will provide in the upcoming sections more details on hyperparameters and tuning in the context of Neural Networks.

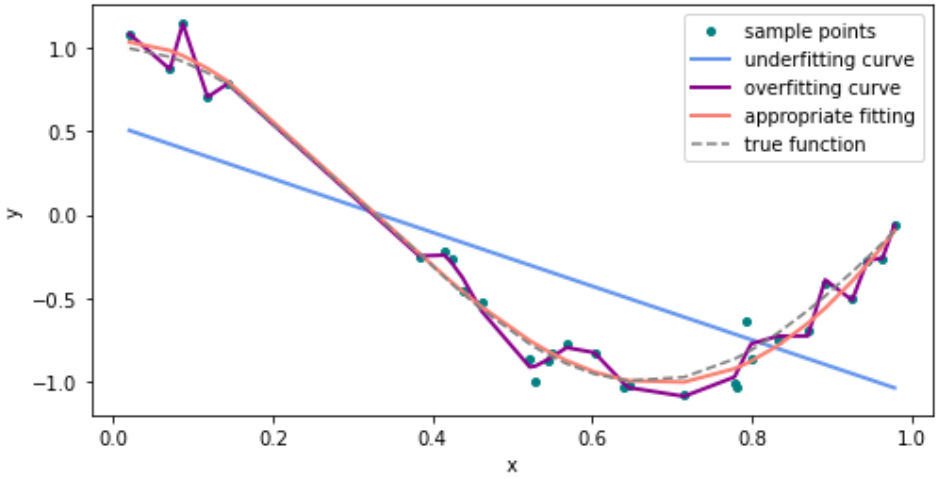

Figure 2.1: Example of underfitting and overfitting problems in Machine Learning regression task.

After the tuning step, the test set $D_{test} = \{(x_m, y_m)\}_{m \in \mathcal{M}_{test}}$ is used to evaluate the generalization of the model for unseen data points. This evaluation provides an insight into the model's performance for the given task.

Loss function and learning approaches

The choice of the loss function depends on the learning problem at hand, which we can classify into three categories: supervised, unsupervised, and semi-supervised learning.

In the case of *supervised learning*, for each sample of input data x used in the training step, an output value y is known and referred to as the *label* of that sample x. The most common tasks in supervised learning are regression and classification. In the case of regression tasks where y is a continuous value, a common objective is the minimization of the Euclidean distance between the predicted value $f_\theta(x)$ and the label y, and a typical loss function is the Mean Squared Error (MSE) [20]:

$$L_{\text{MSE}}(\theta; x_m, y_m) = \|f_\theta(x) - y\|_2^2 \tag{2.18}$$

In the case of classification tasks, y is a discrete value that denotes a specific class, and the objective is to maximize the number of correct discrete-class predictions. A typical loss function is the Cross-Entropy which represents the deviation between two probability distributions. In the case of a binary

classification task (y is either 0 or 1), this loss is defined as [21]:

$$L_{\text{Cross-Entropy}}(\theta; x_m, y_m) = -\left(y \log\left(f_\theta(x)\right) + (1-y) \log\left(1 - f_\theta(x)\right)\right) \tag{2.19}$$

Unsupervised learning, as opposed to supervised learning, is used when labels y are unknown, and we cannot compare the predicted value to any reference of output value. Unsupervised learning does not perform regression or classification tasks as no mapping between inputs and outputs can be approximated without some knowledge of the output. Instead, standard Machine Learning unsupervised problems will aim to learn patterns in input data to discover underlying structures in them. A common task in unsupervised learning is clustering, which aims at grouping unlabeled data following their similarities and differences. Other tasks include Association Rules, and Dimensionality Reduction [22].

Finally, the *semi-supervised learning* approach is used when noisy, limited, or imprecise information about labels is provided. The methodology developed in this thesis is an example of semi-supervised learning, inspired by the Physic-Informed Neural Network (PINN)[23] semi-supervised learning framework, which is itself part of the wider Physic-Informed Machine-Learning (PIML)[24] framework. As a hybrid approach, semi-supervised learning combines supervised learning to learn input-output mapping with the known labels and unsupervised learning to learn hidden patterns as extra information in the learning process [25]. The DSS^2 approach proposed in this thesis is an example of semi-supervised learning, where we use the power flow equations to link our output to the measurements used as input. The power flow equations are therefore used to get some kind of *semi-labels* to learn upon.

2.2.2. Artificial Neural Networks

Artificial Neural Networks (ANN), or Neural Networks (NN), aim to emulate the human brain by transferring and processing input information through a network of artificial neurons to perform a specific task using the output information. This artificial neuron used to process information is the *perceptron*, shown in Figure 2.2.

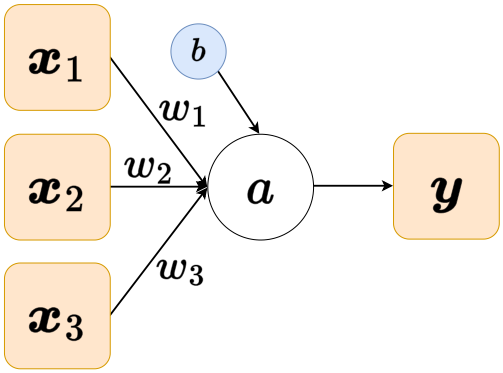

Figure 2.2: Single perceptron representation

Given an input vector x of m sampled values and n information features, a weight vector w and a constant bias b:

$$\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n] \in \mathbb{R}^{n \times m} \tag{2.20}$$

$$\boldsymbol{w} = [w_1, w_2, \cdots, w_n]^\intercal \in \mathbb{R}^{1 \times n} \tag{2.21}$$

$$b \in \mathbb{R}^{1 \times 1} \tag{2.22}$$

The perceptron applies first a linear transformation to the input vector $\boldsymbol{x}$, multiplying it by the weight vector $\boldsymbol{w}$ and adding the bias b, and further apply an "activation function" a to get the output vector $\hat{\boldsymbol{y}}$ defined as:

$$\hat{\boldsymbol{y}} = a\left(\boldsymbol{w}\boldsymbol{x} + b\right) \in \mathbb{R}^{1 \times m} \tag{2.23}$$

The activation function a is a non-linear transformation used to enable the mapping of non-linear relationships between the input $\boldsymbol{x}$ and the output $\hat{\boldsymbol{y}}$. Commonly used activation function includes the sigmoïd function, the hyperbolic tangent, and the Rectified Linear Unit (ReLU), which are shown in Figure 2.3.

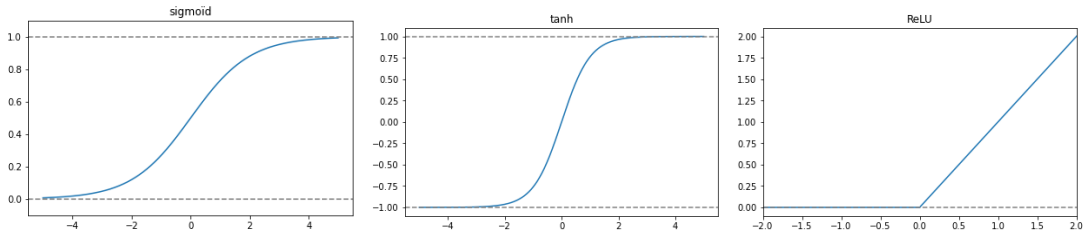

Figure 2.3: Most commonly used activation functions

The fundamental architecture of Deep Learning, the Artificial Neural Network (ANN), uses the concept of perceptron (or neuron) as a building block for a network of neurons to approximate complex relationships between sets of input features and output variables. Such networks vary in layers and number of neurons per layer, depending on the problem. An example of an ANN is given in Figure 2.4 for a network of one hidden layer of 4 neurons followed by an output layer of one output value. For an ANN of T layers, with each layer t of size n_t, the mathematical expression of the relationship input/output (also called the Forward Propagation step) becomes:

$$\boldsymbol{h}(0) = \boldsymbol{x} \tag{2.24}$$

$$\boldsymbol{h}(t) = a_t\left(\boldsymbol{w}_t\boldsymbol{h}(t-1) + \boldsymbol{b}_t\right) \tag{2.25}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{h}(T) \tag{2.26}$$

Where, in the case of a single output variable and for $t = 1, \cdots, T$:

$$\boldsymbol{w}_l = [w_{t11}, w_{t12}, \cdots, w_{tn_{t-1}n_t}] \in \mathbb{R}^{n_t \times n_{t-1}} \tag{2.27}$$

$$\boldsymbol{b}_t = [b_{t1}, b_{t2}, \cdots, b_{tn_t}] \in \mathbb{R}^{n_t \times 1} \tag{2.28}$$

$$\boldsymbol{h}(t) = [a_{t1}, a_{t2}, \cdots, a_{tn_t}] \in \mathbb{R}^{n_t \times m} \tag{2.29}$$

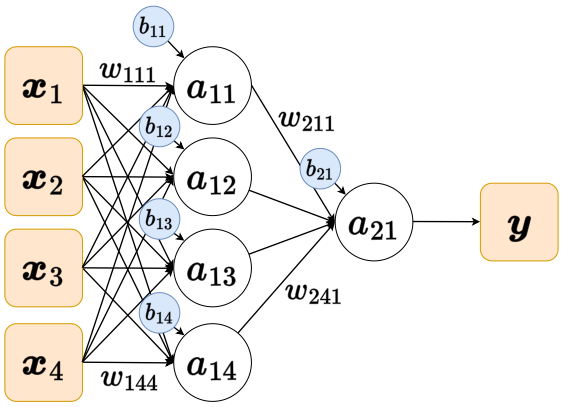

Figure 2.4: Example of Artificial Neural Network with one hidden layer, and a single output. A constant value, namely a bias, is added at each single neuron to fit possible offsets. Most weight symbols are hidden to ease reading.

The combination of more perceptrons allows an increasing complexity of relationships between inputs and outputs to solve increasingly more complex classification/prediction tasks. By adding layers of neurons between the input layer and the output layer, the so-called ”hidden layers,” we can create more abstract computation algorithms to find the mapping that approximates complex relationships. Neural Networks with numerous hidden layers are called *Deep* Neural Networks (DNN) and are the premises of Deep Learning.

The names ANN and DNN both refer to the standard architecture of NN in Deep Learning and are often used interchangeably. Even though there can be some slight variation in the exact definition, Multi-Layer Perceptron (MLP) and Feed-Forward Neural Network (FFNN) also refer to similar architectures, and all four abbreviations are used interchangeably in the literature.

2.2.3. Training Neural Networks

As shown by equation 2.25, the mapping between input and output variables of a NN is defined by its weights and biases. These parameters are called the *trainable parameters* of the NN and are denoted as θ, as opposed to the non-trainable parameters called *hyperparameters* that are set when designing the model. Such hyperparameters include the activation function, the number of hidden layers, the size of a hidden layer, and the learning rate when using gradient descent (defined later in this section).

Consequently, a NN is ”trained” to find a mapping to perform a specific task by tuning its weights θ accordingly. This training step is called the Back Propagation step, and the most common algorithm used to perform this step is called Gradient Descent which uses the gradient of the empirical risk computed with the pre-defined loss function.

Gradient Descent

After defining a loss function $L(\theta; x, y)$ to compute the empirical risk of a dataset(x, y), we use the back-propagation algorithm to compute the derivative of the model with regards to its trainable parameters $\nabla_\theta L(\theta; x, y)$. The gradient descent learning process then uses the first-order gradient of that loss over the trainable parameters θ to update the weights:

$$\theta \leftarrow \theta - \eta \times \frac{1}{|M|} \sum_{m \in M} \nabla_\theta L\left(\theta; x_m, y_m\right) \tag{2.30}$$

With $\alpha \in]0;1[$ defined as the *learning rate* and correspond to a hyperparameter to set during design. We repeat the update step until we observe satisfying performances of the model on the train set. More intuitively, the gradient of the loss function represents its curve with regard to the weights, and we can see the iterative update process as successive steps toward the global minimum of that function. The hyperparameter α defines the size of these steps.

Computing the gradient over the whole train set is computationally expensive, so minibatch gradient descent has been proposed to accelerate the computation of the gradient by computing the gradient over successive minibatches $M_{\text{batch}} \subset M$ of the train set. Minibatch gradient descent represents the foundation of today's training optimizers.

This thesis uses AdaMax, a training optimizer derived from the widely used optimizer Adam [26]. This family of optimizers improves the minibatch gradient descent algorithm by adapting the learning rate, using the estimates of the first and second moments of the gradient.

Hyperparameter tuning

As explained earlier, a subset of the available dataset, called the validation set, is kept aside during training. This subset is then used during the hyperparameter tuning step, where we compare the performance of different models' designs on that subset. Specifically, for each set of hyperparameters, we build a model and train it using the train set, and we evaluate its performance on the validation set. We then keep the model that performs best as the final design, and we fix the hyperparameters.

A standard method to explore the hyperparameters' space is called *Grid search*. This method selects a few possible values for each hyperparameter and browses the space through the defined grid. In programming, this corresponds to a succession of for-loops that browse all the possible sets of hyperparameters.

For a high number of hyperparameters, it is highly time-consuming to try all the possible sets. As an alternative, *Random search* is a method that randomly selects values of hyperparameters and, hence, searches randomly for the best possible set [27]. This method can outperform the grid search if only a few hyperparameters affect the model's performance.

To further increase the efficiency in hyperparameter tuning, *Bayesian optimization* uses educated guesses to optimize the search by balancing exploration and exploitation. Specifically, the method browses through the hyperparameters while learning from the performances of the searched sets [28]. It usually outperforms grid and random searches but requires a more complex tuning algorithm.

2.2.4. Convolutional Neural Network

Standard NN such as DNN introduced in previous sections shows exceptional abilities to approximate the relationship between input and output values. Studies demonstrated the property of DNN to approximate any relationship given a large and deep enough architecture [29]. However, training a DNN on a complex mapping gets increasingly expensive with the complexity of the problem. For example,

nonlinear data such as images are very bulky when linearized, and training a standard DNN for computer vision is tedious. A small picture of 256x256 pixels represents an input layer of 256x256x3 = 196 908 units (each pixel containing three different values to express its color). If we connect this input layer to a hidden layer containing only 32 neurons, the built network already has 256x256x3x32 = 6 291 456 weights to train on in its first layer. Hence, DNN models used for image analysis quickly have millions of parameters, which means the need for millions of samples and hours (or days) of training. Moreover, such architectures with many parameters show poor generalization performance and tend to overfit.

As an alternative, Convolutional Neural Networks (CNN) account for the invariants in the processed data to increase generalization abilities and ease the training process. The primary assumption is that, when processing natural images, the information in a sample is unaltered by translations, and the input features should not be spatially dependent. This property is known as translation invariance, and CNN intrinsically encodes this invariance in the data through its convolution layer [30].

The convolution layer of a CNN uses the convolution operation, which, for the convolution between two discrete functions f and g, is defined in the discrete domain as:

$$h_i = (f * g)(i) = \sum_j f(j) \times g(i - j) \tag{2.31}$$

With j being a multi-index. Now, if we set $k = i - j$, introduce $w_k := g(k)$ and define $f(j) = x_j$, we get:

$$h_i = \sum_k x_{i-k} \times w_k \tag{2.32}$$

Which shows the convolution layer's behavior: a defined filter w_k browses through the input picture x_k, acting as a sliding window and outputting a value that depends on the local pattern, and then provides a new filtered picture as output. An example of such a filter using matrices is provided in Figure 2.5, where a given input matrix is convoluted with a 2x2 matrix filter, namely a convolution kernel. We can notice a higher output when the window encounters a pattern similar to the kernel, which is better observed when representing the numerical cells as gray pixels. Another important property is the respect of the translation invariance, as the upper-left output cell has an equal output value as the lower-right cell, as expected given the same local pattern encountered by the filter.

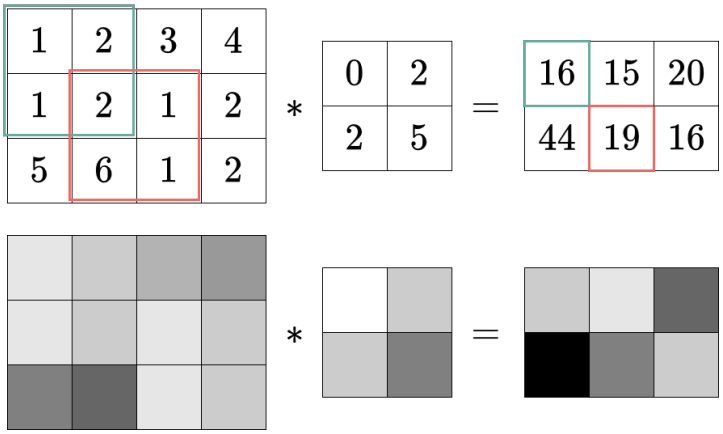

Figure 2.5: Example of convolution on a matrix. The 2x2 matrix is used as a convolution kernel, sliding through the 3x4 matrix and aggregating the information into a unique output cell at each position.

The example above shows how a unique filter can highlight specific patterns in an input grid and aggregate the information. A convolution layer can stack multiple filters together to detect specific patterns such as horizontal and vertical lines simultaneously. Using more layers in the network allows the detection of more complex patterns: In face recognition, the first layer of a CNN will detect elementary edges, a second layer will detect facial elements such as mouths and eyes, and a third layer will be able to detect whole specific faces. These detection abilities can be learned by updating the weights of the filters during training, similar to a DNN but with drastically fewer weights to train.

Using convolution layers to intrinsically encode the invariance of grid-structured data such as images and texts, CNN showed ground-breaking performances while learning drastically faster than standard ANN. With convolution as a local operation, the convolution layer focuses on local patterns rather than global information, improving local features' detection. Moreover, using defined filters allows sharing the weights across the input features, drastically reducing the number of trainable parameters. Finally, the resulting network is intrinsically translation-invariant because of the convolution properties. CNN can successfully process input data that share the same translated information, increasing the model's generalization abilities.

2.2.5. Graph Neural Networks

In recent years, Graph Neural Networks (GNN) has been introduced to generalize the convolution technique to more abstract and flexible data such as graphs. The success of CNN, showing outstanding performance and outperforming DNN by encoding invariants of the data in the model, has inspired the development of new models that can generalize these ideas. Indeed, we can see images as ordered meshed graphs where a pixel represents a node, and every pixel/node is connected to its neighbors. Even texts that CNN well processes in a 1-dimensional framework can be seen as ordered graphs where a word is a node and a sentence is a chain of nodes.

As a matter of fact, CNN can successfully process images mainly due to the grid-shaped arrangement of pixels in such data. Indeed, pixels can be described using Euclidean coordinates, and relative positions between pixels are easily defined, which eases the learning of local information.

GNN aims at generalizing this ability to graph data. However, other techniques are needed as graphs' vertices cannot be described with Euclidean coordinates, and no intrinsic ordering exists between vertices. The only structural information to work with, analogous to the relative positions between pixels in an image, is the defined set of *neighborhood* of a graph which describes the connections between vertices. Thus, using local information similarly to CNN relies on aggregating information between neighbors, using local operations such as the sum or the mean. Moreover, as there is no intrinsic structure for a specific graph, the latent and output representations obtained from the input graph data also need to keep the same graph structure, as opposed to the condensing techniques used in CNN.

Graph data

Graphs' structure consists in a set of $n \in \mathbb{N}$ vertices, denoted by (V) connected to each other by a set of edges (E). These edges are either undirected or directed depending on the type of graphs at hand, and $(E) \subseteq [n]^2$ (considering the notation $[n] := \{1, \dots, n\}$, and therefore $\mathcal{V} = [n]$). The vertices of the graph are usually denoted by their index $i \in [n]$, and we denote the edges by the multi-index $(i, j) \in [n]^2$. The structure of a graph is then denoted by $((V), (E))$

Describing graphs as samples in a dataset, the features of the graphs are added as continuous values to the structure defined above. Each vertice or edge of the graph can carry features, which can be input features x as well as output features y. Following the notation defined in [17], we denote the vertices' input features by $x^v = (x_i^v)_{i \in [n]}$ and the edges' input features by $x^e = (x_{ij}^e)_{(i,j) \in (E)}$, and $x = (x_v, x_e)$. Similarly, we denote the output features by $y = (y^v, y^e)$. We also denote the set of input and output features of a graph $([n], (E))$ by $(X)_{n,(E)}$ and $(Y)_{n,(E)}$, and the sets of all input and output features defined over any graph structure are denoted by $\mathcal{X}$ and $\mathcal{Y}$. An example of graph carrying input features is shown in Figure 2.6.

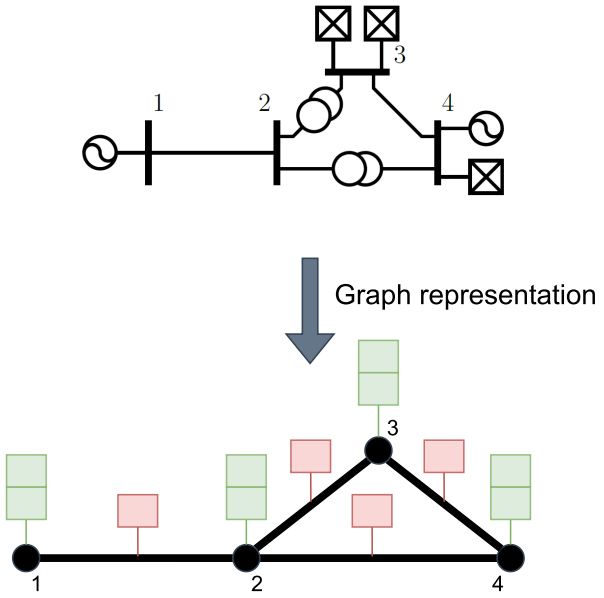

Figure 2.6: Example of a homogeneous graph representation for a simple power grid. The homogeneous graph consists in vertices connected by edges, so generators and loads are aggregated together in the vertice features while the lines and transformers are aggregated together into edge features.

Encoding equivariance under permutation

Given this graph representation and the vertices' ordering, essential properties are the *invariance* of the graph's structure and the *equivariance* of the graph's features under permutations. Indeed, the ordering of vertices is often an arbitrary choice, and graph structures are unaltered by re-ordering, while graph features are re-arranged. Given a permutation $\sigma \in \Sigma_n$ switching vertices ordering, and applying it to a graph input x, the features are preserved such as [17]:

$$\begin{aligned} \sigma \star x^v &= \left(x^v_{\sigma^{-1}(i)}\right)_{i\in[n]} \\ \sigma \star x^e &= \left(x^e_{\sigma^{-1}(i)\sigma^{-1}(j)}\right)_{(i,j)\in\mathcal{E}} \end{aligned} \tag{2.33}$$

The impact of permutation on output features y are similar, and highlights the *equivariance over permutation* property of such features. The objective of GNN designs is to be able to intrinsically encode this equivariance property.

We previously defined the problem in Machine Learning as finding an approximation $\boldsymbol{f}_\theta$ to a target mapping $\boldsymbol{f} : \mathcal{X} \to \mathcal{Y}$ for an input x and a possible label y sampled on a distribution $p(x, y)$. Defining both the input space $\mathcal{X}$ and the output space $\mathcal{Y}$ as sets of graphs, the mapping $\boldsymbol{f}$ approximated by a GNN is stated to preserve the input graph structure, and pairs x and y have the same graph structure.

A function $\boldsymbol{f} : \mathcal{X} \to \mathcal{Y}$ is defined as equivariant to permutation if [17]:

$$\forall n \in \mathbb{N}, \forall \mathcal{E} \subseteq [n]^2, \forall x \in \mathcal{X}_{n,\mathcal{E}}, \begin{cases} \boldsymbol{f}(x) \in \mathcal{Y}_{n,\mathcal{E}} \\ \forall \sigma \in \Sigma_n, \boldsymbol{f}(\sigma \star x) = \sigma \star \boldsymbol{f}(x) \end{cases} \tag{2.34}$$

As stated above, the features of the graphs are equivariant over permutations, which means that the mapping $\boldsymbol{f}$ is permutation-equivariant. Therefore, the parameterized function $\boldsymbol{f}_\theta$ defined by the GNN model and that approximates the mapping $\boldsymbol{f}$ must be permutation-equivariant as well in order to keep strong performances under a change of vertice ordering. This is where lies the strength of GNN: instead of training a DNN over all the permutations (which means to include $n!$ samples for a single graph instance), GNN processes graph data directly while preserving the structure of the graph and intrinsically encoding the permutation-equivariance of the given data.

GNN implementation

For the GNN to provide a mapping $\boldsymbol{f}_\theta$ that is equivariant to permutation, the operations implemented within the model need to be permutation equivariant as well. There are multiple methods developed in the literature to implement GNN with permutation-equivariance. The foundation of the Deep Statistical Solver approach relies on the Spatial Graph Neural Network implementation, which decomposes the process of graph data into three steps: the encoding step, the message-passing step, and the decoding step [17]. The overall process is schematized in Figure 2.7.

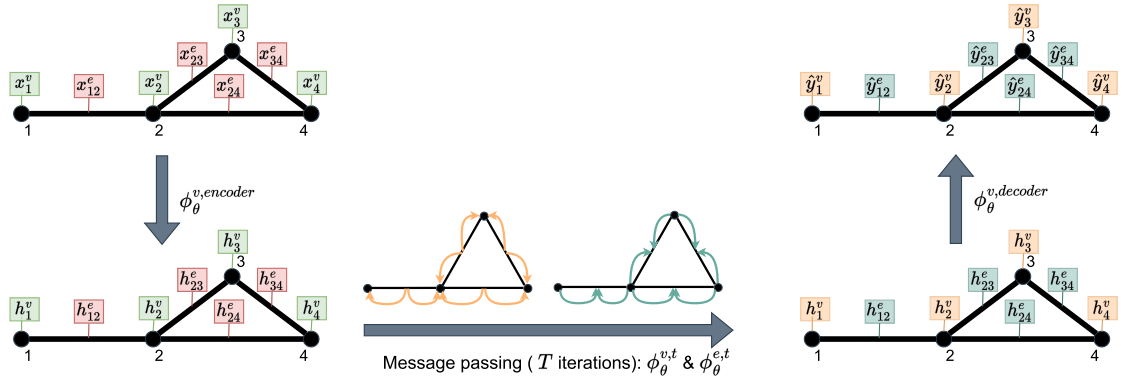

Figure 2.7: Data processing operations in a GNN. The input data x is first encoded to latent variables h, then the information is spread and aggregated through the graph during the message passing step, and finally the predicted output $\hat{y}$ is given by decoding the updated latent variables.

Firstly, the **encoding** step embeds the input of each vertex and edge to a latent space:

$$\begin{aligned} h_i^v &\leftarrow \phi_\theta^{v,encoder}\left(x_i^v\right) \\ h_{ij}^e &\leftarrow \phi_\theta^{e,encoder}\left(x_{ij}^e\right) \end{aligned} \tag{2.35}$$

For each vertice $i \in [n]$, the input features of the vertice x_i^v are mapped to a latent variable $h_i^v \in \mathbb{R}^d$ using a neural network $\phi_\theta^{v,encoder}$, which is the same for every vertice. Similarly, the input features x_{ij}^e of each edge $(i,j) \in \mathcal{E}$ are mapped to a latent variable $h_{ij}^e \in \mathbb{R}^d$ using the same neural network $\phi_\theta^{e,encoder}$. The dimension of the latent variables are defined by d, which is a hyperparameter of the model. This process is done independently and in parallel for each vertice and edge.

Then, the **message passing** step iteratively spread the information between neighbours of the graph. This second step is the most important one, allowing to process the global information of the graph and build a mapping between input and output features. At each iteration $t = 0, \cdots, T-1$, the message passing step is performed as [17]:

$$\begin{aligned} h_i^v &\leftarrow \phi_\theta^{v,t}\left(h_i^v, \left\{h_{ij}^e\right\}_{(i,j)\in\mathcal{E}}, \left\{h_{ji}^e\right\}_{(j,i)\in\mathcal{E}}\right) \\ h_{ij}^e &\leftarrow \phi_\theta^{e,t}\left(h_{ij}^e, h_i^v, h_j^v\right) \end{aligned} \tag{2.36}$$

With the total number of iteration T being a hyperparameter of the model. During this step, each vertice $i \in [n]$ is updated using the same neural network $\phi_\theta^{v,t}$, which takes as input the current latent variable of the vertice h_i^v as well as the latent variables of the connected edges. Similarly, the latent variable of each edge h_{ij}^e is updated through another neural network $\phi_\theta^{e,t}$, taking its current latent value and the latent value of the connected vertices as input.

Finally, a **decoding** step processes the result of the message passing step to final output values:

$$\begin{aligned} \hat{y}_i^v &\leftarrow \phi_\theta^{v,decoder}\left(h_i^v\right) \\ \hat{y}_{ij}^e &\leftarrow \phi_\theta^{e,decoder}\left(h_{ij}^e\right) \end{aligned} \tag{2.37}$$

Similarly to the encoding step, the output values of the vertices $\hat{y}_i^v$ are obtained through the mapping of the related latent variable h_i^v by a unique neural network $\phi_\theta^{v,decoder}$, and the output values of the edges $\hat{y}_{ij}^e$ through the mapping of the edge latent variable h_{ij}^e by a neural network $\phi_\theta^{e,decoder}$.

The overall architecture is made of $2+2T+2$ neural networks ϕ. These neural networks are usually built as Multi-Layer Perceptrons (MLP) trained together in a standard fashion. Using a recurrent architecture where we use the same neural network at each iteration of the message passing step, this amount of MLPs is reduced to only 6 $(2+2+2)$.

With their networks of buses and lines, it is possible to represent power systems' data as a graph-structured type of data. The GNN implementation described above provides a method to efficiently process data with such properties. The model used in this thesis uses this foundation to propose a state estimation model that successfully processes data from the distribution system.

2.2.6. Physic-Informed Machine Learning

In the task of simulating multiphysics problems, traditional numerical simulators cannot seamlessly incorporate noisy data, and solving inverse problems with hidden physics remain an expensive and complex task. Solutions proposed in the literature using ML showed promising results; however, training deep neural networks for such tasks requires a massive amount of data, which is not always available. Such motivation, similar to the ones presented in this thesis, lead to the development of networks trained with additional information obtained by enforcing the physical laws [24]. With such semi-supervised learning approaches, ML models can be used to simulate multiphysics, which allows incorporating noisy data and an easier solving of inverse problems while alleviating the need for big data by enhancing the physical properties of a given problem. Depending on the data availability, the design of a PIML model can be divided into three main categories, as shown in Figure 2.8, which depicts how physics can be used to counter the lack of data in training an ML model.

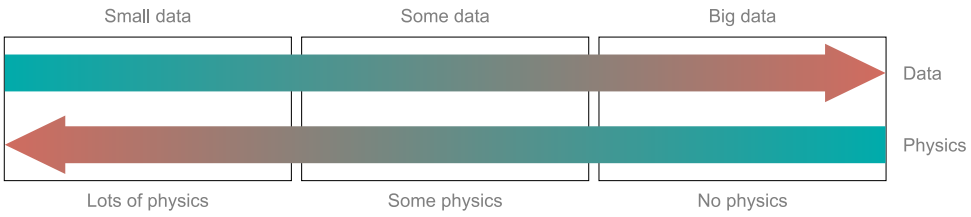

Figure 2.8: PIML data and physics scenarios as described in [24]. Enforcing physics into a ML model can enhance the performance of the model when data is missing.

Numerous PIML approaches have been proposed in the literature in the past decade, and we can list them into three main principles [24]:

- **Observational biases** If sufficient data that covers the input domain and reflects the underlying physics of the problem is provided, ML techniques show remarkable power in learning the complex physical relationships directly from the data. Observational biases can therefore be seen as the main principle of ML, where a model learns the underlying input-output mapping directly

from data. However, this is a weak learning mechanism, and a large volume of data is typically necessary to enforce these biases, especially for over-parameterized DL models.

- **Inductive biases** By embedding prior knowledge associated with a given task to a NN architecture using a specialized design, it is possible to guarantee to satisfy specific physical laws or constraints through the tailored architecture. CNN is the most famous case of inductive bias: its architecture implicitly respects the invariance along groups of symmetries and distributed pattern representations and significantly improved computer vision. GNN, as a generalization of CNN for graph-structured data, is also an excellent example of inductive bias that respects the equivariance of graph permutation and rotation.
- **Learning biases** As softer constraints to the model than inductive biases, learning biases are introduced by applying physical laws or constraints as penalties to the loss function. In this approach, we explicitly favor the convergence to solutions that comply with the physics of a given problem. It can be viewed as multi-task learning where we simultaneously constraint the learning process to fit the observed data and to satisfy the given physical constraints. PINN is an example of learning biases where, as shown in Figure 2.9, the output of the NN is passed through a set of PDE, and the result is added to the loss function.

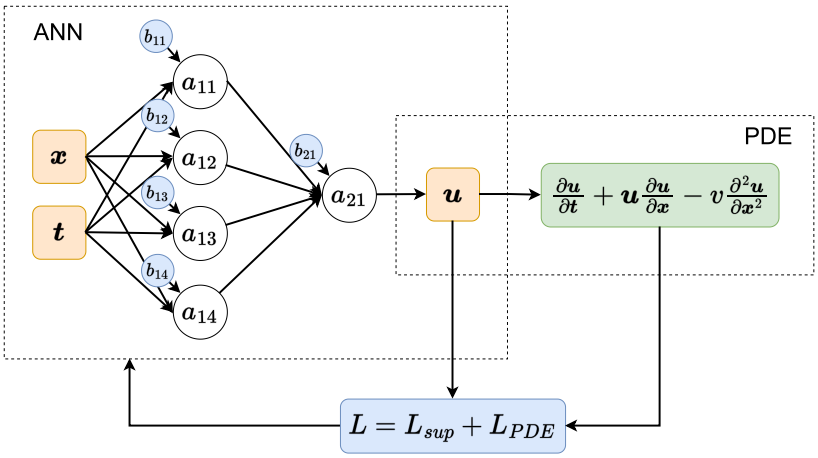

Figure 2.9: Physic-Informed Neural Network (PINN) example for solving viscous Burger's equation, as developed by [23].

2.3. Deep Statistical Solver

The methodology developed in this thesis is mainly based on the Deep Statistical Solver (DSS) architecture proposed by B. Donon [17]. This architecture is an innovative and promising framework for the seamless development of GNN applications for Power Systems operations. This framework is based on two main components: the Hyper Heterogeneous Multi Graph Neural Network (H2MGNN) model, which aims at modeling power systems as graphs more accurately, and the Statistical Solver Problem (SSP) approach, which is used to train a deep learning model to learn to optimize on a target problem.

2.3.1. Hyper Heterogeneous Multi Graph (H2MG)

The first component of the DSS architecture, the Hyper Heterogeneous Multi Graph (H2MG), is a new data formalism proposed by [17] to simplify the integration of power grid data into models compared to traditional graphs while avoiding any loss of information.

Indeed, traditional graphs made of vertices and edges cannot accurately represent power grids, where multiple objects such as generators, lines, buses, and loads are connected to form a very complex network. On top of that, having multiple objects of the same kind connected to the same bus is very common in power networks but impractical to implement in traditional graphs. A workaround is to usually aggregate these components on the bus, which implies loss of information and inaccuracy in the model.

To answer these limitations, [17] uses the concept of hyper-graphs and hyper-edges to accurately model power grids as formal graphs. The H2MG formalism is defined by:

- **Objects as hyper-edges** Every object forming the network at hand is modeled as a hyper-edge, a kind of edge that can be connected to any number of vertices. For example, in the power grid, a line or a transformer would be represented as hyper-edges connected to two vertices, whereas a bus, a load, or a generator would be modeled as hyper-edges connected to one vertice.
- **Vertices as port** Using traditional graphs, vertices would represent the buses of our grid, connected by the lines as edges. Here, even buses are seen as hyper-edges connected to one vertice, and these vertices are identified ports of our model where components are connected.
- **Hyper Heterogeneous Multi Graph** The collection of hyper-edges connected through vertices is called a hyper-graph, and this graph is called “heterogeneous” if it’s made of multiple classes of objects, such as a power grid. If multiple objects of the same class lie on the same hyper-edge (same connections to port), these objects are called multi objects, and a graph containing such a concept is called a multi-graph. In this work, we left multi objects out and modeled the power system as a Hyper Heterogeneous Graph.
- **Structure and features** Following previous points, The H2MG formalism can be defined as a set of objects of specific classes connected to vertices through connection patterns. In mathematical notation, H2MG datasets can be expressed as:

$$\mathcal{G}_x = \{(c, e, m) \mid c \in \mathcal{C}, e \in \mathcal{E}^c, m \in \mathcal{M}_e^c\} \tag{2.38}$$

With $c \in \mathcal{C}$ the objects’ class, $e \in \mathcal{E}^c$ the objects’ hyper-edge and $m \in \mathcal{M}_e^c$ the set of multi-objects

(which is neglected in this thesis). On top of this structure, hyper-edges carry features of the datasets (vertices, as connection port, do not carry any features). In this formalism, the data collections of couple (x, y) can be written as:

$$x = \left(x^c_{e,m}\right)_{(c,e,m)\in\mathcal{G}_x} \tag{2.39}$$

$$y = \left(y^c_{e,m}\right)_{(c,e,m)\in\mathcal{G}_y} \tag{2.40}$$

And we denote by $\mathcal{X}_{n,\mathcal{C},\mathcal{E},\mathcal{M}}$ and $\mathcal{Y}_{n,\mathcal{C},\mathcal{E},\mathcal{M}}$ the input and output graph sets over the previously defined graph structures.

An example of such H2MG modelling is presented below in Figure 2.10, compared to a standard graph model.

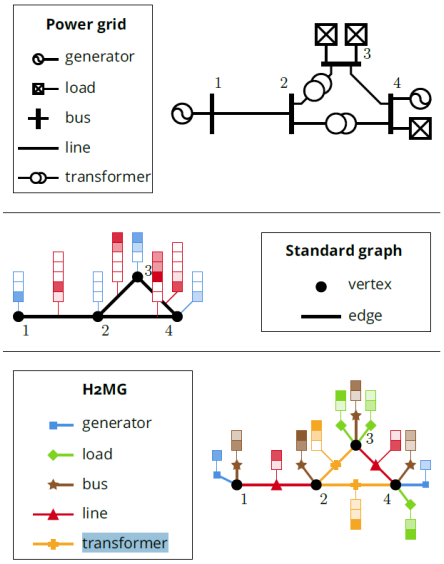

Figure 2.10: Example of H2MG model of a power grid compared to standard graph model [17]

2.3.2. Hyper Heterogeneous Multi Graph Neural Network (H2MGNN)

With the introduction of the H2MG formalism, we can develop a Deep Learning architecture, namely the Hyper Heterogeneous Multi Graph Neural Network (H2MGNN) introduced by [17], that works with such data structure.

The H2MGNN architecture is inspired by the Neural Ordinary Differential Equation (NODE) literature [31]. In the NODE literature, the main idea is to consider deep neural networks as dynamic systems of latent variables. Indeed, considering a fully-connected neural network of T hidden layers, we have the following recurrence equation at each layer t:

$$h(t+1) = \phi^t_\theta(h(t)) \tag{2.41}$$

Where ϕ^t_θ is the weighted mapping between layers and θ the set of trainable weights. In such large

neural networks, it is common solution against vanishing gradient to skip connections [32]:

$$h(t+1) = h(t) + \phi_\theta^t(h(t)) \tag{2.42}$$

In the NODE literature [31], it is suggested that equation (2.42) is similar to Euler scheme for solving differential equations over time T:

$$\frac{dh}{dt} = \phi_\theta(t, h(t)) \tag{2.43}$$

Considering t as time instead of layer, we can observe now a link between deep neural networks and dynamical systems of latent variable if we consider a time interval of $[0, 1]$ with a time step of $\Delta t = \frac{1}{T}$. This similarity is used in NODE and inspires the H2MGNN algorithm used in this thesis. This algorithm is presented in the pseudo-code in Algorithm 1, and is shown visually in Figure 2.11.

Algorithm 1 H2MGNN algorithm as proposed by [17]

procedure $\boldsymbol{f}_\theta(x = \left(x^c_{e,m}\right)_{(c,e,m)\in\mathcal{G}_x})$

▷ Initialization
  for $i \in [n]$ **do**
    $h_i^v \leftarrow 0^d$
  end for
  for $(c, e, m) \in (G)_x$ **do**
    $h^c_{e,m} \leftarrow 0^d$
    $\hat{y}^c_{e,m} \leftarrow \hat{y}^c_\theta$
  end for

▷ Latent interaction
  $t \leftarrow 0$
  while $t < 1$ **do**
    for $i \in [n]$ **do**
      $h_i^v \leftarrow h_i^v + \Delta t \times \sum_{(c,e,m,o)\in\mathcal{N}_x(i)} \phi_\theta^{c,o}\left(t, h_e^v, h^c_{e,m}, \hat{y}^c_{e,m}, x^c_{e,m}\right)$
    end for
    for $(c, e, m) \in \mathcal{G}_x$ **do**
      $h^c_{e,m} \leftarrow h^c_{e,m} + \Delta t \times \phi_\theta^{c,h}\left(t, h_e^v, h^c_{e,m}, \hat{y}^c_{e,m}, x^c_{e,m}\right)$
      $\hat{y}^c_{e,m} \leftarrow \hat{y}^c_{e,m} + \Delta t \times \phi_\theta^{c,y}\left(t, h_l^v, h^c_{e,m}, \hat{y}^c_{e,m}, x^c_{e,m}\right)$
    end for
    $t \leftarrow t + \Delta t$
  end while

  return $\hat{y} = \left(\hat{y}^c_{e,m}\right)_{(c,e,m)\in\mathcal{G}_x}$
end procedure

In the H2MGNN algorithm, we consider 3 types of time-dependent variables:

- Vertice latent variables, considering n vertices: $(h_i^v)_{i\in[n]}$
- Hyper-edge latent variables $\left(h^c_{e,m}\right)_{(c,e,m)\in\mathcal{G}_x}$
- Hyper-edge outputs $\left(\hat{y}^c_{e,m}\right)_{(c,e,m)\in\mathcal{G}_x}$

With the hyperparameter d setting the dimension of the latent variables. These latent variables are initialized with a flat start (zero values), whereas predicted output variables are set to values dependent of the given task.

Then, the H2MGNN algorithm runs through interactions of the time-dependent variables in the differential system with trainable mappings ϕ_θ, in a message-passing fashion similar to traditional GNN algorithms [17]:

$$\forall i \in [n], \quad \frac{dh_i^v}{dt} = \sum_{(c,e,m,o)\in\mathcal{N}_x(i)} \phi_\theta^{c,o}\left(t, h_e^v, h_{e,m}^c, \hat{y}_{e,m}^c, x_{e,m}^c\right) \tag{2.44a}$$

$$\forall (c,e,m) \in \mathcal{G}_x, \quad \frac{dh_{e,m}^c}{dt} = \phi_\theta^{c,h}\left(t, h_e^v, h_{e,m}^c, \hat{y}_{e,m}^c, x_{e,m}^c\right) \tag{2.44b}$$

$$\frac{d\hat{y}_{e,m}^c}{dt} = \phi_\theta^{c,y}\left(t, h_e^v, h_{e,m}^c, \hat{y}_{e,m}^c, x_{e,m}^c\right) \tag{2.44c}$$

With $\mathcal{N}_x(i)$ the set of hyper-edges connected to vertice i, c the hyper-edge class, and o the port of connection of hyper-edge (if hyper-edge connected to multiple ports).

Equation (2.44a) describes the update of vertices latent variable from the aggregation of messages from every connected hyper-edge. The transmitted messages depend on local information such as the time-dependent variables and the input features, as well as the time as global information. The trainable mapping is different for each hyper-edge class (to model different components' behavior) and port (to translate directions into the model, similar to directed graphs).

Then, equation (2.44b) defines the update of latent variable each hyper-edge, influenced by local values on itself and from connected vertices. As it depends on the same number of variables for all hyper-edges of a given class, the trainable mapping $\phi_\theta^{c,h}$ to update the message is also the same.

Finally, equation (2.44c) represents the update of predicted output variables influenced by local inputs, current local predictions, and latent variables on neighboring vertices. It is analogous to the update of the hyper-edge latent variable, with a single trainable mapping for a given class.

These equations describe the algorithm of the H2MGNN, and this architecture can be seen as a recurrent and residual GNN architecture, with trainable mappings implemented as fully-connected neural networks and trained through standard back-propagation.

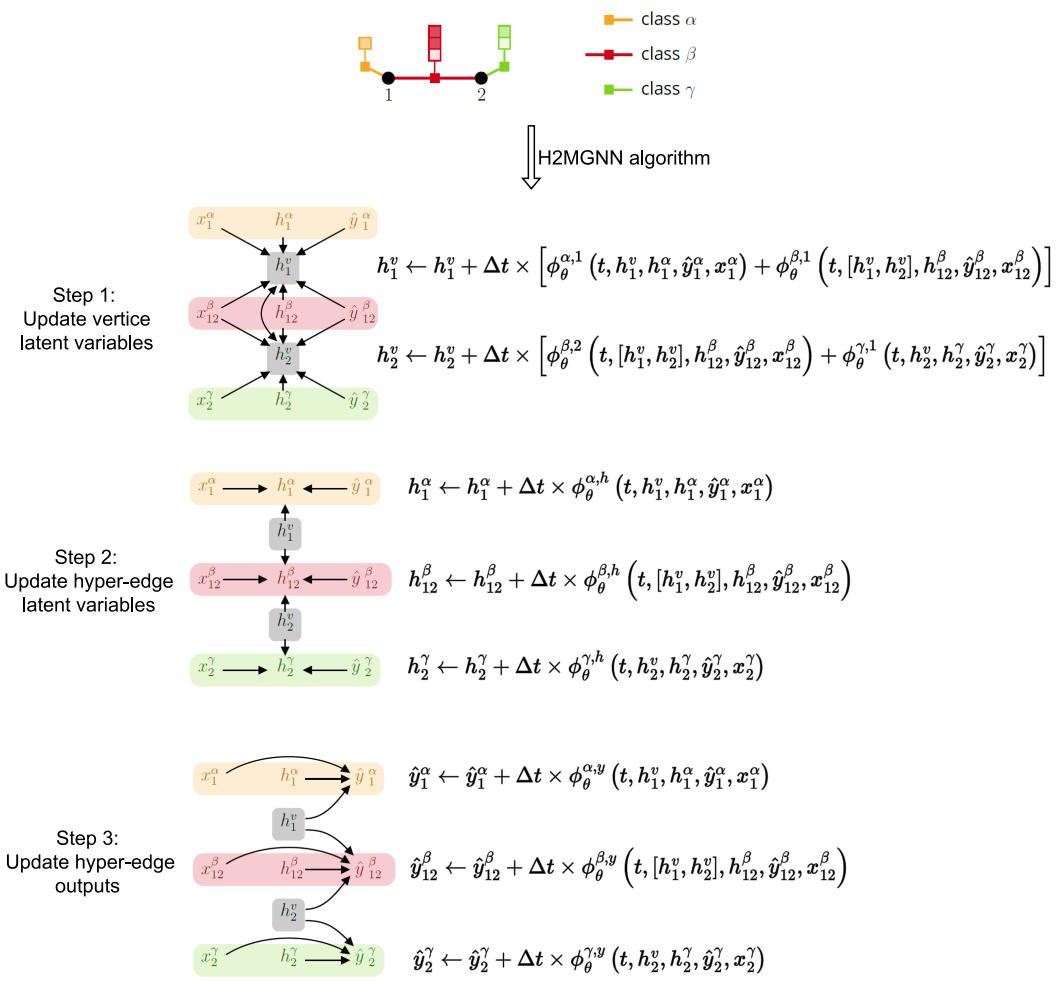

Figure 2.11: 3-class network example of H2MGNN [17]

2.3.3. Statistical Solver Problem (SSP)

The second component of the DSS architecture, the Statistical Solver approach, defines our training model and aims to minimize the optimization problem's cost function in an unsupervised fashion. This means that the train dataset $D_{train} = \{(x_m)\}_{m \in \mathcal{M}_{train}}$ does not contain any label y_m, and the data-points x are sampled on a probability distribution $p(x)$ instead of $p(x, y)$.

In the context of optimization in power systems, the main idea is to propose "an alternative solution to the classical Newton-Raphson solver using Neural Networks mapping" [17]. In the present case, the proposition is "to convert the target optimization problem as a learning problem". In this learning approach, we assume that our cost function $L(x, y)$, for a given $x \in \mathcal{X}$, has a unique minimum $y^*(x)$ such as:

$$y^*(x) = \underset{y \in \mathcal{Y}}{\arg\min}\, L(x, y) \tag{2.45}$$

In the Statistical Solver approach, we make a rephrasing in probabilistic terms of the optimization problem to use a density estimation method and get a statistical learning problem. In probabilistic terms,

given a sampling of inputs x following a probability distribution $p(x)$, we are interested in finding the conditional probability distribution $q(y|x)$ that connects problem instances x to their solutions, which can be defined as a Dirac measure located at $y^*(x)$:

$$q(y \mid x) = \delta_{y^*(x)}(y) \tag{2.46}$$

This conditional probability distribution is unknown. In our model, we consider a set of distributions $q_\theta(y \mid x)$ parameterized by $\theta \in \Theta$, and the goal of the training process is to find θ^* such that $p(x)q_{\theta^*}(y \mid x)$ is as close as possible to the unknown true generative process $p(x)q(y \mid x)$. This is equivalent to minimizing the Kullback-Leibler divergence between the two distributions:

$$D_{KL}\left(pq_\theta \| pq\right) := \int_{x\in\mathcal{X}} \int_{y\in\mathcal{Y}} p(x)q_\theta(y \mid x) \log\left(\frac{q_\theta(y \mid x)}{q(y \mid x)}\right) \tag{2.47}$$

The computation of the KL-divergence is not possible in our case, so [17] derived a theorem through an intermediate relaxation of the target distribution for which the KL-divergence can be written. The derived theorem states that it's possible to find a unique minimizer θ^* to solve the following learning problem:

$$\theta^* = \arg\min_{\theta\in\Theta} \mathop{\mathbb{E}}_{\substack{x\sim p(x)\\ y\sim q_\theta(y|x)}} [L(x,y)] \tag{2.48}$$

This forms the base of our Statistical Solver Problem, which can be expressed as: "given spaces $\mathcal{X}$ and $\mathcal{Y}$, distribution p defined over $\mathcal{X}$ and cost function L defined on $supp(p) \times \mathcal{Y}$, we wish to find the distribution $q_\theta(y \mid x)$ that minimizes the cost function L, when x follows $p(x)$."

The defined loss function can then be expressed as:

$$\text{Loss}(\theta) = \mathop{\mathbb{E}}_{\substack{x\sim p(x)\\ y\sim q_\theta(y|x)}} [L(x,y)] \tag{2.49}$$

And, given the facts that we rely on an empirical dataset $D = \{x_m\}_{m\in(M)_{train}}$, that we assume our cost function to be differentiable with regards to y, and that we choose $q_\theta(y \mid x)$ to be a deterministic mapping instantiated as a trainable H2MGNN function $\boldsymbol{f}_\theta(x)$, our loss function become:

$$\text{Loss}(\theta; D) = \frac{1}{|M|} \sum_{m\in M} L\left(x_m, \boldsymbol{f}_\theta\left(x_m\right)\right) \tag{2.50}$$

The loss function's gradient can be estimated using automatic differentiation, and our model can be trained using a standard gradient-descent method.

3

Methodology

3.1. WLS as target optimization problem for SSP

In [17], the author applied the DSS architecture to perform AC Power Flow analysis. The target optimization problem to learn was based on the minimization of flow imbalance in every bus:

$$\ell(x, y) = \sum_{e \in \mathcal{E}^{\text{buses}}} |\Delta s_e|^2 \tag{3.1}$$

In a similar approach, we can apply the DSS architecture to perform Distribution System State Estimation. In such a task, we can choose the traditional Weighted Least Squares algorithm as the target optimization problem to learn. The WLS approach is widely used in the industry and has good accuracy when given enough information. Therefore, it is interesting to use it as a target problem to reach similar performance while aiming at improving numerical stability, computation time, robustness, and observability requirements.

Our cost function to minimize becomes then:

$$\ell(z, x) = \sum_{i \in m} \frac{|z_i - h_i(x)|^2}{R_{ii}} \tag{3.2}$$

With $\boldsymbol{z}$ the set of provided measurements, $\boldsymbol{x}$ the state variables to find, m the number of measurements, $h(\boldsymbol{x})$ the measurement function and $\boldsymbol{R}$ the covariance matrix with $R_{ii} = \sigma_i^2$ representing the uncertainty of measurement i.

In this cost function, we define the measurement function $h(\boldsymbol{x})$ by the power flow equations, and its use allows our model to consider the physics of the distribution system in the training phase. This measurement function links the state variables, the voltage amplitude V_i and voltage angle φ_i at bus i,

to all measurements using the Power Flow equations. We derive the "predicted" measurements through this measurement function and compute the training's loss function by comparison with the input. We can define this approach as a semi-supervised learning approach, comparing the model's output to the noisy measurements instead of proper labels. This approach to *learn to optimize* is presented in Figure 3.1, together with the supervised approach and the WLS method to compare them directly.

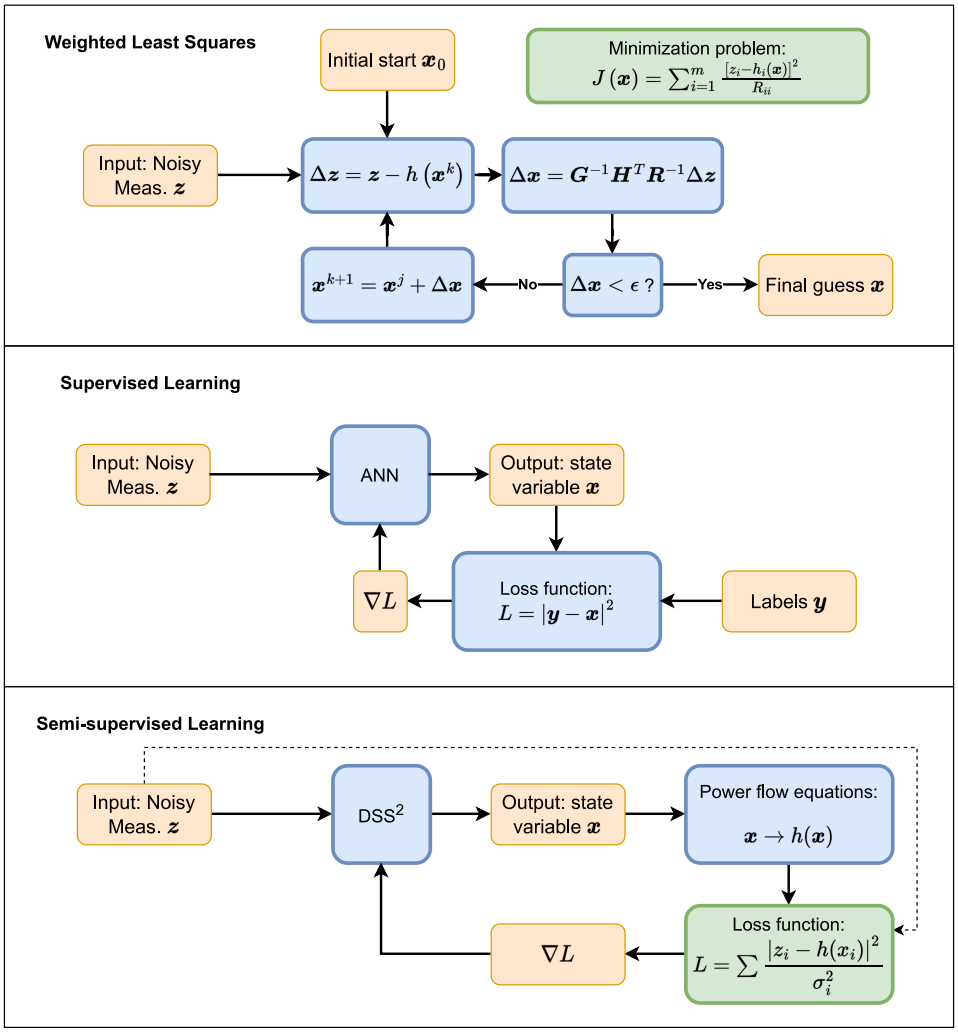

Figure 3.1: Comparison of the different approaches to perform State Estimation.

3.2. Loss function and Power Flow equations

In the previous section, we introduced the loss function used to train our model. This loss function contains the measurement function $h(\boldsymbol{x})$, which consists of the Power Flow equations and provides knowledge of the physics of the problem at hand to the model. Taking into account that $\boldsymbol{x}$ refers to the state variables:

$$\boldsymbol{x} = [V_0, V_1, \cdots, V_n, \varphi_0 = 0., \varphi_1, \cdots, \varphi_n] \tag{3.3}$$

With V_i the voltage amplitude at bus i, φ_i the voltage phase angle at bus i (instead of conventional

notation θ to avoid confusion with the neural networks trainable weights), and n the number of buses in the grid. This gives us $2n-1$ state variables (φ_0 is set to 0 by the convention of the slack). The measurement function, built upon the Power Flow equations, is defined as:

$$h(\boldsymbol{x}) = \begin{cases} V_i = V_i \\ \varphi_i = \varphi_i \\ P_{ij_\rightarrow} = -V_i V_j \left[\mathbb{R}(Y_{ij})\cos(\Delta\varphi_{ij}) + \mathbb{I}(Y_{ij})\sin(\Delta\varphi_{ij})\right] + V_i^2 \left[\mathbb{R}(Y_{ij}) + \dfrac{\mathbb{R}(Y_{s_{ij}})}{2}\right] \\ P_{ij_\leftarrow} = V_i V_j \left[-\mathbb{R}(Y_{ij})\cos(\Delta\varphi_{ij}) + \mathbb{I}(Y_{ij})\sin(\Delta\varphi_{ij})\right] + V_j^2 \left[\mathbb{R}(Y_{ij}) + \dfrac{\mathbb{R}(Y_{s_{ij}})}{2}\right] \\ Q_{ij_\rightarrow} = V_i V_j \left[-\mathbb{R}(Y_{ij})\sin(\Delta\varphi_{ij}) + \mathbb{I}(Y_{ij})\cos(\Delta\varphi_{ij})\right] - V_i^2 \left[\mathbb{I}(Y_{ij}) + \dfrac{\mathbb{I}(Y_{s_{ij}})}{2}\right] \\ Q_{ij_\leftarrow} = V_i V_j \left[\mathbb{R}(Y_{ij})\sin(\Delta\varphi_{ij}) + \mathbb{I}(Y_{ij})\cos(\Delta\varphi_{ij})\right] - V_j^2 \left[\mathbb{I}(Y_{ij}) + \dfrac{\mathbb{I}(Y_{s_{ij}})}{2}\right] \\ I_{ij_\rightarrow} = \left|\dfrac{P_{ij_\rightarrow} - jQ_{ij_\rightarrow}}{\sqrt{3}V_i e^{-j\varphi_i}}\right| = \dfrac{|P_{ij_\rightarrow} - jQ_{ij_\rightarrow}|}{\sqrt{3}V_i} \\ I_{ij_\leftarrow} = \left|\dfrac{P_{ij_\leftarrow} - jQ_{ij_\leftarrow}}{\sqrt{3}V_j e^{-j\varphi_j}}\right| = \dfrac{|P_{ij_\leftarrow} - jQ_{ij_\leftarrow}|}{\sqrt{3}V_j} \\ P_i = -\sum_{j\in\mathcal{N}_x(i)} P_{ij_\leftarrow} + P_{ij_\rightarrow} \\ Q_i = -\sum_{j\in\mathcal{N}_x(i)} Q_{ij_\leftarrow} + Q_{ij_\rightarrow} \end{cases} \tag{3.4}$$

With $\Delta\varphi_{ij} = \varphi_i - \varphi_j + \phi_{ij}$ the voltage angle difference across the line that connects bus$i$ to bus j, ϕ_{ij} being the shift angle of the transformer if any (modelling transformers as lines with added shift angle), and Y_{ij} and $Y_{s_{ij}}$ being respectively the line and shunt admittance of the line between bus i and bus j. In the output, we have $P_{ij_\rightarrow}$ and $Q_{ij_\rightarrow}$ as the active and reactive power flow in the line from bus i to bus j and measured at bus i, and $P_{ij_\leftarrow}$ and $Q_{ij_\leftarrow}$ as the power flow in the line from bus j to bus i and measured at bus i. Current flow I_{ij} follows the same convention. Finally, we derived from the power flows P_i and Q_i, which are the active and reactive power injections at bus i.

This loss function is differentiable in regards to the output state variables, and its gradient can be expressed using the measurement Jacobian matrix $H(x) = \frac{dh(x)}{dx}$ of the problem at hand. In practice, the gradient is computed using automatic differentiation.

3.3. Adding physical constraints to the loss function

During the early stages of training the DSS^2 model, we observed that, even with excellent voltage magnitude accuracy, the line loading accuracy could be terrible, and vice-versa. This behavior is due to the high complexity of the proposed loss function, which can be seen as a multi-objective function with high non-convexity. Some local minima of the objective function can include outputs with some very accurate variables, while others are outside the physical boundaries of the given problem.

As a method to reduce the number of local minima and constraints our outputs to physically-feasible areas, we set different constraints in the form of aggressive regularizers added to the loss function. In the current implementation, there are three different constraints, all based on the stability assumption:

- **Voltage level stability criteria** In order to remain stable, power systems need to ensure a voltage

level between 95% and 105% per unit. Most regulators enforce to stay within 97% and 103% as a safe stability margin. Therefore, a heavy two-sided ReLU penalization is added to the loss function to enforce this criterion when any voltage level output exceeds the 95-105% threshold.

- **Phase angle stability criteria** Power systems ensure stability and quality of power flows by keeping steady phase angles throughout the network. Whereas the phase angles vary by some degree, considerable variation in phase angles is improbable in stable systems. A phase angle difference of more than 15 degrees between two neighboring buses would characterize an unstable network. To enforce physically realistic phase outputs in the context of stable networks, we add a second heavy two-sided ReLU penalization to the loss function when the phase angle difference between two buses exceeds 15 degrees.
- **Line loading stability criteria** Power systems regulators ensure the network's security by applying safety margins to the loading in the lines. Even though power lines and cables can withstand overloading for a short period, regulators assume a line is *congested* when its loading is higher than 90% of its maximum capacity. To keep the model's outputs within physical range, we apply a third ReLU penalization on the line loading when the prediction gives a loading higher than 100%.

Adding these constraints to the loss function, the equation used in the training process becomes:

$$L(\boldsymbol{z}, \boldsymbol{x}) = \sum_{i \in m} \frac{\left|z_i - h_i(\boldsymbol{x})\right|^2}{R_{ii}} + \lambda[\mathsf{ReLU}(V - 1.05) + \mathsf{ReLU}(0.95 - V) + \mathsf{ReLU}(\text{ loading } - 100) + \mathsf{ReLU}(\varphi - 0.25) + \mathsf{ReLU}(-0.25 - \varphi)] \tag{3.5}$$

Where $\lambda \in [0; 1]$ is a hyperparameter set to balance the effect of the constraints during training.

These constraints enforce the model outputs to stay within physically plausible boundaries and to avoid diverging toward local minima of the loss function that are well beyond the physical margins of the system. Even though these constraints are wide enough to allow the model to predict weak, close to instability conditions of the system, they still rely on the assumption of stable systems. Therefore, the model will not perform well when asked to predict clear unstable states because it has been trained within stable margins. Specifically, the third constraint on line loading provides better outputs as it helps the model to find a reasonable minimum. However, it also limits the capabilities of the model to detect high overloading in the lines. Depending on the use case of the state estimation, a direction for future works would be to further specialize the model in performing well in near-instability conditions and detecting overloading.

3.4. H2MGNN implementation for State Estimation

The optimization task stated above describes the trainable H2MGNN function as an approximation of the inverse of the measurement function:

$$\boldsymbol{f_\theta}(\boldsymbol{z}) := \boldsymbol{h}^{-1}(\boldsymbol{z}) \to \boldsymbol{x} \tag{3.6}$$

Where $\boldsymbol{f_\theta}$ is the parameterized function defined by the model and that approximates the mapping from

input measurements z to output state variables x.

In the previous subsection, we showed that estimating accurately the voltage magnitude V at every bus and the voltage angle φ_i provides enough information to compute any significant variables of the power grid using the power flow equations. Using the H2MG architecture is very useful in such a case; it allows to model buses and lines separately and directly provides input features for both lines and buses. We can then compute the loss function with the power flow equations without any loss of information. The features and parameters assigned to each class of components are listed in Table 3.1, and a visual example of the current H2MGNN implementation is provided in Figure 3.2.

	Topology parameters	**Input features**	**Output features**
Buses	Bus port i Zero-inj. boolean $\mathbb{1}_{zero-inj.}$ Slack boolean $\mathbb{1}_{slack}$	Voltage magnitude: V_i - σ_{V_i} Voltage angle: φ_i - σ_{φ_i} Active power injection: P_i - σ_{P_i} Reactive power injection: Q_i - σ_{Q_i}	Voltage magn.: V_i Voltage angle: φ_i
Lines	Line ports i, j Closed line boolean $\mathbb{1}_{closed}$ Transformer boolean $\mathbb{1}_{trafo}$ Phase-shift ϕ_{ij}	Active power flow: P_{ij} - $\sigma_{P_{ij}}$ Reactive power flow: Q_{ij} - $\sigma_{Q_{ij}}$ Current magnitude: I_{ij} - $\sigma_{I_{ij}}$ Line admittance: $\mathbb{R}(Y_{ij})$ - $\mathbb{I}(Y_{ij})$ Shunt admittance: $\mathbb{R}(Y_{s_{ij}})$ - $\mathbb{I}(Y_{s_{ij}})$	

Table 3.1: Model's features and topology parameters. Buses and lines are explicitly modeled separately. Booleans are used to distinguish system's characteristics such as the slack and transformers, which are not modelled separately in this instance of the model. Every measurement is described by two features: the measured value and the underlying uncertainty σ. Complex admittances in the lines are separated in real and imaginary values.

The choice of input features is inspired by the WLS algorithm, where, for each measurement, we consider both the measured value and its underlying uncertainty to get enough information to estimate the global state of the system. Similarly, the output features are the voltage amplitude and angle of every bus, which is a common choice for the task of State Estimation. Other choices of output features can be considered in future work by using, for example, the branch currents, which might help improve the indirect predictions of line loading.

Even though it is possible in the DSS framework to model any power grid components, we simplify the problem in this thesis by only accounting for the lines and buses with a simplified model of transformers. We assimilate then generators and loads to buses' power injection, and we model transformers as lines with extra voltage angle shift. Moreover, while modeling them as either bus or line, some booleans are added to the topology parameters to distinguish other components (the slack, zero-injection buses, open lines, and transformers). Such simplifications are not impactful for the problem of Distribution System State Estimation that we are tackling here. However, complete modeling of the power grid, powered by the DSS framework, could be investigated in future works.

The implementation of the H2MGNN provides the last building block of the methodology for training the model. We summarize this training methodology in Figure 3.3.

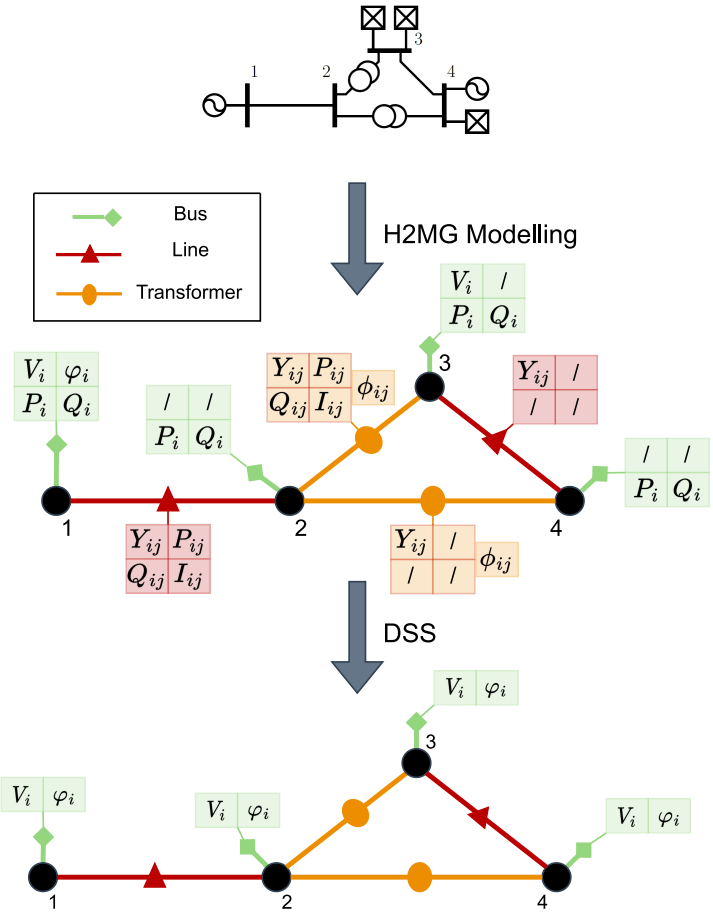

Figure 3.2: Simple example of the State Estimation task performed by the DSS model, following the H2MG architecture. Each class of components has its own characteristic input features. The output of the model is the voltage magnitude and angle at each bus. The symbol "/" in the input features defines a missing measurement, which is characteristic of the DSSE problem.

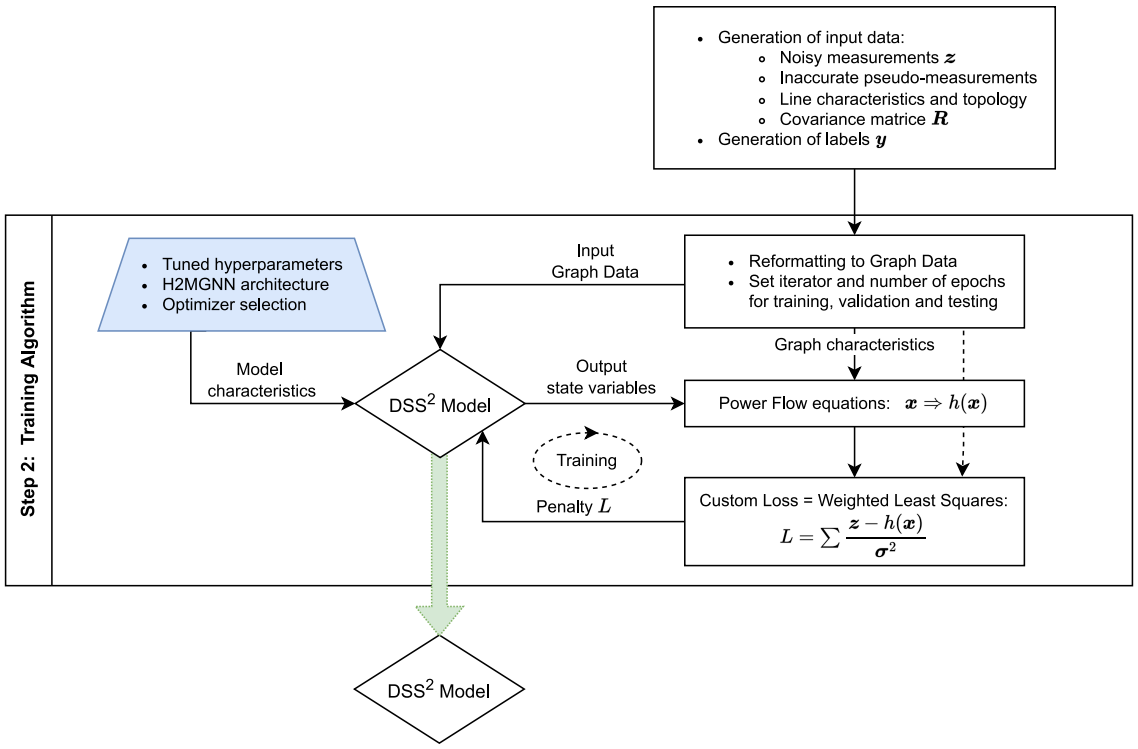

Figure 3.3: Training step of the DSS^2 algorithm

3.5. Implementation considerations

3.5.1. Pseudomeasurements as implicit regularizer

Learning the physics of the network through this training process depends on the available measurements z_i, and the model might not learn correctly for parts of the network with limited observability. However, using the covariance matrix R as weights allows intrinsically to use pseudomeasurements as a regularizer of our cost function, providing some boundary conditions in the grid.

Neglecting small deviations of relative uncertainty between measurements, we can approximate the standard deviation of any measurement i, for $i = 1, \cdots, m$:

$$\sigma_{r,meas_i} \approx \sigma_{r,meas} \tag{3.7}$$

Where σ_r denotes the approximated standard deviation. As pseudomeasurements carry way more uncertainty, we can distinguish the variance of measurements and pseudomeasurements such as:

$$\sigma^2_{r,pseudom.} > \sigma^2_{r,meas.} \Rightarrow \sigma^2_{r,pseudom.} = \frac{\sigma^2_{r,meas.}}{\lambda_{pseudom.}} \tag{3.8}$$

With $\lambda_{pseudom.} \in]0, 1[$ defined as the regularizer coefficient. With such a definition, we can redefine the cost function to highlight the regularizer effect of the pseudomeasurements:

$$\ell(z, x) = \frac{1}{\sigma^2_{meas.}} \left(\sum_{i \in meas.} |z_i - h_i(x)|^2 + \lambda_{pseudom.} \sum_{j \in pseudom.} |z_j - h_j(x)|^2 \right) \tag{3.9}$$

By applying pseudomeasurements of power injection to every non-observable bus, the regularizer effect stated above will enforce some range of output values in non-observable areas. At the same time, the model primarily learns from more accurate measurements in observable areas.

3.5.2. Interface between physical data and well-distributed data

To make our model train faster, we apply standard normalization as a data pre-processing step to standardize the data distribution and get a well-distribute range of values more easily distinguishable by the model. The statistical values used in these steps (mean and standard deviation) are calculated beforehand on a large set of samples, and fixed throughout the training, validation, and evaluation steps. Also, the hidden layers of the model include hyperbolic tangent activation function: latent values inside the model vary within a range of $[-1; 1]$. High bias in the last layer of the model can be caused if we enforce the model to provide non-normalized physical values as output, so we include a post-processing step to un-normalize the output data. Keeping normalized values throughout the model and un-normalize the output in a post-processing step limits bottlenecks and speeds up training. These data processing steps are presented in Figure 3.4.

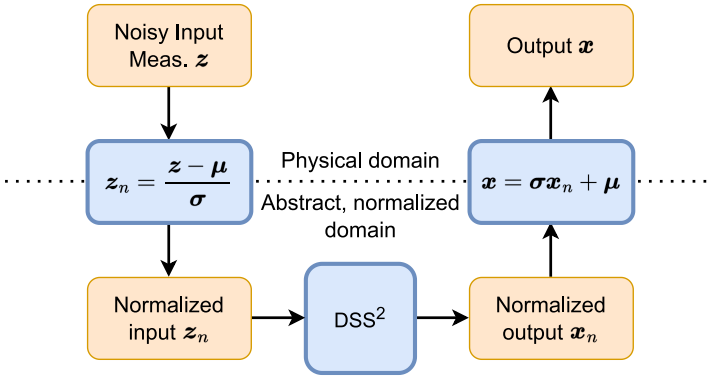

Figure 3.4: Data Processing steps

3.5.3. Improved MLP layers

The original DSS implementation by [17] uses a simple dense ANN layer in the MLPs used for message-passing, as defined in section 2.2.2. We improve these layers to increase the robustness and training abilities of the model by adding a dropout layer between each dense layer and an L2 regularizer in each dense layer.

The dropout layer is used during training to randomly ignore some outputs of the previous layer to increase the noisiness of the training process and make the nodes more robust to random noise. The dropout layer is designed with a hyperparameter called *dropout rate* r and set between 0 and 1, which defines the level of probability to drop a layer connection.

The l2 regularizer adds a penalty term on the squared of the trainable weights in the loss function. We use it to avoid overfitting by penalizing the loss function during training if the training weights get too important. We implement it in a dense layer and tune its impact with the *L2 coefficient* ℓ_2. We kept this value small to avoid the vanishment of important features.

These hyperparameters, together with the previously defined ones, are optimized through Grid search in the tuning process. The values to search in for the dropout rate are between 0.4 and 0.8, while the l2 coefficient is bounded between 0.002 and. 0.006. These boundaries are chosen following common implementations found in the literature.

4

Results

4.1. Scenario generation

To study the performance of the DSS^2 model, we generate different scenarios using Python's package PandaPower [33]. We chose three test networks for experiments and implementations: The 14-bus CIGRE Medium Voltage Distribution Network with PV and Wind DER activated [34], the 70-bus Oberrhein MV/LV sub-grid, and the whole 179-bus Oberrhein grid [33]. These networks are presented in Figures 4.1, 4.2, and 4.3.

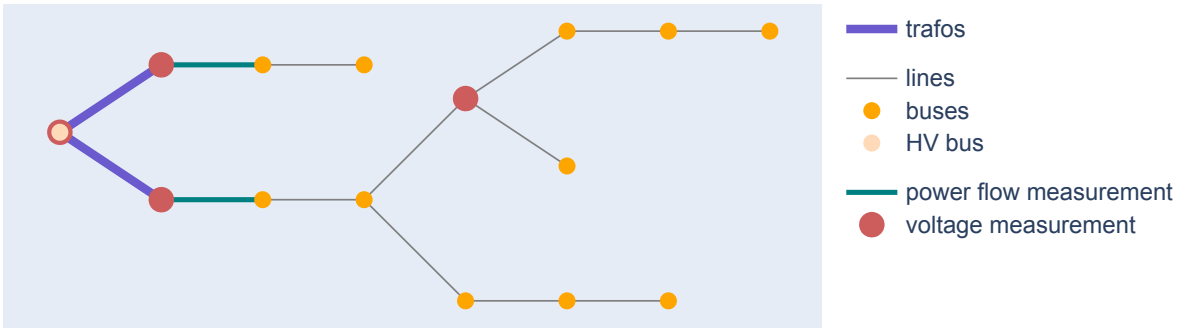

Figure 4.1: 14-bus CIGRE Medium Voltage Distribution Network [34], with arbitrary set of measurements chosen as default.

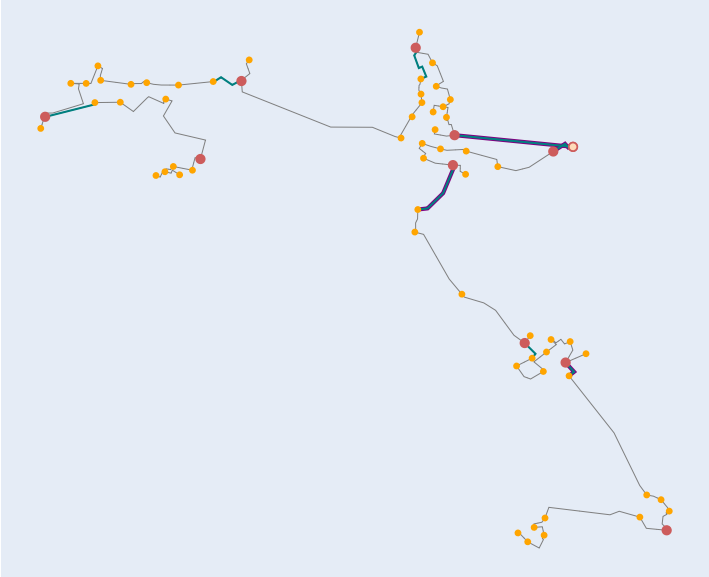

Figure 4.2: 70-bus Oberrhein MV sub-grid [33], with arbitrary set of measurements chosen as default.

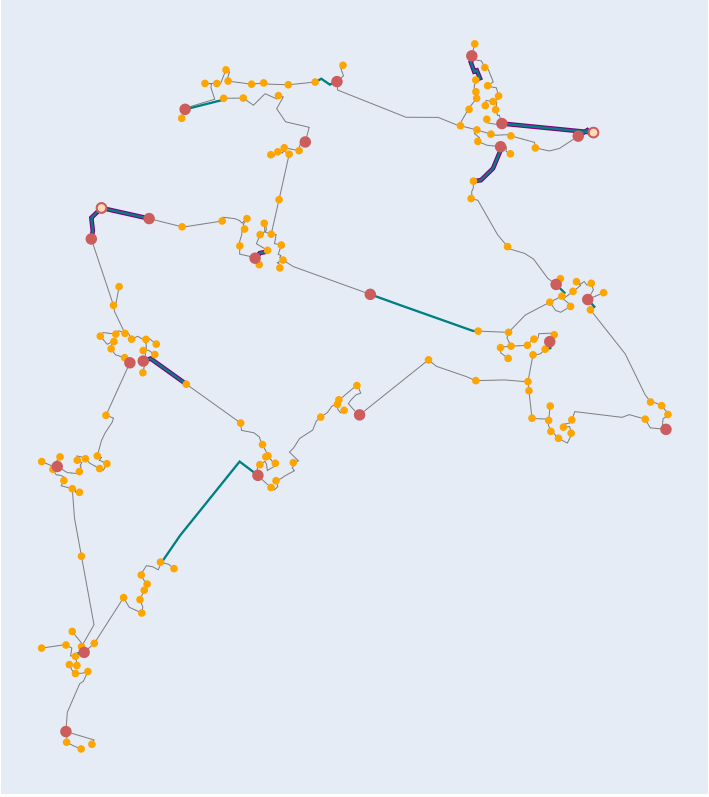

Figure 4.3: 179-bus Oberrhein MV grid [33], with arbitrary set of measurements chosen as default.

The objective is to compare the model's performance on these networks with the standard WLS algorithm and other DL architectures. To do so, we attribute load profiles to every load of the grids, some loads being labeled as residential and others as commercial/industrial. Profiles of generation are also set to the DER in the grid. The load and generation profiles, taken from [35], are shown in Figures 4.4

and 4.5.

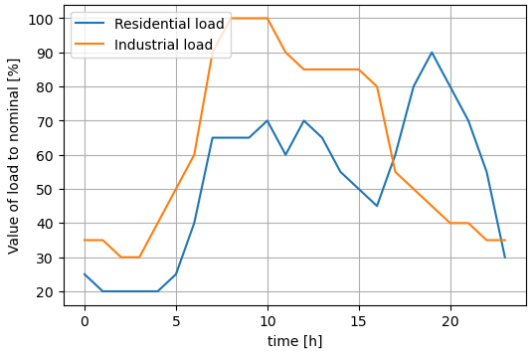

Figure 4.4: Typical load daily profiles for a sunny day [35]

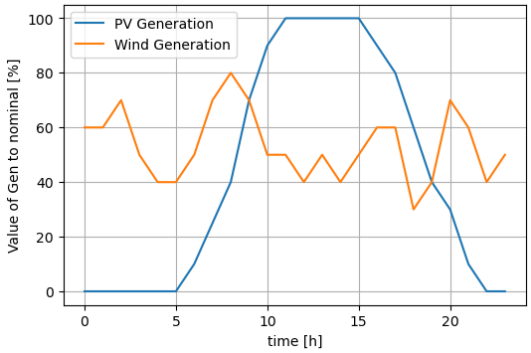

Figure 4.5: Typical generation daily profiles for a sunny day [35]

We are interested in monitoring a more dynamic and unstable grid, which is more insightful for comparison purpose. To get a more loaded and less stable grid, we multiply the nominal values of load by 1.2 and the nominal values of DER generation by 1.4 in the case of the CIGRE grid. In the Oberrhein networks, nominal loads are divided by two, and DER generation is multiplied by two to get more voltage variation in these networks. After getting daily load profiles, we use a Monte Carlo sampling to get random load scenarios. We use the load profiles as mean values and apply a standard deviation equal to 15% of the mean value for the sampling. The standard deviation used for sampling is high to model the high uncertainty of the loads. The process of getting load scenarios and pseudomeasurements is schematized in Figure 4.6.

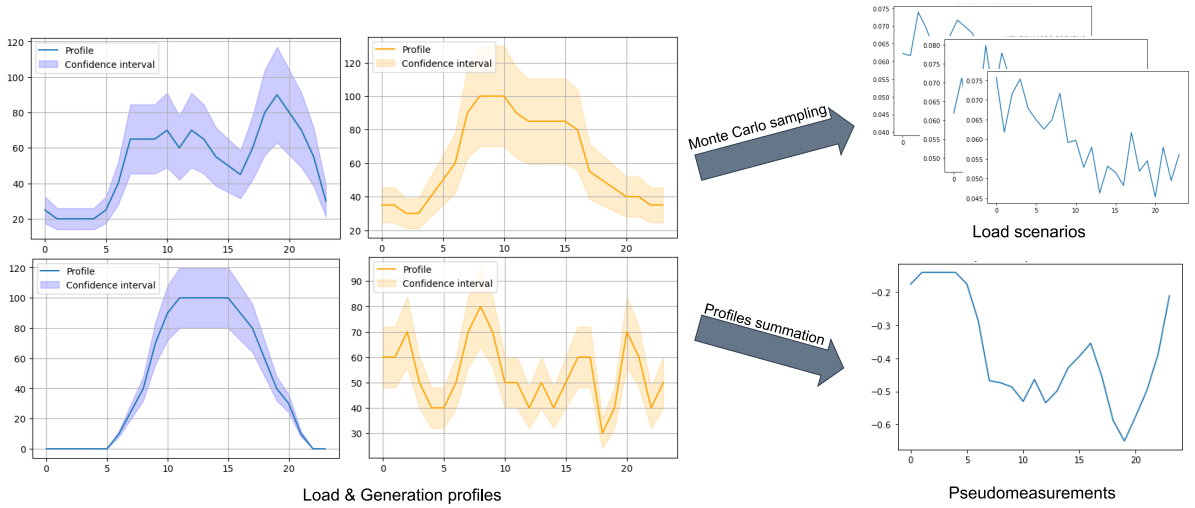

Figure 4.6: Scenarios and pseudomeasurements generation from load profiles. Pseudomeasurements per bus are directly provided by appropriate summation of profiles on each bus of the network. Monte Carlo sampling on the profiles provides random load scenarios on each bus.

To get data for training and testing, we perform AC Power Flow analysis with PandaPower on each network scenario. The results are stored either as measurement inputs (depending on the measurement sets) or state variables for labeling, as represented in Figure 4.7. The retrieved labels are used to test the state estimators and to perform supervised learning for comparison.

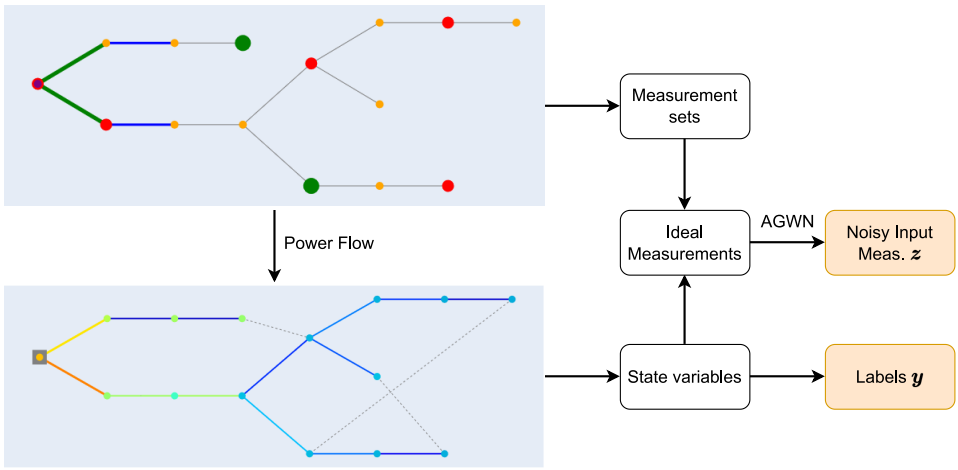

Figure 4.7: Generation of measurements in CIGRE 14-bus grid for input data and labels. Labels are gathered for supervised learning or as references in performance calculation. AGWN refers to Additive Gaussian White Noise, a zero-mean random value that we add on top of our measurement to simulate noisy data. To get a more complex noise, the standard deviation of this noise is proportional to the value of the measurement.

To get close to realistic scenarios and inputs, we add Gaussian noise to the results given by the Power Flow to get noisy data for the input measured values. Indeed, State Estimation is performed when assuming errors in the measurement devices, so we set a specific standard deviation for each kind of measurement, and normal distributions around the exact values are assumed. In such a setup, the input data is noisy, and we can assess the performance of the state estimators fed with such noisy data using the labels, which are the exact output values of the Power Flow solver. We use different sets of measurement uncertainty to analyze the robustness to noise of the state estimators. In this thesis, we arbitrarily choose a standard deviation proportional to the mean value. Further work should include an analysis of robustness to different kinds of uncertainty.

As stated in earlier chapters, the distribution system is characterized by a lack of observability, and pseudomeasurements usually provide extra information to the state estimator. In our case, we define pseudomeasurements by the load and generation profiles shown above in Figure 4.4, where we use the profiles as an estimation of power injection in every unobserved bus.

This data generation process is the first step of the framework presented in this thesis. The second step of the framework consists of the training methodology described in the previous chapter. The last step of this framework is to evaluate our model using the generated labels, as presented in the following sections. The overall framework presented step by step is summarized in Figure 4.8.

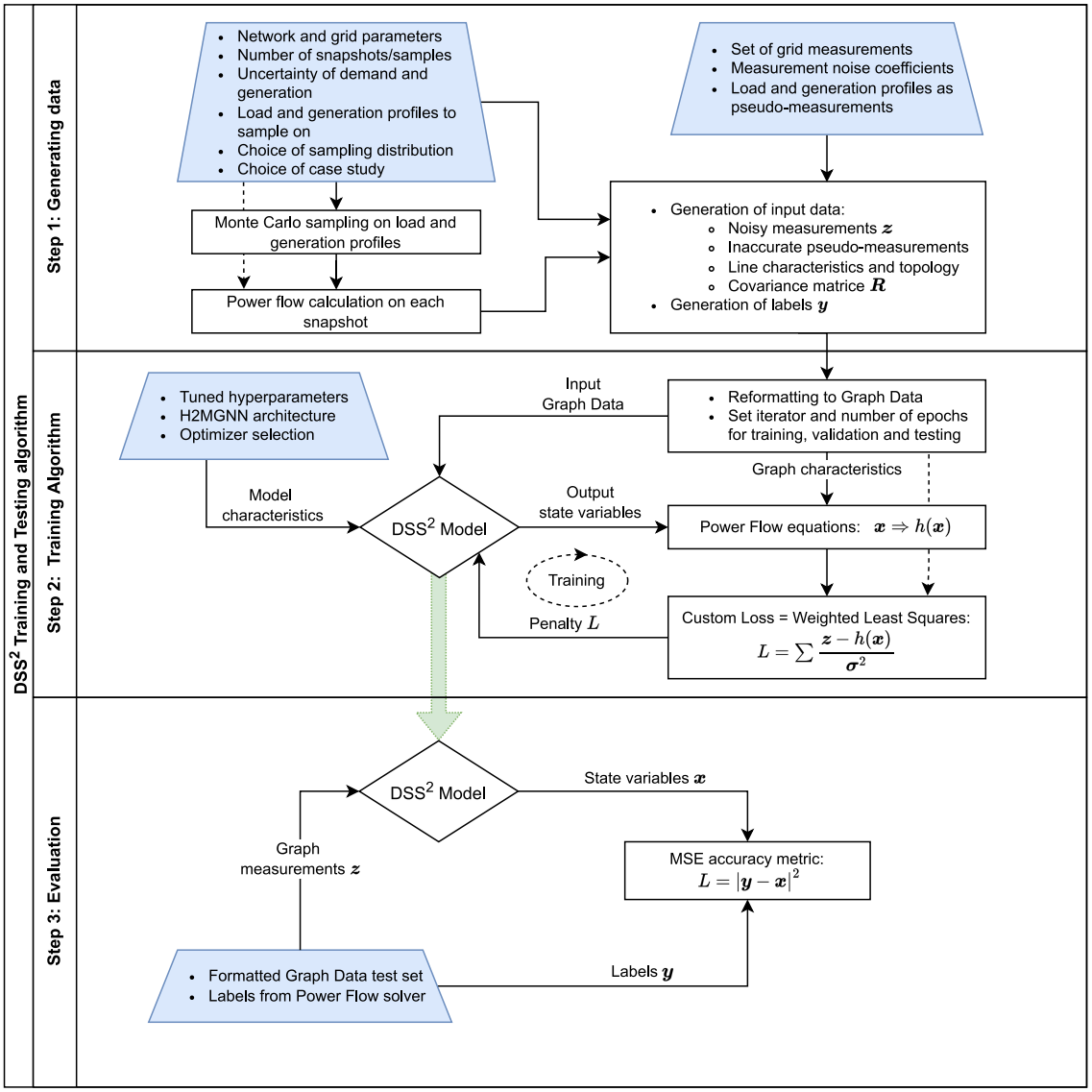

Figure 4.8: DSS^2 framework summary. First step consists in generating the datasets as explained in section 4.1, then the training methodology explained in section 3.4 and shown in Fig 3.3 is used as step 2. The final step consists in evaluating the model, which is done in the following sections.

4.2. Defining the case studies

In the introductory chapter, we presented different research sub-questions to determine a research direction and analyze the contributions of the present thesis. These sub-questions are:

- How well can the Deep Statistical Solver architecture perform the Distribution System State Estimation task compared to the traditional Weighted Least Square approach?
- How accurate is the estimation provided by the semi-supervised DSS^2, compared to supervised models?
- What are the advantages provided by the DSS architecture?

To validate the DSS^2 model, we design and present different case studies to answer these research sub-questions in this section.

We answer the first sub-question by comparing the DSS^2 model to the traditional WLS method for State Estimation. The objective is to show that the developed model can perform at least as well as the WLS algorithm while successfully tackling the main problems of the WLS approach: low convergence rate, sensitivity to measurements/pseudomeasurements errors and weights, and time consumption. This first case study will be performed on the three distribution networks: the CIGRE 14 bus MV grid, the Oberrhein 70-bus sub-grid, and the Oberrhein 179-bus MV grid.

To answer the second sub-question, we compare the DSS^2 model to two models trained through supervised learning: a standard NN, the Feed-Forward Neural Network (FFNN), and the same DSS^2 architecture. We expect the model trained in a supervised approach to perform better in normal conditions; the objective is, therefore, to assess the loss of performance when training our model without knowing any labels. We also compare our model to a supervised DSS^2 to see the difference of performance from the unique change of training, and not the architecture. Adding this model also provides insight on the advantages of the DSS architecture compared to a standard FFNN, to answer the third sub-question.

Finally, We assess further the third sub-question by comparing the DSS^2 model to a more traditional GCN architecture, both trained in a semi-supervised fashion. We expect the DSS^2 model to perform better, confirming the relevance of such architecture.

In this chapter, we use different metrics to fully assess our model's accuracy and compare it with the WLS algorithm and the other comparative models: RMSE, RMSE%, and MAE. The RMSE highly penalizes big errors and assesses the quality of predictions *on average*. The RMSE% provides insight into normalized output, and the MAE assesses the quality of median prediction without penalizing more large errors. This combination of metrics provides enough information to discuss the performance of the models.

We consider different measurement sets and errors representing different case studies to assess the robustness to measurement uncertainty and observability. For the CIGRE grid, these measurement sets are summarized in Table 4.1, and the different considered uncertainties are shown in Table 4.2.

	V amp.	V angle	PQ bus injection	PQ line flow	I line flow
Set 1	0,1,12,8	No bus	0	1-2, 12-13	No line
Set 2	No bus	No bus	0,1,12	1-2,12-13	1-2, 12-13

Table 4.1: Sets of measurements used as input for state estimators

	V meas. error	I meas. error	PQ meas. error	PQ pseudomeas. inaccuracy
Standard set	1%	1%	2%	10%
Worse case set	3%	3%	5%	15%
Better case set	0.5%	0.5%	1%	7.5%

Table 4.2: Sets of measurement uncertainty, characterized by a standard deviation proportional to mean value (% of mean value)

The main measurement set is set 1, represented in Figure 4.9. This set has been designed arbitrarily to have more observability in the upstream and an unobservable area in the downstream; however, real networks usually have a lower observability than the present case. We include measurement errors as input features of the model, and all three cases of measurement errors are included in the training data to learn from measurements with different weights.

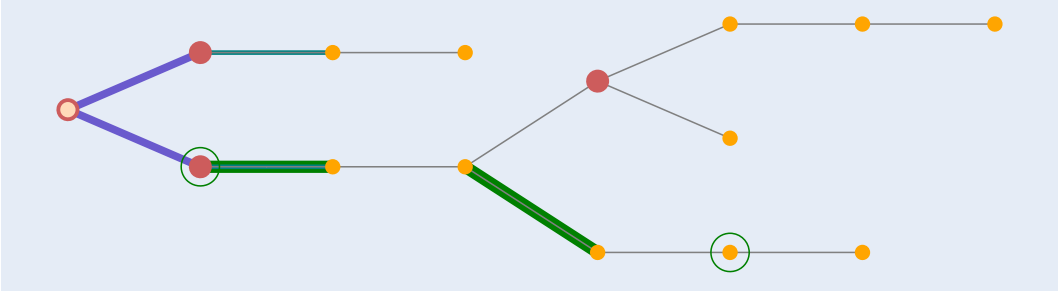

Figure 4.9: 14-bus system with measurement set 1. Geographic coordinates of the buses are normalized for ease of reading. Lines and buses chosen for analysis in section 4.4 are highlighted in green.

To compare the models, we develop specific case studies to assess the model's performance in certain conditions. The characteristics of these case studies for all networks are summarized in Table 4.3.

In this setup, we define wrong measurements as measurements with a higher deviation from the true value than the expected deviation. Also, we replace missing values with the empirical mean value of that specific measurement.

The first three case studies assess the model's performance for different measurement errors. The fourth one assesses the accuracy when using another set of measurements (which was not part of the training set). Then, different events known to be harmful for the state estimation, such as wrong values of measurements and missing measurements, are analyzed in the following four case studies. Finally, the last case studies aim to assess the robustness of the model when varying the overall load and

	Meas. set	Error set	Specific events CIGRE	Specific events Ober. grids
CS1	Set 1	Default	/	/
CS2	Set 1	Bad	/	/
CS3	Set 1	Good	/	/
CS4	Set 2	Default	/	/
CS5	Set 1	Default	Wrong meas. of P_{12-13}	Missing meas. V_{39}
CS6	Set 1	Default	Wrong meas. of V_4 and V_8	Wrong meas. of V_{58}, V_{39} and V_{80} Wrong meas. of P_{39-80} and P_{39-86}
CS7	Set 1	Default	Missing meas. V_4 and V_8	Wrong meas. of V_{58} Missing meas. V_{39} and V_{80}
CS8	Set 1	Default	Wrong meas. of V_8 Missing meas. V_4	DER decreased by 30% Load increased by 30%
CS9	Set 1	Default	DER increased by 20% Load increased by 15%	DER increased by 25% Load increased by 100%
CS10	Set 1	Default	DER decreased by 40%	DER decreased by 75% Load increased by 60%

Table 4.3: List of case studies for each grid. Similar case studies are used for the 70-bus and the 179-bus networks. Measurement (for the CIGRE grid) and error sets can be found in Tables 4.1 and 4.2.

generation distribution to observe how well it behaves when getting sensibly different input distributions. For each case study, we create 120 samples representing different load scenarios and compute the accuracy of our model on these samples.

We use a similar methodology of evaluation for the other networks. Measurement 1 for the 70-bus grid, used as the main set of comparison, is shown in a re-scaled format in Figure 4.10 (instead of a table, for ease of visualization). The measurement set 1 for the 179-bus was already presented in default form in Figure 4.3. These sets already provide both networks with more observability than in real cases. However, the measurements are scarce enough to see the performance under low observability in bigger grids while providing enough information.

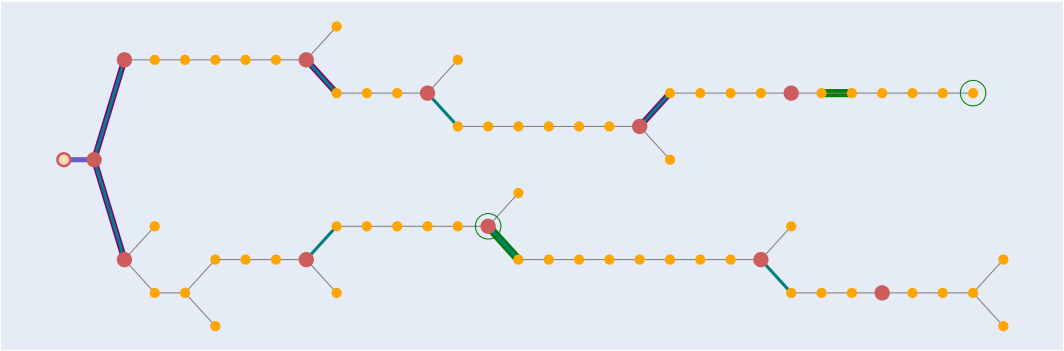

Figure 4.10: 70-bus system with measurement set 1. Geographic coordinates of the buses are normalized for ease of reading. Lines and buses chosen for analysis in section 4.5 are highlighted in green.

4.3. Training and tuning

For each network, 8640 samples are collected, equivalent to one year of hourly data. The dataset is split into train, validation, and test sets, following an 80/10/10 split (80% training, 10% validation, and 10% testing).

The loss function developed in this thesis is complex, highly non-convex, and non-linear. It makes the training process long and tedious, leading to a high amount of epochs to train our model. To keep track of the training process, we train the model per iteration of 30 epochs and save it after each iteration to create checkpoints and avoid overfitting.

During the training process, we could observe that after a certain point in the training, there is a trade-off between accuracy in bus voltage and line loading. Indeed, it was observed that the measurement weights apply to the loss function impact the training process, similarly to hyperparameters. When tuning these measurement weights, different scenarios have been observed:

- **Too high weights on voltage errors** The model's training converges rather fast. The accuracy on bus voltage is very high, but inferior performance is observed for the line loading.
- **Too high weights on power flow errors** The training process is slower and converges toward a high accuracy for the line loading, but also toward a high error in voltage level.

Moreover, the trade-off between voltage and line loading accuracy was not linear with the weights. Some weights balancing performed very poorly, while other mid-balancing provided better results.

The hyperparameter tuning process used in this thesis is kept simple due to time constraints. Several hyperparameters are fixed to some values following efficiency assumptions, and others are tuned using a Grid search approach. The complete list of chosen hyperparameters for each network is summarized in Table 4.4.

	epochs	λ	α	**batch size**	d	**layers**	Δ**t**	r	ℓ_2
14-bus CIGRE	630	0.8	0.006	64	40	3	$\frac{1}{7}$	0.4	0.002
70-bus MV	540	0.8	0.006	64	40	3	0.05	0.4	0.002
179-bus MV	1020	0.8	0.006	64	40	3	0.04	0.4	0.002

Table 4.4: List of hyperparameters for each network, with: the constraints coefficient λ, the learning rate α, the hidden layer size d, the time step Δt, the dropout rate r, and the L2 regularizer coefficient ℓ_2

These hyperparameters were chosen based on the satisfying performance observed. However, the search for a satisfying set of hyperparameters was not straightforward: numerous iterations were made during the search, and the model was highly sensitive to the choice of hyperparameters.

4.4. Performances comparison on the 14-bus CIGRE grid

In this section, we compare the performance of the DSS^2 model on the CIGRE grid to 3 alternatives: the standard SE WLS algorithm, an FFNN model trained with supervised learning, and the same DSS^2 model but trained with supervised learning. The objective is to analyze the viability of the semi-supervised DSS^2 to be an alternative to traditional WLS and to discuss the trade-off between accuracy and the need for labels by comparing it to supervised models.

4.4.1. Estimation accuracy in normal conditions

A summary of the results for the case study 1 (normal conditions) is shown in Table 4.5.

	DSS^2	**WLS**	**FFNN**	**DSS^2 sup.**
Convergence [%]	100	100	100	100
Duration [ms]	5.5	86.3	**3.53**	4.72
Voltage level RMSE [-]	3.4 x 10^{-3}	9.9 x 10^{-3}	2.7 x 10^{-3}	**2.5** x 10^{-3}
Voltage level RMSE% [%]	0.348	0.998	0.277	**0.254**
Voltage level MAE [-]	2.6 x 10^{-3}	7.1 x 10^{-3}	2.4 x 10^{-3}	**1.9** x 10^{-3}
Line loading RMSE [%]	8.02	**4.55**	39.39	14.32
Line loading RMSE% [%]	167.14	**25.53**	787.05	284.29
Line loading MAE [%]	6.58	**3.18**	38.5	11.05
Loading w/out trafos RMSE [%]	3.84	**3.4**	41.98	12.7

Table 4.5: DSS^2's performance on case study 1 (normal condition) in CIGRE grid compared to the WLS algorithm, a supervised FFNN and a supervised DSS^2.

First of all, we observe that the WLS algorithm converges in normal conditions, and no advantages are brought by the DL models there. However, we can see a high gain in computation speed using a DL approach, with the DSS^2 being more than 15 times faster than the WLS algorithm to perform a state estimation. This improvement is expected as we alleviate the need for an iterative process using the DL tools. The simple FFNN architecture has the highest speed of computation, although the speed of computation of the DSS^2 architecture is in the same order of magnitude.

Then, regarding the accuracy in voltage level, we can observe that the DL models all outperform the WLS algorithm. This observation is expected for the supervised model trained with labels, but the semi-supervised DSS^2 also shows outstanding performances, with an RMSE three times smaller than the WLS. Moreover, the loss of accuracy compared to the supervised models is negligible.

Finally, we can notice a reasonable estimation of the line loading by the semi-supervised DSS^2. Transformers loading aside, the DSS^2 reaches performances equivalent to the WLS algorithm and outperforms the supervised models by a wide margin. However, we also notice a high RMSE% from the DL models compared to the WLS algorithm. This difference shows a poorer accuracy of these models when estimating low line loading values, which significantly impact the RMSE%.

The line loading estimation is indirect; we pass the outputs of the model (voltages magnitude and angles) through the power flow equations to get the current flows. A coupling between the model outputs must then be satisfied to get accurate line loading estimations. The semi-supervised approach shows a more accurate coupling of outputs than other supervised DL models, and the supervised DSS^2 performs better than the FFNN in such a task.

We can further analyze the performances by comparing the estimation error on each grid component. The voltage accuracy on each bus is shown in Figure 4.11, and the loading accuracy in each line and transformer is presented in Figure 4.12.

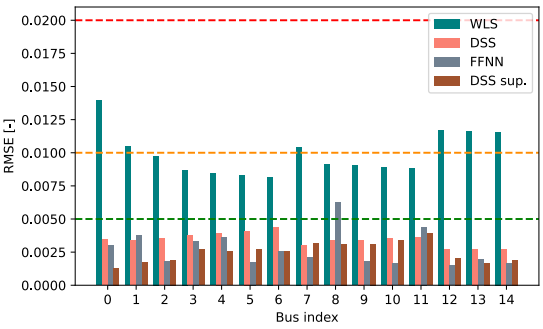

Figure 4.11: RMSE in voltage level estimation. Index 0 represents the slack.

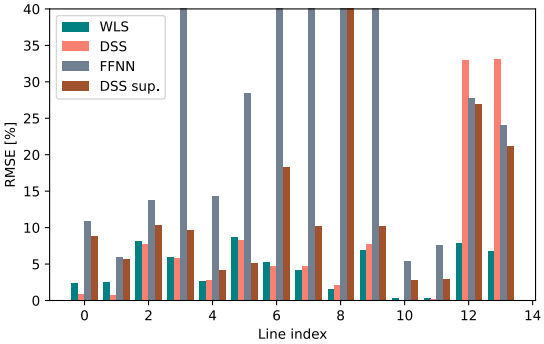

Figure 4.12: RMSE in line loading estimation. Indexes 12 and 13 represent the transformers of the grid.

The accuracy in voltage estimation follows the same trend as the average presented in Table 4.5, with the supervised models being the most accurate and the WLS algorithm showing the worst accuracy. For line loading accuracy, it is interesting to observe similar accuracy in every line for both the DSS^2 and the WLS. However, a clear drop in performance is noticed for the estimation of transformers' loading, shown at indexes 12 and 13.

To complete the analysis of performances in normal conditions, we can investigate the evolution of estimation throughout the sampling period in different components and compare it to the real value from the power flow. The estimation of voltage level for the measured bus 1 is shown in Figure 4.13, and for the unmeasured bus 5 in Figure 4.14. The line loading estimation for the measured line 0 is presented in Figure 4.15, and for the unmeasured line 2 in Figure 4.16. Neither unmeasured components have a measurement in their direct neighborhood.

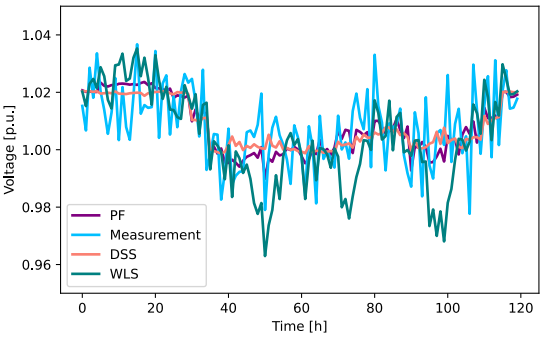

Figure 4.13: Estimation of voltage in the measured bus 1. PF represents the real value.

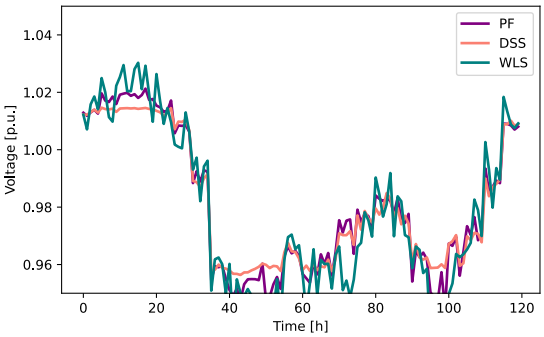

Figure 4.14: Estimation of voltage in the unmeasured bus 5. PF represents the real value.

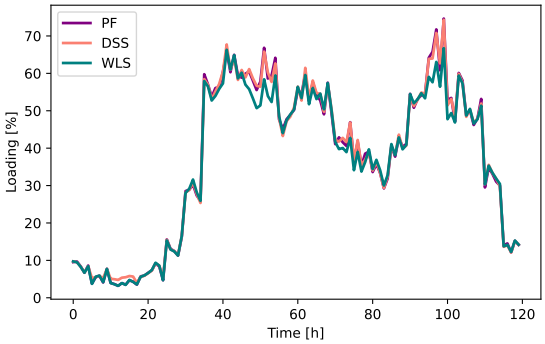

Figure 4.15: Estimation of loading in the measured line 0. PF represents the real value.

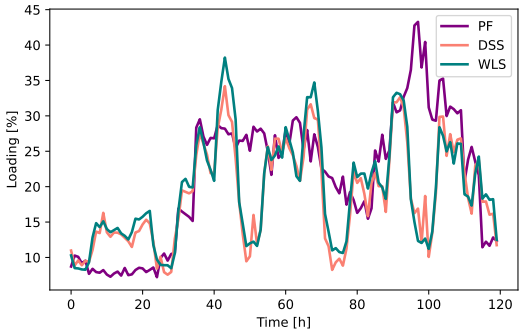

Figure 4.16: Estimation of loading in the unmeasured line 2. PF represents the real value.

We can first observe remarkable performances of the DSS^2 for the voltage estimation, outperforming the WLS in canceling measurement noise and estimating unmeasured buses. For the line loading, both methods estimate the loading in line 0 very accurately, but both fail to estimate the loading in line 2 accurately. Interestingly, the inaccurate estimation of the loading in line 2 follows the same oscillating behavior for both models.

4.4.2. Analysis of robustness

We presented in Table 4.3 different case studies to compare the robustness of the models to measurement noise, measurement error, and changes in load distribution. Figures 4.17 and 4.18 present the performance of the models for each of them. Case study 4 misses the results for the WLS algorithm, as it could not converge with the measurement set 2 due to the inclusion of current flow measurements, which are poorly handled by the algorithm. The WLS algorithm had a convergence rate above 90% in all other case studies on the CIGRE grid.

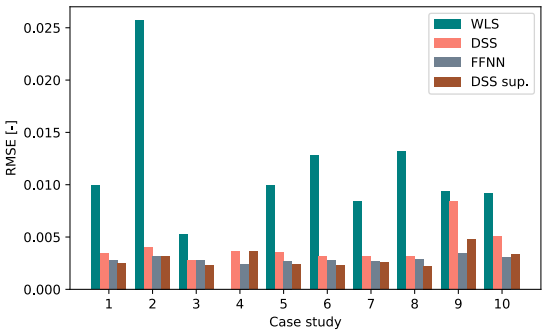

Figure 4.17: RMSE in bus voltage estimation for each case study presented in Table 4.3.

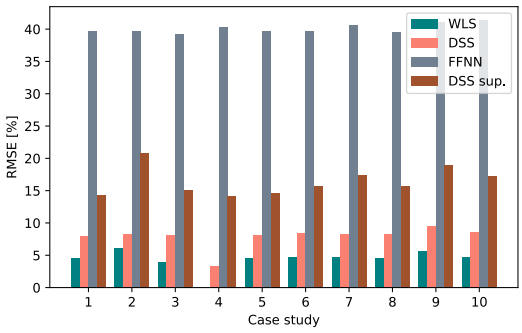

Figure 4.18: RMSE in line loading estimation for each case study presented in Table 4.3.

We can first observe strong robustness from the DSS^2 model to measurement noise and error in these results. The DSS^2 shows a more accurate voltage estimation than the WLS in every case study and is less impacted by events. Especially the increase in measurement noise (case study 2), where the WLS accuracy decreases drastically more than the DSS^2. However, the DSS^2 is more impacted by changes in load distribution (case studies 9 and 10), although the accuracy in voltage remains higher than the WLS.

The accuracy in line loading is less impacted by the random events than the voltage level. Nevertheless, we can first notice that the semi-supervised DSS^2 shows an increased accuracy in case study 4, positively using the current flow measurements included in the measurement set 2. Moreover, the semi-supervised DSS^2 is more robust to measurement noise and error than its supervised version.

4.5. 70-bus and 179-bus MV networks

4.5.1. Estimation accuracy in normal conditions

In this section, we compare the performance of the DSS^2 and the WLS when scaling up the network. The supervised models are left out of this comparison. The performances in normal condition (case study 1) are summarized in Table 4.6 for the 70-bus network, and in Table 4.7 for the 179-bus network.

First, we can notice a drop in convergence rate from the WLS algorithm when scaling up the model and the number of measurements. The convergence rate drops to 25% in the 70-bus grid when using all the measurements from set 1, and it does not converge at all in the 179-bus. To allow comparison without convergence issues, we compare the DSS^2 in the 70-bus system to a WLS algorithm only fed with voltage measurements and pseudomeasurements to increase its convergence rate. This workaround did not increase the convergence of the WLS in the 179-bus, so no further comparison is possible on this network.

Then, we can see the DSS^2 drastically outperforming the WLS algorithm in every metric in both networks. While tackling convergence and speed issues, the DSS^2 also shows outstanding accuracy in bus voltage and line loading estimations. The difference between RMSE and RMSE% in line loading remains big: higher loadings are better estimated by the DSS^2 than lower values.

	DSS^2	**WLS**	**WLS w/out P,Q,I meas.**
Convergence [%]	**100**	25	100
Duration [ms]	**22.1**	122.77	74.6
Voltage level RMSE [-]	**1.5** x 10^{-3}	3.14 x 10^{-2}	6 x 10^{-3}
Voltage level RMSE% [%]	**0.15**	3.01	0.59
Voltage level MAE [-]	**1.3** x 10^{-3}	3.11 x 10^{-2}	4.8 x 10^{-3}
Line loading RMSE [%]	**2.57**	39.3	27.92
Line loading RMSE% [%]	**133.48**	378.55	232.63
Line loading MAE [%]	**2.2**	34.88	20.15
Loading w/out trafos RMSE [%]	**2.29**	17.3	16.07

Table 4.6: DSS^2's performance on the Oberrhein 70-bus sub-grid compared to the WLS algorithm. An instance of the WLS algorithm not fed with flow measurements is added to increase its convergence rate and compare further.

	DSS^2	**WLS**
Convergence [%]	**100**	0
Duration [ms]	58.2	-
Voltage level RMSE [-]	2.3 x 10^{-3}	-
Voltage level RMSE [%]	0.227	-
Voltage level MAE [-]	1.9 x 10^{-3}	-
Line loading RMSE [%]	3.47	-
Line loading RMSE% [%]	362.36	-
Line loading MAE [%]	2.7	-
Loading w/out trafos RMSE [%]	3.39	-

Table 4.7: DSS^2's performance on the Oberrhein 179-bus grid compared to the WLS algorithm.

We further analyze the performance of the DSS^2 model when scaling up to the 70-bus network by looking at the estimation of specific components through the sampling period, as shown in Figures 4.19 to 4.22. We can first notice accurate estimations of measured components by the DSS^2, successfully canceling measurement noises. Using the power flow measurement, as in line 60, also provides better results compared to the WLS. Secondly, the DSS^2 also shows better performance when estimating the voltage in unmeasured, remote buses such as bus 223. However, the model tends to underfit with a straight line the estimation of these buses. For the estimation of unmeasured lines, the performance of neither model is satisfying. Contrarily to the estimations of unmeasured lines in the 14-bus network, the models do not show similar behavior in this case. Although, both models show an oscillating pattern through the sampling period.

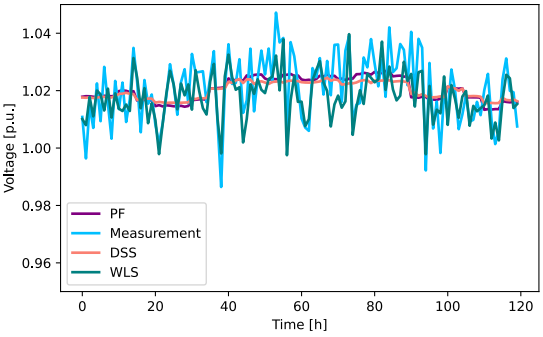

Figure 4.19: Estimation of voltage in the measured bus 34. PF represents the real value.

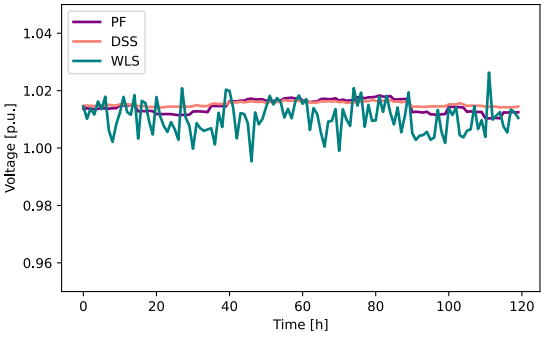

Figure 4.20: Estimation of voltage in the unmeasured bus 223. PF represents the real value.

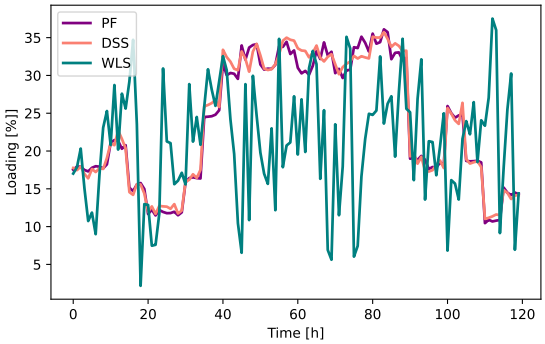

Figure 4.21: Estimation of loading in the measured line 60. PF represents the real value.

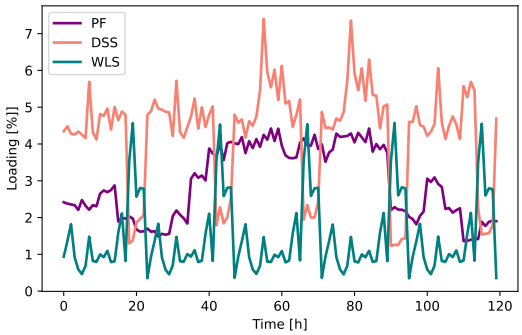

Figure 4.22: Estimation of loading in the unmeasured line 68. PF represents the real value.

4.5.2. Analysis of robustness

The performances of the models for each case study presented in Table 4.3 are shown in Figures 4.23 and 4.24, for the 70-bus network. Similar to the analysis made on the 14-bus grid, the DSS^2 shows great robustness to measurement noise and error. However, a significant drop in performance is observed for case studies 4, 9, and 10. Whereas it is interesting to see some robustness from the model in case study 8 (small variation of load distribution), we notice a clear lack of generalization from the model when modifying the measurement set or changing the load distribution more importantly.

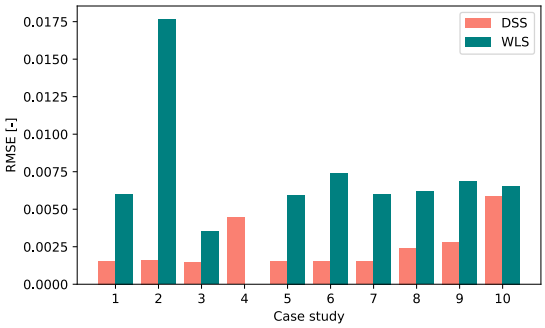

Figure 4.23: RMSE in bus voltage estimation for each case study presented in Table 4.3.

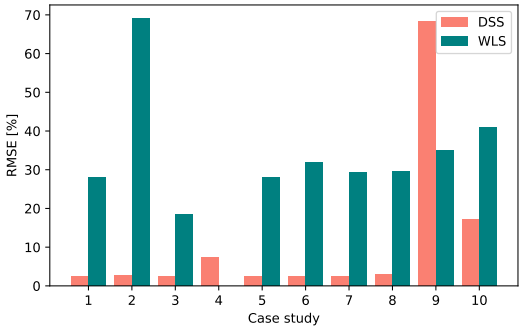

Figure 4.24: RMSE in line loading estimation for each case study presented in Table 4.3.

4.6. Comparison to Graph Convolutional Neural Network

To compare the DSS^2 model to a Standard GCN trained with the semi-supervised methodology, we attempted to build a GCN model using the Spektral library[36]. However, this model did not train successfully with the semi-supervised methodology and thus did not perform the estimation task successfully. The difficulties encountered during training can be observed in Figures 4.25 and 4.26, where the fast decrease in training loss does not couple with the accuracy in voltage.

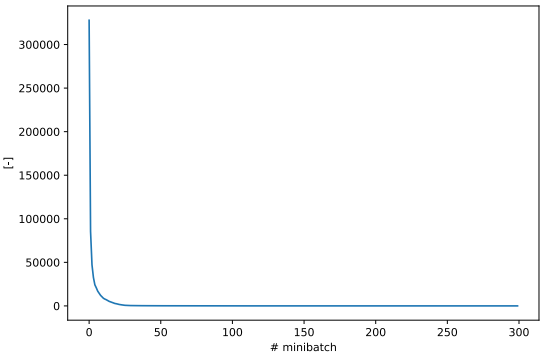

Figure 4.25: Evolution of the training loss when training the GCN model

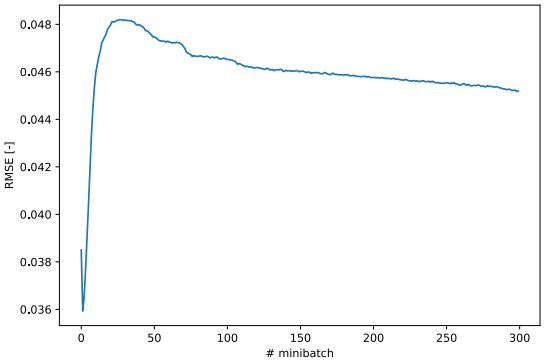

Figure 4.26: Evolution of the RMSE of voltage estimation during training

The results during training were unsatisfying and no insightful comparison can be carried on. More details on the issue of training a standard GCN is provided in the discussion chapter.

5

Discussion & Conclusion

In this chapter, we analyze the results gathered in the previous chapter and provide some interpretations. Then, these interpretations allow us to answer the thesis' research questions, mention the limitations of the thesis, and provide recommendations.

5.1. Interpretation of results

5.1.1. Training and tuning

Impact of measurement weights

During the training process, we observed at first a trade-off between voltage and line loading accuracy. This trade-off is due to the nature of the optimization problem: we want to minimize the error against the *noisy* measurements using the power flow equations. The model's output values are correlated through the power flow equations, but the intrinsic noisiness of the measurements alters the correlations between the state variables. Therefore, for the same reason the WLS algorithm tends to diverge, the model learns slowly as it has to handle the intrinsic measurement errors that alter the minimization process of an already complex and non-convex loss function. This inherent problem brings another issue from the WLS algorithm to the Deep Learning model: measurement weights tuning. Whereas tuning the measurement weights can highly impact the WLS algorithm during the estimation process, the DSS^2 model is highly impacted by it during the training process. Following the observations made when tuning the measurement weights, we concluded that finding the most optimal balance between the weights is not a straightforward task and can be study-dependent. For the final version of the model, we decided to keep the standard measurement weights to focus on the tuning of hyperparameters. Although, a more extensive analysis of the weights' impact on the performances should be performed in the future.

Sensitivity to hyperparameter tuning

For the tuning process, we saw that the model was highly sensitive to the choice of hyperparameters. The final design was inspired by previous work [17] and fine-tuned using Grid Search. However, it is possible to perform more in-depth research on the choice of hyperparameters, using, for example, Bayesian Optimization. This high sensitivity to hyperparameters brings the need for such extensive tuning in the future, which might considerably improve the model's performance.

5.1.2. Performance evaluation

Gain in computation speed

From comparing the four models in the 14-bus network, we observed a high gain in speed using Deep Learning models, as expected given our hypothesis. This gain is important even for a small network, around 15 times less. Interestingly, only a slight decrease in speed was observed when comparing the DSS^2 to an FFNN architecture. This increase can be due to the message-passing steps and the architecture's added complexity but does not significantly impact the overall speed.

However, the computation speed gain decreased when scaling up the problem to 70-bus and 179-bus networks. Even though the DSS^2 shows a higher computation speed in every case, the gain factor compared to the WLS was expected to increase with the scale, which was not the case. This decrease in speed by the DSS^2 can be explained by the decrease of the hyperparameter Δt. By decreasing its value, we increase the number of message-passing iterations, which induces more computation. On the contrary, the WLS computation speed showed a smaller decrease in speed than expected. Further work should analyze further the gain in computation speed for different networks.

Great performance against WLS

The results gathered from case studies showed great performances from the DSS^2 compared to the WLS. The model was faster and alleviated the convergence issues while improving the estimation accuracy. Notably, the WLS did not converge in the 179-bus due to the problem's high dimensionality, whereas the DSS^2 offered satisfying results.

The DSS^2 showed remarkable performance in voltage estimation, outperforming the WLS in canceling measurement noise and estimating unmeasured buses in all studied networks. The model was three times more accurate in voltage estimation than the WLS in the 14-bus network, and scaling up the network increased this gain.

For the line loading estimation in the 14-bus network, the DSS^2 showed similar accuracy to the WLS (not counting transformers). As stated earlier, accuracy in both voltage and loading estimation is difficult due to the coupling of these values and the noisy measurements used as reference points. It is, therefore, interesting to see the DSS^2 estimating the line loading as accurately as the WLS while being more accurate in voltage estimation. Moreover, in the 70-bus network, the DSS^2 was ten times more accurate in loading estimation, partly due to the inability of the WLS to handle all the measurements. It also showed promising results in the 179-bus, while the WLS could not provide any estimation.

However, the difference in RMSE and RMSE% in line loading estimation was always significantly bigger for the DSS^2 than the WLS. The RMSE% is a version of RMSE normalized by the mean value of interest. This means that it is impacted by the relative distance from the true value, not the absolute error as in the RMSE. For example, an estimation of 4% instead of 2% for the loading in one line will have the same impact as an estimation of 40% instead of 20% in another line. The increased RMSE% for the DSS^2 means a higher relative error for small loadings than the WLS. Hence, the DSS^2 does not estimate small loadings as accurately as the WLS; however, in the case of line loadings in a power system, higher relative error on small loadings is not the main consideration. Accurate absolute error is more important to estimate high loadings, so a low RMSE is more critical.

Robustness to noise & sensitivity to new distributions

The study of robustness using different case studies showed strong robustness by the DSS^2 model to measurement noises and errors. The DSS^2 outperformed the WLS in every case study involving changes in measurement values and was especially robust to high measurement uncertainty.

However, the DSS^2 was highly impacted by load distribution and measurement set changes. Whereas the model showed some robustness in case of a slight variation of load distribution (see case study 8 on the 70-bus grid), there is a lack of generalization when modifying the measurement set or when changing the load distribution more importantly (cases 4,9 and 10 on the 70-bus grid). This was an expected result as the model has not been trained on these new distributions. Nevertheless, better generalization abilities should be investigated in future work, notably by taking advantage of the physic-informed framework.

Bad performance on transformer loading

The DSS^2 showed great accuracy in all tested networks. However, the accuracy of transformer loading was not satisfying. The RMSE of line loading estimation doubled when including the transformers in the 14-bus network, and a drop in accuracy was also seen in the other networks. This bad performance can be due to the simplified modeling of the transformers or the impact of the slack that changes the local behavior. Future work should investigate the impact of the simplifications and improve the modeling of slack and transformers. Also, increasing the number of independent classes in the DSS^2 architecture (separating lines and transformers) may increase the expressivity of the model and give better results.

Poor estimation of unobserved lines & underfit of remote buses

The analysis of the estimation of unmeasured lines highlighted poor results from the DSS^2 and the WLS. This failure from both models can be due to the inaccurate information provided around that line. Indeed, only pseudomeasurements are provided in the neighborhood and might negatively impact the estimation if they are too inaccurate. This is especially true with an inaccurate prediction of DER generation, which impacts the direction of the flow. Further work should improve the estimation of such lines, notably by investigating and improving the impact of DER prediction on the estimation process. Alternatively, improving the quality of pseudomeasurements can also be investigated, as such improvements will benefit the estimation process.

Also, we observed an underfitting of remote buses' voltage levels in the 70-bus and 179-bus networks

when no measurements were provided. Similar to the wrong estimation of line loading, this can be due to the poor quality of information surrounding these buses. It is also caused by the low variation of these voltage levels. Indeed, remote buses in large distribution systems do not experience high variation in voltage, so the estimation task without underfit is even harder for Deep Learning architecture.

Satisfying performance against supervised models

Comparing the semi-supervised DSS^2 against supervised FFNN and DSS^2 highlighted the quality of estimation from the former. All three models showed an excellent quality of voltage estimation, but only the semi-supervised DSS^2 provided satisfying results for the line loading estimation. The supervised models showed poor accuracy in that task due to a lack of coupling in the model's outputs, which is needed for the indirect estimation of the line loadings. This gain difference in performance highlights the strengths of the semi-supervised approach: we alleviate the need for labels while keeping satisfying performances and provide extra knowledge to accurately couple the model output state variables.

Moreover, we observed that the semi-supervised DSS^2 is more robust to measurement noise and error than its supervised version. It can be due to the ”noisier” nature of the semi-supervised training, where the model learns using the noisy measurements as reference values. Such noisier training can have a regularization effect, similar to dropout, which helps to increase robustness. Although, this difference in robustness can also be due to the difference in training length, so further investigation is needed.

Advantages of DSS architecture

Through the case studies, some advantages of the used architecture were also highlighted. The supervised DSS^2 outperformed the FFNN in this task with an RMSE three times lower. It shows the advantages of using an architecture suitable for graph data and optimization tasks on power systems, as the DSS^2 learns better from local patterns to provide more accurate local outputs.

5.2. Answers to research questions

How well can the Deep Statistical Solver architecture perform the optimization task of Distribution System State Estimation, compared to traditional Weighted Least Squares approach?

Taking full advantage of the Machine Learning tools, the DSS^2 model shows a significant decrease in computational time. Moreover, using Machine Learning enables us to eliminate the convergence issues that can arise with the WLS algorithm. This is especially true when increasing the network size or decreasing the measurements' quality. These two situations are likely in the distribution system and impact the speed and convergence of the WLS algorithm. The same conclusions can be drawn for the robustness of the model. Whereas the WLS algorithm tends to diverge when introducing wrong or very noisy measurements, the DSS^2 model showed to be robust against such events.

Then, regarding the overall accuracy, the DSS^2 model showed clear improvements in all tested networks. When estimating the values that are direct outputs of the model, which are the voltage magnitudes, the DSS^2 model outperformed the WLS algorithm in all cases. The estimation of indirect values such as loading/current flows through the lines was equivalent in the 14-bus network, but a clear im-

provement was observed when scaling up the network.

Finally, a significant advantage of the DSS^2 model that has been observed is its scalability compared to the WLS algorithm. When performing state estimation on the 70-bus and 179-bus networks, the WLS algorithm showed a drastic decrease in accuracy and convergence, whereas the DSS^2 model kept similar accuracy and outperformed the WLS.

The results highlight great performances from the DSS^2. Although, more work is needed to improve the performance further, and validation on other networks should be performed to develop an accurate and trustworthy DSSE tool. The DSSE task is a complex challenge, and beating the baseline that represents the WLS algorithm with a semi-supervised learning approach is an exciting achievement.

How accurate is the estimation provided by the semi-supervised DSS^2, compared to supervised models?

The supervised models showed great accuracy when estimating voltages, which are the direct outputs of the model. Knowing the voltage levels to reach, they achieved the best results from all compared solutions when estimating the voltage values. Fed with noisy data, they also showed robustness and good accuracy in every case study. This was an expected outcome, as providing labels is an easier learning task than learning from the power flow equations. However, as stated in the introduction, labels are rarely provided in the context of DSSE, and the slight decrease in accuracy for the DSS^2 model showed to be negligible compared to the benefits of a semi-supervised learning approach.

More importantly, the supervised models only estimated the voltage values accurately. Indeed, we observed very high errors in the estimation of line loading as these models failed to learn a coupling between the state variables to estimate the line loadings accurately. With such observation, the DSS^2 model showed great strengths in learning from the power flow equations as it incorporates a soft constraint in coupling voltage magnitude and voltage angles by fitting measurement of power flows. Moreover, this also attests to the difficulty for a Deep Learning model to perform the DSSE task, as the model needs to couple estimated values together while being fed with scarce noisy inputs. In such a case, implementing a constraint to couple the estimated values is essential to get satisfying performances, and the semi-supervised approach overall outperforms supervised models. For further improvement, hard physical constraints in the model can be investigated in the future.

What are the advantages provided by the DSS architecture?

As highlighted in the interpretation section, the supervised DSS^2 provided better results than the supervised FFNN, which shows the advantages of using an architecture suitable for graph data. Indeed, the model could learn from the graph's local patterns, which helped estimate local values.

Also, the DSS architecture appeared more suitable for such a learning approach than standard GCN. The GCN model used in this thesis for comparison consists of the most simple architecture of GNN, implementing message-passing layers between nodes without any edge features. This simple GCN model performed very poorly, and difficulties have been observed when training this model with the semi-supervised methodology. With such observation, it is clear that the DSS^2 model is better suited for implementing power flow equations in the loss function. However, the choice of such simple GNN

architecture can be discussed, and a comparison to more sophisticated GNN models should be made in future works.

5.3. Limitations

The DSS^2 model showed promising results and can outperform the WLS algorithm in most situations. However, we highlighted in this thesis several limitations that should be overcome to enable the use of the DSS^2 model for DSSE.

Simple tuning method

We highlighted in this thesis the impact of hyperparameter tuning on the model's performance. Even though we found satisfying performances with simple tuning methods (trial and error, grid search), too little knowledge on the impact on performances has been gathered. More in-depth study of the system's behavior to hyperparameters can greatly enhance the model's capabilities.

Complex training process and weights sensitivity

The loss function built in this thesis incorporates the WLS algorithm as an optimization problem to minimize. This means the integration of the power flow equations in the loss function, and the objective is to find an estimation of the state that minimizes the global error to the measured values. With this loss function, the model's output variables get coupled together and aim at fitting a global estimate provided by *noisy* measurements. The loss function becomes a multi-objective function in a sum of many terms, where providing too many weights to one noisy measurement may pull away the training from the optimum solution, as the function is non-convex in such case.

With such a complex training process, the accuracy of the trained model is highly impacted by the measurement weights. A different set of weights can affect the optimum of the loss function, and the model becomes as sensitive to these weights as the WLS algorithm. However, the difference from the WLS algorithm is that this sensitivity is felt during training and not during the estimation process, where the WLS algorithm can diverge.

Underfitting of voltage levels for remote buses

As observed in the analysis of the 70-bus and 179-bus networks, the DSS^2 model tends to reach a constant value when estimating remote buses' voltage levels that underfit the true values. This is due to the lack of variation in these voltage levels through the data, and the lack of measurements near these remote buses. This underfitting leads to satisfying accuracy in normal conditions but shows a lack of flexibility when varying the conditions of the system.

An approach to counter this issue would be to increase the size of the neural networks used in the model, which can provide more expressivity and improve the performance if combined with proper weight tuning. However, this solution may not suffice alone, and improving the observability of the system and the accuracy of measurements and pseudomeasurements will also be needed.

Estimation of unmeasured line and transformer loading

This thesis aims to develop a solution to perform DSSE in its traditional form, using the voltage magnitudes and angles as the problem's outputs. By doing so, one of the main values to estimate is the current/loading in the lines, which is only *indirectly* estimated by the model after feeding the estimated state variables in the power flow equations. As stated previously, this leads estimation errors to magnify through the equations because of the lack of direct coupling between the output values of the model, and the estimation of the loading becomes an arduous task. The DSS^2 showed great accuracy for measured and neighboring lines. While the model performed better than the WLS algorithm, it did not accurately estimate lines with no measurements in their neighborhood.

This limitation becomes even worse for the transformers, as the DSS^2 model performs poorly in estimating transformer loading. The sharp decrease in accuracy compared to the line loading suggests the existence of another issue related to the sole transformers of the system and should be investigated further.

Sensitivity to input variations and load distribution

The results presented in the previous chapter showed a decrease in performance when the DSS^2 model is fed with another measurement set as input or with a different load distribution within the network. Even though it was an expected behavior as it deviates significantly from the training dataset, robustness to such variations is needed to provide an efficient alternative to the WLS algorithm.

Even though the model showed poor results when varying the load distribution, the strength of the model is to learn a mapping from the power flow equations directly, which means that, in theory, the model could learn such mapping for a broader range of load distribution, if provided in the training set. A more heterogeneous training set should be investigated to analyze the effect on the model's robustness and accuracy.

Lack of validation on real networks

The DSS^2 showed exciting performances; however, no validation has been performed on real networks. Early trials showed difficulties in performing on such networks, but numerous factors can impact the model's accuracy, such as low quality of data or model. Indeed, the model's performance appeared to be sensitive to erroneous datasets and models, which are likely in real systems, and more work should investigate these issues.

Limited use of GNN equivariance property

In chapter 2, we explained how GNN is used to efficiently process graph data by learning from local neighborhoods and ensuring equivariance to permutation. The current implementation of DSS^2 does learn from local connectivity; however, it is not currently able to process different graph structures. Indeed, we used a fixed adjacency matrix in the current implementation to simplify the problem.

Nevertheless, it is possible to improve the implementation for handling varying graph structures, which should be investigated in future work to improve generalization abilities.

Model's architecture	Training algorithm & SE implementation	Comparison & validation
In-depth hyperparameter tuning	Different output: branch-current, mix V/I	Validation on real systems
Hard physical constraints / Inductive bias	Overloading estimation	Analysis of robustness to noise distributions
Improved components' modeling	Integration of PMU & Distributed SE	Comparison to better-suited GNN models & Kalman Filters

Table 5.1: Summary of recommendations for future work, divided in three categories and ordered from top to bottom by importance level.

5.4. Recommendations

The analysis performed in this thesis showed promising results for the DSS^2 model to become a robust alternative to the WLS algorithm. However, further work is needed to overcome the model's limitations, improve its accuracy, and increase its use case. A summary of the recommended future work is given in Table 5.1, where the tasks are divided into three categories and ordered from top to bottom by importance level.

Model's architecture

The first category of improvements concerns the model's architecture, as better Machine Learning expertise in the architecture is expected to improve the model's performance. Most notably, in this category, weights and hyperparameters tuning should be investigated. An in-depth study of the impact of the measurement weights on the training process is needed, and a better strategy for hyperparameter tuning such as Bayesian Optimization can help to improve the model. Then, inductive bias and intrinsic physical constraints should be studied as a promising solution to overcome the lack of consistency in estimating indirect values such as current flows. Finally, better modeling of the components such as transformers and slack in the H2MG format can help to provide more expressivity to the H2MGNN architecture and improve the overall performance.

Training algorithm & SE implementation

The second category of improvements concerns the training algorithm and the chosen implementation of State Estimation. Firstly, adding new outputs to the model might improve the estimation accuracy. Instead of imitating the WLS algorithm with voltage amplitudes and angles, it is interesting to use the strength of the DSS framework and add specific output features not only to buses but also to other components, such as branch current on the lines. Secondly, the current implementation assumes stable conditions where the loss function is penalized if the loading prediction is above 100%. It can be interesting to allow such predictions also to estimate line overloading. Lastly, integrating PMU measurements can further increase the use cases of the model and improve the estimation accuracy. Moreover, using PMU as reliable synchronized values can enable distributed SE where multiple models work in parallel.

Comparison & validation

Finally, further validation of the model should be performed to assess the scalability and robustness of the method, as well as further comparison. This validation should include a case study on a real dataset, lacking in the present analysis. Also, analyzing the robustness of different noise distributions can assess the model's advantages compared to WLS and Kalman Filters, that only work with Gaussian distributions. For further comparisons, the DSS^2 should be compared to more sophisticated GNN architectures and to Kalman Filters, which is also lacking in this thesis.

References

[1] F. C. Schweppe and J. Wildes, “Power system static-state estimation, part i: Exact model,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-89, pp. 120–125, 1970, issn: 00189510. doi: 10.1109/TPAS.1970.292678.

[2] M. Ahmad, *Power System State Estimation*. 2013.

[3] F. Ahmad, A. Rasool, E. Ozsoy, R. Sekar, A. Sabanovic, and M. Elitaş, “Distribution system state estimation-a step towards smart grid,” *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 2659–2671, 2018, issn: 18790690. doi: 10.1016/j.rser.2017.06.071.

[4] A. Boricic, J. L. R. Torres, and M. Popov, “Comprehensive review of short-term voltage stability evaluation methods in modern power systems,” *Energies*, vol. 14, 2021, issn: 19961073. doi: 10.3390/en14144076.

[5] A. Primadianto and C. N. Lu, “A review on distribution system state estimation,” *IEEE Transactions on Power Systems*, vol. 32, pp. 3875–3883, 2017, issn: 08858950. doi: 10.1109/TPWRS.2016.2632156.

[6] K. Dehghanpour, Z. Wang, J. Wang, Y. Yuan, and F. Bu, “A survey on state estimation techniques and challenges in smart distribution systems,” *IEEE Transactions on Smart Grid*, vol. 10, pp. 2312–2322, 2019, issn: 19493053. doi: 10.1109/TSG.2018.2870600.

[7] X. B. Jin, R. J. R. Jeremiah, T. L. Su, Y. T. Bai, and J. L. Kong, “The new trend of state estimation: From model-driven to hybrid-driven methods,” *Sensors*, vol. 21, pp. 1–25, 2021, issn: 14248220. doi: 10.3390/s21062085.

[8] H. Liu, F. Hu, J. Su, X. Wei, and R. Qin, “Comparisons on kalman-filter-based dynamic state estimation algorithms of power systems,” *IEEE Access*, vol. 8, pp. 51 035–51 043, 2020, issn: 21693536. doi: 10.1109/ACCESS.2020.2979735.

[9] N. Kumari, R. Kulkarni, M. R. Ahmed, and N. Kumar, “Use of kalman filter and its variants in state estimation: A review,” in *Artificial Intelligence for a Sustainable Industry 4.0*, S. Awasthi, C. M. Travieso-González, G. Sanyal, and D. Kumar Singh, Eds. Cham: Springer International Publishing, 2021, pp. 213–230, isbn: 978-3-030-77070-9. doi: 10.1007/978-3-030-77070-9_13. [Online]. Available: https://doi.org/10.1007/978-3-030-77070-9_13.

[10] P. A. Pegoraro, A. Angioni, M. Pau, A. Monti, C. Muscas, F. Ponci, and S. Sulis, “Bayesian approach for distribution system state estimation with non-gaussian uncertainty models,” *IEEE Transactions on Instrumentation and Measurement*, vol. 66, pp. 2957–2966, 2017, issn: 00189456. doi: 10.1109/TIM.2017.2728398.

[11] K. R. Mestav, J. Luengo-Rozas, and L. Tong, “Bayesian state estimation for unobservable distribution systems via deep learning,” *IEEE Transactions on Power Systems*, vol. 34, pp. 4910–4920, 2019, issn: 15580679. doi: 10.1109/TPWRS.2019.2919157.

[12] M. Q. Tran, A. S. Zamzam, P. H. Nguyen, and G. Pemen, "Multi-area distribution system state estimation using decentralized physics-aware neural networks," *Energies*, vol. 14, 2021, issn: 19961073. doi: 10.3390/en14113025.

[13] B. Zargar, A. Angioni, F. Ponci, and A. Monti, "Multiarea parallel data-driven three-phase distribution system state estimation using synchrophasor measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, pp. 6186–6202, 2020, issn: 15579662. doi: 10.1109/TIM.2020.2967512.

[14] S. de Jongh, F. Müller, H. Li, B. Georgieva, M. Suriyah, and T. Leibfried, "Machine-learning-based bayesian state estimation in electrical energy systems," English, *CIRED - Open Access Proceedings Journal*, vol. 2020, 341–344(3), 2020. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/oap-cired.2021.0053.

[15] G. Revach, N. Shlezinger, R. J. Sloun, and Y. C. Eldar, "Kalmannet: Data-driven kalman filtering," vol. 2021-June, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 3905–3909. doi: 10.1109/ICASSP39728.2021.9413750.

[16] B. Habib, *Github — implementation of deep statistical solver for distribution system state estimation*, https://github.com/TU-Delft-AI-Energy-Lab/DSS2, [Online; accessed 21-August-2022], 2022.

[17] B. Donon, "Deep statistical solvers & power systems applications," Artificial Intelligence [cs.AI]., Université Paris-Saclay, 2022. [Online]. Available: https://tel.archives-ouvertes.fr/tel-03624628.

[18] A. Abur and A. G. Expósito, *Power system state estimation : theory and implementation*. Marcel Dekker, 2004, p. 327, isbn: 0824755707.

[19] M. Bazrafshan and N. Gatsis, "Comprehensive modeling of three-phase distribution systems via the bus admittance matrix," *IEEE Transactions on Power Systems*, vol. 33, pp. 2015–2029, 2018, issn: 08858950. doi: 10.1109/TPWRS.2017.2728618.

[20] Wikipedia contributors, *Mean squared error — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=1097028555, [Online; accessed 28-July-2022], 2022.

[21] ——, *Mean squared error — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=1097028555, [Online; accessed 28-July-2022], 2022.

[22] K. Pykes, *Unsupervised machine learning explained*, https://towardsdatascience.com/unsupervised-machine-learning-explained-1ccc5f20ca29, [Online; accessed 28-July-2022], 2021.

[23] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019, issn: 10902716. doi: 10.1016/j.jcp.2018.10.045.

[24] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, pp. 422–440, 2021, issn: 25225820. doi: 10.1038/s42254-021-00314-5.

[25] K. Pykes, *Semi-supervised machine learning explained*, https://towardsdatascience.com/semi-supervised-machine-learning-explained-c1a6e1e934c7, [Online; accessed 28-July-2022], 2021.

[26] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. doi: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[27] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, pp. 281–305, 2012, issn: 1532-4435.

[28] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms*, 2012. doi: 10.48550/ARXIV.1206.2944. [Online]. Available: https://arxiv.org/abs/1206.2944.

[29] Wikipedia contributors, *Universal approximation theorem — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Universal_approximation_theorem&oldid=1100923966, [Online; accessed 2-August-2022], 2022.

[30] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," vol. 2018-January, Institute of Electrical and Electronics Engineers Inc., 2018, pp. 1–6, isbn: 9781538619490. doi: 10.1109/ICEngTechnol.2017.8308186.

[31] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *CoRR*, vol. abs/1806.07366, 2018. arXiv: 1806.07366. [Online]. Available: http://arxiv.org/abs/1806.07366.

[32] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. doi: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[33] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "Pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, 2018. doi: 10.1109/TPWRS.2018.2829021.

[34] K. Strunz, E. Abbasi, R. Fletcher, N. Hatziargyriou, R. Iravani, and G. Joos, *TF C6.04.02 : TB 575 – Benchmark Systems for Network Integration of Renewable and Distributed Energy Resources*. 2014, isbn: 9782858732708.

[35] K. Rudion, A. Orths, Z. A. Styczynski, and K. Strunz, "Design of benchmark of medium voltage distribution network for investigation of dg integration," IEEE Computer Society, 2006, isbn: 1424404932. doi: 10.1109/pes.2006.1709447.

[36] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral," *CoRR*, vol. abs/2006.12138, 2020. arXiv: 2006.12138. [Online]. Available: https://arxiv.org/abs/2006.12138.