Karlsruher Institut für Technologie

Institut für Elektroenergiesysteme
und Hochspannungstechnik
Prof. Dr.-Ing. Thomas Leibfried

Bachelor Thesis

of

Mr. *Fares Kallel*

„Deep Learning for Optimization in Electrical Power Grids “

Start:	7th of Mai 2020
End:	25th of November 2020
Advisor:	Steven De Jongh M.Sc.
Supervisor:	Prof. Dr.-Ing. Thomas Leibfried

Ich versichere hiermit, dass ich meine Bachelor arbeit selbstständig und unter Beachtung der Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) in der aktuellen Fassung angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, den 25th of November 2020

Fares Kallel

Abstract

The tremendous growth in power demand and the increasing number of renewable energy stations is facing energy distributors with new challenges. Electrical power supply has to be well scheduled and distributed over all the system ensuring at all times availability, security and stability. The Optimal Power Flow calculation is at the centre of optimizations in power systems, however a robust, reliable and fast enough technique to solve the problem is yet to be developed. Artificial intelligence, present itself as a new tool that can contribute in developing a solution for the problem. In this work, a graph convolutional neural network is implemented, tested and compared to a standard artificial neural network in order to predict the optimal power flow values for an electrical power grid. The outcome of this thesis has revealed that using the graph convolutional architecture compared to the non-graph-based architecture considerably improves the predictions of the optimal power flow values. We note a decrease of the prediction error by 95% for a 14-bus grid and 60% for a 39-bus grid. The results also show that the trained architectures are able to make predictions 200 times faster than traditional techniques like the interior point method.

Content

Chapter 1

Introduction

In this chapter we present the motivation behind this work as well as an outline that briefly summarizes the objectives and the main results that were reached.

1.1 Background

For centuries, energy has been one of the most discussed and investigated topics in the scientific community. The way the energy is produced and distributed has always been tightly connected to the development of civilizations. Our generation is witnessing major advancements all over the world with new technologies emerging everyday in different fields with the majority of them relying on electrical energy for power supply.
Therefore, the demand in distributed energy has reached the highest point so far which poses technical and financial challenges for electrical energy distributors as they have to cover the demand in power changing over time and space, while securing a stable and optimal distribution over the entire system [1].
In addition to that, the way the energy is produced is being questioned and revisited all over the world due to environmental reasons by urging to switch from traditional energy resources such as burning fossil fuel, to renewable energy resources such as solar photovoltaic and wind energy.
Such power systems however, have high variability as they are bound to natural factors like wind velocity and solar irradiations constantly changing over time [2] which poses additional challenges for power distributors as they have to manage not only load variability but also the power generation variability while ensuring a stable supply of energy over the entire system and an appropriate scheduling of power distribution [1].

1.2 Outline of Work

The energy sector is facing new challenges emerging from the ever-growing demand in electrical power and from the recent expansion of renewable energy stations. These challenges include the necessity to provide a secure, stable and optimal distribution over all the electrical power system.
For that purpose, we address in this thesis the problem of the Optimal Power Flow (OPF).
The OPF problem dates back to the 1960's [3] and it is since then considered one of the most important optimization problems in the energy industry. In some applications like the RT-OPF, the OPF must be run up to every six seconds [4] which requires a robust but also a very computationally fast solver.
To tackle this problem, researchers have developed several methods, some of them are referred to

as traditional such as the interior point method or the Newton method, and some of them can be categorized as contemporary and the reason for using this term, is that these methods are taking advantage of the huge developments materializing in artificial intelligence.

In this Thesis, we explore an artificial intelligence variant: Deep learning and more precisely, artificial neural networks and graph convolutional neural networks. Since a power grid can be represented with a graph, it is insightful to investigate the impact of including this graph in a prediction model and evaluate this model according to the prediction accuracy in comparison to non-graph-based artificial neural network models, but also evaluate the computational time compared to traditional techniques for calculating the optimal power flow.

The results from the different experiments conducted in this work, show that using graph convolutional neural networks improves the prediction accuracy in comparison to artificial neural networks with a reduction of the prediction error of 95% for a 14-bus network and 60% for a 39-bus network. It is to be noted however, that the accuracy of the models decreases with larger networks due to their complexity. The results also show that both artificial intelligence based models, after the training, are able to make predictions in a much shorter amount of time compared to calculations made using the interior point method.

In this work we contribute to the research about applying artificial intelligence to the energy field by showing that graph convolutional neural networks have high capabilities in extracting features from electrical power grids. These features can be used to make predictions about the state and functioning of the power systems for optimization purposes but also for security and innovation purposes.

Chapter 2

Theoretical Fundamentals

In this chapter, we present an overview on the Optimal Power Flow. We provide a definition of the concept and a formulation of the problem as well as a classification of previously used methods to solve it. The chapter also includes theoretical fundamentals about artificial neural networks and graph convolutional neural networks.

2.1 Optimal Power Flow

2.1.1 Definition

The Optimal Power Flow (OPF) is one of the most important optimization problems in the energy industry. It is used for planning, establishing prices on day-ahead markets, grid extensions and most importantly efficiently distribute stable energy overall the system throughout the day.

The OPF calculations for a given power grid, aim at finding the optimal set points for all generators in the system that are required to satisfy a given demand but also stay within the limits of a set of physical constraints and also respect the power flow balance equations [5]. OPF is at the center of Independent System Operator (ISO) power markets, and is solved in some form every year for system planning, every day for day-ahead markets, every hour, every five minutes and even every 6 seconds [3, 4]. However, 60 years after the problem's first formulation by Carpentier in 1960's, there is still no robust and fast enough solution for calculating the AC-OPF. Finding this solution could potentially save tens of billions of dollars annually [3].

2.1.2 Applications

Since it's formulation by Carpentier in the 1962, the OPF was applied for a number of optimization tasks taking various forms [6]. Some of these applications are shown in figure (2.1) which represents two heat maps.
In the upper heat map, green denotes up to a 24-hour lead time for the decision making process, whereas red denotes an increase in the time-criticality.
In the lower heat map, grey denotes highly approximated mathematical representations of the OPF whereas dark red denotes the most representative formulation of steady-state operations in AC networks. The two heat maps follow the same direction, which means the shorter the time is to make the calculations, the more precise has to be the representation of the OPF.
The computational challenge today is to find a consistent global optimal solution with speeds faster than the existent solvers by considering the most accurate formulation which is the AC-OPF.

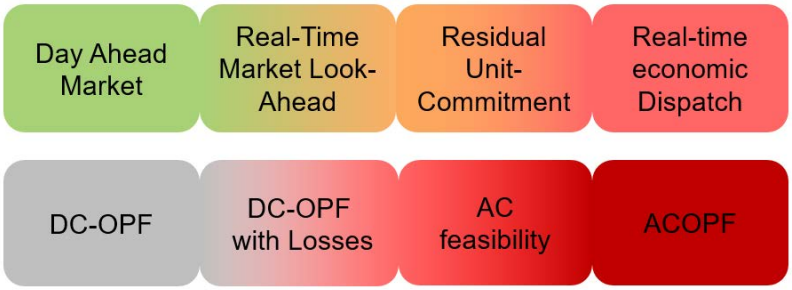

Fig. 2.1: Applications and formulations of the optimal power flow

2.1.3 Problem Formulation

The problem of the Optimal Power Flow consists of determining the power that every generator needs to deliver into the system at a given time, in order to cover the power demand while simultaneously respect power flow balance equations and a set of physical constraints [5].
The state of a given electrical power grid can be described using several variables. Among these variables, one can consider for each bus of the system the voltage magnitude V and the voltage angle θ along with the active and reactive power p and q. These measurements can be collected across all buses and stored in a vector x such that $x = \begin{bmatrix} V & \theta & p & q \end{bmatrix}$.

Mathematically speaking, the OPF problem can be brought down to minimizing an objective function in order to satisfy a purpose, such as minimize a cost, minimize total network losses [7] and/or maximizing the total yield from a network [8].
For example if we consider C to be a cost function that depends only on the active power generated at each generator p_g, the problem can then be formulated as to minimize the cost function C relative to all generators in the grid while considering the balance equations that control the power flow as well as the set of the physical constraints (2.1).
The active and reactive power p and q at one bus are determined by the voltage angles θ and voltage magnitudes V at all other buses and also the topological structure of the power grid W [9].
These variables describe physical entities which makes them naturally and physically bound to some constraints such as active and reactive power injections p and q at each bus but also limits on voltage angle θ and voltage magnitude V. It is also common to consider limits on line currents. These constraints are all stacked and noted with the vectors X^{min} and X^{max}. (2.1d) .

$$\text{minimize} \sum_{g \in \mathcal{V}_g} C(p_g) \tag{2.1a}$$

$$\text{considering } p = \mathcal{P}(V,\theta,W) \tag{2.1b}$$

$$q = \mathcal{Q}(V,\theta,W) \tag{2.1c}$$

$$X_{min} \leq X \leq X_{max} \tag{2.1d}$$

The balance equations of the active and reactive power (2.1b),(2.1c) are what mainly differs one representation of the OPF from another. The most representative formulation of OPF can be described with the full AC- power flow equations.

2.1.4 AC-Power Flow Equations

The most accurate and representative formulation of the Optimal Power Flow considers the following full AC- power flow equations.

$$p_i = V_i \left[\sum_{k=1}^{n} Y_{i,k} \cdot V_k \cdot cos(\theta_i - \varphi_k - \alpha_{i,k}) \right] \tag{2.2}$$

$$q_i = V_i \left[\sum_{k=1}^{n} Y_{i,k} \cdot V_k \cdot sin(\theta_i - \varphi_k - \alpha_{i,k}) \right] \tag{2.3}$$

These equations model the flow of power over the network. They are composed of a set of coupled non-linear equations that follow the Kirchoff laws.

Although these laws are linear in voltages and currents, power is a product of voltage and current which becomes a quadratic term making the equations non-linear and the problem non-convex and therefore hard to solve [10].

Some applications of the OPF consider the DC-OPF setting which is a linearized form of the optimal power flow where the voltages are not taken intou account in the power constraints [11]. The DC-OPF formulation can be sufficient for some applications but for other applications, it is not suitable for the actual conditions of the OPF.
A number of methods have been developed over the years in order to find a suitable and acceptable solution to calculate the OPF. These are introduced in the following section.

2.1.5 Methods

The term Optimal Power Flow first emerged in 1961 and since then, many methods have been developed to solve the optimization problem. These different methods can be classified into two categories: traditional methods and modern intelligent methods [12].

2.1.5.1 Traditional Methods

The traditional methods rely mostly on mathematical programming and even though these conventional methods produce accurate results and they are widely used in the energy industry, they tend to be very slow and some of them perform linearizations and approximations that are not always compatible with the real power flow conditions. This can be considered a problem when dealing with continuously changing electrical systems. One of these methods and one that is widely used and shows the most reliable results is the Interior Point Method (IPM). The IPM can be dissected into four steps [13]:

1. The inequality constraints of the OPF problem are transformed into equality constraints by adding slack variables to the inequalities.
2. The non-negativity conditions are handled by adding them to the objective function of the OPF as logarithmic barrier terms.

3. The Problem is transformed from a constrained optimization problem to an unconstrained problem using the Lagrange theory of optimization.

4. Solve the Karush-Kuhn-Tucker perturbed first order optimality conditions by the Newton method.

2.1.5.2 Modern Methods

To cope with the limits of the traditional methods, and to reach global optimal solutions, intelligent computational strategies took part in solving the OPF.
Most of these artificial intelligence tools are architected in a way that mimics a natural phenomenon such as species evolution, thermal dynamics of a metal cooling process or data processing and interpretation in human brain.
These methods have shown high capabilities when dealing with problems that have non-differentiable regions or problems that need to reach global solutions to be considered acceptable results. The focus of this thesis will be on variants of Artificial Neural Networks.

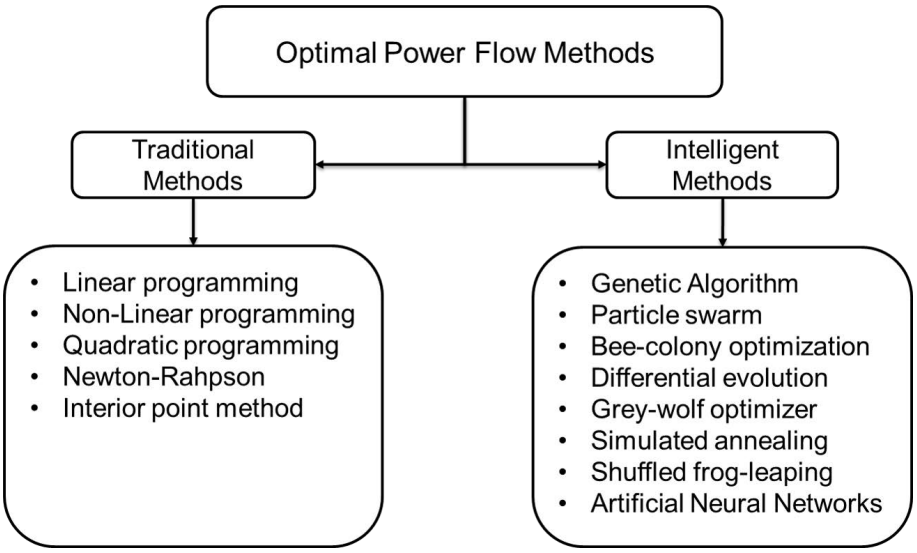

Fig. 2.2: Methods used to calculate the optimal power flow

2.2 Artificial Neural Networks

Artificial Neural Network (ANN), were first imagined by Rosenblatt in 1958 [14]. Like the expression suggests, they are an attempt to replicate the network and the behavior of the neurons that exist in a human brain. These fundamental units are used to process and transmit information from on region to another in order to understand and learn behavior.

2.2.1 Basics of Artificial Neural Networks

ANN consist of an input layer, an output layer and a number of hidden layers with each layer holding a number of neurons. These Neurons, also called units, hold, receive, and pass information from one layer to next. This information is passed through lines of connection that can be considered as the ground that holds the neural network together. These connection lines hold variable values called weights that describe the intensity of the connection between two units [15].

Neural networks are structured to satisfy a certain purpose and they have to be trained in order to learn that particular purpose and be able to make predictions that meet with the initial intentions. Neural networks can either be trained in a supervised [16] , unsupervised [17] or a semi-supervised manner [18].
Supervised learning involves a mechanism that includes the desired target in the learning process while in unsupervised learning, the neural network has to figure out, based only on the inputs, the correct outcome of the neural network. The semi-supervised learning uses a combination of the two previous mechanism where both labeled and unlabeled data is used.
In this work, the different artificial neural network models will be trained in a supervised way and this mechanism will be further explored in the next section.

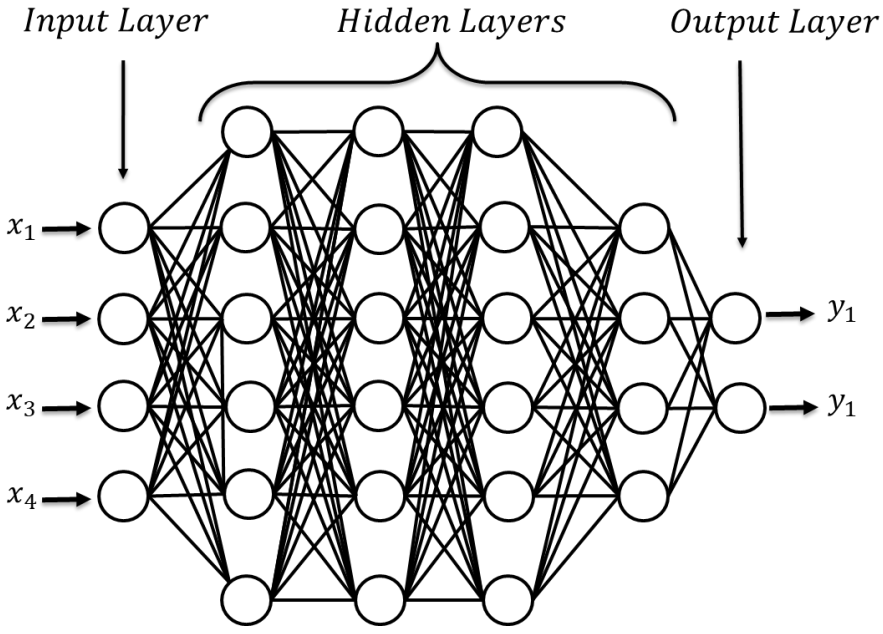

Fig. 2.3: An example of a neural network

2.2.2 Supervised Learning

A supervised training or supervised learning is one way out of many to train machine learning algorithms. What makes a training supervised is the presence of target or output data in the dataset which means the dataset is labeled or, in other words, the outcome of the samples of the input data is known and the aim is to find a mapping function between input and output that can generalize to new data.
If we consider a dataset, that includes input values and output values. In that case, a neural network can be built between the input and output and trained in a supervised way in order to solve the mapping function. The training of the neural network consists of making a number of runs between input and output layer and with every run, the coefficients of the mapping function are updated to decrease a loss value. These coefficients are precisely the weight values mentioned in the previous section (2.2.1).
This procedure allows very complex, nonlinear patterns to be mapped between input and output. Figure (2.4) shows a black box that represents an unknown mapping function between input X and output Y and the matrix inside the black box is the weight matrix of the built neural network between X and Y but represents also the coefficients of the mapping function.

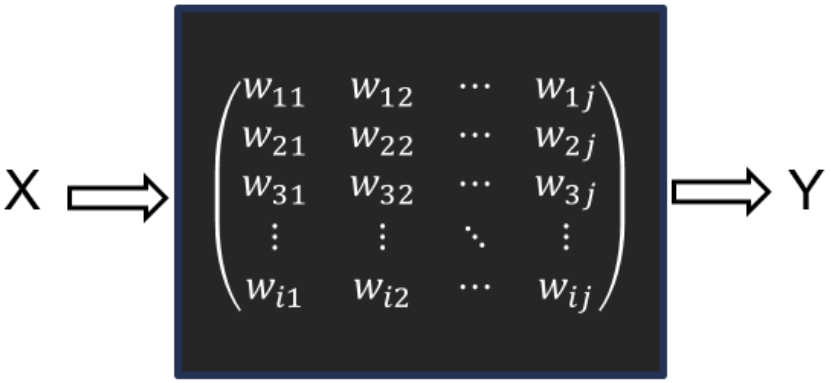

Fig. 2.4: The black box of a mapping function

2.2.3 Forward Propagation

Forward propagation refers to the calculation and storage of intermediate variables for the neural network (including outputs) [19]. Once a neural network is built between a given input and output data in order to fulfill a particular prediction task, it is then ready to be trained.
For the first run, since the network has no way to know the right weights, the input feature vector x is propagated with each edge assigned a random weight value.
The process that then takes place inside each unit of hidden or output Layers, happens in two steps:

First step is called pre-activation. It consists of calculating the weighted sum of the inputs from the previous layer according equation (2.4).

$$x_i{}^{(l)} = \sigma\left(w_0 + \sum_{j=1}^{n} w_i \cdot x_j{}^{(l-1)}\right) \tag{2.4}$$

Each unit computes an affine transformation where w_i are the weight parameters of the units of the hidden layer and w_0 the bias which acts like the intercept added to a linear equation and is used to shift the output to adjust it to fit more perfectly to the given data.
The second step, consists of determining whether the neuron should transmit the information it is holding to the neurons of the next layer, based on the value of the aggregated sum. This step is performed by the activation functions.

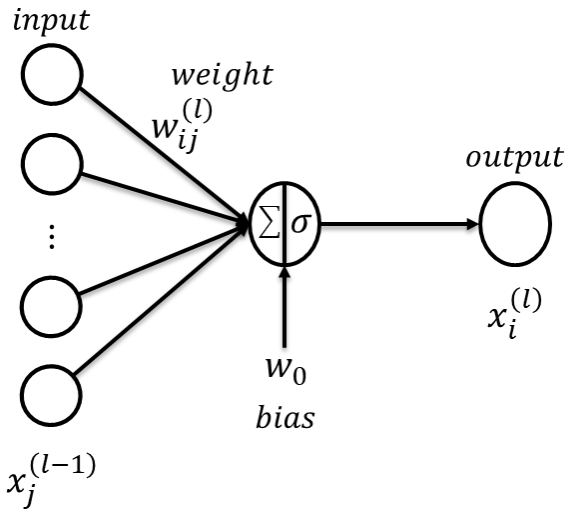

Fig. 2.5: Forward propagation of information across a layer of an artificial neural network

2.2.4 Activation Functions

In neural networks, the activation function is a mathematical function, usually non-linear, applied to the output signal of a neuron [20].
The task of an activation function in a forward propagation is to determine whether a neuron should transmit the information or not based on the relevance of the information carried by the neuron. The activation function as well as its derivative need to be simple since they will be used back and forth in every run and for every neuron in the network. Table (2.1) presents the equations and derivatives of some commonly used activation functions and figure (2.6) shows a plot of the equations of some of these functions.
In complex networks with a high number of hidden layers, the choice of the activation functions have a significant impact on the performance of the network, as they introduce non-linearity to the model.

Activation function	Equation	Derivative
Sigmoid Function	$\sigma_1(x) = \frac{1}{1+e^{-x}}$	$\sigma_1'(x) = \sigma_1(x) \cdot (1 - \sigma_1(x))$
Rectified Linear Unit ReLU	$\sigma_2(x) = max(0,x)$	$\sigma_2'(x) = 1$ for $x > 0$ $\sigma_2'(x) = 0$ otherwise
Hyperbolic Tangent	$\sigma_3(x) = tanh(x)$	$\sigma_3'(x) = 1 - tanh(x)^2$

Tab. 2.1: Commonly Used Activation Functions

After the information is propagated through the network, and the output layer is reached, an error is calculated using a loss function and then back propagated through the neural network using a method based on gradient descent in order to improve the weights so they better represent the mapping function between input and output data. In the following sections both the concepts of loss functions and back propagation are further explored.

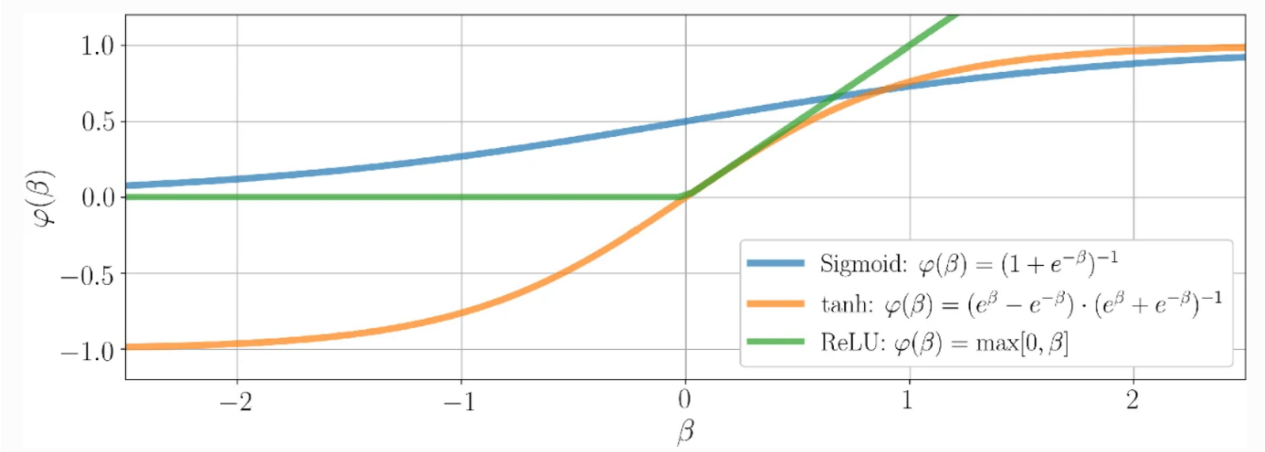

Fig. 2.6: Plot of commonly used activation functions [21]

2.2.5 Loss Functions

The task a loss function performs in a machine learning algorithm is to evaluate how good the performance was in modeling the given data.
If the predictions of the neural network have a high deviation from the actual target values, then the loss function would output a high value. The appropriate loss function to choose for a certain machine learning task depends most of all of the type of the task, either it is a regression or a classification task. For a regression task the most used loss functions are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE).
MSE [22] is the mean of the sum of squared distances between the target values and the predicted values. Its mathematical description is as follows:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{2.5}$$

Where y_i is the target value, $\hat{y}_i$ the prediction and n the total number of the observations.

MAE is the average of the sum of absolute differences between target values and predictions and it varies from zero to $+\infty$. It is mathematically described in [23] as follows:

$$MAE = \frac{\sum_{i=1}^{n} |(y_i - \hat{y}_i|}{n} \tag{2.6}$$

Where y_i is the target value, $\hat{y_i}$ is the prediction and n the total number of the observations.

2.2.6 Back Propagation

When an Artificial Neural Network is presented with data, in the first run, based only on the input and the random values assigned to the weights, it will make predictions that have a high error value. To eventually be able to make accurate predictions, the neural network has to update the weights after each forward propagation. Back propagation is the tool that allows the neural network to perform that task.
Back propagation was first introduced by David Rumelhart, Geoffrey Hinton and Ronald Williams in 1986 [24] and it represented a huge step for optimizing artificial neural networks. It consists of an implementation of the gradient computation which in its turn consists of calculating the partial derivatives of the computed loss for each neuron of each layer relative to all the weights and accordingly, adjust the value of the weights and biases to minimize the error value.[25]
Algorithm 1 shows the steps performed to realize the basic gradient descent Algorithm.

Algorithmus 1 : Gradient Descent Algorithm

Input : Loss function $\mathcal{L}$, initial weigtht matrix $W^{(0)}$, Number of iterations T, Learning rate η
Output : Final weigtht matrix $W^{(T)}$

```
for t in T do
    for w_i^(t) in W^(t) do
        compute the gradient of the Loss :  ∇L(w_i^(t))
        update the weight :  w_i^(t+1) := w_i^(t) − η∇L(w_i^(t))
return W^(T)
```

2.3 Graph Convolutional Neural Networks

Artificial Neural Networks have achieved promising results in different applications and fields of research. They were successful in extracting hidden features within the data using them to make accurate predictions. However, some studies have developed a way to improve the capabilities of artificial neural networks for some types of data that have a structure that can be represented with graphs.

2.3.1 Graphs

Graphs are composed of nodes that represent smaller regions or smaller objects of bigger entities. These nodes are connected to one another with edges that describe a relation or the intensity of the relation between two nodes if the graph is weighted. Graphs can either be directed or undirected.

Graphs are a very efficient way to represent structure, they simplify notions and make them easier to understand and to work with. They present a way to dissect data into smaller entities or regions and encode the resulted structural information to model the relations among the entities and therefore offer more insights underlying the data [26].

A weighted graph $\mathcal{G}(\mathcal{V},\mathcal{E},A)$ can be mathematically represented with $\mathcal{V} = \{v_0,\dots,v_{N-1}\}$ a set of $N = |\mathcal{V}|$ nodes, and $\mathcal{E} = \{e_0,\dots,e_{M-1}\}$ a set of $M = |\mathcal{E}|$ edges. A represents the weighted adjacency matrix of the graph which is a matrix with shape $A = N \times N$ where each element of the matrix is either a zero, when there is no connection between two certain nodes, or the element takes the value of the weighted edge $w(e_{ij})$ connecting the two nodes v_i and v_j.

$$A_{ij} = \begin{cases} w(e_{ij}), & \text{if } v_i \cap v_j = e_{ij} \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

The idea of combining graphs with deep learning was motivated by LeCun et al. in 1998 [26] and the idea was derived from the conventional convolutional neural networks on regular structures making use of the spatial relations of the nodes.
This notion of exploiting spacial relation in a structure, motivated spatial graph convolutional neural networks (SGCNN) to represent a node in graph based on the nodes surroundings it, also called neighborhood of a node. In this work, we use the notation $\mathcal{N}_v$ to represent the neighborhood of a node.
(SGCNN) is one of two variant of graph convolutions. The second variant is called spectral graph convolutional neural networks. It was introduced by Bruna et al. [27] and it defines the convolution operation on graphs in the spectral domain by exploiting the eigenvectors of the graph laplacian matrix. In the following section we will further explore the spatial convolutional neural networks.

2.3.2 Convolution on Regular Structures

The regular structure that digital images and times series have, also called euclidian structure, represent a key feature in deep learning areas such as image recognition and natural language processing. Images, audio data and many other similar kinds of data share these important properties:

- They are stored as multi-dimensional arrays.
- They are arranged over one or more axes for which the ordering matters.
- One of the axes, called channel axis, is used to access the different views of the data (e.g., the red, green and blue channels of a colored digital image).

The pixels in a digital image, represent nodes of the regularly structured graph and they hold values. Each pixel is connected to four adjacent pixels and with all edges holding the same weight value, the relation between nodes can be called symmetric .
A convolutional layer uses convolutional kernels also called filters (2.7). These learnable filters slide over small regions of the image combining every pixel with the surrounding pixels to extract higher level representations of the image.

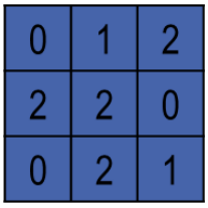

Fig. 2.7: A convolutional kernal

This process is illustrated in figure (2.8) where light blue represents the input feature matrix with the pixel values of the image and the darker blue representing the filter that slides across the input feature map and at each location, the product of each element of the filter and the feature input element it is overlapping, is computed and the results summed up to produce the output in that location [28].

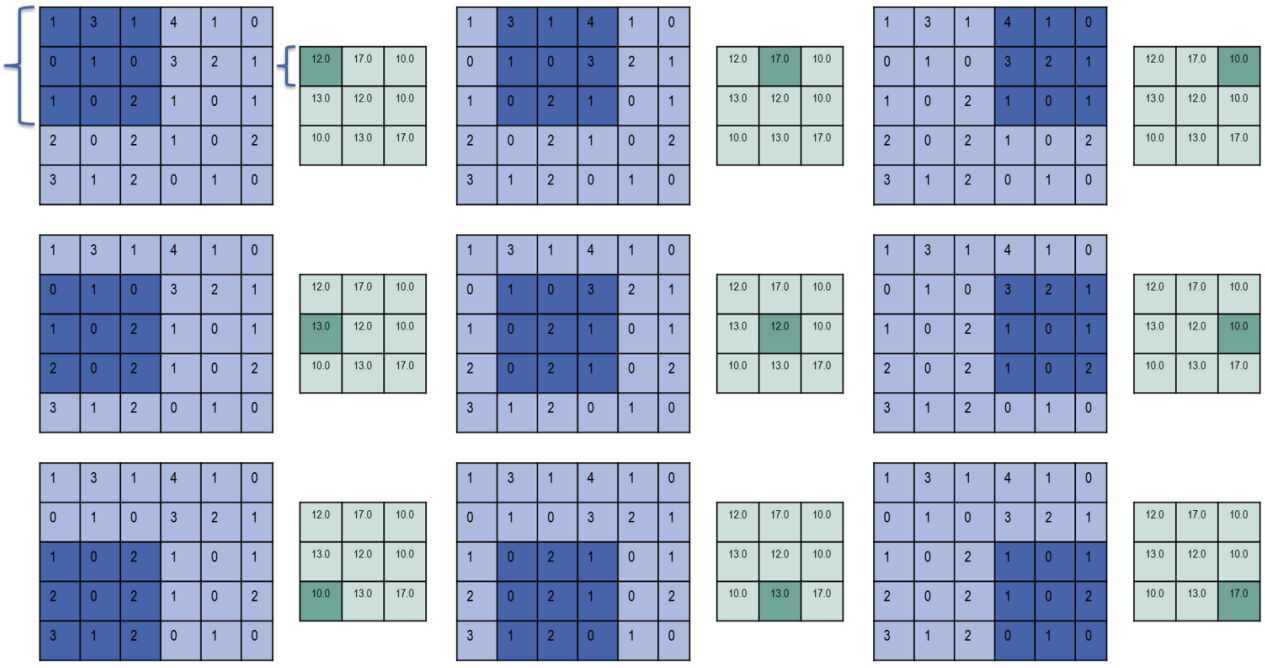

Fig. 2.8: Computing the output of a convolution on a digital image

2.3.3 Convolution on Graphs

Unlike images or audio data, structures like irregular graphs do not have the symmetric euclidian characteristics. Nodes are unordered and they can have a different number of neighbors and the weights between the nodes can have different values. This makes filtering on graphs not as straightforward as on digital images. For that reason, in order to perform convolutions on irregular graphs, a way to represent nodes using the neighbors and a mechanism that allows filtering over this neighborhood was in need. To understand this mechanism, it is important to define signal processing on graphs and for that, we explore in the next two subsections two important notions for graph convolutions: graph shift and graph filtering.

2.3.3.1 Notion of Graph Shift

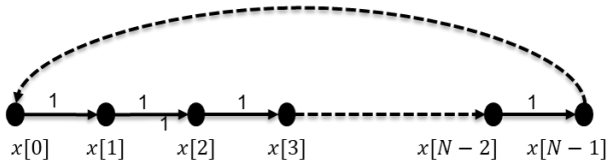

Fig. 2.9: Time series

In classical discrete signal processing (DSP), a finite discrete time series signal like shown in figure (6.2), can be represented with a vector x . The multiplication of that vector with a simple shift invariant delay filter C results in a shifted version of the input signal.

$$\hat{x} = Cx \tag{2.8}$$

However in graph theory, that finite discrete time series signal is seen as a graph with its adjacency matrix defined as:

$$A_{ij} = \begin{cases} 1, & \text{if } v_i \cap v_j = e_{ij} \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

If this adjacency matrix is compared to the shift operator, used to shift the time series signal, it will appear that it is the same matrix.
This notion opens the door for graph signal processing as for any graph, when multiplied with its adjacency matrix, the result is a shifted version of the input graph.

$$\hat{x} = Ax = Cx \tag{2.10}$$

In graph signal processing, this notion of shift generalizes to graph signals, where each node of a graph can be represented by a vector x of features describing the state of the node and the connections between nodes described with an adjacency matrix [29].
The graph shift operator applied over the graph signal of a node v, would result in a new representation of that node that includes the representation of its neighbors $\mathcal{N}_v$.

$$\hat{x_v} = \sum_{j \in \mathcal{N}_v} A_{vj} \cdot x_j \tag{2.11}$$

2.3.3.2 Notion of Graph Filtering

Filtering on graphs is similar to filtering in classical DSP. It is a multiplication of the signal vector with the matrix of the operator. The used filter however, has to be linear and shift invariant [29]. These characteristics are needed to validate theorem 1 that states:

Theorem 1:

If a graph is considered with its adjacency matrix A and assuming that its characteristic and minimal polynomials are equal $p_A(x) = m_A(x)$, then a graph filter G is linear and shift invariant if and only if, G is a polynomial in the graph shift meaning if and only if there exists a polynomial:

$$g(x) = g_0 + g_1 x + \ldots + g_K x^K \tag{2.12}$$

with possibly complex coefficients $g_k \in \mathbb{C}$, such that:

$$G(A) = g_0 I + g_1 A + \ldots + g_K A^K \tag{2.13}$$

The previous theorem can be used in both directions, resulting in the possibility to represent any linear shift invariant filter as a linear weighted combination of shifts. This represents the building blocks for graph filtering and with the convolution operator being linear and shift invariant, it becomes possible to define a convolution operation on graphs using just the adjacency matrix of the graph.

$$G = \sum_{k=0}^{K} g_k A^k \tag{2.14}$$

With the graph convolution defined for any arbitrary graph, a graph convolutional layer can be given by the formula:

$$x^{(l+1)} = \sigma(G x^{(l)} + b) \tag{2.15}$$

where $x^{(l+1)}$ represents the feature vector at layer $(l + 1)$, b a bias and G the learnable graph convolutional filter.

Chapter 3

Concept

This chapter, details the concept of this work and presents a primordial part in building prediction models which is the data.

3.1 Data

For a machine learning model to understand the actions that have to be followed, to be able to learn and to finally build a mapping function, a collection of samples or instances have to be fed to the model. This collection of data samples is the dataset and it has to be homogeneous and big enough for the model to improve the predictions step by step until an acceptable solution is reached. Datasets can either be extracted from real situations or generated from a simulator.

3.1.1 Data Generation

For this work, a data generator is implemented, one that can consider an electrical power grid, simulate an optimal power flow and generate the needed input and output samples. In this work, two datasets were constructed based on IEEE 14-bus and the IEEE 39-bus test cases shown in figures (3.1) and (3.2).

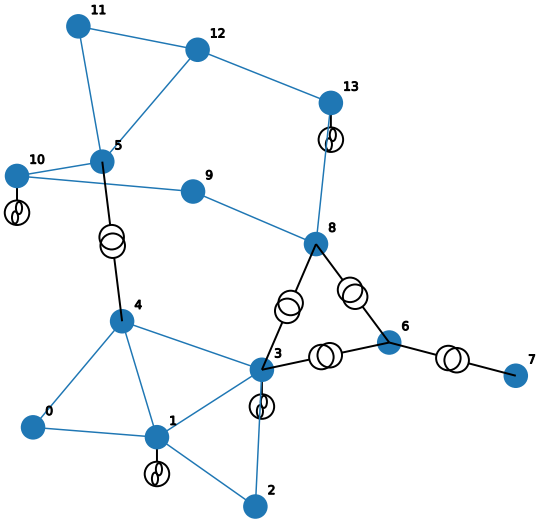

Fig. 3.1: The 14-bus test-case network

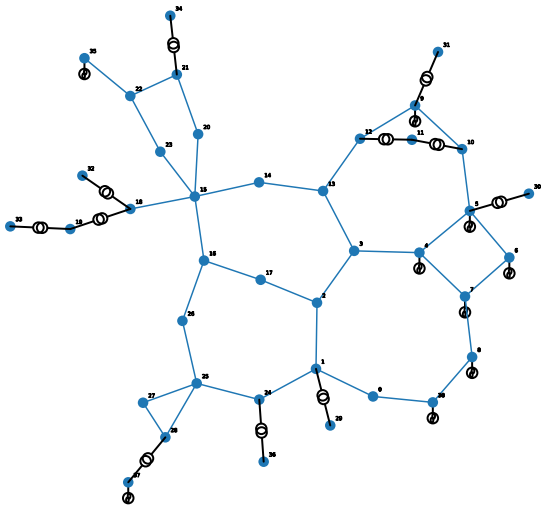

Fig. 3.2: The 39-bus test-case network

Each of the two datasets include, first the input data. In this setting, the input data is an initial state of the grid that includes only the variables that can be known and fixed before running any power flow calculation while the unknown variables are set to the value -1.
Second part in a dataset is the output data that is used as the optimal target that the different models are trying to imitate. The datasets will also include an adjacency matrix that will describe the topological structure of the electrical power grid.

To simulate the power flow in an electrical power grid and make the necessary calculations, the powerful python library **PandaPower** [30] was used, a network calculation program that aims at analysis and optimization in power systems.
Through **PandaPower**, the network examples can be loaded and the next step after that, is to fix boundaries for the voltages of each node. In this work, the lower bound was fixed to 0.9(p.u) and the upper bound to 1.1(p.u).
Since the values are fixed for the test-case networks, it is important to have for each sample of the dataset different values for the loads. To overcome this, a reference is randomly sampled using the uniform distribution between two values and it is then multiplied by the load of the test case network. The boundaries that are set for the reference determine the closeness of the load samples to one another.

To build the input feature matrix, that includes only the known variables to the system, it is important to classify the nodes of the network into three categories where each category has two variables known and two variables unknown.
Any node in an electrical power system, can either be a PV-node, a PQ-node or a slack node [5] and generally only two out of the four feature variables $[V, \theta, p, q]$ are known before computing a power flow analysis depending on the type of the node. The classification of known and unknown variables is presented in table (3.1).

- PQ-nodes are the load nodes they are nodes that only consume power so the real and reactive-power are supposed to be known.
- PV-nodes also called generator nodes are nodes that generate power and therefore have values of real Power and voltage magnitude known.
- Slack-node or swing-node is generally one node and it is necessary for power flow calculations to keep balance between active and reactive power transmissions considering the losses that are yet to be calculated. So until the final values are fixed, the slack node either provides or absorbs power from the different lines to cover for the losses. It is the only node in the network to have the angular voltage value to be known [3].

Type of node	**known**	**unknown**
Load Node (PQ node)	P, Q	$\|V\|, \theta$
Generator Node (PV node)	$P, \|V\|$	Q, θ
Slack or Swing Node	$\|V\|, \theta$	P, Q

Tab. 3.1: Known and unknown variables for each type of node

After the nodes are classified, for each node depending on its type, the known values are inserted in their assigned column in the following order $input = [V, \theta, p, q]$.
For PV_{nodes} , the total power is the difference between the generated power and the load in that node while for PQ_{nodes}, power will be only consumed.

$$p_v = [p^G]_v - [p^L]_v, \quad v \in PV_{nodes} \tag{3.1a}$$
$$q_v = [q^G]_v - [q^L]_v, \quad v \in PV_{nodes} \tag{3.1b}$$
$$p_v = -[p^L]_v, \quad v \in PQ_{nodes} \tag{3.1c}$$
$$q_v = -[q^L]_v, \quad v \in PQ_{nodes} \tag{3.1d}$$

In order to add a helpful clue to the prediction models, another column is added to the matrix. For each node in the network, this column holds a value that indicates the type of the node from the three types cited in the table (3.1).
The resulting input features are built in the following form: $input = [V, \theta, p, q, Type]$ with shape ($N\times$ 5).
The output of the dataset, which in this case, is the optimal solution that we are trying to imitate, can be calculated using the **PandaPower** framework. **PandaPower** offers the possibility to run an optimal power flow calculation using the interior point solver by calling the method **pandapower.runopp(network)** and then to extract the values of voltage magnitude and angle along with active and reactive power by calling the method **network.res_bus['values']** resulting in a matrix with shape ($N\times$ 4).

In order to produce a large dataset, the steps conducted to generate one sample of input and output, are repeated for a large number of times with discarding the samples that failed to converge when calculating the OPF values.

One final part has to be included in the datset before it is ready and it is the adjacency matrix that will be used in the graph convolutional model to describe the topology of the network.
As explained in section (2.3.1), the weighted adjacency matrix holds values for the connections between two adjacent nodes and zeros if the nodes are disconnected. In an electrical power system, a matrix with such characteristics already exists and is called node admittance matrix. This matrix holds admittances for two physically connected buses and it can be extracted from the network using the **PandaPower** method **network._ppc['internal']['Ybus']**. Algorithm 2 shows the steps performed to generate a dataset for a case network.

Algorithmus 2 : Data generator algorithm

Input : Number of samples S, network net, the set of nodes in the network $\mathcal{V}$

Output : Dictionary Dataset

for s *in* S **do**
  assign values to the loads
  create input matrix with shape $N \times 5$: X_{init}
  create output matrix with shape $N \times 5$: Y_{opt}
  for v *in* $\mathcal{V}$ **do**
    if $v \in net.loads$ **then**
      $X_{init}[v,2] \leftarrow -[p^L]_v$
      $X_{init}[v,3] \leftarrow -[q^L]_v$
      $X_{init}[v,4] \leftarrow 0.02$
    else if $v \in net.generators$ **then**
      $X_{init}[v,0] \leftarrow |V|_v$
      $X_{init}[v,2] \leftarrow [p^G]_v - [p^L]_v$
      $X_{init}[v,4] \leftarrow 0.03$
    else
      $X_{init}[v,0] \leftarrow |V|_v$
      $X_{init}[v,1] \leftarrow \theta_v$
      $X_{init}[v,4] \leftarrow 0.04$
  $Y_{opt} \leftarrow$ OPF results
Y $\leftarrow$ net.admittance matrix
Dataset = $[X_{init}, Y_{opt},$ Y]
return Dataset

3.1.2 Data Preparation

3.1.2.1 Data Splitting

Before starting the training process, the data set has to go through a few additional steps. The first consists of splitting the data in three parts. Train, validation and test sets.

The train set is to be used during the training part and it is usually the set that has the largest number of samples so that the model is able to train on a wider range of samples from the dataset.

The test set is composed of samples unknown to the models. These samples are presented to the different models after being trained to see how they react to unseen data and therefore to compare their performance.

The validation set is an optional set that can be used to visualize the performance of a model along the training process, to have an understanding on how the different model parameters influence the training and also to have a sense of the degree of complexity the model needs to have.
The validation set, similar to the test set is composed of data unknown to the model. This process

is important because one can also compare the training results with the validation results and based on this, can check whether the model is overfitting or underfitting which are both not ideal characteristics for any model.

A very complex model, can lead to overfitting to the training data. This means the model is so complex that it is recognizing and learning patterns in the training data that are only relevant to the training data. Having that, leads the model to perform very well on training data but fails to converge for unseen data. Having a shallow model is also not preferred, as in that case, the model is said to be underfitting which means it is failing to capture the deep and relevant patterns hidden within the data.
This compromise between overfitting and underfitting is called the bias-variance tradeoff [31], where bias is the difference between the average of the predictions and the correct values while variance is the variability of the model predictions. A model having a high bias, tends to oversimplify relations within the data which leads to high errors wheras a model that has high variance over-adjusts to the data and therefore fails to generalize well. Figure (3.3) illustrates the bias-variance-tradeoff.

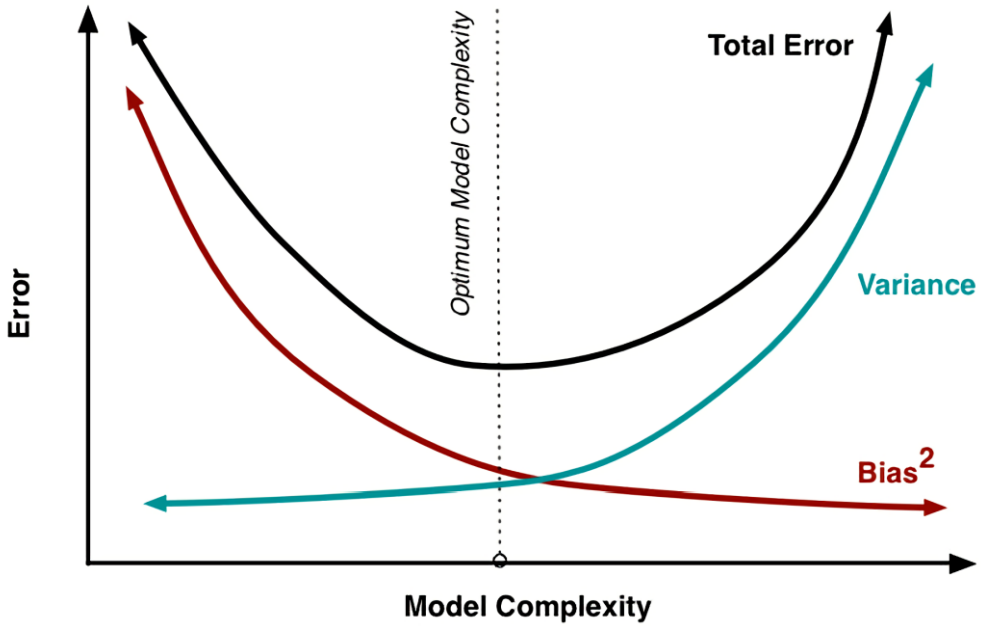

Fig. 3.3: The bias-variance-tradeoff [31]

3.1.2.2 Data Scaling

The generated data that was extracted from the electrical power grid and split into three sub-datasets needs to go through another additional step before it is ready to be fed to the different models, the scaling process.
The features that will be used to describe the state of the nodes in the network have different magnitude ranges and this can cause the model to be more influenced by high values while ignoring the smaller ones. Therefore, scaling is used to bring all different features down to the same value range while maintaining the same general distribution and ratios as the original data.
For this purpose a **StandardScaler()** was used for scaling the input data and a **MinMaxScaler()** for

scaling the output data. Both of these scalers can be found in the **sklearn.preprocessing python library** [32].
The **StandardScaler()** transforms the data such that the distribution will have a mean value of 0 and standard deviation of 1. Given the distribution of the data, each value x in the dataset will have the mean value subtracted, and then divided by the standard deviation of the whole dataset.

The **MinMaxScaler()** transforms all feature variables to the range [0, 1] by subtracting the smallest value in the dataset from the feature value to transform and then divide by the range between the maximum and minimum value. The mathematical formula of the **MinMaxScaler()** is given by:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.2}$$

Using different scalers for input and output data does not affect the performance of the model in a bad way. The opposite was noticed as having the above cited combination, resulted in the best results.
Figure (3.4) shows a plot of the active power values for node number eight and node number four for 25 different samples in the 14-nodes network before and after scaling using the **MinMaxScaler()**. As can be seen in the figure, the feature values before scaling have a wide range and after scaling all values were downsized to the range [0, 1].

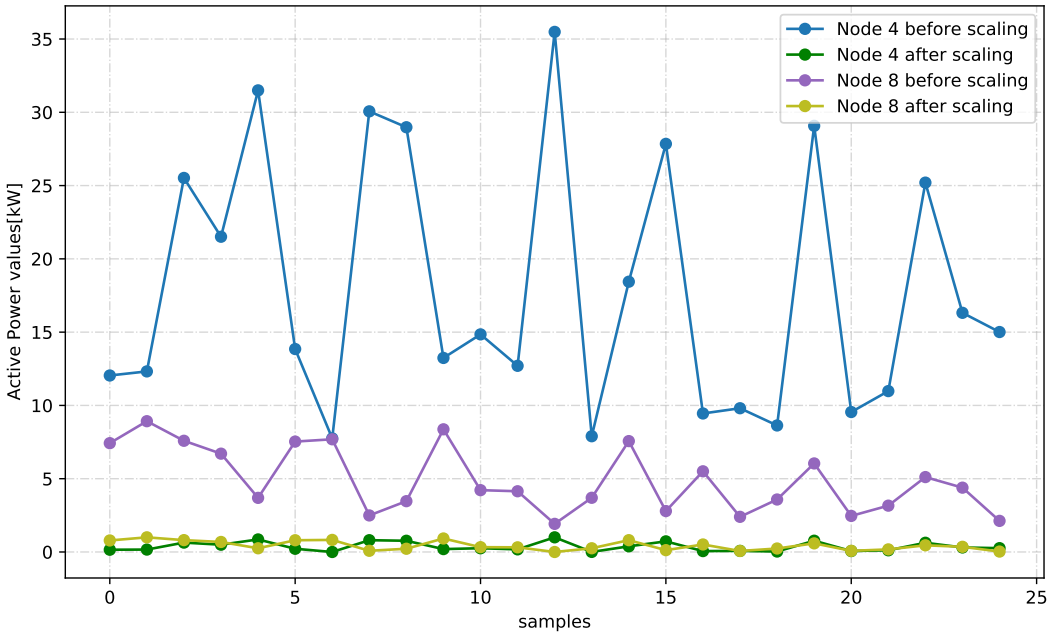

Fig. 3.4: Plot of samples from output data before and after scaling

Chapter 4

Implementation

In this chapter, we present the implementation of the different models. The chapter includes a description of the architectures and states the steps performed during the training process as well as the optimization approaches that were used.

4.1 Model Selection

4.1.1 Feed Forward Model

A feed forward neural network, also called fully connected neural network is one of the most basic architectures of deep learning. It proved to be very efficient when applied to tasks like failure detection, or material mechanical property detection [33]. This architecture was introduced by Rosenblatt [14] as a succession to linear regression models that were unable to learn non-linear mappings between inputs and outputs. His idea was to introduce a number of hidden layers between input and output layer as intermediate predictors that can learn the mapping function to solve in a graduate way.
A fully connected neural network can be seen as a stack of multiple linear regressions and to allow non-linearity in the model, a non-linear element-wise operation is added for each unit of the layer before passing its value to the next layer. These are the activation functions.

In a feed forward neural network, all neurons in a layer are bound to all the neurons of the next layer which means it learns features from all the combinations of features from the previous layer. However, in some cases a technique called dropout can be used which consists of randomly cutting off some of the connections between neurons during training to make the learning less vulnerable to overfitting to the training data.
In this work, the feed forward architecture will be used as a reference to assess the improvements that can be acquired by using a geometrical based model that includes a graph in the learning process.

4.1.2 Graph Convolutional Model

The main purpose of this work is to study the performance of a geometric deep learning model that, not only makes use of the features that describe the state of the nodes, but also includes a graph in the learning process describing the topological structure of the grid. Combined with the powerful feature extraction abilities of the convolutional filter, graphs have proven to be very efficient in many applications of deep learning by incorporating the relational features in the learning process.

In this work, different graph convolutional architectures were investigated and trained such as the architecture proposed in the paper Topology Adaptive Graph Convolutional Networks [34] and also the one proposed in the paper Simplifying Graph Convolutional Networks [35]. However, the architecture that showed the best results and the one that is further investigated in this thesis, is the one proposed by W.L Hamilton et al. [36] in the paper Inductive Representation Learning on Large Graphs where the autors called the architecture: GraphSAGE.

4.1.2.1 Architecture

GraphSAGE is a representation learning technique that generates embeddings for nodes in a graph in the spatial domain by training a set of aggregator functions. These functions learn to collect feature information from the local neighborhood of a node gradually by using different search depths for each graph convolutional layer. These functions however, must be invariant to permutations of node orderings such as the functions proposed by the authors: the mean, pool and lstm aggregators.

Learning aggregator functions instead of just learning weights, makes the model more suitable for generating embedding for unseen nodes or new sub-graphs. This particularity is what makes the algorithm inductive. Inductivity means the model can be trained with some of the nodes in the graph missing but still be able to generate embedding for new nodes when tested. This ability can be very useful in situations where systems often encounter new nodes or have evolving graphs. This inductivity approach in generating embeddings, also makes models more favorable to generalize to other graphs of the same nature.

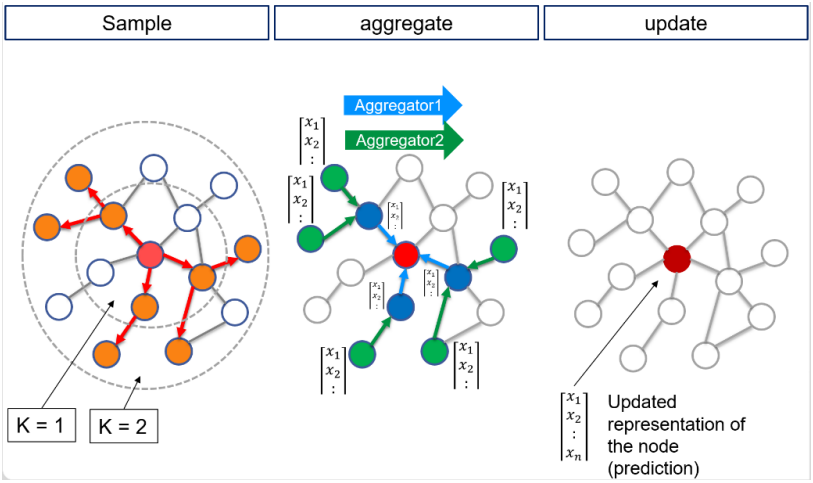

Fig. 4.1: Illustration of the GraphSAGE feature collection process

4.1.2.2 Forward propagation algorithm

Algorithm 3 shows the learning process or the forward propagation process of GraphSAGE. The algorithm can be divided into three steps. Sampling the neighborhood, aggregating the features of the neighboring nodes and finally update the representation of nodes.
Sampling means fixing the depth of the search. This is represented by the outer loop of algorithm 3, for example if $k = 1$, the algorithm samples for each node, the neighbors that are located one hop

away. The number of neighbors that are sampled for a certain depth is also fixed in order to keep a low computational footprint.

Second step is aggregating the features. All nodes in the sampled neighborhood send their node feature representations to be aggregated into one single vector $h^k_{N(v)}$. These aggregated representations are the representations generated for each node at the previous iteration of the outer loop. For $k = 0$, the representations are the initial input features held by each node.
The last step, is updating the representations of the nodes. This is done by concatenating the current representation (from the last iteration) with the vector holding the aggregated representations of the neighbors. It is then assigned a weight value and fed to a non-linear activation function.

Algorithm 3 works in an incremental manner. This means it starts by updating the representation of every node depending on the local neighborhood and as the process iterates, it is gaining more and more information from the nodes that are further located in the graph and is therefore gaining more knowledge about the structure of the network.
The last representation for each node, noted in algorithm 3 by z_v , is the representation acquired in the last search depth ($k = K$) and it represents the output of the neural network and therefore the predicted optimal state for each node.

Algorithmus 3 : GraphSAGE forward propagation algorithm

Input : Graph $\mathcal{G}(\mathcal{V},\mathcal{E})$,input features $\{x_v,\forall v \in \mathcal{V}\}$, depth K,
weight matrices $W^k,\forall k \in \{1,\dots,K\}$,
non-linearity σ, differentiable aggregator functions AGGREGATE$_k,\forall k \in \{1,\dots,K\}$,
neighborhood function $\mathcal{N} \rightarrow 2^v$

Output : Vector representations z_v for all $v \in \mathcal{V}$

$h^0_v \leftarrow x_v, \forall v \in \mathcal{V}$
for k *in* K **do**
  for v *in* $\mathcal{V}$ **do**
    $h^k_{\mathcal{N}(v)} \leftarrow$AGGREGATE$_k(h^{k-1}_u \forall u \in \mathcal{N}(v))$
    $h^k_v \leftarrow \sigma\left(W^k \cdot CONCAT(h^{k-1}_v, h^k_{\mathcal{N}(v)})\right)$
  $h^k_v \leftarrow h^k_v / \|h^k_v\|^2$
$z_v \leftarrow h^K_v, \forall v \in \mathcal{V}$

4.2 Model Building

4.2.1 Feed Forward Model

To implement the different models, the framework **pytorch** [37] is used. **Pytorch** is an efficient and easy to use framework that enables the inteoperability with the rich ecosystem of the python libraries. In **pytorch**, layers and models can be expressed as python classes whose constructors enable the creation and initialization of parameters and layers. These classes include a forward method where layers can be stacked to build networks.
The **pytorch.nn** library provides linear layers that can be employed as fully connected layers. These linear layers when stacked, form a fully connected model. they require input and output size to

be entered as argument for initialization along with a boolean parameter that specifies whether a bias term is to be taken into account. The network is composed of four linear layers with ascending number of neurons in each layer from one layer to the next and each layer except the last layer is followed by a Relu activation function.

4.2.2 Graph Convolutional Model

To implement the graph convolutional model, we worked with the deep learning framework **DGL** [38] with a pytorch backend. **DGL** is short for Deep Graph Library and it is a powerful framework that revolves around the usage of graphs in deep learning model. **DGL** supports arbitrary message passing computation over graphs which means communication between the nodes. In addition, **DGL** is very memory and time efficient and contains a variety of built-in layers based on architectures developed in different papers that designed algorithms for deep learning purposes.

The first step before implementing the graph convolutional model was to convert the adjacency matrix, extracted from the test-case networks, into a **DGL** graph as the convolutional layers require a **DGL** graph along with the input tensors. This can be made using the method **DGL.from_scipy(Y)** where Y is the adjacency matrix.

The architecture GraphSAGE is included in **DGL** under the name **SAGEConv()**. To initialize the layer, input and output sizes have to be specified as well as the aggregator type, it is also possible to activate a bias for the layer.

To incorporate the parameter K, described in section (4.1.2.2), into the model, the power of the adjacency matrix is used . When squared, the adjacency matrix becomes a two hop adjacency matrix. This means the matrix holds a values for a connection between two nodes, if the nodes are two hops away from each other. The same process continues for the cube of the adjacency matrix which translates to a three hop distance between the nodes.

The graph convolutional model is composed of three GraphSAGE layers. The first layer is built using the adjacency matrix and is followed by the sigmoid activation function. A new graph is then constructed using the square of the adjacency matrix and fed to the second layer along with the output features of the first layer. The same process is repeated with the third GraphSAGE layer with a graph built with the cube of the adjacency matrix.

After the three convolutional GraphSAGE layers followed each by the sigmoid activation function, a fully connected layer is added to the network representing the output layer. Since the convolutional neural networks are localized and rely on small regions to extract the features, this fully connected layer will add globalization to the network by dealing with the different combination of high level features extracted from the convolutional layers in order to output the estimation for the feature values at each node of the graph [39].

4.2.3 Model Training

Some functions have to be initialized before starting the trainig process, the optimizer and the loss function. The optimizer is responsible for computing the gradient descent algorithm and updating the weights while the loss function will compute the loss after each run through the neural network.

For the optimizer we use the Adam optimizer [40], an efficient method for stochastic optimization that only requires the first order gradients of the loss function. It is an extension to the stochastic gradient descent algorithm and its particularity is that for every step, it is computing individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients. This makes the magnitude of parameters updates invariant to the rescaling of the gradient and leads to a faster convergence.
For an optimizer, a learning rate has to be entered as parameter as well as the exponential decay rate for the first and second moment estimates and also an epsilon value used to prevent any division by zero in the implementation.
Since before training a model, the best learning rate is unknown, a random value is given to then be optimized to the needs of the model. for the moment estimates, the values 0.9 and 0.999, suggested by the authors of the adam paper [40], were used.

The loss function that was used during training is the Mean Squared Error(MSE). Its main advantage is that by squaring the error, it penalizes the high deviations more severely which leads to a better correction of the weight values.

A convenient way to load the different datasets during any part of the implementation is to use data loaders. The data loader is a tool provided by **pytorch** and it offers the possibility to store pre-processed parts of the dataset in the form of tensors, for instance the training, validation and test datasets. A batch size is specified along with other arguments such as the possibility to shuffle the data. The different datasets are then easily loaded and handed over to the model in the form of batches with the size specified within the data loader.

The training process requires almost the same steps for both models. The difference lies in the fact that the input for the graph convolutional model includes a graph along with the feature matrix. Apart from that, the input features have to be reshaped to fit the input shape of the model.
The training steps generalized for both models, are the following:

1. For each epoch, the samples are loaded using the training-data-loader in the form of batches.
2. For each batch, the input tensors are reshaped to match the appropriate shape required by each model.
3. The input is fed to the model in order to make a prediction.
4. A loss is calculated using the MSE loss function.
5. The gradients of the optimizer are cleared.
6. The loss is back propagated through the network using the adam optimizer.
7. The parameters of the model are updated.

These steps are also summerized in algorithm 4.

Algorithmus 4 : Training algorithm

Input : Number of epochs E, number of batches B, the model *model*, the input for the model *input*

Output : Weight matrix W

```
for e in E do
    for x, y in B do
        a prediction is made : ŷ = model(input)
        a loss is calculated L(y,ŷ)
        the gradients are set to zero ∇L := 0
        the loss is backpropagated
        the parameters are updated W
return W
```

To accelerate the training of the different models, the software platform **CUDA** [41] is used. **CUDA** stands for "Compute Unified Device Architecture" and it is a software platform provided by **Nvidia**. **CUDA** offers the possibility to control GPUs and by doing so, computationally intensive sections of programs are moved to the GPU which shortens the computational time considerably. Moreover, it is a requirement for the Deep Graph Library **DGL**, to use a GPU.

4.3 Model Optimization

4.3.1 Regularization

Regularization in its general definition, include any additional technique that aims at making a deep learning model generalize better on the testing dataset. Some of the most commonly used regularization techniques are dropout, lasso and ridge regularizations also called respectively L_1 and L_2 regularizations.
These techniques did not show any improvement in the performance of the graph convolutional model. However, applying dropout on the feed forward model slightly increased the performance.

Dropout is a technique used in neural networks with a large number of parameters in order to address the issue of overfitting [42]. It consists of temporarily removing some of the units in a network during training along with their incoming and outgoing connections. The choice of the neurons to be dropped occurs randomly by setting a probability p which in our case was set to $p = 0.5$.

4.3.2 Bayesian Optimization

After selecting and training a neural network model, there is a number of parameters that play an important role in the performance of the model. These parameters are called hyperparameter and they include the number of epochs to run, the number of layers in a network, the learning rate and can also include regularization parameters and many other variables that influence the learning process of the model.
These hyperparameters have to be tuned in order to find the best set that improves the performance of the model. The idea is to run many combinations of different hyperparameter until this set of parameters that minimizes an objective function is found.

One way to achieve this is to run a grid search which is a brute force method that computes all possible combinations of hyperparameters. However, when the objective function in question requires training a neural network, this can become a very computationally expensive task.
In this work, the bayesian optimization technique is used [43]. It is also a search method that aims at finding the parameters for which an objective function, in our case the error function, reaches its minimum.
What differs the bayesian optimization from other search procedures, is that it builds a probabilistic model for the objective function to minimize and then based on this model, makes decisions about the next set of parameters to evaluate, while taking into account uncertainty.
When performing a bayesian optimization, two major decisions have to be made. The first is the choice of a prior to the objective function that would make the assumptions about the function to be optimized. The most used prior is the Gaussian process due to its flexibility and tractability. The second decision to make is the acquisition function. It is used to build a utility function from the model posterior in order to determine the next evaluation point.
Although this technique includes extra computation, this extra computation is well justified as it contributes to making better decisions in the search procedure.

Chapter 5

Results

In this chapter, the results obtained after training the implemented models are presented and compared. The chapter also outlines the strengths and weaknesses of both models.

5.1 Prediction Results

In order to reach the best results, a bayesian optimization algorithm is used to combine multiple settings of hyperparameters. The resulting parameters for each model are listed below along with plots comparing the predicted feature values and target values.

5.1.1 Feed Forward Model

The feed forward model is architected using four layers each followed by the ReLU activation function and a fully connected output layer and the parameters of the models are the following:

parameter	**value**
Training Loss function	ReLU
Learning rate	$7.66 \cdot 10^{-4}$
Batch size	32
Number of Epochs	2000
Optimizer	Adam
Dropout probability	$p = 0.5$

Tab. 5.1: Hyperparameters of the Feed Forward model

Figure (5.1) is composed of four plots, one for each of the four feature variables describing the state of the nodes in the network. The different plots include the values predicted by the feed forward model along with the optimal target values for one samples for all nodes in the 14-nodes network. The results show that the model is able to recognize patterns from the data as the predicted values, represented with red dots in the plots, follow the same trajectory as the target values, represented with blue stars. However the prediction of the unknown values does not seem to be very accurate.

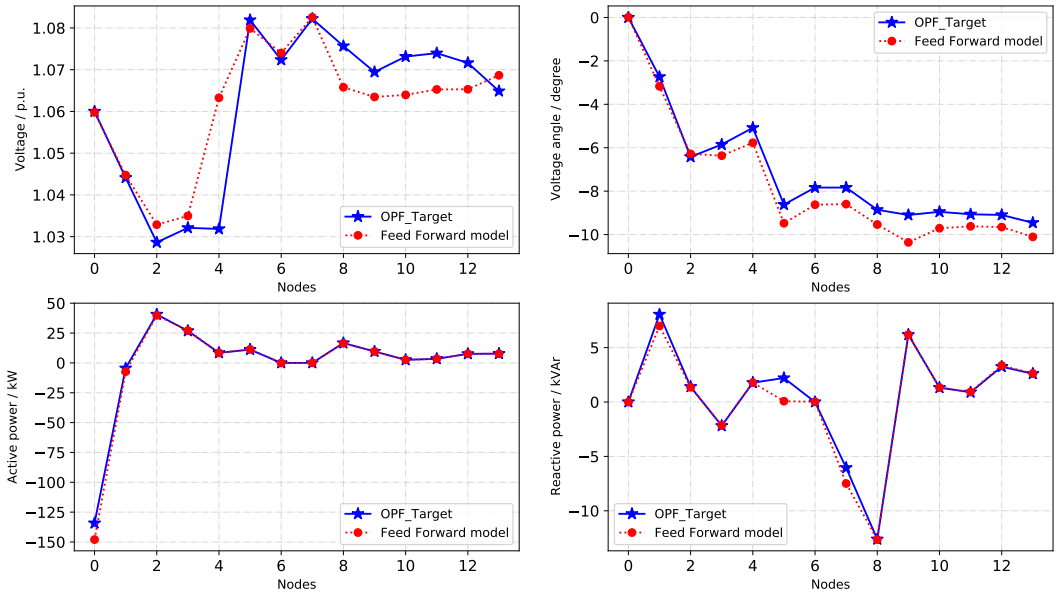

Fig. 5.1: Plot of the feature variables predictions for the Feed Forward model

5.1.2 Graph Convolutional Model

For the graph convolutional model, we build a network with three GraphSAGE layers each followed by the sigmoid activation function and a fully connected layer as output layer. The rest of the used parameters are summarized in table (5.2):

parameter	**value**
Training Loss function	Sigmoid
Learning rate	$8.24 \cdot 10^{-4}$
Batch size	16
Number of Epochs	2000
Optimizer	Adam
Aggregator function	pool
Depth of the search K	3

Tab. 5.2: Hyperparameters of the Graph Convolutional model

Figure (5.2), like figure (5.1), includes the plots of the feature variables describing the nodes in the network where each plot displays the perdicted and the optimal target values. In comparison to the results from the feed forward model, a clear improvement in the accuracy can be noticed as the red line with dots representing the predictions, not only follows the trajectory of the blue line with stars representing the target values, but a precision in the prediction can also be noticed as the two line are almost perfectly overlapping.

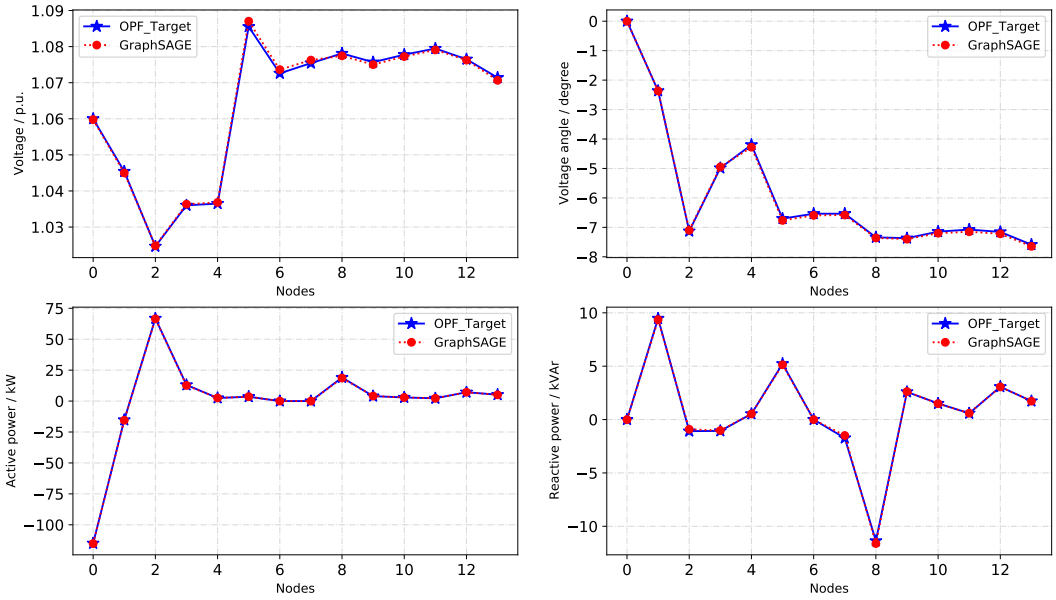

Fig. 5.2: Plot of the feature variables predictions for the Graph Convolutional model

5.2 Comparison of Mean Absolute Error

The Mean Absolute Error (MAE) is a measure of calculating the deviation of the predicted values from the expected target. The smaller its value, the better are the predictions. We evaluate the models using the MAE rather than the widely used Root Mean Squared Error (RMSE) because the variability of the RMSE function depends on the variation of several characteristics of error rather than just the average error and this can lead to an ambiguous interpretation of the results [23].
In order to evaluate the models and to compare them to one another, three experiments are conducted based on the computed MAE values. The two first experiments consist of comparing the MAE's of the models for each feature variable separately, one by training the 14-bus network and the second by training a larger network with 39 nodes. For the third experiment, the reconstruction abilities are tested and thus by progressively hiding nodes from the network during training.

5.2.1 MAE of Variables

The 14-nodes network, composed of 4 generators, 9 loads and a slack node, is a relatively small network. However, selecting models, training them and searching for the best hyperparameters, can be a daunting task and takes a very long time even when a GPU hardware accelerator is used, and considering that the time needed to train a neural network increases with larger networks, we opted for a small network to do most of the model building process.

The plot in figure (5.3), shows the MAE of the four feature variables for both models. We note that the hardest variable to predict, is the active power p due to the wide range of values it can have which may lead to imprecision in the predictions. The results are presented in table (5.3).

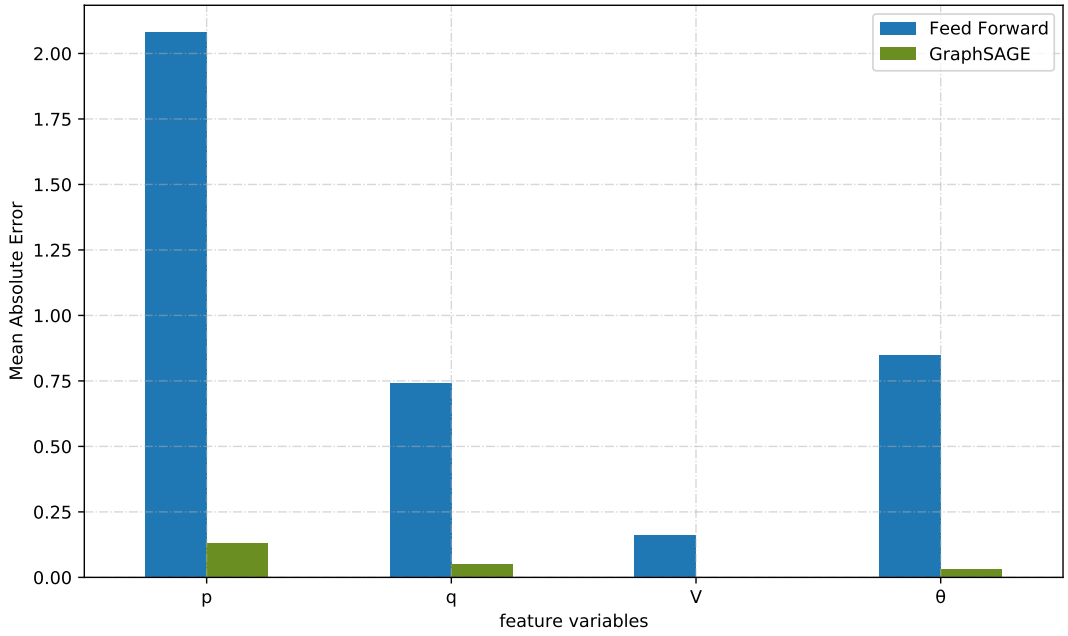

Fig. 5.3: Comparison of the Mean Absolute Error for all feature variables for the 14-nodes network

Model	voltage magnitude	voltage angle	active power	reactive power
Graph Convolutional	0.0006	0.03	0.13	0.05
Feed Forward	0.16	0.85	2.08	0.74

Tab. 5.3: Mean Absolute Error of feature variables for the 14-nodes network

5.2.2 Training on a Larger Network

To test and further compare the results of both models, we train on a 39-nodes network composed of 10 generators, 28 loads and one slack bus. The graph convolutional model is still outperforming the feed forward model, however, the results of both models reveal, an increase in the error for all the feature variables. Table (5.3) as well as figure (5.4) show the results from the experiment.

Model	voltage magnitude	voltage angle	active power	reactive power
Graph Convolutional	0.004	0.42	3.89	2.15
Feed Forward	0.01	2.16	9.23	4.28

Tab. 5.4: Mean Absolute Error of feature variables for the 39-nodes network

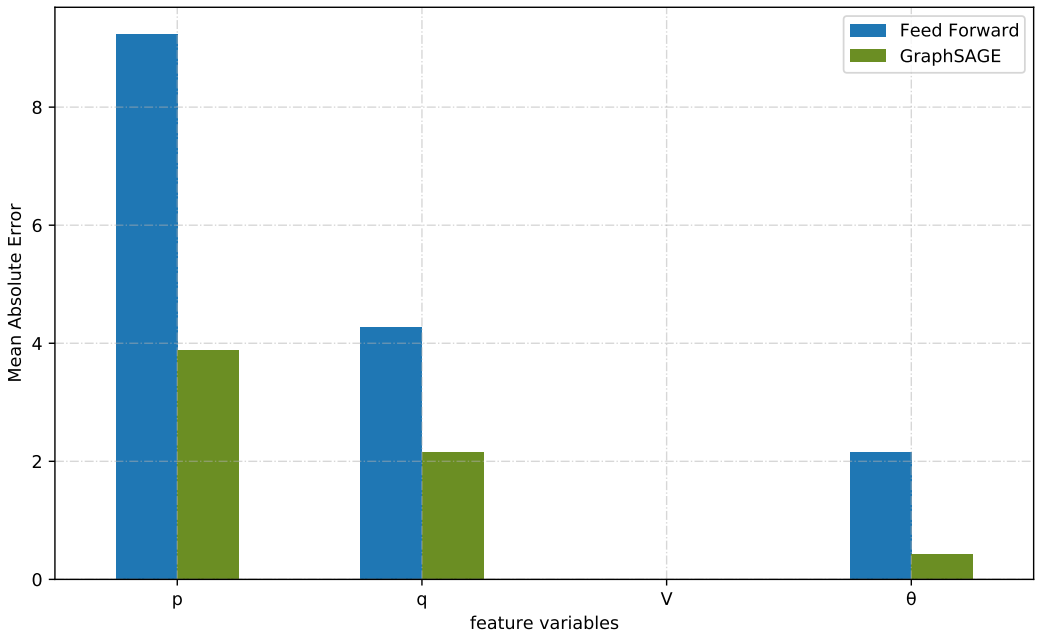

Fig. 5.4: Comparison of the Mean Absolute Error for all feature variables for the 39-nodes network

5.2.3 Training with Missing Nodes

In the previous two experiments, the graph convolutional model outperformed the feed forward model. For that reason, the third experiment will only include the graph-based model.
This experiment consists of training the model, while hiding the features of percentages of the nodes and the model still has to make predictions for all nodes in the network.

We start by randomly hiding 10% of the nodes then 20% and so on until 100% of the nodes are hidden which translates to an input feature matrix with all values unknown. The model is then trained and tested by computing the global MAE which is an average of the error values from all feature variables.
In real life scenarios, it can occur that the feature variables of some nodes in the network are unavailable. For that reason, this experiment is performed to evaluate the reconstruction abilities of the graph convolutional model.
Figure (5.5) is a plot that presents the results obtained from the experiment.

These result show a gradual increase of the error every time 10% more of the nodes are hidden. An average leap of 0.21 in the MAE value can be noted indicating that the model is affected by leaving nodes out during training. This experiment demonstrates that the model performs much better when all the features are available during training.
We also note that hiding nodes does not affect particularly the predictions of the hidden nodes, the effect can be seen on all nodes in particular generator nodes which are the most sensitive nodes in the network.

Nonetheless, the prediction results can still be considered acceptable despite having an increase in the error value. Figure (5.6) displays the predictions of the model, having all 14 nodes in the network hidden during training and testing. It is visible from the plots that, regardless of the lack of precision, the model still has a good sense of the direction to follow as both predictions and target values follow the same track and the model does not diverge completely.

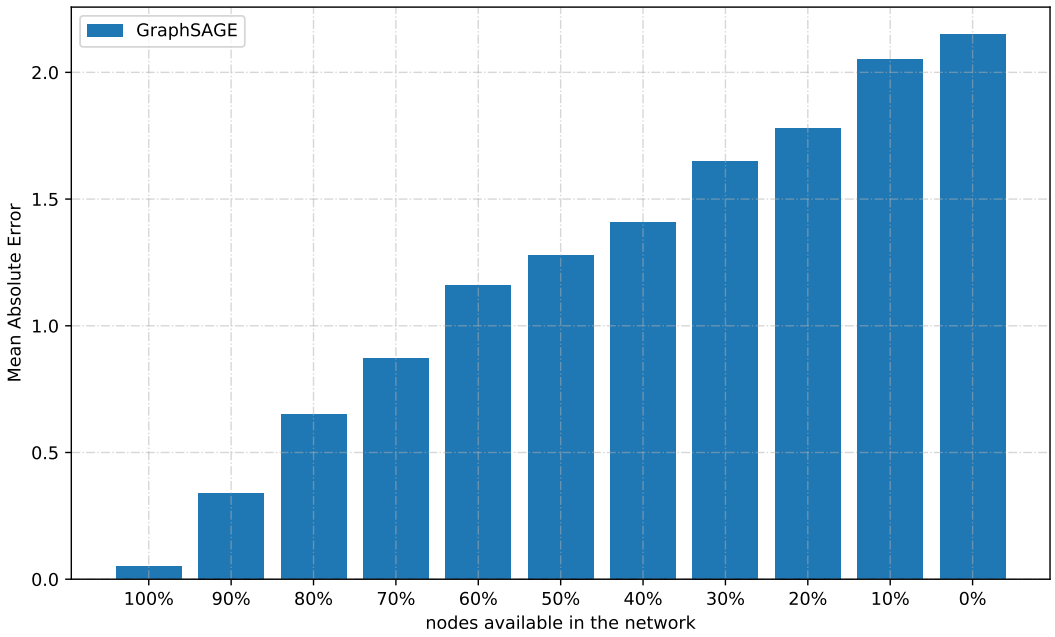

Fig. 5.5: The global Mean Absolute Error with hidden features of nodes

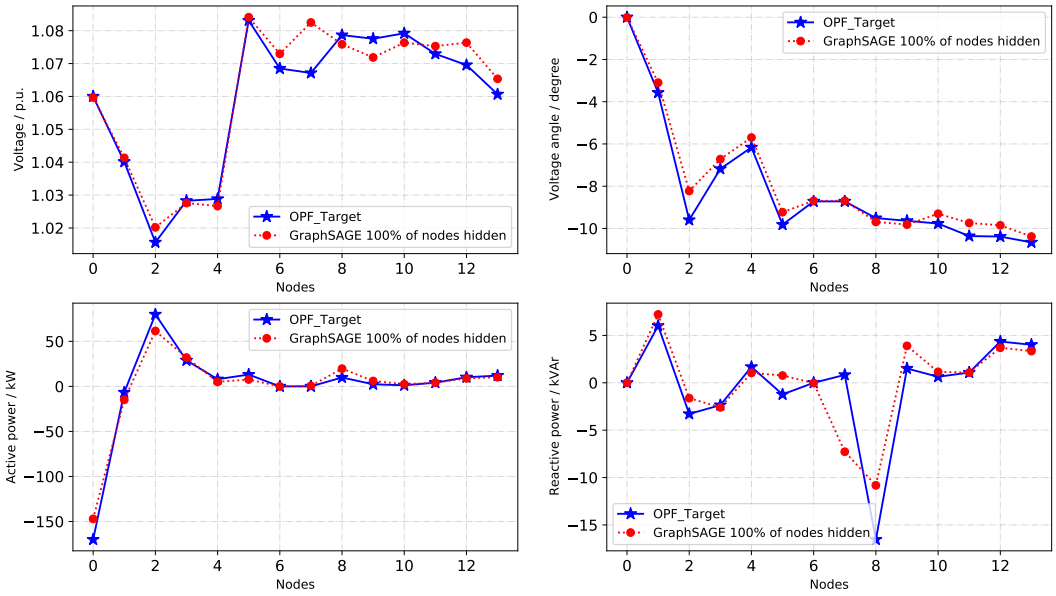

Fig. 5.6: Plot of the predicted feature variables of the convolutional model trained with 100% of nodes hidden

5.3 Time Comparison

In this work, the aim is to predict the Optimal Power Flow for a given network by using a method based on geometric deep learning which is building a graph convolutional neural network and with doing so, including the topological structure of the grid in the model.

For this purpose, a supervised learning approach was followed which means the values that were predicted can only be as accurate as the values of the output dataset that was extracted using the framework **Pandapower** that calculates the OPF values using the interior point method (IPM).

When considering the time needed to make the OPF calculations for one sample, the IPM from **Pandapower** takes approximately 1.092 seconds for the 14-bus-test-case network and 1.877 seconds for the 39-nodes network. However, using the artificial neural network methods developed in this work, making the OPF calculations takes approximately 0.005 seconds for the 14-bus-test-case network and 0.008 seconds for the 39-nodes network. This means the artificial intelligence architectures can make calculations around 200 times faster than the traditional IPM solver. These results are summarized in table (6.2) in the next chapter.

Chapter 6

Conclusion

This chapter summarizes the main steps and the results that were reached in this work. We also provide a conclusion and an outlook to future work.

6.1 Overview and Conclusion

In this thesis, we addressed the Optimal Power Flow problem in order to tackle a challenge faced by the energy sector, providing a secure, stable and optimal distribution of power over all the system. For that purpose, we explored a deep learning tool called artificial neural networks and we investigated and noted the benefits of using graph convolutional neural networks in order to predict the OPF values for an electrical power grid starting from a state that includes only the fixed values known to the system before running a power flow.

For this purpose, a data generator was implemented with the purpose of creating different datasets for two test case networks: the 14-bus and the 39-bus test cases. These datasets were used to train, test, compare and evaluate the results of two different models: The feed forward model and the graph convolutional model.

The test results from chapter 5 showed that the graph convolutional model outperforms the feed forward model in all accuracy tests. The results of the global MAE are presented in table (6.1).
We can conclude that this is due to the integration of the graph representing the power grid in the training process of the model and also thanks to the powerful feature extraction abilities of the convolutional filter.

Indeed, the graph convolutional model was able to capture the patterns hidden within the features of the nodes but also the relational patterns underlying the graph that were primordial for producing accurate predictions for the electrical power grid.

We note however, that the accuracy of the predictions decreases with larger networks having a higher number of nodes and edges. The increase of the number of units and connections in the network, adds complexity to the electrical power grid which possibly requires building models more precise able to deal with these more developed networks in order to produce better results.

The reconstruction abilities of the graph convolutional model were also tested and this was by hiding nodes from the network during training and the results showed that the model produced relatively good results with up to 50% of the node features present in the input.

Also to be noted is the computation time of both artificial neural network models that is around 200 times lower than the OPF computation time when performed by **PandaPower** with the Interior point method.
This improvement in computational speed compared to traditional methods, can be a valuable asset for some applications of the OPF such as the Real-Time-OPF where the calculations have to be made up to every six seconds. The numerical results are presented in table (6.2).

Architecture	**14-bus-testcase**	**39-bus-testcase**
Graph Convolutional	0.05	1.61
Feed Forward	0.95	3.92

Tab. 6.1: Global Mean Absolute Error

Architecture	**14-bus-testcase**	**39-bus-testcase**
Graph Convolutional	0.005 s	0.008 s
Feed Forward	0.002 s	0.004 s
PandaPower OPF	1.092 s	1.877 s

Tab. 6.2: Comparison of computation time

6.2 Outlook

This work can be considered a further step in understanding and developing techniques that can be helpful in the optimization of the distribution of stable energy throughout the entire system.
The conclusions that were made, hopefully open the door a bit further for future studies and research where more complex and detailed models can be designed to consider other formulations for the Optimal Power Flow, involving more constraints that would make the calculations closer to the real physical conditions of the OPF.

Artificial Neural Networks also have the possibility to take the predictions to a higher level where a semi-supervised or a completely unsupervised setting can be constructed that will eventually be independent from calculations made by other methods and will possibly be able to make more accurate and optimal calculations for the electrical power distribution.

Moreover, extracting features from the electrical grid using graph convolutional neural networks, not only is useful for optimization tasks but can also serve other purposes such as security in power grids where line failures can be detected.

Hopefully, artificial intelligence will continue to improve and grow in the right direction in order to solve other existent world problems and also the ones that we are yet to face.

Figures

Tables

Bibliography

[1] S. Surender Reddy and P. R. Bijwe, "Day-ahead and real time optimal power flow considering renewable energy resources," *International Journal of Electrical Power & Energy Systems*, vol. 82, pp. 400–408, 2016.

[2] E. Mohagheghi, M. Alramlawi, A. Gabash, and P. Li, "A survey of real-time optimal power flow," *Energies*, vol. 11, no. 11, p. 3142, 2018.

[3] M. B. Cain and R. O'Neill, "History of optimal power flow and formulations," 2012.

[4] Y. Tang, K. Dvijotham, and S. Low, "Real-time optimal power flow," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2963–2973, 2017.

[5] T. Leibfried, *ELEKTRISCHE ENERGIENETZE (EEN)*. WS 2017/2018.

[6] Richard O'Neill, Anya Castillo, and Mary Cain, "The iv formulation and linear approximations of the ac optimal power flow problem: Optimal power flow paper 2," 2012.

[7] A. Gabash and P. Li, "Active-reactive optimal power flow in distribution networks with embedded generation and battery storage," *IEEE Transactions on Power Systems*, vol. 27, no. 4, pp. 2026–2035, 2012.

[8] A. Gabash and P. Li, "Flexible optimal operation of battery storage systems for energy supply networks," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 2788–2797, 2013.

[9] D. Owerko, F. Gama, and A. Ribeiro, "Optimal power flow using graph neural networks," 10 2019.

[10] K. Dvijotham, M. Chertkov, and S. Low, "A differential analysis of the power flow equations," in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 23–30, IEEE, 15.12.2015 - 18.12.2015.

[11] B. C. Lesieutre, M. Schlindwein, and E. E. Beglin, "Dc optimal power flow proxy limits," in *2010 43rd Hawaii International Conference on System Sciences*, pp. 1–6, IEEE, 05.01.2010 - 08.01.2010.

[12] Fatma Sayed Mahmoud Sayed, "Optimalpower flow methods a survey," *International Electrical Engineering Journal*, vol. 7, no. 4, pp. 2228–2239, 2016.

[13] F. Capitanescu, M. Glavic, and L. Wehenkel, "An interior-point method based optimal power flow," 06 2005.

[14] F. ROSENBLATT , "The perceptron: A probabilistic model for information storage and organization in the brain," vol. 65, no. 6, pp. 386–408, 1958.

[15] Sonali. B. Maind,Priyanka Wankar , "Research paper on basic of artificial neural network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96–100.

[16] Erik G. Learned-Miller , "Introduction to supervised learning," *University of Massachusetts, Amherst Amherst, MA 01003*, 2014.

[17] L. K. Hansen and J. Larsen, "Unsupervised learning and generalization," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1, pp. 25–30 vol.1, 1996.

[18] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised learning*. Adaptive computation and machine learning, Cambridge Mass.: MIT Press, 2006.

[19] J. Zupan, "Introduction to artificial neural networks (ann) methods: What they are and how to use them," 2014.

[20] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese Control And Decision Conference (CCDC)*, pp. 1836–1841, IEEE, 09.06.2018 - 11.06.2018.

[21] K. Fukami, K. Fukagata, and K. Taira, "Assessment of supervised machine learning methods for fluid flows," *Theoretical and Computational Fluid Dynamics*, vol. 34, no. 4, pp. 497–519, 2020.

[22] J. Fürnkranz, P. K. Chan, S. Craw, C. Sammut, W. Uther, A. Ratnaparkhi, X. Jin, J. Han, Y. Yang, K. Morik, M. Dorigo, M. Birattari, T. Stützle, P. Brazdil, R. Vilalta, C. Giraud-Carrier, C. Soares, J. Rissanen, R. A. Baxter, I. Bruha, G. I. Webb, L. Torgo, A. Banerjee, H. Shan, S. Ray, P. Tadepalli, Y. Shoham, R. Powers, S. Scott, H. Blockeel, and L. De Raedt, "Mean squared error," in *Encyclopedia of machine learning* (C. Sammut and G. I. Webb, eds.), p. 653, New York and London: Springer, 2010.

[23] C. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate Research*, vol. 30, pp. 79–82, 2005.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[25] Ruoyu Sun , "Optimization for deep learning: theory and algorithms," *Department of Industrial and Enterprise Systems Engineering (ISE), and affiliated to Coordinated Science Laboratory and Department of ECE, University of Illinois at Urbana-Champaign, Urbana, IL*, 2019.

[26] Yann LeCun,Patrick Haffner, Léon Bottou and Yoshua Benjio , "Object recognition with gradient-based learning," *Journal of Electronic Imaging*, vol. 7, no. 3, pp. 410–425, 1998.

[27] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *CoRR*, vol. abs/1312.6203, 2014.

[28] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv*, vol. abs/1603.07285, 2016.

[29] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[30] 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.

[31] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, "A modern take on the bias-variance tradeoff in neural networks," 2019.

[32] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot and ´Edouard Duchesnay , "Object recognition with gradient-based learning," *Journal ofMachine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] P. G. Asteris, P. C. Roussis, and M. G. Douvika, "Feed-forward neural network prediction of the mechanical properties of sandcrete materials," *Sensors (Basel, Switzerland)*, vol. 17, no. 6, 2017.

[34] Jian Du, Shanghang Zhang, Guanhang Wu, Jos´e M. F. Moura Soummya Kar , "Topology adaptive graph convolutional networks," *Carnegie Mellon University, Pittsburgh, PA*, 2018.

[35] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 6861–6871, PMLR, 09–15 Jun 2019.

[36] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[38] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2020.

[39] S. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing*, vol. 378, pp. 112–119, 2020.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[41] N. Zlatanov, "Cuda and gpu acceleration of image processing," 03 2016.

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[43] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.