POLYTECHNIC OF TURIN

MASTER's Degree in DATA SCIENCE AND ENGINEERING

MASTER's Degree Thesis

OVER VOLTAGE SITUATIONS FORECAST AND ACTIVE CONTROL IN A MEDIUM-VOLTAGE POWER GRID

Supervisors	Candidate
Prof. LUCA CAGLIERO	MAURIZIO VASSALLO
Prof. DAMIEN ERNST	

July 2022

Abstract

The recent aspirations for a more sustainable energy system and the reduction of CO_2 have started a transformation in the power networks. Traditionally considered as passive systems, the power grids are undergoing a rapid change with the introduction of more distributed energy resources. Their introduction requires a better control of the networks to ensure reliability and avoid energy losses. A technical consequence of these devices is the increased number of network's problems, like over voltages of the lines. These problems could damage the devices connected to the grid, with social and economic consequences.

This thesis intends to investigate some possible solutions when dealing with the issues introduced by these grid changes. It also suggests different techniques to address the challenges of network forecast and control. In particular, to test whether it is possible to predict and respond in time to solve the voltage problems in the network system, some machine learning models are implemented to forecast and to control the network's devices. Two main learning algorithms are used: supervised learning, for the forecasting part; and reinforcement learning, for the controlling part.

The thesis focuses on a medium-voltage network and the analysis of one-year time series measurements of its devices. The time series are built using the SimBench dataset, and they are adapted to the MV Oberrhein network in order to have a real network with realistic time series.

The methods' results revealed that, starting from the network's devices measurements, it is possible to forecast the over voltage problems with a certain level of accuracy and in a similar way control these devices to reduce the number of voltage issues.

Riassunto

Le recenti aspettative di un sistema energetico più sostenibile e la riduzione delle emissioni di anidride carbonica hanno dato il via a una trasformazione delle reti elettriche. Tradizionalmente considerate come sistemi passivi, le reti elettriche stanno subendo un rapido cambiamento con l'introduzione di un numero sempre maggiore di risorse energetiche rinnovabili. La loro introduzione richiede un migliore controllo delle reti per garantire la sicurezza ed evitare sprechi di energia. Una conseguenza tecnica di questi dispositivi è l'aumento di problemi sulla rete elettrica, come le sovratensioni delle linee elettriche. Questi problemi potrebbero danneggiare i dispositivi collegati alla rete, con conseguenze sociali ed economiche.
Questa tesi intende studiare alcune possibili soluzioni nell'affrontare i problemi introdotti da questi cambiamenti all'interno della rete. Suggerisce inoltre diverse tecniche nell'affrontare le sfide della previsione e del controllo del sistema elettrico. In particolare, per verificare se sia possibile prevedere e rispondere in tempo per risolvere i problemi di tensione nella rete, sono stati implementati alcuni modelli di apprendimento automatico. Vengono utilizzati due principali algoritmi di apprendimento: l'apprendimento supervisionato, per la parte di previsione, e l'apprendimento rinforzato, per la parte di controllo.
La tesi si concentra su una rete di media tensione e sull'analisi delle serie temporali di un anno dei suoi dispositivi. Le serie temporali sono state costruite utilizzando il dataset SimBench e sono state adattate alla rete MV Oberrhein, in modo da avere una rete reale con serie temporali realistiche.
I risultati dei due metodi hanno rivelato che, a partire dalle misure dei dispositivi della rete, è possibile prevedere i problemi di sovratensione con un certo livello di precisione e, allo stesso modo, controllare i dispositivi per ridurre il numero di problemi di tensione.

Acknowledgements

I would like to express my gratitude to professor Luca Cagliero as my academic tutor. A special thanks to professor Damien Ernst for accepting me as part of his research group and involving me in this project by believing in my abilities and skills. I am thankful for the time he dedicated to the development of the project, for the discussion of new ideas, and for sharing with me all the material needed. Thanks to this project, his comments and encouragements, I was able to improve both academically and personally.
A special thanks to Laurine, to dedicate her time to discuss the technical parts and to check the different part of this thesis.

I would like to thank my family, my friends and everyone who has been close to me during these years.

Ringraziamenti

A mio padre che ho sempre visto come punto di riferimento, che mi ha insegnato molte nozioni sulla vita di cui ne sono riconoscente e mi ha ispirato sin da piccolo alle materie scientifiche. Grazie a te ho intrapreso il percorso che mi ha portato fino a qui e senza i tuoi insegnamenti e incoraggiamenti non sarei mai arrivato dove sono arrivato adesso. Nonostante il passare del tempo non ti ho mai dimenticato e so che sei lassù che mi osservi e mi guidi nel compiere le scelte giuste. Ricorderò sempre i bei momenti passati insieme: i viaggi in macchina e in moto, le uscite al mare, il coltivare l'orto in campagna e anche quando mi portavi in posta con te per farmi vedere il tuo lavoro.
Grazie per avermi dato il regalo più grande che una persona potesse ricevere e aver creduto in me. Anche adesso so che in questo momento tu mi stia osservando e tu sia fiero di tuo figlio. Grazie papà..

"Continuiamo a vivere nel ricordo di chi ci ama." Zafón

A mia madre che mi ha sempre sostenuto moralmente, non facendomi mai fatto mancare nulla. Grazie per esserci sempre stata nei miei momenti di ansia e di gioia, per aver creduto in me e per avermi appoggiato su ogni mia decisione senza impormi limiti e lasciandomi libero di fare ciò che volessi anche se qualche volta significava stare lontani.

A mio fratello che mi ha sempre parlato in modo sincero facendomi notare cosa fosse giusto e cosa sbagliato, avermi fatto capire i miei limiti e avermi aiutato a superarli.

Alle due mie nonne per cui sono grato di avervi conosciuto e aver passato indimenticabili e bei momenti insieme.

A mio zio Franco che è stato sempre presente, che mi ha dato consigli su cosa fare per la mia carriera futura lavorativa e accademica e con cui ho avuto lunghe discussioni di attualità.

A Salvatore, Salvatore, Ivan e Giada con cui siamo sempre insieme e abbiamo passato e passeremo molti momenti belli. Grazie per essermi stati sempre vicini e sopportato anche quando ero per le mie. Probabilmente senza di voi starei sempre a casa.

Ad Alessandro e Simone con cui ci conosciamo da una vita, che mi hanno sempre spalleggiato e che sono in grado di vedere in me cose che neanche io sono in grado di cogliere.

A Chiara e Jessica che sono entrate nella mia vita in un momento un po' difficile e mi hanno aiutato a superarlo e con cui abbiamo passato bei e divertenti momenti insieme.

Vorrei anche ringraziare il tutor accademico, il professore Cagliero, professore Ernst e Laurine per avermi sostenuto nello sviluppo di questa tesi e per avermi aiutato a maturare a livello personale e lavorativo.

Ringrazio in generale tutti gli amici, parenti e le persone che mi sono state vicini in questi anni di università, che mi hanno sostenuto moralmente, mi hanno dato suggerimenti per i tanti progetti a cui mi sono candidato e soprattutto mi hanno dedicato un po' del loro tempo.

Table of Contents

List of Tables

List of Figures

Abbreviations

ANN	Artificial neural network
ANM	Active network management
API	Application programming interface
BDEW	German Association of Energy and Water Industries
CNN	Convolution neural network
DDPG	Deep deterministic policy gradient
DER	Distributed energy resources
DES	Distributed energy storage
DNN	Deep neural network
DPG	Deterministic policy gradient
DSO	Distribution system operator
FC	Fully connected
FN	False negative
FP	False positive
i.i.d.	Independently and identically distributed
GHG	Greenhouse gas
LSTM	Long short-term memory
MAE	Mean absolute error
MDP	Markov Decision Process
MLP	Multi layer perceptron
PF	Power flow
p.u.	Per unit
PV	Photovoltaic

ReLU	Rectified linear unit
RES	Renewable energy sources
RL	Reinforcement learning
RNN	Recurrent neural network
r.m.s.	Root mean square
SLP	Standard load profile
SMOTE	Synthetic minority oversampling technique
Tanh	Hyperbolic tangent function
TN	True negative
TP	True positive
WP	Wind park
w.r.t.	With respect to

Symbols

CO_2	Carbon dioxide
R	Electric resistance
I	Electric current
V	Voltage magnitude
P	Active power
W	Active power unit: Watts
Q	Reactive power
VAR	Reactive power unit: Volt-Ampere reactive
G	Directed graph
$\mathcal{N}$	Set of positive integers representing the bus (or node) in the network
$\mathcal{E}$	Set of directed edges linking buses together
e_{ij}	Directed edge with sending bus i and receiving bus j
$\mathcal{D}$	Set of all devices
$\mathcal{L}$	Set of all loads
$\mathcal{G}$	Set of all generators
$\mathcal{T}$	Set of all transformers

Chapter 1

Introduction

The last decade has experienced a rapid growth in the world population, and, in the following years, the number of people it is expected to increase with a rate of 1.1% for year, reaching 9.7 billions by 2050 [1]. As the world population increases, so the energy demand does. This high energy demand has required an intensive usage of fossil energy, causing environmental pollution and changes in the climate. Indeed, the increase of global temperature and the worsening of the air quality are posing a real problem for the environment. In the last years, some changes have been observed in Earth's climate, primarily driven by human activities, particularly fossil fuel burning.

The biggest disadvantage of fossil fuels is that during the process of combustion in addition to energy, greenhouse gases (GHG) are emitted [2].
These gases form a cope in the atmosphere, similar to glass in greenhouses, that traps the heat, increasing the Earth's temperature.
For around a century, humans have relied on fossil fuels, like oil, natural gas and coil for everyday tasks: heating, transportation and to produce electricity. For this reason, the GHG emissions have reached historical peaks and are expected to increase in the following years [3]: in the year 2020, the concentration of CO_2 in the atmosphere had risen to 48% above its pre-industrial level (before 1750).

In order to improve the situation, the 2015 Paris Agreement set an ambition to limit global warming to well below 2 °C above pre-industrial levels and pursue efforts to limit it to 1.5 °C - in part by pursuing net carbon neutrality by 2050. The substantial reduction of global greenhouse gas emissions (including CO_2) would limit the increase of global temperature [4].
Countries were asked to go through a process of decarbonisation: the reduction of carbon dioxide emissions through the use of low carbon power sources.
These sources convert the energy coming through naturals elements (sun, wind,

geothermal heat) in another form of energy, electricity for example, with low or no waste products such as CO_2 or other chemical pollutants.

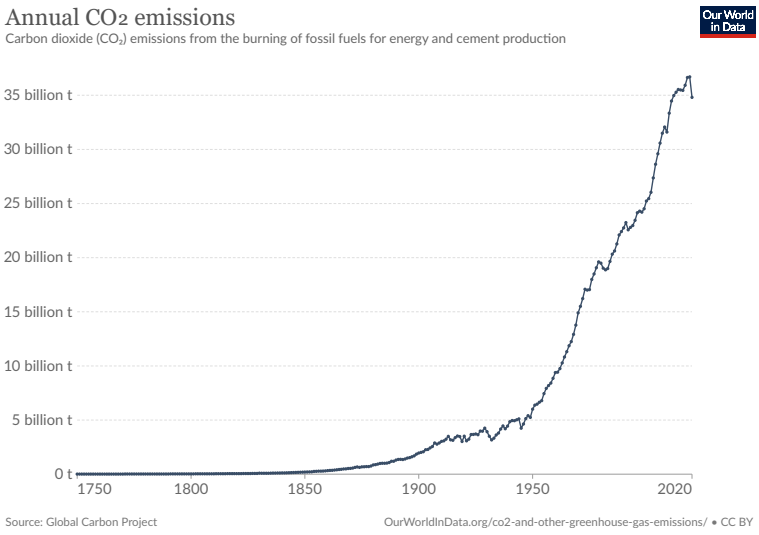


Figure 1.1: World CO_2 production over the years [5].

Thanks to this emerging trend of decarbonisation, more and more renewable power energy devices are introduced in the distribution networks [6]. With the advantages of inexhaustibility and low impact on the environment, the high penetration of these renewables devices bring in some technical complications for the transmission and distribution of power in the grids.
The networks, that have been designed around the conventional centralized energy production, have to adapt to the new generators in the system. The distribution networks are moving from unidirectional power flow (from the distribution system to the consumers) to a bidirectional power flow (in this case the consumers are also producers and the exceed energy can be transported from the consumers to the distribution system. They are also known as prosumers [7]). This switch from unidirectional to bidirectional power flow requires a smarter system that can handle in an efficient way the generation and distribution of energy.

In the literature, this smarter way to control a power grid is known as active network management (ANM) and it refers to the design of control schemes that control the distributed energy resources (DERs), the loads, and the distributed energy storages (DESs), as well as other elements like switches, connected to the grid.

1.1 Aim of the thesis

The aim of this thesis is to exploit data-driven approaches to forecast and control the future nodes' voltage problems based on historical measurements in a medium-voltage distribution system using machine learning techniques, with a particular focus on deep artificial neural networks. The objectives of this thesis are to:

I) Build a time series dataset that is used to train the machine learning models. A real medium-voltage network from the Pandapower python library is used. This network is fit with time series taken from the SimBench database, a database generated by the measurements of real loads and generators from Germany.

II) Predict the voltage fluctuation problems in the network under a supervised learning framework. The three artificial neural networks used are: a multi layer perceptron, a convolutional neural network and a recurrent neural network. Three combinations of network information are tested, and some techniques for unbalanced datasets are examined as well.
Different combinations are used since the network operator can have access to a limited amount of data, so the performances for each combination are reported.

III) Control the active and reactive power of the network's generators using reinforcement learning framework. The model used is a deep deterministic policy gradient algorithm that allows to work with continuos state and action space. The agent changes the generators' output in order to keep the voltage magnitude of the network's buses inside a safe voltage range.

Predicting the contingency problems and controlling the network's devices would allow reducing possible consequences related to voltage issues, like for example over voltages problems.

1.2 Thesis outline

The document is organised into six chapters.

Chapter 2 provides the background information needed to understand this thesis's work, in particular in 2.1 it is presented a short description of how power networks work, an important calculation in power networks: the power flow; and some observations on voltage problems. In 2.2, it is presented an overview of machine learning, with emphasis on artificial neural networks. In 2.3 a description of the concepts of supervised learning, the models and evaluation metrics used in this

thesis. Moreover, some techniques to handle unbalanced datasets are presented.
In 2.4 a description of the concepts of reinforcement learning, a model and some evaluation metrics is presented.
After an introduction of the different terms and concepts, the literature review is proposed in 2.5.

Chapter 3 provides the information about the use case studied. In particular, the description of the MV Oberrhein network is presented in 3.1, an overview of the SimBench database in 3.2 and how the time series are selected in 3.3.

Chapter 4 provides the main development of the project. The problem is defined in a rigorous way in 4.1. In 4.2 the solving methodology is proposed with the description with the forecast of possible over voltage situations and the control the network's devices in order to avoid these situations.

Chapter 5 provides the results of the different combinations of the forecasting part in 5.1 and of the controlling part in 5.2. A discussion of the values obtained is presented as well.

The thesis ends with **chapter 6** that provides the conclusion and a discussion on some possible future works.

Chapter 2

Background

In this chapter an overview of power systems is presented followed by a description of supervised and reinforcement learning concepts. Finally, the literature review is proposed.

2.1 Power system

Nowadays, electricity is given for granted and that flipping the switch will turn the light on instantly and effortless. But electricity performs a long journey before arriving to houses or where it is consumed. To reach its destination, electricity circulates in power networks or power systems.

2.1.1 Description of a power system

A power system is a complex infrastructure that produces and distributes electricity to different consumers. A power network consists of generation, transmission and distribution system and each of them has a different function.

In the traditional power system, electricity is generated in large, centralised power plants. The electricity is then transferred to the loads using the transmission and distribution networks. Transmission substations are located near the power plants. Their main function is to increase the voltage level to high and extra-high voltages levels. The reason for power transmission at high and extra-high voltage levels is to increase efficiency. The lower current that accompanies high-voltage transmission allows thinner and lighter cables to be used. This reduces the construction costs of towers and power lines. In Belgium, high and extra-high voltages refer to voltage magnitudes $36kV \leq |V| < 380kV$ for the high voltage and $|V| \geq 380kV$

for the extra high voltage [8].
Large industrial complexes and factories that require a substantial amount of power often utilise medium supply voltages. The high voltage coming from the transmission lines is sent to the primary substation, this can supply step-down power to secondary substations or to single buildings. Secondary substations can have transformers to further step-down the power, and they are generally located in areas that can serve one or more buildings. Medium voltages refer to voltage magnitude $6kV \leq |V| < 36kV$.
Then the medium supply power is step down again to a low voltage and sent to the domestic household or home appliances power supply. Low voltages refer to voltage magnitude $|V| < 6kV$.

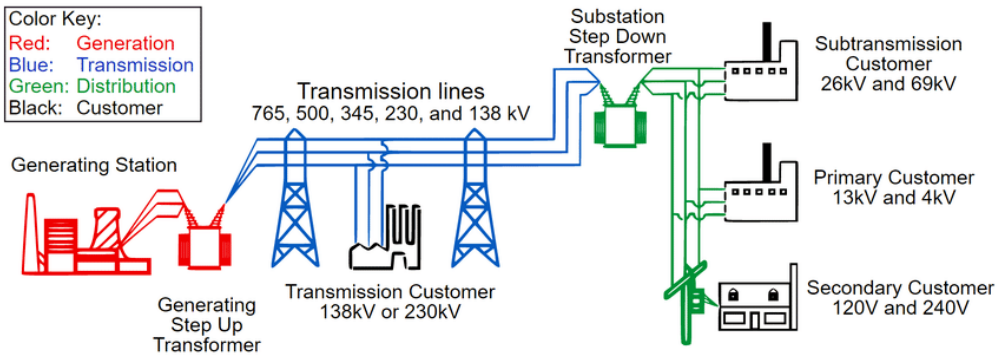


Figure 2.1: Typical Power network distribution [9].

These power networks include many elements and devices that are needed to generate, transport and assure that there are no problems in a network.
These elements are:

- **Generator**. These generate electricity starting from a different form of energy. In general, electricity is produced when a magnet rotates within closed loops of wire to create a steady flow of electrons.
  For this reason, many generators produce energy using turbines: a fluid spins the generator's blades, producing electricity. This fluid, either water or air, can derive from natural sources: hydroelectric, wind or geothermal turbines, or generated by combustion of some fuel, for example coal, natural gas, oil or nuclear source.
  There are other generators that do not need a turbine to generate electricity, for example solar panels.

- **Lines**. These transport the power from where energy is generated to where it is consumed. These lines have to be long enough to reach any destination.

One of the main issues about transportation lines is insulation.
There are different types of lines: overhead cables, they use air to insulate the bare conductors, or underground cables. For latter cables, particular attention must be taken to insulate them from other conductors and from the earth (ground). Also, the material used must be resistant to damages, corrosion, and it must avoid that the water is being absorbed.

- **Transformers**. Transformers are used to interlink systems operating at different voltages. These can increase the voltage magnitude near a generator power plant or decrease it near the consumptions facilities.
  Changing the voltage magnitude allows reducing the power loss due to transportation. One of the main causes of power loss is the Joule effect: some part of the energy transmitted is converted in heat generated by the current flowing through a conductor. This power lost is given by the equation $P = RI^2$, where R is the resistance and I is the current through a line, so decreasing the current reduces the energy loss.

- **Loads**. These are electric components that consume the electric power generated by power plants. The types of loads can be divided base on the consumption in:

  Domestic loads, the domestic loads mainly consist of lights, fan, refrigerator, air conditioners, mixer, grinder, heater, ovens, small pumping, motor, etc. The domestic loads consume very little power.

  Commercial loads, the commercial loads mainly consist of lightning, fans, heating, air conditioning and many other electrical appliances used in establishments such as markets, restaurants, shops. This type of load consumes energy for more hours during the day as compared to the domestic load.

  Industrial loads, the industrial loads refer from a small-scale industry, to a heavy industry. Industrial loads may be connected during the whole day [10].

2.1.2 Power flow

An important procedure in power systems is to perform a numerical analysis to determine the electrical state of the network, starting from some parameters that are known. This analysis is called power flow (PF).
The objective of a PF study is to calculate the voltages, magnitude and angle, for a given bus, load, generation device. After this information is known for all elements, line flows and losses can be calculated.

A bus is a node of the network where a line or several lines are connected to. Loads and generators can be connected to it as well. There are different types of bus:

- *Slack bus.* It is taken as a bus reference, where the magnitude and phase angle of the voltage are specified. Slack bus magnitude considers 1 p.u. and the phase angle 0 of degrees. This bus provides the additional real and reactive energy to cover the demand.
- *Load bus or PQ bus.* At these buses, the real and reactive powers are specified. The buses' magnitude and phase angles are unknown until the final solution is obtained.
- *Voltage controlled bus or PV bus.* Instead, at these buses, the real power and voltage magnitude are specified, and the phase angles of the voltages and the reactive power are known only after the final solution is obtained. The limits on the value of reactive power are also specified.

Solution techniques

Defining and solving the PF equations are the main tasks in load flow analysis. The definition of the PF equations is based on Ohm's Law, which is the relationship between voltages and currents. For a network, it can be expressed in matrix notation as follows:

$$\mathbf{I} = \mathbf{Y} \times \mathbf{V}$$

Where:

- $\mathbf{Y}$ is the bus admittance matrix
- $\mathbf{V}$ is an array of bus voltages
- $\mathbf{I}$ is an array of bus current injections

The PF formulation is based on the application of Kirchhoff's laws to meshed electric networks. The concept of the PF calculation is that the sum of all flows into each node is equal to the sum of flows flowing out.
The flows equations are in complex form, they consist of real and reactive components. That means that if there are n nodes, then there are n complex equations. Solution methods for this system of equations are primarily iterative with the objective of reducing the sum of flows in all nodes to some acceptably small value known as the mismatch tolerance.

All these iterative methods follow the same basic concept: they assume starting values for the dependent variables, primarily voltage at nominal voltage magnitude

(i.e. 1 p.u.) and zero phase angle; compute new values for those voltages using the nodal network equation or a numerical approximation and repeat until the convergence criteria are met.

Convergence

The PF is a non-linear and non-convex numerical analysis, with a large number of constraints and variables. It is, therefore, a hard problem, whose cost of finding a solution can increase exponentially, particularly with the increasing size of the network. Moreover, there is no guarantee to find the global optimum.
When a solution exists, and it is reached, it is said that the network has converged. The calculation has converged when all nodes have met the mismatch tolerance. The main PF solution methods are:

- Gauss-Seidel method updates the voltage one node at a time until all nodes are within the mismatch tolerance.
- Newton-Raphson method uses a first order expansion of the PF equations to approach convergence. It is generally faster than the Gauss-Seidel method and able to converge to small tolerances. This method is prone to **divergence**, when mismatches increase instead of decrease from iteration to iteration. This occurs when the solution vector exits outside the feasible solution space at any point during the algorithm. As soon as the solutions are outside the feasible space, the gradient tends to further increase mismatches, leading to solutions that "blow-up" in the numerical sense.
  This method requires calculating the first order approximation matrix, known as the Jacobian.
- Interior-Point Newton method forces the solution inside feasible space to avoid divergence. The interior point method uses a second order expansion of the PF equations to find a solution. The method is more computationally intensive than either the Gauss-Seidel or Newton-Raphson, but is less susceptible to numerical divergence [11].

Divergence

Divergence is the condition of the power network when the numerical solution can not be found any more due to some possible issues:

- the power system is going to "blow-up."
- the power system is in voltage collapse.
- the power system is unstable.

- the initial conditions defined were bad or poor.
- some issues related to software or input data.

Divergence of the PF solution is usually associated with the singularity of the Jacobian matrix. Since some methods require an inverse of the Jacobian as part of its solution algorithm, singularity of the Jacobian means division by zero [12].

2.1.3 Voltage problems in power systems

The power systems are highly secure, but they are far from perfect, and some issues may arise at any moment, for example voltage problems. Voltage problems arise when the voltage through a line or a device is more or less than what it is expected. These cases should be handled correctly. There are two voltage problems that can raise on a network: under and over voltage problems.
Generally, electronic devices have defined voltage limits to work in safety conditions. The voltage magnitude in the different parts of a network is not always constant during time, but it fluctuates. These fluctuations, both positive and negative, can be large: when negative, the voltage can drop below the device's minimum allowed voltage limit, in this case there would be an under voltage problem; or when positive, the voltage can increase above its maximum, in this case there would be an over voltage problem.

The voltage control problem has been studied for years, but it only came under the spotlight in the last years for the increasing number of distributed resources introduced in the networks.
The introduction of more and more DER devices in the networks increases the number of voltages problems, in particular over voltage problems. These devices generate electric power and when this power is greater than the energy consumed, the extra energy is emitted back in the network.

For this reason, it is important to control the voltage in an electrical power system for a regular operation of the electrical equipments. It can prevent damages such as overheating of devices and lines, reduce transmission losses and maintain the ability of the system to last and avoid voltage collapses. Over voltages other than shorten the lifetime of equipments have a negative impact on the stable operation of both supply side devices and demand-side appliances.

A possible way to control and manage the voltage on the network to avoid voltage problems and to ensure its stability is active network management (ANM). Indeed, the principle of ANM is to address congestion and voltage issues via short-term decision making policies [13].

ANM creates a smarter network infrastructure providing automated control of various components in the network and provides the information needed to ensure that every device performs in an optimal manner. This control allows grid companies to avoid the traditional approach of reinforcing the network with expensive upgrades; so reducing the costs. For example, in case of energy generation, from the renewable devices, higher than what a particular line can handle, a grid company, to avoid congestions and possible overvoltages, has three main options:

- Replace the existing line with a line that can handle a higher voltage. This usually means replacing the existing line with a cable with a larger cross-sectional area.

- Add another parallel line.

- Handle the situation with ANM.

The first two solutions require some infrastructure investment that can be expensive and troublesome, especially in the case of overhead or underground lines.
The solution with ANM does not require construction cost for the grid company; to keep the network working, in this case, the output of the renewable devices can be curtailed, or the reactive power modulated to reduce lines' overloading.

The control voltage problem has some interesting properties:

- It is a combination of local and global problems: the voltage at each node is influenced by the powers of all other nodes, but the impact depends on the distance between them.

- It is a constrained optimisation problem with many constraints, for example to keep the voltage in a given range, and the objective is to minimise the total power loss.

- Voltage control has a relatively large tolerance, and there are no severe consequences if the control fails to meet the requirements for short periods of time. [14]

- It is a hierarchical problem where the information available can be represented as a pyramid: much information is available at the top of the pyramid (distribution stations and substations) and it decreases at the base of the pyramid (houses, factories) mainly due to the absence of many sensors.

2.2 Machine learning overview

Machine learning is a subset of artificial intelligence that trains a machine to learn. In particular machine learning is the study of how a computer algorithm improves its performances at some task through experience or more precisely:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance P, if its performance at tasks in T, as measured by P, improves with experience E [15].

where generally, in a machine learning problem, T is a task too complex to be solved with human written algorithms.

Machine learning differs from the traditional computer science methods. In traditional approaches, algorithms are sets of explicitly programmed instructions, or rules, used by computers to solve a problem. Machine learning algorithms instead allow computers to train on data inputs and use statistical analysis in order to output values or answers.

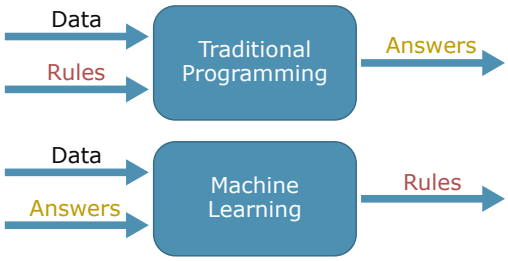


Figure 2.2: Difference between traditional programming and machine learning approach.

Figure 2.2 shows the main difference between traditional methods and machine learning approach: when solving a problem, traditional programming required someone (usually an expert in the field) to generate some rules that would be used to get answers from the input data; while in machine learning the model tries to find some rules that link the input data and the output data (or answers). Machine learning approach has demonstrated to outperform humans in finding this kind of rules, moreover no real expert is required.

In machine learning, tasks are classified into some categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed. There are three main types of machine learning algorithms:

supervised, unsupervised and reinforcement learning.
This thesis focuses on supervised and reinforcement learning.

2.2.1 Artificial neural networks

Nowadays, a popular approach to solve machine learning problems is to use artificial neural networks (ANNs), whose main elements are the *neurons*.

ANN is a computational model that consists of several processing elements that receive inputs and deliver outputs based on their predefined activation functions. They have been proved to provide a strong approach to approximate functions in order to solve continuous and discrete problems.

They have been inspired by the biological neural networks that constitute animal brains. For the first time, in *1943* Walter Pitts and Warren McCulloch published a paper with the mathematical modelling of a neural network. They thought a human neuron cell as a threshold logic unit working together with other neurons to build a complex system: a neuron cell collects multiple signals arriving at the dendrites, elaborate them and if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon [16]. So, the idea behind ANNs is to link many simple units (neurons) to develop a more complex system (brain).

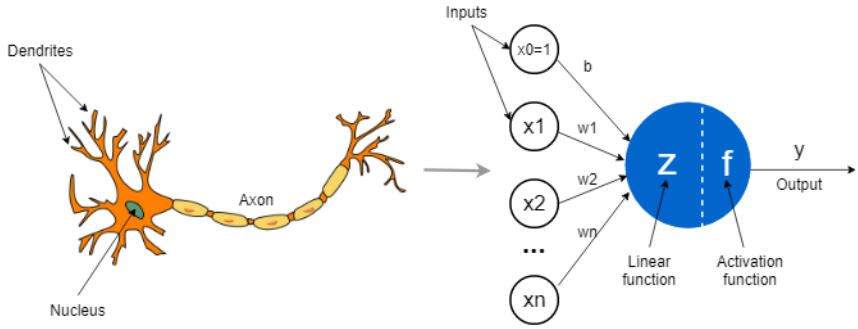


Figure 2.3: Representation of a biological neuron (left) and its artificial equivalent (right).

Perceptron

The simple unit in an ANN is called a *perceptron*. The perceptron is a mathematical function that takes some inputs, weights them separately, sums them and pass the sum through a nonlinear function to produce an output.

This mathematical function can be written as follows:

$$o(x_1, x_2, ..., x_{n-1}, x_n) = f(\sum_{i=0}^{n} w_i x_i + w_0) \tag{2.1}$$

where n is the number of connected neurons, x_i is the input from the neuron i, w_i is the weight that determines the contribution of input i, and f is a nonlinear function. A possible example of the function f is:

$$f(x) = \begin{cases} 1 \text{ if } x > T \\ -1 \text{ otherwise} \end{cases}$$

with T a real value representing the threshold that x has to surpass for the function to output 1. In the formula 2.1 the threshold T is given by the value w_0.

A single perceptron can be used for binary classification tasks: it builds a hyperplane that separates the data, and outputs giving a value between -1 and 1 whether a point is on a side of the hyperplane or on the other side. The perceptron can find a hyperplane in any n-dimensional space as long as this decision boundary exists; this happens if the data points are *linearly separable*.

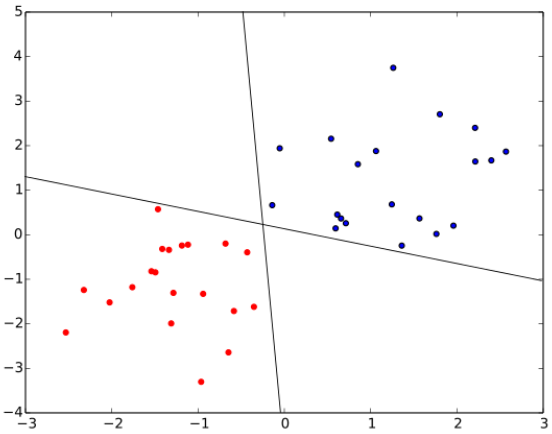


Figure 2.4: Example of linearly separable data. The two lines are some possible representations of hyperplanes that divide the data [17].

The weights' values are generally initialised randomly, and their values are changed during training using the *perceptron training rule* where each weight w_i in $\mathbf{w}$ is changed according to:

$$\begin{aligned} \mathbf{w} &= \mathbf{w} + \Delta\mathbf{w} \\ \Delta\mathbf{w} &= \alpha(\mathbf{t} - \mathbf{o})\mathbf{x} \end{aligned} \tag{2.2}$$

where $\mathbf{x}$ are the inputs, $\mathbf{o}$ the outputs of the perceptron, $\mathbf{t}$ the target outputs and α is a scaling factor called learning rate, used to reduce the change of the weights $\mathbf{w}$ at each step. Equation 2.2 assures convergence under the specific condition of the linearly separable training samples.

A better training rule is the *delta rule* that uses the *gradient descent* to find the weights that best fit the training examples. This training rule requires the definition of a loss function L that has to be differentiable.
This loss function is used to calculate the derivative of the loss w.r.t. to the weights:

$$\nabla L(\mathbf{w}) = \left[\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \cdots, \frac{\partial L}{\partial w_{n-1}}, \frac{\partial L}{\partial w_n}\right] \tag{2.3}$$

where $\nabla L(\mathbf{w})$ is the gradient of L w.r.t. the weights $\mathbf{w}$. Geometrically, $\nabla L(\mathbf{w})$ is a vector in the weights' space that represents the steepest increase in L. So, the formula 2.2 is updated as follows:

$$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$$
$$\Delta\mathbf{w} = -\alpha L(\mathbf{w})$$

where α is the learning rate and the minus sign is used to move towards the direction that *minimise* the loss L.

This gradient descent rule is the base of the backpropagation algorithm commonly used when training deep artificial neural networks.

Perceptrons are the basis for more complex models like for example: multi layer perceptrons, convolutional neural networks and recurrent neural networks.

2.3 Supervised learning

In supervised learning, the goal is to learn a function that maps an input X to an output Y based on example input-output pairs and applying this learnt function to predict the output of future unseen data.

2.3.1 Formal definition

In a supervised learning problem, the goal is to find a function $f : X \to Y$, from a sample data S_n composed by pairs of (input, output) points:

$$S_n = ((x_1, y_1), \ldots (x_n, y_n)) \in (X \times Y)^n$$

Typically, $x_i \subset R^n$ and $y_i \subset R$ for regression problems or y_i discrete for classification problems, for example $y_i \in \{-1,1\}$ for binary problems.

In the statistical learning settings, an important hypothesis is that the training data is independently and identically distributed (i.i.d.) from a probability distribution function $P(X, Y)$. The term i.i.d. means that the random samples are chosen: *independently*, the samples are considered as independent events, having the value of one sample does not give information about the other samples; and *identically distributed* so the probability of choosing one sample or another is the same, all the samples are equally likely to be chosen.
The goal of the learning is to find a mapping function f that can encode the property of $P(X, Y)$ between the inputs X and the output Y.

Another important concept is to evaluate how well the function f performs, calculating the error or loss between the predicted values $f(x)$ and the actual value y. This error is evaluated with a loss, or cost, function $L : Y \times Y \to R^+$. There are many loss functions depending on the problem and requirements, one example is the mean absolute error (MAE) loss function:

$$L(f(x), y) = \frac{1}{N} \sum_{i=0}^{N} |f(x_i) - y_i|$$

Many supervised learning algorithms consider the minimisation of this loss function as an optimisation problem to find the best predictor among all the possible candidate input-output mappings in the solution space $\mathcal{B}$.

With the loss function $L(x, y)$, the definition of risk of the function f, also called generalisation error, must be introduced:

$$R(f) = \int L(f(x), y))\, dP(x, y).$$

The objective is to find the function f in $\mathcal{B}$ that minimises the generalisation error, $R(f)$. Since it is not possible to solve $R(f)$, because of the joint probability distribution $P(x, y)$ is unknown, f inferred from available data set S_n.

Given this loss, a way to minimise it, especially with ANN is calculation the gradient of the loss w.r.t. the weights and then back propagate the gradient to change the weights' value.

2.3.2 Models

In this section is reported the main ANNs used for the forecasting part where the models are trained under a supervised learning framework.

Multi layer perceptron

As mentioned in 2.2.1, perceptrons can express only linear decision boundaries. To solve this problem, it is possible to use more perceptrons to represent more complex decision surfaces.

Multi layer perceptron (MLP) networks are constructed by many perceptrons. These perceptrons are organised in layers: there are always at least two layers, input and output layer, and one or more hidden layers; from here the term *multi* layer perceptron. Each layer is composed by many neurons and each neuron in one layer is connected to all the neurons in the next layer, so the information from the input layer is propagated to hidden layers and then to the output layer. These models are also known as deep neural networks (DNNs) since the networks' hidden layers make the models "deep". The output of a layer, before being propagated to the next layer, passes through a non-linear activation function, for example the Rectified Linear Unit (ReLU), the Sigmoid function or the hyperbolic tangent function (Tanh). The non-linearity of the activation function is needed since it introduces more complexity to the model; summing operations of many linear layers is still a linear operation, so, in such case, a deep network would perform similarly to a single layer network.

The main idea behind stacking many layers is that each layer represents a boundary region, that it will pass to another layer to represent an even more complex boundary region. Using many layers, it is possible to represent very complex decision boundaries. This sequence of operations can be written as follows:

$$f(x) = f^{(n)}(f^{(\dots)}(f^{(1)}(x))) \tag{2.4}$$

where n is the number of hidden layers, $f^{(n)}(x)$ is the boundary representation at the last hidden layer before the output layer, $f^{(1)}(x)$ is the decision boundary representation at the first hidden layer after the input layer. The result composition of all these functions, $f(x)$, is the mapping function that can solve the requested problem. Equation 2.4 can be viewed as a chain, where the output of the first decision boundary is propagated to the next decision boundary function up to the final hidden layer and then to the output layer to get the predictions. This propagation is known as *forward propagation*.

The weights in a MLP are updated with formula 2.3. This process in called backpropagation, since the after calculation of the loss, the gradient of the loss w.r.t. each weight is propagated back through all layers.

Convolutional neural network

Convolutional neural networks (CNNs) are a particular type of ANN that process data with a grid-like topology. These kinds of networks are usually used with images, considering an image as a 2D matrix of pixels or for with time series, considering a time series as a 1D structure.

The term convolutional comes from the usage of a mathematical operation called convolution. Convolution is a linear operation that involves two functions (or matrices in the discrete case), x and w:

$$(x * w)(t) \stackrel{def}{=} \int_{-\infty}^{\infty} x(\tau) \cdot w(t - \tau) d\tau \tag{2.5}$$

where $*$ is the sign for the convolution and $\cdot$ is the sign for the dot product.
The output of this linear operation, given by the input x and the weights w (also called **kernel** or **filter**) is referred to as **feature map**.

The main idea is to *convolve* the input, for example an image, with a filter of size f. The filter is applied to an area of the image, and the dot product between the that portion of the input image and the filter is computed. Then the filter is shifted to the next portion of the input image, and this way the dot product is calculated for the full width and height of the input.
Since the convolution is a linear operation, the output of the convolution must go through a non-linear activation function, usually ReLU in the case of CNNs.

The output of a convolutional layer is passed to a pooling function. This pooling function aggregates the output of the convolutional layer at a certain location with a number that statistically represents their values. This allows to reduce the data dimensionality, to shorten the training time and to reduce overfitting.
Usually, the most common pooling function are max pooling, which takes the max value of the window, or average pooling, which averages the values of the window.

With these series of operation, a single convolutional block can extract some important features from the input data. Generally, many convolutional blocks are stacked together so that each of the next block can represent more complex and specific features.
The output of the last block is flatten and passed to one or more fully connected layers to get the final prediction (in case of a classification task).

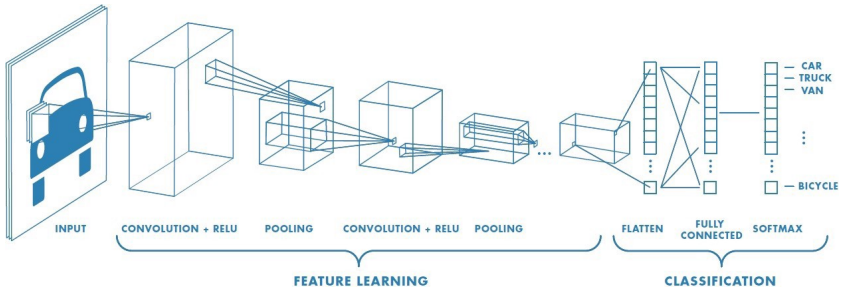


Figure 2.5: Graph representation of a CNN. It is possible to notice some convolutional and pooling layers (left) and the final fully connected layer (right) [18].

CNNs have shown to perform very well on some task, especially image classification, and their main advantages are lower number of weights compared with a MLP network and the ability to automatically learn how to extract important features.

Recurrent neural network

Recurrent neural networks (RNNs) are a particular type of ANN that work well with time series data or data that can be represented as a sequence. RNNs use the output of the network units at time t as the input of the other units at time $t+1$.

Generally, RNNs are represented as follows:

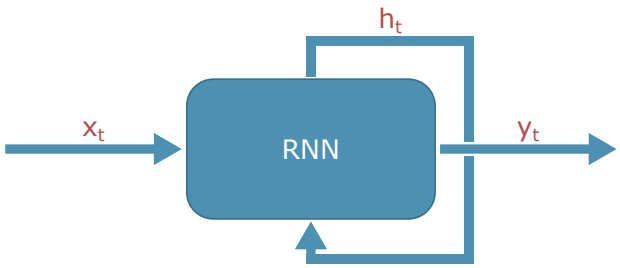


Figure 2.6: Graph representation of a RNN.

where x is the input, the RNN rectangle is the recurrent network block and y is the output of the model. The arrow coming out and back in the RNN block is what the term *recurrent* refers to: after receiving an input, at time x_t the RNN computes some operation and a hidden state, h_t, is saved and used for the next input, at time $t + 1$.

Mathematically, this process can be represented with the following formula:

$$h_t = f_W(h_{t-1}, x_t)$$

where f_W is a function that takes as input the hidden state of the previous time step h_{t-1} and the input at the current time step x_t and it outputs the hidden state at the current time step h_t. At the next time step $t+1$, the previous hidden state h_t would be passed with the next input x_{t+1} to f_W and so on until all the input time steps are consumed.

An important concept to notice is that the function f_W depends on some weights W, and these weights are shared for every time step of the computation. For example, the function f_W can be represented as:

$$f_W = Tanh(W_h \cdot h_{t-1} + W_x \cdot x_t)$$

where Tanh is the hyperbolic tangent function, W_h is the matrix of weights that multiplies the hidden state h_{t-1} and W_x is the matrix of weights that multiplies the hidden state x_t.

Long Short-Term Memory networks

A popular problem with RNN networks is exploding or vanishing gradient. Due to the recurrent nature of the model, during backpropagation, the partial derivatives generate chains of matrix multiplications. In case of large derivates, the gradient increases exponentially, resulting in too large values to be handled by a calculator; this is often referred to as *exploding gradient*. In the opposite case, if the gradient is small, the gradient decreases and the model would stop learning; this is often referred to as *vanishing gradient*.

A model that can solve these gradient problems is the Long Short-Term Memory (LSTM) networks. They were designed to handle the long-time dependency of the input. The main difference between a simple RNN and a LSTM network is the complexity of the hidden block: while in a RNN there is only a Tanh function, in a LSTM there usually is the Tanh function and as well some Sigmoid functions.
This more complex model allows the network to keep in memory (as a hidden state) or forget some information that are considered as not too relevant [19].

2.3.3 Evaluation metrics

An important part of a machine learning problem is to evaluate whether a model performs well or not. There are many ways to evaluate a model, these are commonly

referred to as evaluation metrics.

The evaluation metrics differ from task to task. There are many specific evaluation metrics for classification problems. Some of the most common are presented in this section.

Some terms have to be introduced. During classification, there are four outcomes that can occur:

- **True positive (TP)**: when the result of a test tells that a subject belongs to a particular class (its result is positive), and it actually belongs to that class (the result is true, correct).
- **False negative (FN)**: when the result of a test tells that a subject does not belong to a particular class (its result is negative), but it actually belongs to that class (the result is false, wrong).
- **True negative (TN)**: when the result of a test tells that a subject does not belong to a particular class, and it actually does not belong to that class.
- **False positive (FP)**: when the result of a test tells that a subject belongs to a particular class, but it actually does not belong to that class.

When a test is wrong (either FP or FN) a misclassification occurs. The evaluation metrics try to quantify how well a model performs, elaborating how many miss classifications were done.

Accuracy

Accuracy measures how often the model classifies correctly. Accuracy is defined as the ratio between the number of correct classifications and the total number of predictions.

$$accuracy = \frac{\text{correct predictions}}{\text{total predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall

Recall, also called sensitivity or true positive rate, gives the proportion of positive cases correctly identified by the tests. Recall is defined as the ratio between the positive subjects correctly identified and the total number of positive subjects.

$$recall = \frac{TP}{TP + FN}$$

Precision

Precision gives the proportion of correct positive tests and the number of all tests whose result was positive.
Precision is defined as the number of true positives divided by the number of predicted positives.

$$precision = \frac{TP}{TP + FP}$$

Trade-off recall precision

Generally there must be a trade-off between recall and precision, or equivalently between the number of false negative and false positive, since increasing the recall (precision) would decrease the precision (recall).

This trade-off is even more important when a misclassification would be worse than the other: predicting a subject to have an illness, but actually it is sane (FP) or predicting a subject sane, but actually it has an illness (FN). In this medical case, FN would be a worse case, since the illness of the subject would not be treated while FP would have only to take some medications.
In general, the trade-off depends on the task and there is not a specific way to tell a priori if FP is a better case than FN.

F1-score

There are other situations where having FP or FN does not change much, they are equally important.
In these cases, it is often convenient to combine recall and precision in a single evaluation metric. It is defined by the harmonic average of the recall and precision.

$$F1-score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

2.3.4 Unbalanced dataset

Unbalanced datasets are common in real life classification problem. An unbalanced dataset is a dataset where a class is unrepresented w.r.t. the other classes; so the classes' distribution is not even. As it usually happens, the observations in the minority class are the most important and the problem is more sensitive to misclassification of that class: fraud detection, for example.

Generally this is a difficult problem since some models may not generalise well: the model receiving more observation of a class tends to be more biased towards it

and fails to understand the patters that separate the classes.
In these cases, it is also important to consider which evaluation metric to use. Accuracy metric, in general, is not a meaningful metric, since the model predicting every observation as belonging to the over-represented class would get a high score. More meaningful metrics are precision, recall or an average of the two, for example F1-score.

There are few methods to solve or mitigate the unbalanced dataset issue. Some of them will be reported here:

- Resample the dataset: this can be done, increasing the number of observations in the minority class (*oversampling*) or decreasing the observation in the majority class (*undersampling*).
  The two main methods to resample the dataset are described in the following part:

  – Undersampling: the main idea is to reduce the number of instances in the majority class to the underrepresented class' level.
    This is usually done with a random downsampling, randomly discarding samples.

    A possible problem with this technique is that it does not care to discriminate importance that the different observations may have.

    There are some other more informative techniques like for example nearest neighbours algorithms that try to include samples from every cluster in the majority class.

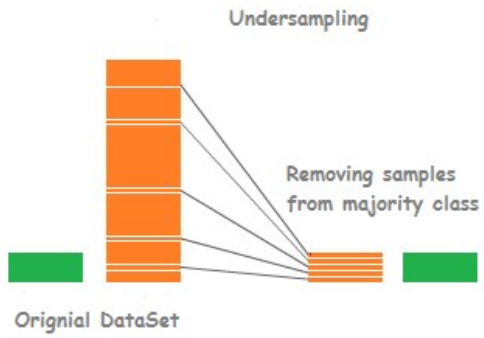


Figure 2.7: Undersampling technique [20].

  – Oversampling: opposite to undersampling, its main idea is to increase the

number of instances in the minority class.
This is usually done generating synthetically observation of the underrepresented class base on the available data.
Some popular techniques include Variational Autoencoders (VAE), SMOTE (Synthetic Minority Oversampling Technique) or MSMOTE (Modified Synthetic Minority Oversampling Technique).

Figure 2.8: Oversampling technique [20].

In this thesis, the method SMOTE will be used. SMOTE generates synthetic samples of the minority class. The main idea is to find examples that are close in the feature space, draw a line between them, and then a new sample is chosen on that line. In particular, a random sample from the minority class is chosen, some other k samples are chosen close to it (usually k=5, and they are chosen using k nearest neighbor algorithm), among these k, one is selected randomly, and then a synthetic element is created on the line that links the two selected samples.

- Penalise misclassification: the idea is to penalise misclassification of the minority class more than the majority class. In this way the model should put more focus on the underrepresented observation since a penalty in case of error is larger than the major class error.
  These penalties are commonly referred to as weights, and finding the right weights' values is usually challenging. Commonly, the weights are calculated as follows:

$$w_i = \frac{total\_samples}{num\_classes \cdot num\_samples_i}$$

  where w_i is the weight for class i, $total\_samples$ is the number of total samples available, $num\_classes$ is the number of unique classes (2 in a binary

classification task), $num\_samples_i$ is the number of sample belonging to class i.

2.4 Reinforcement learning

Reinforcement Learning (RL) is a machine learning technique that tries to solve a problem in which a decision-maker, agent, can perform some actions inside a world, called environment. The agent senses the environment through the state and for each state the agent has to perform an action. These actions have two results: they change the agent's state and give him a feedback, the reward. The reward is a value which indicates if the action performed is good, then the reward is positive, or it is bad, the reward is negative. Given these rewards, the agent understands what action to perform in a given state to get a positive reward. This cycle of state-action-reward is repeated many times. The goal of the agent is to find a policy such that the actions performed lead to the maximum reward possible.

RL has become popular in the last years thanks to results obtained in some game environment like, Backgammon ([21]), Atari games ([22]), Go ([23]) but also robotics ([24]), self-driving cars, finance and other fields ([25])

2.4.1 Formal definition

Reinforcement learning can be formulated as a Markov decision process (MDP), indeed a MDP express the problem of sequential decision-making, where for each state s, the decision maker can choose any action a available in that state s. The process responds by moving with some probability to the state s' and giving the decision maker a reward $R_a(s, s')$.
The MDP is defined as a tuple of 4 elements (S, A, P, R), where:

- S is a set of states, called the *state space*.
- A is a set of actions, called the *action space*.
- P is the probability from state s, at time t, of reaching state s', at time $t + 1$ with action a:
$$P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$$
- $R_a(s, s')$ is the immediate reward received after transitioning from state s s to state s', due to action a; so the reward at time t, also r_t.

The state and action space may be finite or infinite.

The MDP is controlled by a sequence of discrete time steps that create a trajectory υ:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

where the states follow the state transition $P_a(s, s')$. The transition function and the reward function are determined only by the current state, and not from the previous states. This property is called Markov property, which characterises the MDP and it means that the process is memoryless and that the future state depends only on the current one and not on its history.

The goal of the MDP is to find a good policy for the decision maker, that is, a function π that specifies the action a that will be chosen when in state s. The policy π found will maximise the cumulative reward over a trajectory υ:

$$G(\upsilon) = \sum_{t=0}^{\infty} r_t$$

This return value has the problem that all the rewards contribute in the same weight and this can create some problems due to the lack of temporal information. A better return value would be to give more importance to the short-term memories and giving less importance to the ones far in the future. This is solved by introducing a discount factor, denoted with γ. Then the corrected formula is:

$$G(\upsilon) = \sum_{t=0}^{\infty} \gamma^t r_t$$

with value of γ satisfying $0 \leq \gamma \leq 1$. When γ is closer to zero, the agent will tend to consider only immediate rewards whether if γ is closer to one, the agent will consider future rewards with greater weight, willing to delay the instant reward in favour of a greater cumulative reward. This new definition of $G(\upsilon)$ is the total discounted reward.

A simple decomposition of $G(\upsilon)$ is :

$$G_t(\upsilon) = r_t + \gamma G_{t+1}(\upsilon)$$

so the return value G can be divided in the reward at time t plus the discounted total reward at time $t + 1$.

Another important notion in MDP and RL is the value function, known as V-function. While the return $G(\upsilon)$ gives the reward over a trajectory, it does not tell much about how good the single states are. The value function does exactly this: it estimates how good it is for the decision maker to be in a given state. The notion of "how good" is defined in terms of future rewards that the decision maker can expect in terms of expected return.
The value function $V_\pi(s)$ can be formally defined as:

$$V_\pi(s) = \mathbb{E}_\pi(G(\upsilon)|s_0 = s) = \mathbb{E}_\pi(\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s)$$

The expected return when starting at state s and following policy π.

Similarly, it is possible to define the action-value function, also known as Q-function as the expected return from state s with an initial action a:

$$Q_\pi(s,a) = \mathbb{E}_\pi(G(\upsilon)|s_0 = s, a_0 = a) = \mathbb{E}_\pi(\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a)$$

Furthermore, the value function and the action-value function are related and satisfy a particular relationship, used in many RL contests, that for any policy π and state s, the following condition holds:

$$V_\pi(s) = \mathbb{E}_\pi(Q_\pi(s,a))$$

Knowing the optimal Q-function ($Q^\star$), to maximise the V-function ($V^\star$), the action best has to be found. This is found with:

$$a^\star(s) = argmax_a Q^\star(s,a) \tag{2.6}$$

That is: the best action is the one that maximises the Q-function. Moreover, the V-function can be decomposed in two terms:

$$V_\pi(s) = \mathbb{E}_\pi(G(\upsilon)|s_0 = s) = \mathbb{E}_\pi(r_t + \gamma V_\pi(s_{t+1})|s_t = s) \tag{2.7}$$

where r_t is the reward at time t and $\gamma V_\pi(s_{t+1})$
the discounted total reward of the next state.

The equation in 2.7 is the Bellman Equation that defines the value function recursively, enabling the estimations of the next states.
Similarly, it is possible to write the Bellman equation for the Q-function:

$$Q_\pi(s,a) = \mathbb{E}_\pi(G(\upsilon)|s_0 = s, a_0 = a) = \mathbb{E}_\pi(r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})|s_t = s, a_t = a)$$

In this way the V-function and the Q-function are updated with the values of the successive states without the need to know the trajectory till the end.

2.4.2 Model

In this section is being reported the main ANNs used for the controlling part.

Deep deterministic policy gradient

As state in equation 2.6, it is possible to see that the best action is chosen in order to maximise the Q-function among all the possible actions.
This works in the case of a discrete action space, where it is possible to easily

compute the $argmax_a$ operation, just comparing the Q-values of each action. This is not obvious for a continuous case.

The deterministic policy gradient (DPG) implementation tries to solve this problem. The idea behind DPG is to learn a deterministic function $\mu_\phi(s)$ that can approximate the $argmax_a Q(s, a)$ operation.
One possible way to find this deterministic function is using ANN and in particular deep neural networks. So, the term deep deterministic policy gradient merges the DPG idea with the use of ANN.

In particular, the DDPG is an actor-critic algorithm, and it uses two neural networks. The actor (characterised by weight parameters ϕ) decides the action to perform in a given state, and the critic (characterised by weight parameters θ) evaluates the action chosen by the actor.

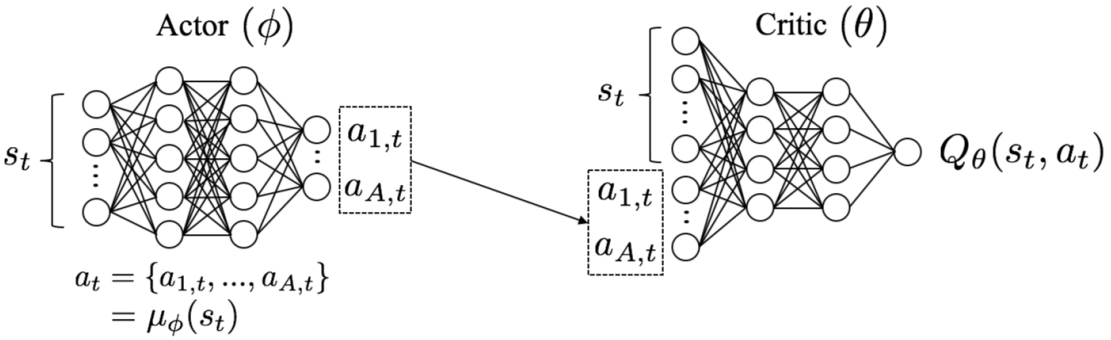


Figure 2.9: Actor-critic model architecture of the DDPG agent [26].

During exploration, give the state s_t the agent performs some action a_t, moving to the next state s_{t+1}, obtaining a reward r_t. This information, with a boolean flag stating if the game ended (d_t) or not, is stored in a replay memory as a tuple $(s_t, a_t, r_t, s_{t+1}, d_t)$. This replay memory is then used during training: a batch B is randomly chosen, and the agent is trained on it.
Both networks are trained, updating the weights ϕ and θ, in particular, the weight from the critic are update with gradient descent and following loss:

$$\mathcal{L}(\theta) = \sum_B \left(\underbrace{Q_\theta(s_t, a_t)}_{i} - \Big(\underbrace{r_t + (1 - d_t)\gamma Q_\theta\Big(s_{t+1}, \mu_\phi(s_{t+1})\Big)}_{ii} \Big) \right)^2 \tag{2.8}$$

At the same time, the actor network's weights ϕ are updated maximising the

following loss function with gradient ascent:

$$\mathcal{L}(\phi) = \sum_{B} Q_\theta\Big(s_t, \mu_\phi(s_t)\Big) \tag{2.9}$$

The goal is to update the weights such that the action chosen by the actor ($\mu_\phi(s_t)$) maximises the Q-value ($Q_\theta(s_t, a_t)$).

Given equation 2.8 it is possible to see that both the predicted value i and the target value ii are calculated with the same network Q_θ, this makes the training unstable. A solution to this problem is to use two other target networks ϕ_{target} and θ_{target}. So the equation 2.8 becomes:

$$\mathcal{L}(\theta) = \sum_{B} \left(Q_\theta(s_t, a_t) - \Big(r_t + (1 - d_t)\gamma Q_{\theta_{target}}\Big(s_{t+1}, \mu_{\phi_{target}}(s_{t+1})\Big)\Big)\right)^2$$

While the equation 2.9 does not change.

The target networks weights are updated after a fixed number of steps, suing the Polyak averaging:

$$\phi_{target} = \rho\phi + (1 - \rho)\phi_{target} \tag{2.10}$$

where ρ is a scaling factor usually very small.
In a similar way, equation 2.10 is used when updating the critic target network.

2.4.3 Evaluation metrics

Generally, the evaluation metrics for RL are different from the ones of supervised learning.
Moreover, in a RL framework there are no standard metrics, but it depends on the task that it is trying to solve.

A popular evaluation of the agent's choices is looking at the reward over each step, or the cumulative reward over each episode. If the agent is learning well, the reward will increase over time.

For this specific task some other statistics about the agent performance are reported, in particular an analysis of when the agent choses an action is reported in the results chapter. These statistics refer to the usage of the active and reactive power and if the action chosen was needed to solve the critical voltage situation or not.

2.5 Literature review

The interest in power networks has increased in the last years, especially for the introduction of distributed energy resources and the progress in the field of artificial intelligence. Particular attention is paid to the dispatching of the required power, as it is important for a utility company to predict the load consumed in order to avoid losses. Predicting the power demand is important because the energy generation must balance the consumption, since there are no efficient ways to store the surplus energy at a reasonable cost. Some papers have tried to forecast the load requests with different techniques: support vector machine [27], wavelet transform [28], some exponential smoothing methods [29], artificial neural network [30]. The authors in [31] use an interesting approach with a recurrent gradient boosting regression model to predict the future load consumption and detect power theft. Similarly, some papers deal with the forecasting of the wind generators' output using radial basis function [32], support vector machine [33] and recurrent neural network [34].
A lot of concern is placed in forecasting devices' critical situations in the network. The authors in [35] perform a deep analysis of a network using classification methods to predict critical loading situations and regression models to predict the bus voltage magnitudes; the tests are limited to only multi layer perceptron as deep models. The authors in [36] predict the over voltage instabilities using a recurrent neural network in a low voltage distribution network. The authors forecast buses critical situations considering the voltage's phase angles information of consecutive buses instead of the voltage magnitudes; the tests are limited to a small network and time series are not taken in consideration. In [37], authors perform voltage security monitoring, with particular attention to blackouts, using different machine learning models; among which multi layer perceptron models.
Some possible ways are proposed to reduce the number of contingencies in the power networks. The authors in [38] use optimal power flow calculation to control generators' active and reactive power in a small distribution network, in [39] the authors propose a droop-based reactive power compensation.
Thanks to its increasing popularity in many fields; when solving voltage problems, some solutions have been proposed with reinforcement learning algorithms: in [40] the authors developed some reinforcement learning environment playgrounds where they tested proximal policy optimisation and soft actor-critic algorithms; in [41] the authors used a deep deterministic policy gradient algorithm to change active and reactive power of the generators; in [42] authors used multi deep reinforcement learning framework to control static var compensators and batteries to balance the voltage magnitude in a network; for a similar task, the same authors, in [43] control generators' reactive power; in [44] authors control batteries and heat pumps to maximise energy consumption and reduce losses.

The mentioned works either: *a)* consider only small test networks, *b)* do not deal with unbalanced datasets, and *c)* have high values of active curtailment. These are some limitations since: *a)* small networks may result in a not realistic analysis, *b)* power grids are highly secure, so the number of voltage problems is small with respect to the number of non-critical situations, *c)* the goal is to minimise losses, so the generators' active power curtailment should be reduced to low values or avoided.

Chapter 3

Case study

This chapter provides the information about the use case studied. In particular, the description of the MV Oberrhein network, an overview of the SimBench database and how the time series are selected.

3.1 MV Oberrhein network

The thesis is developed in Pandapower, an open-source and popular Python library that makes it easy when working with power networks.
Pandapower has many features, some of them are reported here:

- Every element in Pandapower (load, generator, ...) is considered as a Pandas' data frame that holds all parameters for a specific component. This implementation makes it possible to add custom columns without influencing the Pandapower functionality.
- It allows calculating in an easy and fast way the PF, essential for network analysis. Its PF solver is based on the Newton-Raphson method.
  After the PF computation, the results are store in another Pandas'data frame.
- It allows running time series simulations.
- It has many predefined power networks.

Among all the power networks available in the Pandapower library, the one used for these experiments is the MV Oberrhein network. MV Oberrhein is a real distribution located at Upper Rhine (in German: Oberrhein), Germany. This network is a generic 20 kV power system serviced by two 25 MVA HV/MV transformer stations. The network supplies 141 HV/MV substations and 6 MV loads through four MV feeders. The network layout is meshed, but the network operates as a

radial network with some open sectioning points, i.e., redundant lines/cables. This is common in real networks: they are usually meshed, but they operate in a radial way.
The grid also includes geographical information of lines and buses and is assembled from openly available data [45, 46].

The representation of the network is presented in 3.1. The blue dots represent the buses where load and/or generators are connected to, and the yellow squares represent the HV/MV substations.

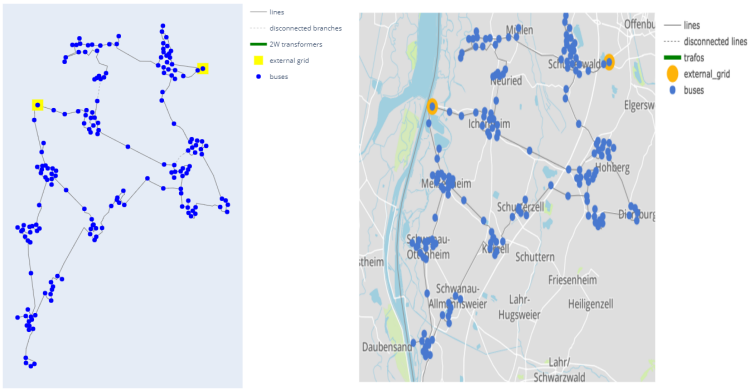


Figure 3.1: MV Oberrhein network from Pandapower. Representation plot on the left and geographical plot on the right.

To simplify the situation, the network can be divided in 2 independent parts:

(a) **(b)**

Figure 3.2: MV Oberrhein network division in the two parts, each with its own external grid.

The network used in this thesis is the one showed in figure 3.2b. The network consists of: one external grid, one transformer, 70 buses, 61 loads and 60 generators.

3.2 SimBench database

The time series used is taken from the SimBench database. This database refers to some real distribution networks in Germany of the year 2016. SimBench includes multiple time series for one year with 15 min resolution for loads, generators and storage units. The time series were grouped by element type, reducing the total number of required time series to a reasonable number, while retaining the possibility to model individual nominal power [47]. All active power values are normalised to the maximum active power value.

Power utilities commonly use generic load profiles to group commercial customers with similar load shapes into categories or standard load profiles (SLPs). The most commonly used profile sets are developed by the German Association of Energy and Water Industries (BDEW). It comprises eleven aggregated profiles, one for residential consumers, three for agricultural, and seven for commercial consumers with different opening hours. They are differentiated into workdays, Saturdays and Sundays as well as three seasonal categories winter, summer, and transitional. The set also includes two profiles for street lightning and band load. The generation time series for photovoltaics (PVs) and wind energy dataset are generated using an agent-based simulation tool for optimised grid expansion planning: SIMONA. SIMONA's power plant models receive real weather data of Germany from the German Weather Service in 2011 for Wind and 2012 for PV time series as input data.
For 2011 and 2012 generation data, the time axis is adjusted to 2016 by shifting days so that they correspond to the nearest weekday[48].

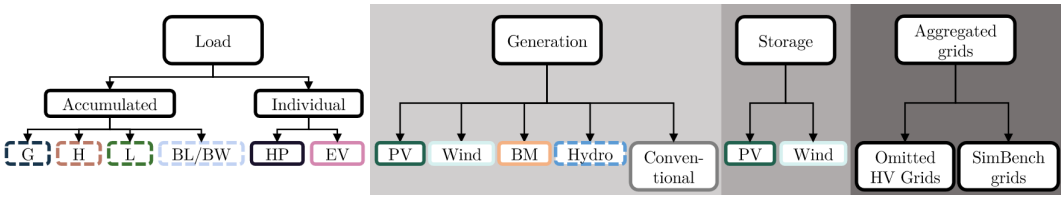


Figure 3.3: Overview of the SimBench time series type.

The load time series were distinguished between real measured accumulated, highlighted with a dash, and simulated individual consumers, marked with a solid frame in figure 3.3.

3.3 Time series selection

Some time series from the SimBench database are taken to adapt to the number of loads and DERs of the MV Oberrhein network in consideration.

As said, each element (load or generator) falls under a specific profile type that represents the consumption or generation over time.

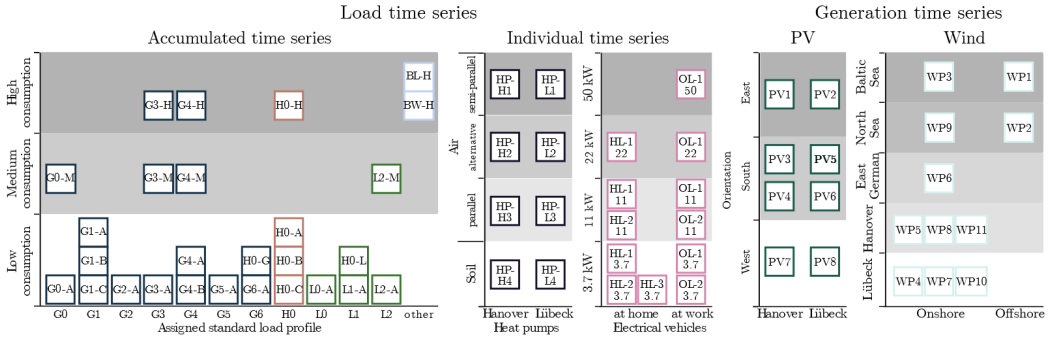


Figure 3.4: Loads and generations standard profiles.

In this case, the profile's types for loads and DERs are chosen, such that every profile type is present, to have a network as close as possible to a real network. The profiles' distribution in the Pandapower network is as follows:

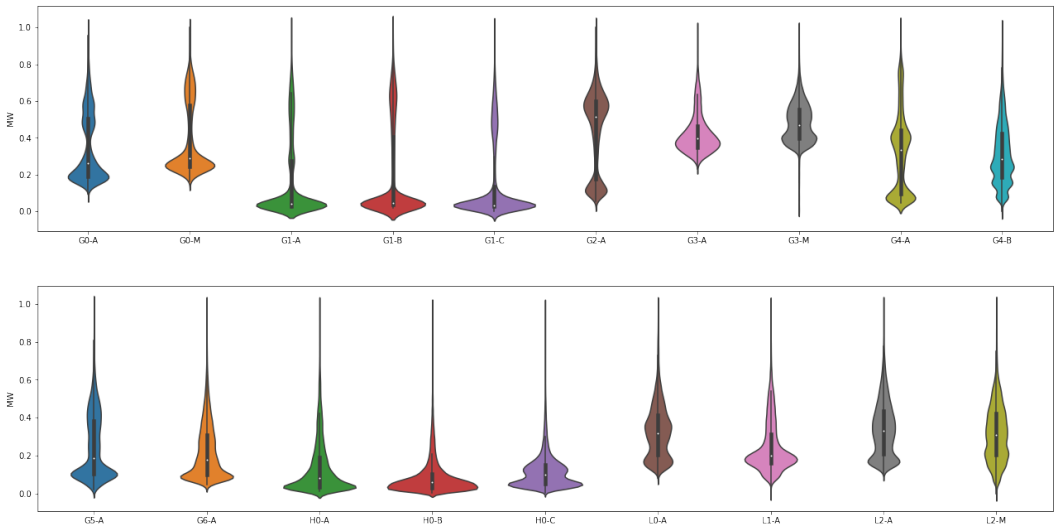
Load elements by type: {"G0-A": 4, "G0-M": 4, "G1-A": 3, "G1-B": 3, "G1-C": 3, "G2-A": 3, "G3-A": 4, "G3-M": 3, "G4-A": 3, "G4-B": 3, "G5-A": 3, "G6-A": 3, "H0-A": 3, "H0-B": 3, "H0-C": 4, "L0-A": 3, "L1-A": 3, "L2-A": 3, "L2-M": 3}

DER elements by type: {"PV1": 7, "PV3": 7, "PV4": 7, "PV5": 8, "PV6": 8, "PV7": 8, "WP4": 7, "WP7": 8}

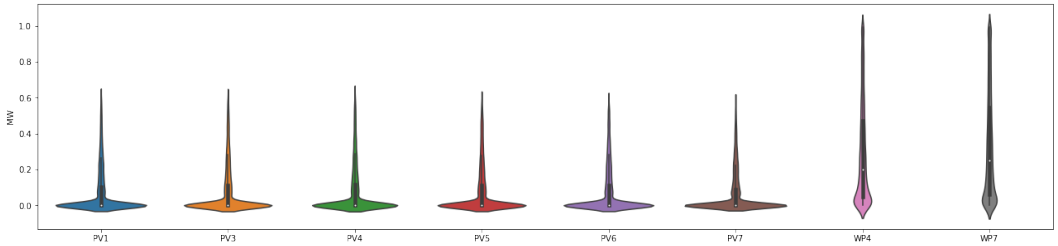
---

where the letters stand for: commercial enterprises (G), households (H) and agricultural holdings (L); with last letters -A to -C indicating low consumption and -M medium consumption customers.
For the DER devices there are photovoltaics PVs and wind parks WPs. It is possible to notice a bigger presence of PVs over WPs.

The loads and DERs are chosen so that different profile types are present.



(a) load profiles



(b) generator profiles

Figure 3.5: Violin plots of loads and generators for each profile type.

From plots in figure 3.5 it is possible to see the distribution of consumptions and generation of the different profiles.

Since the profiles are similar for every element of the same type, some noise is added to increase randomness. In particular, the noise added is a scaling factor in the range [0.85,1.15]. The scaling factor allows avoiding negative values in case of a value lower than 1; subtraction may result in a negative value of active or reactive power for a particular device, either load or generator.

It is possible to use some scaling factors to easily generate different case that can fit with the network considered. The scaling factors, chosen in this case, are:

scale_factor_load = 0.8
scale_factor_der = 1.2

These values are chosen to be close to the load and generation peak capacity of the MV Oberrhein network.

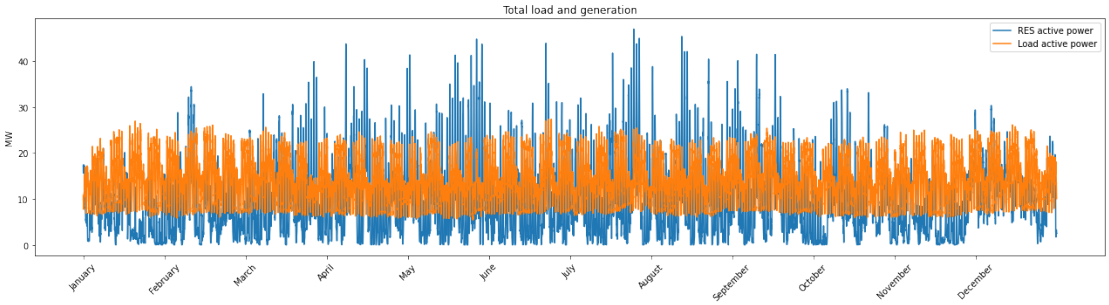


Figure 3.6: Sum of load energy consumption and DER energy generation over the considered year.

The load consumption values are as a typical MV network ([49]), while the generation is higher. This case can represent a future power network when the number and the performance of DER devices increase, so to have a generation of power higher than the demand; especially during the summer period when the consumption is lower and the generation is higher.
Such situations are critical for a network since the surplus energy increases the voltages at the buses that may experience voltage problems.

In this thesis, the network situation, in a given time step t, is considered critical if the voltage of at least one of the buses is out of the boundaries $V_t < v^{\min}$ (under voltage) or $V_t > v^{\max}$ (over voltage), where $v^{\min}$ is the minimum acceptable voltage, 0.95 p.u., and $v^{\max}$ is the maximum one, 1.05 p.u.

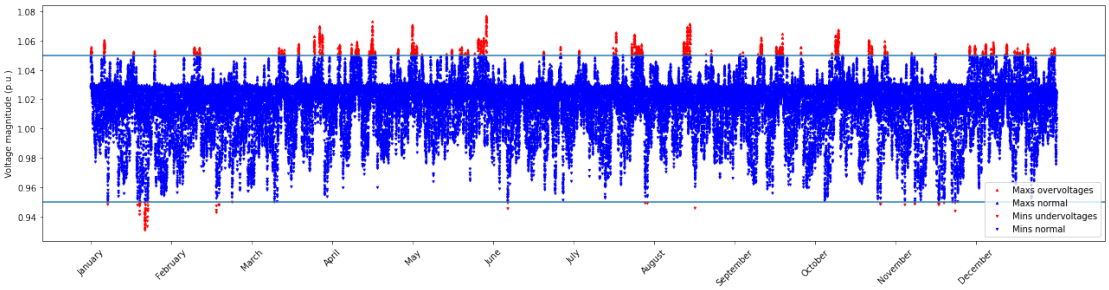


Figure 3.7: Network's over and under voltage critical situations. The two horizontal lines correspond to $v^{\max}$ and $v^{\min}$.

In figure 3.7, it is possible to see that the number of over voltages situations is low compared to the normal situations. It is possible to see that there are also few under voltage situations.

This concludes the part relative to aim **I)** to adapt to a real power network some realistic time series.

Chapter 4

Problem statement and solving methodology

This chapter describes the elements and methodology used to solve the forecasting and control problems.

4.1 Problem statement

This thesis focuses on ANM and the problem faced by a DSO to maintain the network within its operational limits. In particular, the system operator evaluates whether at a given moment there will be a voltage problem. In this way, the DSO can proceed with some actions, like applying curtailment to generator devices or control their reactive power, to maintain the voltage inside a safe range.

For this problem, we assume that the DSO knows the following information:

- The network topology: the number of buses, loads and generators, the lines' length, the distance between the connected buses, and the distance between each load and generator from the bus they are connected to. Moreover, the impedance of the lines is known.
- The active and reactive power of some loads at each time step.
  In a real case it would be possible that some values may not available mainly for two reasons: *a)* for that particular load there are no measurements at all for the lack of sensors or for privacy reasons; or *b)* communication problem, so the sensor recording was lost. These cases are not considered in this thesis, so all the information is available.
- The type of DER device connected to each bus. If the DER device is directly

connected to the medium voltage grid, the active power and reactive power are considered known.

This data is used to calculate the power flow of the network and obtain more information like the voltage magnitude at each bus, lines' loading and other values. This calculation can be performed with any power system analysis tool, like for example Pandapower.

A power network can be represented as a direct graph $\mathsf{G}(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of nodes, also called buses ($\mathcal{B}$), in the network; $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of directed edges linking two buses together, also called lines. The notation $e_{ij} \in \mathcal{E}$ refers to the directed edge with sending bus i and receiving bus j. Each bus might be connected to several devices, which may inject or withdraw power from the grid. The set of all devices is denoted by $\mathcal{D}$ that can be either loads $\mathcal{L}$ or generators $\mathcal{G}$. Let's also define $\mathcal{T}$ as the set of transformers.

The DSO considers the behaviour of the network over a set of discrete time steps $t \in \{1,2,...,T-1,T\}$ of length Δt, with, $T \in \mathbb{N}$, the last time step of the time series' horizon. A time step is considered as a snapshot of the system at one particular point in time.

The DSO can have access to some information, let's define $\mathbf{I}$ as the information domain. This information can be divided in static information like the network topology, and the observation $\mathbf{O_t}$ collected at every time step t like the loads and generators' active and reactive power and the buses' voltage magnitude and lines' loading. The information at time step I_t can be defined as:

$$\mathbf{I_t} \in \mathbf{I}, \text{with}\, t \in \{1,2,...,T-1,T\}$$
$$\mathbf{I_t} = [\mathbf{static\,information}, \mathbf{O_t}]$$

In general power networks are not static, since the operators can modify their topology, for example changing the connections due to some incidents on the lines. In this thesis, the network topology is considered as static, so that no changes are applied on the network during the whole time series' horizon.

An instance of the observation $\mathbf{O_t}$ can be:

$$\mathbf{O_t} = [\mathcal{L}_\mathbf{t}^\mathbf{p}, \mathcal{L}_\mathbf{t}^\mathbf{q}, \mathcal{G}_\mathbf{t}^\mathbf{p}, \dots]$$

where $\mathcal{L}_\mathbf{t}^\mathbf{p}$ and $\mathcal{L}_\mathbf{t}^\mathbf{q}$ are active and reactive power of loads $\mathcal{L}$ at timestamp t; $\mathcal{G}_\mathbf{t}^\mathbf{p}$ is the active power of the generators $\mathcal{G}$ at time t; some other variables can be included in the observation $\mathbf{O_t}$ for example the buses' voltage magnitudes or the loading

percentage of each line, etc.

For the forecasting part, the operator predicts if the system, at some given $t+n$ future time steps, will be in a critical condition. As mentioned in 3.3, the network is considered in a critical condition if any of its elements is in an unsafe situation, for example an over voltage condition. Let define $\mathbf{C}$ as the list of critical situations and C_t the critical situation at time step t, with $C_t \in \{0,1\}$.
For predicting whether the system will be in a critical situation or not, the DSO considers the history of the system only for h preceding steps, with $h \in \mathbb{N}^+$.

4.2 Solving methodology

4.2.1 Forecasting

The goal of this section regards the explanation of the solving methodology for aim **II)**, that is, to train a classifier that can forecast whether in the future there will be some over voltage problems.

As a supervised task, the time series are separated in training and testing dataset; in particular, the data is divided in windows of length, h for the inputs $\mathbf{x}$ and length n for the outputs $\mathbf{y}$.

There are many ways and combinations of how to choose the information used for the input $\mathbf{x}$s. In this thesis, three combinations are tested:

i) Considering only the voltage information at each bus.

$$\mathbf{x} = [\mathcal{B}^{\mathbf{V}}_{\mathbf{t-h+1}}, \mathcal{B}^{\mathbf{V}}_{\mathbf{t-h+2}}, \dots, \mathcal{B}^{\mathbf{V}}_{\mathbf{t-1}}, \mathcal{B}^{\mathbf{V}}_{\mathbf{t}}] \tag{4.1}$$

where $\mathcal{B}^{\mathbf{V}}_{\mathbf{t}}$ is the voltage level V of every bus $\mathcal{B}$ at timestamp t. It can be represented as a matrix, as follows:

$$\mathbf{x} = \begin{bmatrix} \mathcal{B}^V_{t-h+1,1} & \mathcal{B}^V_{t-h+2,1} & \cdots & \mathcal{B}^V_{t-1,1} & \mathcal{B}^V_{t,1} \\ \mathcal{B}^V_{t-h+1,2} & \mathcal{B}^V_{t-h+2,2} & \cdots & \mathcal{B}^V_{t-1,2} & \mathcal{B}^V_{t,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathcal{B}^V_{t-h+1,|\mathcal{B}|-1} & \mathcal{B}^V_{t-h+2,|\mathcal{B}|-1} & \cdots & \mathcal{B}^V_{t-1,|\mathcal{B}|-1} & \mathcal{B}^V_{t,|\mathcal{B}|-1} \\ \mathcal{B}^V_{t-h+1,|\mathcal{B}|} & \mathcal{B}^V_{t-h+2,|\mathcal{B}|} & \cdots & \mathcal{B}^V_{t-1,|\mathcal{B}|} & \mathcal{B}^V_{t,|\mathcal{B}|} \end{bmatrix}$$

where $\mathcal{B}^V_{t,i}$ is the voltage level V of a single bus i at timestamp t.

ii) Considering active and reactive power of loads and active power of generators.

$$\mathbf{x} = [\mathcal{L}^{\mathbf{p}}_{\mathbf{t-h+1}}, \mathcal{L}^{\mathbf{q}}_{\mathbf{t-h+1}}, \mathcal{G}^{\mathbf{p}}_{\mathbf{t-h+1}}, \dots, \mathcal{L}^{\mathbf{p}}_{\mathbf{t}}, \mathcal{L}^{\mathbf{q}}_{\mathbf{t}}, \mathcal{G}^{\mathbf{p}}_{\mathbf{t}}] \tag{4.2}$$

where $\mathcal{L}^{\mathbf{p}}_{\mathbf{t}}$ and $\mathcal{L}^{\mathbf{q}}_{\mathbf{t}}$ are, respectively, the active and reactive power of all loads $\mathcal{L}$ at timestamp t and $\mathcal{G}^{\mathbf{p}}_{\mathbf{t}}$ is the active power of each generator $\mathcal{G}$ at timestamp t. Similarly to the first case **i)**, a matrix can be written, but it is omitted.

iii) Considering active power of generators and the active and reactive power at the transformer on the HV/MV substation.

$$\mathbf{x} = [\mathcal{G}^{\mathbf{p}}_{\mathbf{t-h+1}}\mathcal{T}^{p}_{t-h+1}, \mathcal{T}^{q}_{t-h+1}, \dots, \mathcal{G}^{\mathbf{p}}_{\mathbf{t}}, \mathcal{T}^{p}_{t}, \mathcal{T}^{q}_{t}] \tag{4.3}$$

where $\mathcal{T}^{p}_{t}$ and $\mathcal{T}^{q}_{t}$ are, respectively, the active and reactive power of the transformer at timestamp t.

iv) Since the considered dataset is unbalanced, some techniques usually used when dealing with these kinds of datasets are employed. The techniques are, as mentioned in 2.3.4, over and under sampling and set weights for each class. These techniques are tested on the best and worst performing models of all the aforementioned combinations.

Different combinations are considered because it is interesting to check how the model performs with different kinds of data; moreover, the information a DSO can have access to can be limited, so some cases are more realistic than others. In particular, DSO usually has access to only a limited amount of data: for small loads (households, ...) the active and reactive powers are not available at every instant (they are only after some period of time, after the bill is issued), in general the more a device is close to the transformer, the more information a DSO has access to, so the combination **iii)** would be the most appropriate for a MV network.

For the labels, $\mathbf{y}$ can be defined as:

$$\mathbf{y} = [C_{t+1}, C_{t+2}, \dots, C_{t+n-1}, C_{t+n}] \tag{4.4}$$

where C_t is the condition of the system at timestamp t, with $C_t = 1$ if the network is in a critical situation or $C_t = 0$ if it is in a normal situation and n is the number of future time steps considered, with $n \in \mathbb{N}^+$.

For what concerns h, the number of history step considered and n, the number of future steps to forecast, these are not hyperparameter to be optimised, but are

values that depend on the problem and the task that a DSO is trying to solve. In particular, choosing a value of h depends on the amount of data the DSO can access to, the computation cost and the time to get an answer; while n, depends on the kind of forecasting the DSO wants to apply, either short planning (from minutes to hours), medium-term planning (from hours to days) or long-term planning (many days). It has also to be noted that it is expected that higher values of n would get less accurate forecasting.

The couples $\{\mathbf{x}, \mathbf{y}\}$ are divided in training, validation and test set, with the following ratios: 0.7, 0.2, 0.1.

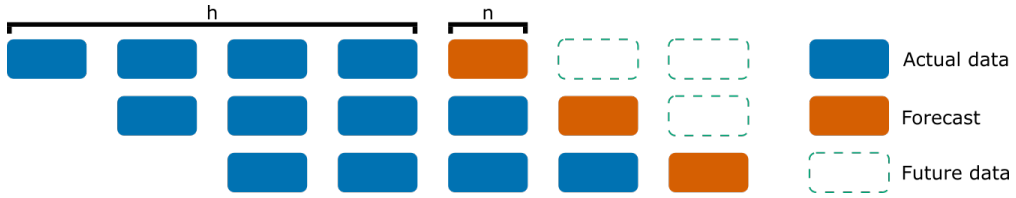


Figure 4.1: Division of the time series in windows of length h as past information and length n as forecasting values.

Figure 4.1 shows how the time series are separated in input and output lists: the input $\mathbf{x}$ is generated taking the raw information of size h ($\mathbf{x}$ is different for each combination used), and the output $\mathbf{y}$ is generated taking the information about the buses' voltage magnitudes of length n and converted to a binary value that represent whether the system is in a critical situation or not as mentioned in 3.3 ($\mathbf{y}$ is the same for each combination considered).

For the forecasting part, only over voltage situations are considered.
The followings are the critical states' distribution in training, validation and test sets.

Number of over voltage situations: 1229, over 35040 time steps, ratio: 3.5%

Number of over voltage situations in Training set: 835, ratio: 3.4%
Number of over voltage situations in Validation set: 211, ratio: 3.0%
Number of over voltage situations in Testing set: 183, ratio: 5.2%

The data is normalised using the formula:

$$\bar{x} = \frac{x - \mu}{(\sigma + \epsilon)}$$

where x is the data to be normalised, μ is the mean of the training data, σ the standard deviation of the training data and ϵ a small number to avoid dividing by zero.
Only the training data is used to calculate the mean and standard deviation, and these values are then used to normalise training, validation and test set.

Three different ANNs are tested:

- a MLP with three hidden layers, composed by 256, 128 and 128 neurons.
- a CNN with one 1-D convolution hidden layer 128 filters followed by two FC layers with 128 neurons each.
- a RNN with one LSTM unit followed by two FC layers with 128 neurons each.

Each FC layer presents a batch normalisation and a dropout layer to improve the score.

The trained model predicts the critical condition of the system at some future time steps n. Let's define $\hat{\mathbf{y}} = \hat{\mathbf{C}}$ as the forecasted values of the system given the information $\mathbf{x} \in \mathbf{I}$. A single instance $\hat{C}_{t+i}$ ($\hat{C}_{t+i} \in \{0,1\}$) states if the system is critical ($\hat{C}_{t+i} = 1$) or not ($\hat{C}_{t+i} = 0$), with $i \in \{1,2,\ldots,n-1,n\}$ and $n \in \mathbb{N}^+$.

The main goal of the model is predicting the future values such that the actual critical values $\mathbf{C}$ and forecasted values $\hat{\mathbf{C}}$ are as close as possible. The performances of the models are evaluated with the metrics defined in 2.3.3.

4.2.2 Active control

The goal of this section regards the explanation of the solving methodology for aim **III)**, that is, to train an agent that can take some action to solve the voltage problems.

The agent is trained within a reinforcement learning context.
Let's define the main RL elements:

- *Environment.* The environment is the entire grid, so the MV Oberrhein network.
- *State.* The state is the information the agent can have access to. In this case, active and reactive power of loads and active power of generators at time step t.

$$\mathbf{s_t} = [\mathcal{L}^{\mathbf{p}}_{\mathbf{t}}, \mathcal{L}^{\mathbf{q}}_{\mathbf{t}}, \mathcal{G}^{\mathbf{p}}_{\mathbf{t}}]$$

where, similarly to the forecast part **ii)**, $\mathcal{L}_\mathbf{t}^\mathbf{p}$ and $\mathcal{L}_\mathbf{t}^\mathbf{q}$ are, respectively, the active and reactive power of all loads $\mathcal{L}$ at timestamp t and $\mathcal{G}_\mathbf{t}^\mathbf{p}$ is the active power of each generator $\mathcal{G}$ at timestamp t.
So the state is a list of continuos variables of size 182 ($|\mathcal{L}| + |\mathcal{L}| + |\mathcal{G}| = 61 + 61 + 60$).

- *Action.* The agent can control active and reactive power of the generators, in particular:

$$\mathbf{a_t} = [\mathbf{a_t^p}, \mathbf{a_t^q}]$$

with $\mathbf{a_t^p}$ an array with the active power reducing factors at time step t ($a_{t,i}^p \in$ [0,1] and $i \in 1, ..., |\mathcal{G}|$) and $\mathbf{a_t^q}$ an array with reactive powers values ($a_{t,i}^q \in [-0.25,0.25]$). So the state is a list of continuos variables of size 120 ($2 \cdot |\mathcal{G}|$).

These values are used to change active and reactive power of the generators at the time step $t + 1$.
$\mathbf{a_t^p}$ represents the proportion of the quantity of energy curtailment decided by the agent. In formula:

$$\mathbf{final\_power_{t+1}} = \mathbf{initial\_power_{t+1}} \cdot (\mathbf{1} - \mathbf{a_t^p})$$

with $\mathbf{initial\_power_{t+1}}$ the initial power the generator was about to output at time step $t + 1$ (same as $\mathcal{G}_{\mathbf{t+1}}^\mathbf{p}$. The notation **initial_power** is used for a clearer understanding of what happens before and after the agent's action); $\mathbf{final\_power_{t+1}}$ the curtailed power, actual power output. Moreover

$$\mathbf{initial\_power_{t+1}} \cdot \mathbf{a_t^p}$$

is the total energy loss.

For the reactive power, $\mathbf{a_t^q}$ represents the quantity of reactive power absorbed or injected in the network by each generator.

- *Reward function.* The reward function consists of four terms:
  - A reward regarding the active power. The agent is punished to choose too high values of active power curtailment, the higher the curtailment, the higher the punishment.

$$reward\_p_t = -\sum_{i=0}^{|\mathcal{G}|} a_{t,i}^p$$

– A reward for the reactive power. The agent is punished proportionally to the change in the values of reactive power.

$$reward\_q_t = -\sum_{i=0}^{|\mathcal{G}|} |a_{t,i}^q|$$

the $|\cdot|$ is needed, since $\mathbf{a_t^q}$ may contain both positive and negative values.

– A reward regarding voltage violation. The agent is punished if the voltage magnitude of the buses is away from 1 p.u. The punishment value is chosen with the following a punishment function f:

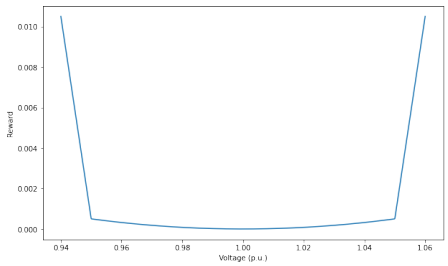


Figure 4.2: Shape of the voltage violation function centred in 1 p.u.

The idea is to punish the agent by a low value when the voltage bus is in the range [0.95,1.05] and to punish it more when the voltage bus is more far away from 1 p.u.

$$reward\_vv_t = -\sum_{i=0}^{|\mathcal{B}|} f(\mathcal{B}_{t+1,i}^V)$$

– A reward for the critical situation solved. The agent is given a positive or negative reward whether it was able to solve the critical situation or not. In particular:

$$reward\_cs_t = C_{t+1} - 4 \cdot network\_status_{t+1}$$

with C_{t+1} the critical status of the network as stated in the forecasting section (expression 4.4) at time step $t+1$, and $network\_status_{t+1}$ a boolean value stating if the system at time step $t+1$, after the agent's action and the PF calculation, is critical or not.
The main idea is to make the agent understand if the chosen action was helpful and solved the critical situation of the network. In particular, there are four possible cases:

case 1) 0 - 0: no critical situation before and after the agent's action (good case, 0 reward).

case 2) 0 - 1: no critical situation before, but it was introduced by the agent's action (the worst case, -4 reward).

case 3) 1 - 0: critical situation before and it was solved by the agent's action (the best case, $+1$ reward).

case 4) 1 - 1: critical situation before and after the agent's action (not good case, -3 reward).

The total reward is given by the algebraic sum of all the previous terms multiplied by a scaling factor:

$$\begin{aligned} reward_t = \quad & \alpha_p \cdot reward\_p_t + \\ & + \alpha_q \cdot reward\_q_t + \\ & + \beta \cdot reward\_vv_t + \\ & + \gamma \cdot reward\_cs_t \end{aligned} \tag{4.5}$$

with $\alpha_p = 4$, $\alpha_q = 2$, $\beta = 100$ and $\gamma = 20$. These values are chosen to balance the rewards get by the agent.
With such articulate reward expressed in 4.5, the agent's goal is to reduce the number of over voltage situation without introducing new under voltage situations.

- *Agent.* The agent model chosen is the DDPG algorithm. As mentioned in 2.4.2, this algorithm can handle continuous state and action spaces, so it is suitable for this task.

  Both actor and critic networks are MLPs composed by three hidden layers with 256, 128, 128 neurons. The weights are initialised with a standard distribution with mean 0 and standard deviation 0.08 so that most of the weights are concentrated between $[-0.1,0.1]$. This is important since with a larger initial standard deviation, the model tended to saturate the output.

  To control both the active and reactive power of the generators, the final hidden layer of the actor networks is sent to two different branches: one for $\mathbf{a^p}$ with a Sigmoid function as activation function, and the other branch for $\mathbf{a^q}$ with Tanh as activation function and a scaling factor of 0.25.

  Some other hyperparameters are:

  – α, the learning rate for the actor networks, set to 10^{-4}.

  – β, the learning rate for the critic networks, set to 10^{-3}.

- τ, the scaling factor for updating the weights with equation 2.10, set to 10^{-3}.
- The weights of the critic networks are updated every 3000 time steps.
- The agent is trained every 3 time steps.
- To the action is added some noise, initially set to 0.1, and it decreases every 100 time steps of a factor of 0.995. The noise minimum is set to 10^{-5}, below this value the noise is not decreased any more.
- For all the networks, an Adam optimiser is used.
- The replay memory is set to 10^6. After the memory is full, the past experiences are overwritten with the new ones.

The agent is trained using 40%, 14016 time steps, of the data and tested on the remaining 60%, 21024 time steps. The training is repeated for 4 time for a total of 56064 time steps.

Algorithm 1 Pseudo-algorithm for the control of the network's devices

Initialise the network
Extract time series for each time step from Simbench database
Calculate PF for each time step and evaluate when the system is in a critical situation, obtaining **C**
Initialise agent with actor and critic networks: ϕ, θ, ϕ_{target} and θ_{target}
for each episode p **do**
  for each time step $t \in \{1,2,\dots,T-2,T-1\}$ **do**
    Get state $\mathbf{s_t}$ from the network
    Choose the action $\mathbf{a_t}$ with $\mathbf{a_t} = \phi(\mathbf{s_t})$
    Get active power of the generators at time step $t+1$: $\mathbf{initial\_power_{t+1}}$
    Apply the action $\mathbf{a_t}$: change active power with the $\mathbf{final\_power_{t+1}}$ and reactive power with $\mathbf{a_t^q}$ of the generators at time step $t+1$
    Calculate the PF at the time step $t+1$
    Calculate the reward r_t, as mentioned in 4.2.2
    Check the status of the environment: d_t
    Get new state $\mathbf{s_{t+1}}$
    Store the experience $(\mathbf{s_t}, \mathbf{a_t}, r_t, \mathbf{s_{t+1}}, d_t)$
    Train the agent
  end for
end for

It is worth mentioning that Pandapower allows performing time series analysis, but training the RL agent required a custom control of the time series execution.

For this reason, the open-source code of Pandapower is modified to allow a finer control of the different steps of the runs.

Chapter 5

Results and analysis

In this chapter, the results of the different methods will be reported.

The results are run on two main machines: Google Colab ([50]) for the forecasting part and my personal laptop for the controlling part.
Two systems are used for two main reasons: the possibility to run both forecasting and controlling tests at the same time and to reduce the training time (especially for the forecasting part) thanks to the powerful hardware from Google.
The laptop used is a Xiaomi Mi Laptop Air 13.3 (2017) with an Intel Core i5 6200U 2.30GHz CPU, a 8GB 1064MHz RAM and an NVIDIA GeForce 940MX 1023MB GPU.

5.1 Forecast results

In this section, the results obtained when solving the aim **II)** are reported.

For the forecasting task, some evaluation metrics are used to test the performance of the ANN. These metrics are: accuracy, recall, precision and F1-score.
During training, the F1-score on the validation set of the model is monitored, and the best weights configuration is stored. The score reported in the following tables, are obtained testing the model on the test set, with the most performing saved model.
It is worth mentioning that, at the time of this thesis development, the core TensorFlow library allows monitoring only the accuracy, recall, precision but not the F1-score; this can take to misleading, and it can lead to suboptimal results. For this reason, a custom monitor of the F1-score is developed.

In this thesis, three values of h are used: 2, 4 and 16 corresponding to have the

past information of the system of 30 minutes, 1 hour and 4 hours. For the n, only one value is used: 1, corresponding to a short planning time of 15 minutes ahead.

The three combinations are tested, and the results are reported as an average of 5 runs. The models are trained for 100 epochs on the training set, for each epoch the neural network is evaluated on the validation set and the best performing model is used to get the score on the testing set.
The batch size is 512. This is important since the dataset is unbalanced; in this way there is a high chance that each batch contains a decent amount of voltage problems to learn from.

- Results of the first combination **i)**:

h	Models	Accuracy	Precision	Recall	F1-score	Time (s)
2	**MLP**	0.979	0.786	0.833	0.804	73
	CNN	0.981	0.793	0.862	0.825	67
	RNN	0.981	0.798	0.858	**0.826**	121
4	**MLP**	0.979	0.817	0.777	0.792	74
	CNN	0.980	0.795	0.851	0.820	70
	RNN	0.981	0.796	0.860	**0.827**	205
16	**MLP**	0.978	0.800	0.797	0.796	77
	CNN	0.798	0.798	0.822	0.810	81
	RNN	0.980	0.786	0.866	**0.823**	441

Table 5.1: Results using as input the voltage information at each bus.

This task is the simplest case, since the goal is to predict the future of a particular quantity (voltage) given the past information of the same quantity. From table 5.1, It is possible to see that the RNN model performs better than the other models, since usually RNN handles better time series data. Moreover, it would be expected that the more data the ANNs receive as input (value of h), the higher the score it gets, but for this task, results look like are independent of the value of h.

- Results of the second combination **ii)**:

h	Models	Accuracy	Precision	Recall	F1-score	Time (s)
2	**MLP**	0.961	0.960	0.263	0.399	72
	CNN	0.969	0.948	0.429	0.570	66
	RNN	0.969	0.907	0.451	**0.591**	122
4	**MLP**	0.963	0.962	0.308	0.454	76
	CNN	0.965	0.945	0.359	**0.510**	71
	RNN	0.963	0.955	0.325	0.473	211
16	**MLP**	0.969	0.815	0.542	0.623	109
	CNN	0.971	0.843	0.580	**0.665**	112
	RNN	0.969	0.912	0.467	0.603	444

Table 5.2: Results using as input the active and reactive power of each load and active power of each generator.

This combination performs the worst. These results can be explained considering the complexity and non-linearity of the PF calculation. So the models are not able to grasp the relationship that maps the loads and generators' information to the buses' voltage. Moreover, the ANNs are missing all the information regarding the network, like for example the lines' resistance, impedance, etc; essential for the PF calculation. Another possible explanation is a large input space.

- Results of the third combination **iii)**:

h	Models	Accuracy	Precision	Recall	F1-score	Time (s)
2	**MLP**	0.974	0.727	0.837	0.774	152
	CNN	0.972	0.683	0.897	0.773	109
	RNN	0.974	0.692	0.915	**0.788**	295
4	**MLP**	0.971	0.683	0.893	**0.769**	181
	CNN	0.966	0.619	0.950	0.748	127
	RNN	0.970	0.669	0.910	0.764	468
16	**MLP**	0.964	0.628	0.887	0.728	236
	CNN	0.970	0.679	0.873	0.757	380
	RNN	0.973	0.703	0.899	**0.782**	617

Table 5.3: Results using as input the active each generator and the active and reactive power of the external grid's transformer.

This case is the more realistic one, since a DSO has always the information at the transform on the external grid and the active power of the generators as well.

The score is lower than combination **i)** but better than combination **ii)**.

- Testing some techniques for unbalanced datasets **iv)**.

The process of over sampling and under sampling are performed such that the ratio between the number of critical situations and the number of total time steps on the training set is around 20%.
Instead, the weights are calculated as mentioned in 2.3.4 and their numerical values are:

Weight for class 0: 0.52
Weight for class 1: 14.69

In the following tables are reported the results for the worst performing model (combination **ii**, h=2, model=MLP) and the best performing model (combination **i**, h=4, model=RNN).

Method	**h**	**Model**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Time (s)**
Reference	**2**	**MLP**	0.960 (0.005)	0.959 (0.021)	0.263 (0.107)	0.399 (0.147)	72.38 (3.93)
Over sampling			0.973 (0.005)	0.912 (0.046)	0.555 (0.149)	0.673 (0.106)	355.50 (120.18)
Under sampling			0.953 (0.003)	0.625 (0.093)	0.422 (0.307)	0.420 (0.215)	67.08 (0.99)
Classes weights			0.971 (0.003)	0.854 (0.083)	0.5710 (0.134)	0.667 (0.073)	215 (3.82)

Table 5.4: Results for the worst performing model using as input the active of each generator and the active and reactive power of the external grid's transformer. The value between brackets is the standard deviation.

It is possible to see that the score increases in all the cases, with the highest improvement with the over sampling method. While the lowest improvement is obtained with the under sampling techniques. A possible explanation is that during this operation, it may remove useful data that can be essential for the models.

Method	h	Model	Accuracy	Precision	Recall	F1-score	Time (s)
Reference	4	RNN	0.981 (0.0004)	0.798 (0.014)	0.858 (0.013)	0.826 (0.001)	120.58 (1.12)
Over sampling			0.980 (0.0006)	0.790 (0.167)	0.857 (0.014)	0.822 (0.003)	619.64 (75.95)
Under sampling			0.976 (0.004)	0.727 (0.054)	0.890 (0.037)	0.798 (0.021)	138.11 (4.42)
Classes weights			0.979 (0.002)	0.761 (0.032)	0.880 (0.012)	0.815 (0.012)	467.14 (9.09)

Table 5.5: Results for the best performing model using as input the buses' voltage magnitudes. The value between brackets is the standard deviation.

It is possible to see that the scores did not increase this time. One possible explanation is that the score is already high, so these techniques did not add useful information to the model.

It is worth noting that the training times, in all the combinations, are highly dependent on the Google Colab's virtual machine.

5.2 Control results

In this section, the results obtained when solving the aim **III)** are reported.

The agent was trained for 4 episodes, over 50k times steps.

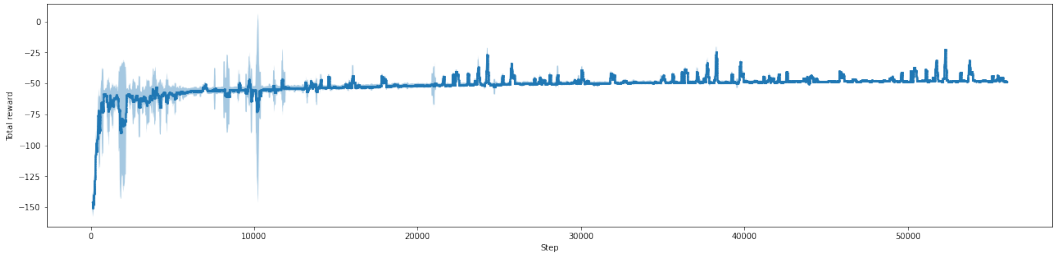
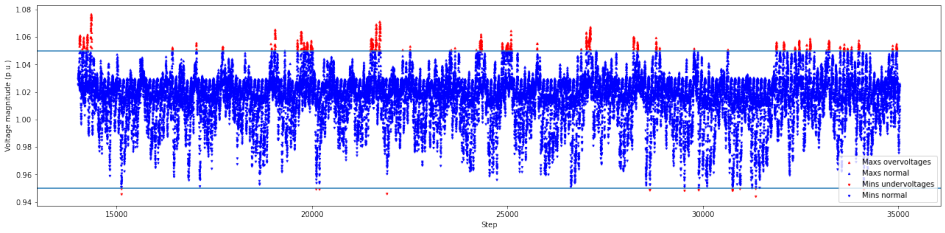


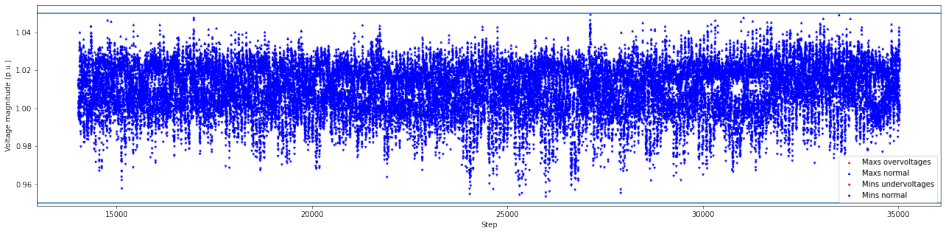
Figure 5.1: Agent's rewards over time. The reward is averaged over a rolling window of 96 time steps. The shadow region is ±2 standard deviations calculated over 5 runs.

From figure 5.1, it is possible to see that the agent is learning, since the total reward is increasing over time.

For all the 5 runs, after training, the agent was evaluated on the test set. Here reported are the results from one of the best run:



(a)



(b)

Figure 5.2: The buses' voltage situation before (**a**) and after (**b**) the agent's actions.

Figure 5.2 shows that the agent was able to solve all the over voltage problems without introducing under voltages problems, that were solved as well.

Total energy 169187 MW, total curtailed energy: 0.91 MW in 21024 time steps (7 months and 13 days), ratio: $5.3 \cdot 10^{-6}$ %

Total reactive power controlled: 287085 MVAR

---

the percentage of curtailed energy is well below the tolerable values of 4%. Indeed, curtailment levels have generally been 4% or less of wind energy generation in regions where curtailment has occurred [51].
While the reactive power corresponds to an average of 0.22 MVAR for device.

Some statistic from figure 5.2 are reported here:

Maximum voltage observed: 1.0494 p.u.
Minimum voltage observed: 0.9533 p.u.

So the agent understood how to control the network minimising the control needed, since the values are really close to the boundaries (1.05 p.u. and 0.95 p.u.) but inside the safe voltage condition.

In all the 5 runs the agent was able to solve the over voltage problems and in only one run some under voltages conditions were not solved.
It has also must be said that the agent performance is not perfect since the agent took some actions even it was not needed, it is also possible to see from figure 5.1 is stuck in a local minimum, since the reward is steady around -50.

Here there are some more statistics for the best run:

Active power curtailment:
needed (case 3: 1 - 0): 0.67 MW
Not needed:
case 1 (0 - 0): 0.24 MW
case 2 (0 - 1): 0 MW
case 4 (1 - 1): 0 MW

Reactive power usage:
needed (case 3: 1 - 0): 106341 MVAR
Not needed:
case 1 (0 - 0): 180744 MVAR
case 2 (0 - 1): 0 MVAR
case 4 (1 - 1): 0 MVAR

Cases, as defined in 4.2.2, are reported here to check in which situation the agent decides to take a particular action.
In particular, the agent is able to understand when it is needed to perform the action (case 3) and in all those cases it can solve the voltage critical situation (case 4 situations are 0). Moreover, the agent newer worsened the situation of the network (case 2 situations are 0) but the agent could perform better since a lot of activity is done when not really needed (case 1).

The time for training the agent on average is 53 minutes and the inference time is 13 minutes.

Chapter 6

Conclusions and further works

This master's thesis has investigated the analysis and the use of machine learning techniques to forecast some possible over voltage problems in a power network and to carry out some possible active control of its devices to reduce the number of critical situations.

In particular, for the forecasting part, different combinations of information were tested, depending on what kind of data a distribution system operator can have access to. These tests were performed using three values of h: 2, 4, and 16 corresponding to 30 minutes, 1 hour and 4 hours of the past network information to consider. Moreover, different artificial neural networks were trained and tested using some evaluation metric to understand their performances.
The results showed that it is possible to forecast the network's critical situation, in particular the most performing model is the recurrent neural network with 30 minutes of past buses' voltage information, obtaining an accuracy of 0.981 and a F1-score of 0.826.
Some techniques for unbalanced databased were applied as well, improving the score of the least performing model and keeping them unchanged for the best performing one.

For the controlling part, a reinforcement learning training method was used to train an agent to avoid over voltages problems. The model used was a deep deterministic policy gradient algorithm. This algorithm was able, using only 15 minutes of the network's past information, to solve the critical situations in the network. In particular, the agent was able to solve 100% of the over voltage problems and, most of the time, 100% of the under voltage problems as well; with

a cost for the distribution system operator of few MWs of active power in the entire time span considered of 7 months and 13 days.

Some future works may examine *a)* predict more time steps in the future, *b)* merge the tasks, forecasting and controlling parts, with a single model, for example under the reinforcement learning framework that could predict and control at the same time; *c)* improve the agent's performance to get even lower values of active and reactive power control; *d)* apply the same procedure to a larger network to check whether the pipeline is robust; *e)* study cases where not all the information about the devices are available due to missing data, lack of sensors and so on.

Bibliography

[1] United Nations. «World Population Prospects 2019». In: (2019). URL: https://population.un.org/wpp/Publications/Files/WPP2019_Highlights.pdf (cit. on p. 1).

[2] Hannah Ritchie and Max Roser. «CO and Greenhouse Gas Emissions». In: *Our World in Data* (2020). URL: https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions (cit. on p. 1).

[3] Statista. «Forecast of carbon dioxide emissions worldwide from 2018 to 2050». In: (2019). URL: https://www.statista.com/statistics/263980/forecast-of-global-carbon-dioxide-emissions/ (cit. on p. 1).

[4] United Nations Framework Convention on Climate Change (UNFCCC). *THE PARIS AGREEMENT*. 2015 (cit. on p. 1).

[5] Max Roser Hannah Ritchie and Pablo Rosado. «CO and Greenhouse Gas Emissions». In: *Our World in Data* (2020). URL: https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions (cit. on p. 2).

[6] Hannah Ritchie and Max Roser. «Energy». In: *Our World in Data* (2020). URL: https://ourworldindata.org/energy (cit. on p. 2).

[7] Axel Gautier, Julien Jacqmin, and Jean-Christophe Poudou. «The prosumers and the grid». In: *Journal of Regulatory Economics* (Feb. 2018). URL: https://doi.org/10.1007/s11149-018-9350-5 (cit. on p. 2).

[8] Elia. «Our infrastructure». In: (2022). URL: https://www.elia.be/en/infrastructure-and-projects/our-infrastructure (cit. on p. 6).

[9] A Muir and J Lopatto. *Final report on the August 14, 2003 blackout in the United States and Canada : causes and recommendations*. Apr. 2004 (cit. on p. 6).

[10] E. Lakervi and E. J. Holmes. *Electricity distribution network design*. English. IEE power series. Peter Peregrinus Ltd., 1995 (cit. on p. 7).

[11] Pterra. *Converging the Power Flow*. https://www.pterra.com/power-flow-analysis/converging-the-power-flow/. 2009 (cit. on p. 9).

[12] MOHAN Ned. *Power electronics: a first course*. 2012 (cit. on p. 10).

[13] Quentin Gemine, Damien Ernst, and Bertrand Cornélusse. «Active Network Management for Electrical Distribution Systems: Problem Formulation and Benchmark». In: (May 2014) (cit. on p. 10).

[14] Jianhong Wang, Wangkun Xu, Yunjie Gu, Wenbin Song, and Tim C. Green. *Multi-Agent Reinforcement Learning for Active Voltage Control on Power Distribution Networks*. 2022 (cit. on p. 11).

[15] Tom M. Mitchell. *Machine Learning*. 1997 (cit. on p. 12).

[16] Warren S. McCulloch and Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». In: (Dec. 1943). URL: https://doi.org/10.1007/BF02478259 (cit. on p. 13).

[17] Wikipedia. *Perceptron*. https://en.wikipedia.org/wiki/Perceptron. 2022 (cit. on p. 14).

[18] *Practical examples of Deep Learning with MATLAB*. URL: https://it.mathworks.com/campaigns/offers/deep-learning-with-matlab.html (cit. on p. 19).

[19] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 20).

[20] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. «Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results». In: Apr. 2020. DOI: 10.1109/ICICS49469.2020.239556 (cit. on pp. 23, 24).

[21] Gerald Tesauro. «TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play». In: *Neural Computation* (). DOI: 10.1162/neco.1994.6.2.215 (cit. on p. 25).

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: 10.48550/ARXIV.1312.5602. URL: https://arxiv.org/abs/1312.5602 (cit. on p. 25).

[23] David Silver et al. «Mastering the game of Go with deep neural networks and tree search». In: (Jan. 2016) (cit. on p. 25).

[24] Jens Kober, J. Andrew Bagnell, and Jan Peters. «Reinforcement learning in robotics: A survey». In: *The International Journal of Robotics Research* (2013). URL: https://doi.org/10.1177/0278364913495721 (cit. on p. 25).

[25] Yuxi Li. *Reinforcement Learning Applications*. 2019. DOI: 10.48550/ARXIV.1908.06973. URL: https://arxiv.org/abs/1908.06973 (cit. on p. 25).

[26] Jean-François Toubeau, Bashir Bakhshideh Zad, Martin Hupez, Zacharie De Grève, and François Vallée. «Deep Reinforcement Learning-Based Voltage Control to Deal with Model Uncertainties in Distribution Networks». In: (2020). URL: https://www.mdpi.com/1996-1073/13/15/3928 (cit. on p. 28).

[27] Song Qiang and Yang Pu. «Short-term power load forecasting based on support vector machine and particle swarm optimization». In: *Journal of Algorithms & Computational Technology* (). URL: https://doi.org/10.1177/1748301818797061 (cit. on p. 30).

[28] N. Amjady and F. Keynia. «Short-term load forecasting of power systems by combination of wavelet transform and neuro-evolutionary algorithm». In: (2009). URL: https://www.sciencedirect.com/science/article/pii/S0360544208002454 (cit. on p. 30).

[29] James W. Taylor. «Short-Term Load Forecasting With Exponentially Weighted Methods». In: *IEEE Transactions on Power Systems* 27.1 (2012), pp. 458–464. DOI: 10.1109/TPWRS.2011.2161780 (cit. on p. 30).

[30] Swaroop Ramaswamy and Hussein Al Abdulqader. «Load forecasting for power system planning and operation using artificial neural network at al batinah region Oman». In: (Aug. 2012) (cit. on p. 30).

[31] Santanu Kumar Dash, Michele Roccotelli, Rasmi Ranjan Khansama, Maria Pia Fanti, and Agostino Marcello Mangini. «Long Term Household Electricity Demand Forecasting Based on RNN-GBRT Model and a Novel Energy Theft Detection Method». In: (2021). URL: https://www.mdpi.com/2076-3417/11/18/8612 (cit. on p. 30).

[32] Hemachandra Reddy Badari Narayana Manjunatha. «WIND ENERGY FORECASTING USING RADIAL BASIS FUNCTION NEURAL NETWORKS». In: (2015). URL: https://ijret.org/volumes/2015v04/i12/IJRET20150412054.pdf (cit. on p. 30).

[33] WangJidong, RanRan, SongZhilin, and SunJiawen. «Short-Term Photovoltaic Power Generation Forecasting Based on Environmental Factors and GA-SVM». In: (Jan. 2017) (cit. on p. 30).

[34] Dhananjay Kumar, H. D. Mathur, S. Bhanot, and Ramesh C. Bansal. «Forecasting of solar and wind power using LSTM RNN for load frequency control in isolated microgrid». In: (2021). URL: https://doi.org/10.1080/02286203.2020.1767840 (cit. on p. 30).

[35] Florian Schaefer, Jan-Hendrik Menke, and Martin Braun. *Evaluating Machine Learning Models for the Fast Identification of Contingency Cases*. 2020. DOI: 10.48550/ARXIV.2008.09384. URL: https://arxiv.org/abs/2008.09384 (cit. on p. 30).

[36] Amr M. Ibrahim and Noha H. El-Amary. «Particle Swarm Optimization trained recurrent neural network for voltage instability prediction». In: (2018). URL: https://www.sciencedirect.com/science/article/pii/S231471721730020X (cit. on p. 30).

[37] Nikita V. Tomin, Victor G. Kurbatsky, Denis N. Sidorov, and Alexey V. Zhukov. «Machine Learning Techniques for Power System Security Assessment». In: (2016). URL: https://www.sciencedirect.com/science/article/pii/S2405896316324028 (cit. on p. 30).

[38] Frédéric Olivier, Petros Aristidou, Damien Ernst, and Thierry Van Cutsem. «Active Management of Low-Voltage Networks for Mitigating Overvoltages Due to Photovoltaic Units». In: (2016). DOI: 10.1109/TSG.2015.2410171 (cit. on p. 30).

[39] Shibani Ghosh, Saifur Rahman, and Manisa Pipattanasomporn. «Distribution Voltage Regulation Through Active Power Curtailment With PV Inverters and Solar Generation Forecasts». In: 8 (2017). DOI: 10.1109/TSTE.2016.2577559 (cit. on p. 30).

[40] Robin Henry and Damien Ernst. «Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems». In: (Sept. 2021). URL: http://dx.doi.org/10.1016/j.egyai.2021.100092 (cit. on p. 30).

[41] Changfu Li, Chenrui Jin, and Ratnesh Sharma. *Coordination of PV Smart Inverters Using Deep Reinforcement Learning for Grid Voltage Regulation*. 2019. URL: https://arxiv.org/abs/1910.05907 (cit. on p. 30).

[42] Di Cao, Junbo Zhao, Weihao Hu, Fei Ding, Qi Huang, Zhe Chen, and Frede Blaabjerg. «Data-Driven Multi-agent Deep Reinforcement Learning for Distribution System Decentralized Voltage Control with High Penetration of PVs». In: (Apr. 2021). URL: https://www.osti.gov/biblio/1781623 (cit. on p. 30).

[43] Di Cao, Weihao Hu, Junbo Zhao, Qi Huang, Z. Chen, and F. Blaabjerg. «A Multi-Agent Deep Reinforcement Learning Based Voltage Regulation Using Coordinated PV Inverters». In: (May 2020). DOI: 10.1109/TPWRS.2020.3000652 (cit. on p. 30).

[44] Brida V. Mbuwir, Davy Geysen, Fred Spiessens, and Geert Deconinck. «Reinforcement learning for control of flexibility providers in a residential microgrid». In: *IET Smart Grid* (2020). URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-stg.2019.0196 (cit. on p. 30).

[45] Leon Thurner, Alexander Scheidler, Florian Schafer, Jan-Hendrik Menke, Julian Dollichon, Friederike Meier, Steffen Meinecke, and Martin Braun. «Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems». In: (Nov. 2018). URL: http://dx.doi.org/10.1109/TPWRS.2018.2829021 (cit. on p. 33).

[46] Salish Maharjan, Ashwin M. Khambadkone, and Jimmy C.-H. Peng. «Enhanced Z-bus method for analytical computation of voltage sensitivities in distribution networks». In: 14.16 (2020), pp. 3187–3197. DOI: https://doi.org/10.1049/iet-gtd.2019.1602. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2019.1602 (cit. on p. 33).

[47] Steffen Meinecke, Džanan Sarajlić, Simon Ruben Drauz, Annika Klettke, Lars-Peter Lauven, Christian Rehtanz, Albert Moser, and Martin Braun. «SimBench—A Benchmark Dataset of Electric Power Systems to Compare Innovative Solutions Based on Power Flow Analysis». In: 12 (2020). URL: https://www.mdpi.com/1996-1073/13/12/3290 (cit. on p. 34).

[48] Christian Spalthoff, Džanan Sarajlić, Chris Kittl, Simon Drauz, Tanja Kneiske, Christian Rehtanz, and Martin Braun. «SimBench: Open source time series of power load, storage and generation for the simulation of electrical distribution grids». In: (May 2019) (cit. on p. 34).

[49] Leon Thurner. *Structural Optimizations in Strategic Medium Voltage Power System Planning*. 2018 (cit. on p. 37).

[50] *Google Colab*. URL: https://colab.research.google.com (cit. on p. 50).

[51] Lori Bird, Jaquelin Cochran, and Xi Wang. «Wind and Solar Energy Curtailment: Experience and Practices in the United States». In: (Mar. 2014). URL: https://www.osti.gov/biblio/1126842 (cit. on p. 55).