POLITECNICO
MILANO 1863

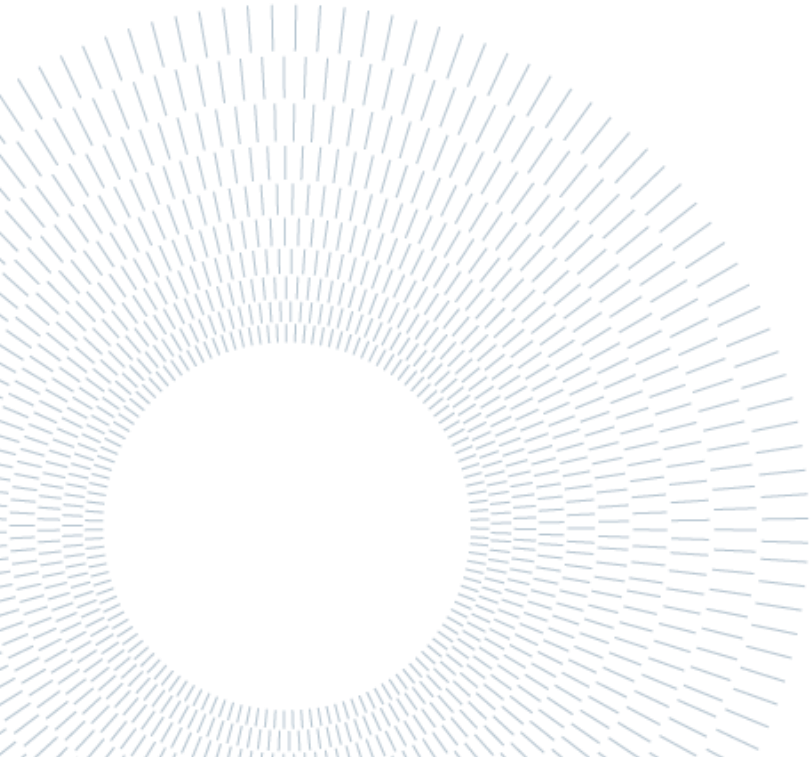
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Voltage regulation in distribution networks in the presence of distributed generation and electric vehicles: LVR and E-OLTC with machine learning approach

TESI DI LAUREA MAGISTRALE IN
ELECTRICAL ENGINEERING
INGEGNERIA ELETTRICA

Author: **Durim Musiqi**

Student ID: 10711652
Advisor: Alessandro Bosisio
Academic Year: 2020-2021



Abstract

This thesis makes use of machine learning (ML) algorithms to solve problems in electric power systems.
The specific issue tackled here is voltage regulation in MV networks with high penetration of distributed generation (DG) and electric vehicles, using Electronic On Load Tap Changers (E-OLTC) and Line Voltage Regulators (LVR).
This thesis contains two studycases, hence, two MV feeders are built from scratch using PowerFactory software. One of them is fictive while the other one is based on a real life feeder, the former is sub-urban and the latter is rural.
The process starts with creating hourly characteristics of loads and photovoltaic systems (PVs) of the feeder. That is done by collecting and processing millions of datapoints. For the PVs irradiation the data comes from PVGIS tools, while for the load characteristics, the data is based on real yearly curves which are then adapted accordingly with the peak loads of the feeder.
After modelling the feeders and preparing the characteristics, two Quasi-Dynamic (Q-D) simulations are run for a period of 2-years in hourly steps (17520 load flows). One Q-D analysis is done without employing automatic tap changers, while the other is done using automatic tap changers. Then comparisons are made.
The results of the Q-D simulations offer information on correct tap positions of transformers given various inputs, which are loads and PV productions.
The last step is to use these results and build a Multi-output Deep Neural Network (DNN).
The DNN, using forward propagation and backward propagation, will be able to learn by fitting the inputs to the relevant outputs. This way, in the future the DNN will be able to predict the correct tap positions of transformers in the feeder (outputs), only knowing the loads and PV productions (inputs).
After the machine learning algorithm is trained, it will then be tested with a set of inputs it has not seen before.
Although this thesis does not address other voltage regulation methods, the DNN model is flexible in a manner that allows to potentially include them in the future with ease. Various software tools are used in this study, some of them are PowerFactory, Python (lib. Pandas, Tensorflow/Keras, Scikitlearn etc.), Matlab etc.

Key-words: Medium Voltage, Voltage regulation, E-OLTC, LVR, Machine Learning, Deep Neural Network, Quasi-Dynamic analysis.

Abstract in italiano

Questa tesi utilizza algoritmi di machine learning (ML) per risolvere problemi nei sistemi di alimentazione elettrica.
La questione specifica qui affrontata è la regolazione della tensione nelle reti MT ad alta penetrazione della generazione distribuita (DG) e dei veicoli elettrici, mediante l'utilizzo di commutatori elettronici sotto carico (E-OLTC) e regolatori di tensione di linea (LVR). Questa tesi contiene due casi di studio, quindi, due alimentatori MV sono costruiti da zero utilizzando il software PowerFactory. Uno di questi è fittizio mentre l'altro si basa su un alimentatore di vita reale. Il processo inizia con la creazione delle caratteristiche orarie dei carichi e degli impianti fotovoltaici (PV) dell'alimentatore. Ciò viene fatto raccogliendo ed elaborando milioni di punti dati. Per l'irraggiamento PV i dati provengono da strumenti PVGIS, mentre per le caratteristiche di carico i dati si basano su curve annuali reali che vengono poi adattate di conseguenza. Dopo aver modellato gli alimentatori e preparato le caratteristiche, vengono eseguite due simulazioni Quasi-Dynamic (Q-D) per un periodo di 2 anni in fasi orarie (17520 flussi di carico). Un'analisi Q-D viene eseguita senza l'utilizzo di commutatori di presa automatici, mentre l'altra viene eseguita utilizzando commutatori di presa automatici. I risultati delle simulazioni Q-D offrono informazioni sulle corrette posizioni di presa dei trasformatori dati i vari ingressi, che sono carichi e produzioni fotovoltaiche. L'ultimo passaggio consiste nell'utilizzare questi risultati e creare una rete neurale profonda multi-output (DNN). Il DNN, utilizzando la propagazione in avanti e la propagazione all'indietro, sarà in grado di apprendere adattando gli input agli output pertinenti. In questo modo, in futuro il DNN sarà in grado di prevedere le corrette posizioni di presa dei trasformatori nell'alimentatore (uscite), conoscendo solo i carichi e le produzioni fotovoltaiche (ingressi). Dopo che l'algoritmo di apprendimento automatico è stato addestrato, verrà quindi testato con una serie di input che non ha mai visto prima. Sebbene questa tesi non affronti altri metodi di regolazione della tensione, il modello DNN è flessibile in un modo che consente di includerli potenzialmente in futuro con facilità. In questo studio vengono utilizzati vari strumenti software, alcuni dei quali sono PowerFactory, Python (lib. Pandas, Tensorflow/Keras, Scikitlearn ecc.), Matlab ecc.

Parole chiave: Media Tensione, Regolazione della tensione, E-OLTC, LVR, Machine Learning, Deep Neural Network, Analisi Quasi-dinamica.

Contents

Abstract ..... **i**

Abstract in italiano ..... **iii**

Contents ..... **v**

Motivation **1**

1 Introduction **3**

1.1. Operation of classic networks 3

1.2. Operation of modern networks 4

1.3. Introduction to voltage regulation methods 5

2 Line Voltage Regulators vs Electronic – On Load Tap Changers **7**

2.1. Line Voltage Regulator (LVR) 7

2.2. Electronic On Load Tap Changer (E-OLTC) 15

2.3. Comparing LVR and E-OLTC 17

3 Machine learning approach on voltage regulation **19**

3.1. Introduction to Machine Learning 19

3.1.1. Linear regression and gradient descent 20

3.1.2. Polynomial regression 24

3.1.3. Bias/Variance tradeoff 25

3.2. Deep Neural Networks and activation functions – Utilization of machine learning tools in distribution networks 27

3.2.1. The perceptron 27

3.2.2. The basic architecture of Neural Networks 32

3.2.3. Activation functions 33

3.2.4. Choosing the right ML algorithm 38

3.2.5. Building a Multi-Output Deep Neural Network for voltage regulation with E-OLTCs 40

4 CASE STUDY I – 20 [kV] feeder Sub-Urb, Sicily. 45

4.1. Introduction 45

4.1.1. Tap changing settings in PowerFactory 48

4.1.2. Running a Quasi-Dynamic Simulation 50

4.2. Modelling feeder Sub-Urb in PowerFactory 52

4.3. Generating, extracting, and preparing the data 59

4.3.1. Creating time characteristics for PVs 59

4.3.2. Creating time characteristics for loads 62

4.4. Running the Quasi-Dynamic simulation and exporting the results of Tap Positions 65

4.5. Cleaning, re-organizing and filling the missing data points 74

4.6. Dividing and exporting the dataset features and labels 75

4.7. Coding the Deep Neural Network in Python 76

4.8. Additional results and visualizations 85

5 CASE STUDY II – 10 [kV] feeder Rugova, Kosovo 87

5.1. Introduction 87

5.2. Modelling Rugova feeder in PowerFactory 87

5.3. Generating, extracting, and preparing the data 94

5.3.1. Creating time characteristics for PVs 94

5.3.2. Creating time characteristics for loads 96

5.4. Running the Quasi-Dynamic simulation and exporting the results of Tap Positions 98

5.4.1. Combining LVRs and E-OLTC 106

5.5. Cleaning, re-organizing, and filling the missing data points 108

5.6. Dividing and exporting the dataset features and labels 109

5.7. Coding the Deep Neural Networks in Python 111

5.8. Additional results and visualizations 118

6 Conclusion and future developments 121

7 Bibliography 125

List of Figures 127

List of Tables **131**

Acknowledgments **132**

Motivation

The electric power system has been developing continuously parallel to the development of various other technologies, especially communications and materials technologies.

The goal is to achieve a highly reliable, automated, 'self-healing' power system. A power system which will require almost no inputs from an engineer's hand, rather, a system that knows exactly how to react on any occasion.

EU Directive 2018/2001 has set a target that requires 32% of total energy demands to be covered by RES by 2030 [1].
This is becoming more and more important every day now that the system needs to support and accommodate renewable energy sources [2].
RES are key to a sustainable system, however, they are not easy to integrate in the grid.
For example, although everyone having solar panels on their roofs may seem like a great idea at first glance, it also has a serious downside, which is the unpredictability it brings to the system's parameters if the sun doesn't shine, if there are clouds, or if the irradiation prediction is wrong for any reason. A wind farm may decrease the overall CO_2 emissions, but if the predictions on wind velocity turn out to be incorrect, it may cause severe trouble to whichever entity that is trying to balance the system. Maybe it will even be necessary to import hundreds of megawatts of power, potentially spiking the energy prices that consumers pay.

So, on one hand more work is needed to move towards a more sustainable system which is good for the planet and the economy. On the other hand, there will be many more problems trying to maintain the system's parameters within allowed limits, due to the hardly predictable nature of these resources [3].
One of the parameters that proves problematic is the voltage. The presence of many DGs may cause voltage fluctuations, over-voltages and even flicker [4].
Ultra-modern networks may be H-shaped [5] or 2-step ladder shaped [6]. These networks are designed to have very low voltage drops, which may be problematic when we deal with high DG.

This brings us to the motivation for this thesis. Therefore, this study will try to come up with some sort of a machine learning solution to voltage regulation.

1 Introduction

This chapter offers a quick introduction to power systems relevant to this study and briefly discusses basic voltage regulation methods.

The goal is to understand the general evolution of networks as far as the power flow direction is concerned.

1.1. Operation of classic networks

The basic components of any power system are generators, transformers, lines, and loads.

The first ever electric network was developed by Edison in New York in 1882, where 59 consumers were connected through a 110 [V] DC line to a 30 [kW] generator. Just three years later William Stanley built the first commercial transformer. The power system continued developing extremely fast because long distance lines, three-phase lines, Tesla's alternative motors and other elements, were invented and put to use within that first decade.

The networks continued to grow and improve constantly for the next 100 years. The power flows had a quite intuitive direction, the energy would flow from the generators downstream to the consumers, relays and protection were easy to design, voltage drops were easy to calculate and simple to fix from the engineering point of view.

Figure 1 shows a portion of a random network that operates in the classical way. The direction of power flow is intuitive. Power flows downstream from the generators down to the loads. The voltages drop throughout the lines in the load's direction.

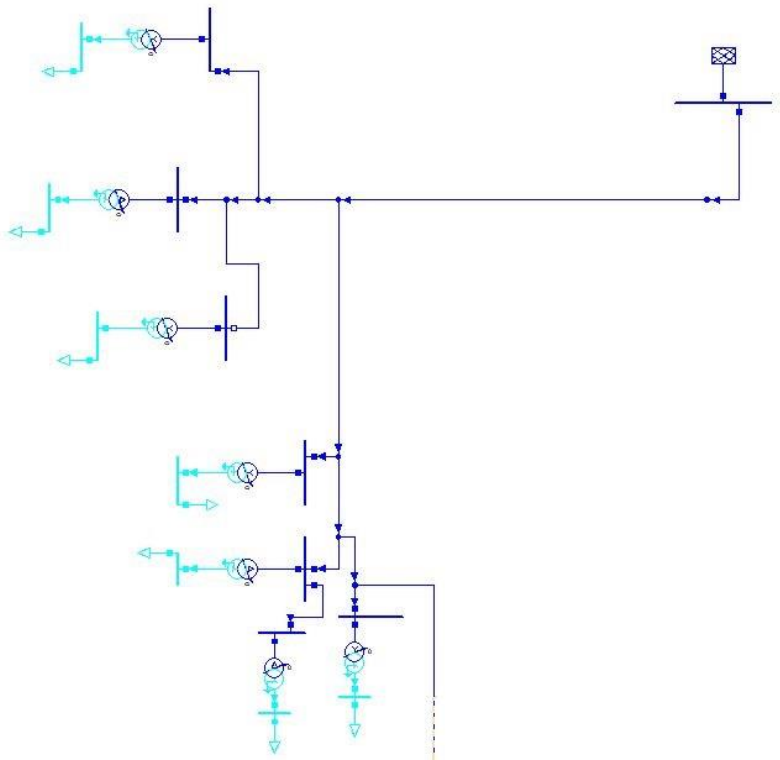


Figure 1. Classic network: Downstream power flow.

1.2. Operation of modern networks

The power system started to change in the 2000s. That is when global warming attracted more attention and a lot of financial resources were allocated to researching more sustainable ways of power generation. Intense research and worldwide interest on solar panels made it possible for their price to drop from 105 [$/Watt] in 1975 to just 0.2 [$/Watt] in 2021 [7].

This made it possible for PV systems to be used commercially, which in turn would mean that anyone could install a few panels on their rooftop and inject energy into the grid. Many incentive schemes were offered by governments, attracting businesses to invest in large scale PV systems investments.

When the DGs power output surpasses the feeder's load, the power will start flowing upwards towards the substation that supplies that feeder.

Such a situation is shown in the next figure, taken from a real case of a 3 [MW] PV system near Ulpiana, rep. of Kosovo.

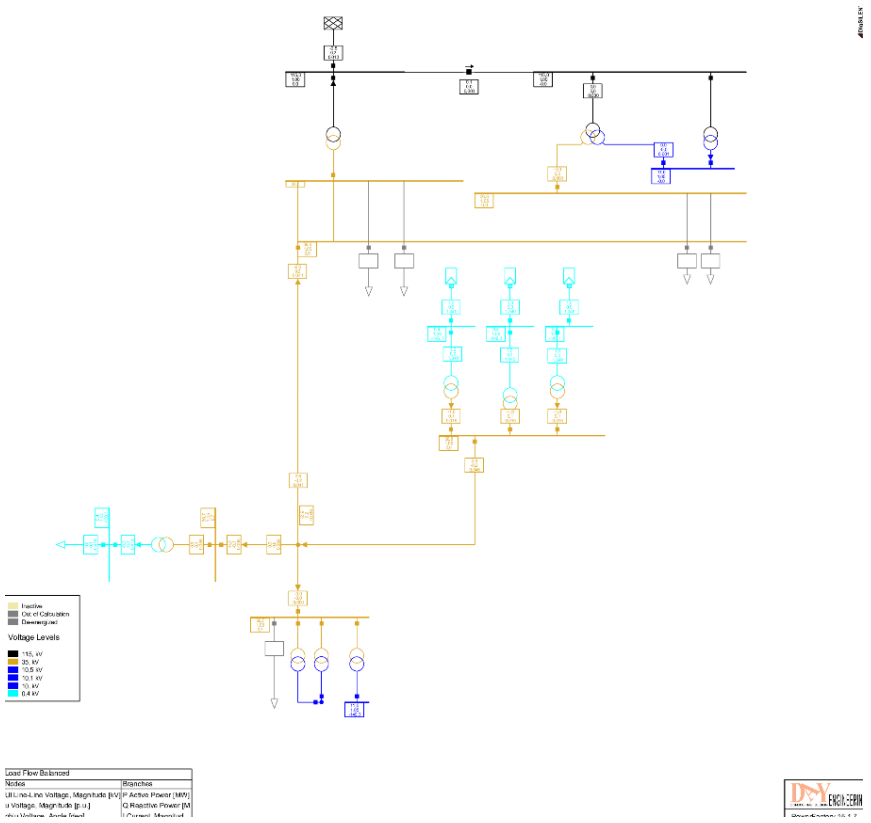


Figure 2. Modern network operation: Upstream power flows

If the power injected by the PVs happens to also be larger than the sum of all loads connected to the LV side of the SS, then the power will flow upstream through the transformer and it will be injected to the HV side of the SS. As indicated by the arrows in the PowerFactory model in Figure 2, the power flows upstream towards the substation that has supplied that feeder.

These complications are some of the traits of modern networks. In these networks, voltage regulation becomes a troublesome process which will be dealt with here, using machine learning.

1.3. Introduction to voltage regulation methods

Depending on the country or region, there are different ranges of allowed voltage values. In Europe it is usually ±10% Un [8]. In feeders where there is injection from renewable sources, voltage varies, especially if the reactive power output changes unpredictably. If the voltage increases, there will be problems.

In both classical and modern networks there are several ways of voltage regulation,

the main ones are the following:

a) **Transformer Tap Changers** – every transformer in the network has several levels of voltage it can operate on. This is made possible by either mechanically or digitally changing the effective number of turns in one of the windings in the transformer [9]. For example, a transformer with a ratio of 10000:400 which is equal to 25, can change its ratio to 25.25. This is done in increments of small percentages such as 1.25%, 2.5% or similar. Depending on the voltage level, transformers may have anywhere from ±2x1.25% to ±11x1.25% or similar.

   So, transformers have multiple levels of voltage they can switch to. This helps the overall power system to operate smoothly, as changing the voltage level in various busbars across the network helps balance the system parameters. In the more recent times, taps are mainly changed from control rooms while on load (E-OLTC), while some older and usually smaller transformers must be off load and their tap positions must be changed mechanically, which is their biggest disadvantage.

b) **Reactive power flow settings** - The voltage in the system depends mainly on the reactive power, therefore, manipulating the reactive power will change the voltage as well. This is mainly done using capacitors installed in shunt to the line or using static VAR regulators. There are several types of configurations to realize such systems.

c) **Line voltage regulators** – Also referred to as simply 'voltage regulators' are transformers installed somewhere along the feeder. They make up for the voltage drop by using tap changers. The difference from usual transformers is that LVRs have a smaller ratio of transformation.

d) **Others**

This report is only concerned with **a)** and **c)**.

2 Line Voltage Regulators vs Electronic – On Load Tap Changers

This chapter will cover Line Voltage regulators (LVRs) and Electronic – On Load Tap Changers (E-OLTCs). First, they will be discussed independently, then compared to each other. This chapter will raise and support hypotheses regarding the most suitable feeders for each type of voltage regulator to operate on.

2.1. Line Voltage Regulator (LVR)

Line voltage regulators are basically transformers installed somewhere along the lines.

Voltage drops along the lines are caused by loads and line losses. If there is distributed generation as well, the voltage will vary also depending on PVs reactive power output.

In case the infeed is high, the voltage will steeply increase and the local PV or wind plants will have to reduce their power output or even completely stop injecting to avoid faults. This is where LVR comes in handy. LVR will be able to automatically adjust the voltage level to a specific range specified by the distribution company.

That translates to more distributed generation allowed in the feeder/area, which now will not cause violations on voltage parameters. In other words, LVR allows the infeed to increase by avoiding overvoltages.

Many companies have been paying extra attention to LVRs in the past few years since it is a feasible way to accommodate the ever-increasing distributed generation. LVRs are cheaper and easier to install than having the entire feeder upgraded with higher section lines or similar interventions

LVRs are appropriate in feeders that are longitudinal instead of feeders that are short and branchy. That means that LVRs are not the optimal choice in networks with high penetration of DGs, as overvoltages mainly occur on busbars where PVs inject directly. However, as mentioned before, LVRs are useful in long feeders. One such feeder is the following:

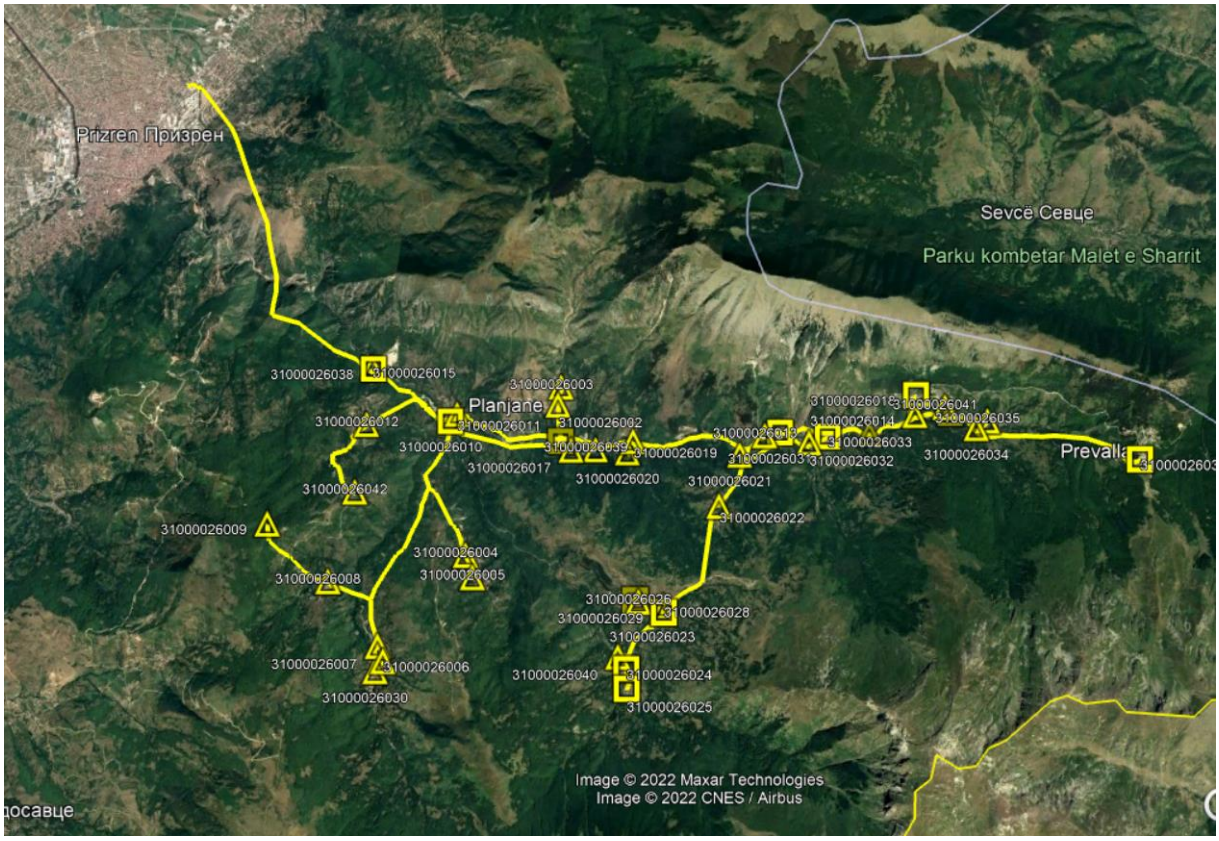


Figure 3. Prizren-Zhupa feeder in the Sharr Mountains.

The feeder shown in Figure 3 is a 10 [kV] feeder that is over 35 [km] long in total. Its main axis is around 22 [km] long and it also has some shorter side branches.

Due to line lengths at a voltage level this low, there are always under-voltages at the busbars located the end of the feeder.

According to KEDS measurements at the last 10/0.4 [kV/kV] transformer of this feeder, the voltage drops at around 0.82 [p.u]. That is around 0.8 [kV] under the allowed voltage drop range in normal conditions, and around 0.3 [kV] under the allowed voltage drop range in extra-ordinary conditions.

This is exactly the type of line and feeder where a LVR can prove to be an excellent solution.

The LVR can be installed somewhere around the middle of the main branch, with its tap settings at one of the levels that would increase the voltage for around 5% Un. This would make it possible for the busbars at the end of the line to operate at a voltage level within the allowed range set by the Distribution Code.

LVR would not be as useful in short feeders with many branches as it would only solve the voltage of the problem for a branch, which in this case would be a very

small portion of the feeder. To tests this hypothesis, LVRs were installed in one of the feeders of this study and their impact is analyzed at times of large PV production, such as noontime of summer days.

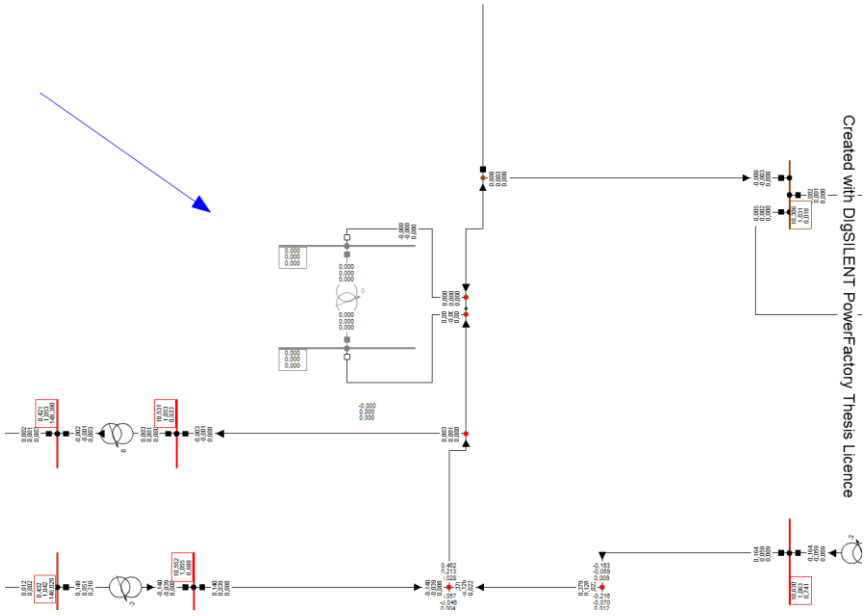


Figure 4. LVR installed along the feeder - inactive.

Figure 4 offers a view of how LVRs is installed somewhere along the main branch of the feeder. In this case, the LVR is not active (LVR is by-passed), and since it is a critical time of high PV injection, busbars experience overvoltages (colored in red). The impact of the LVR being activated is showcased in Figure 5.

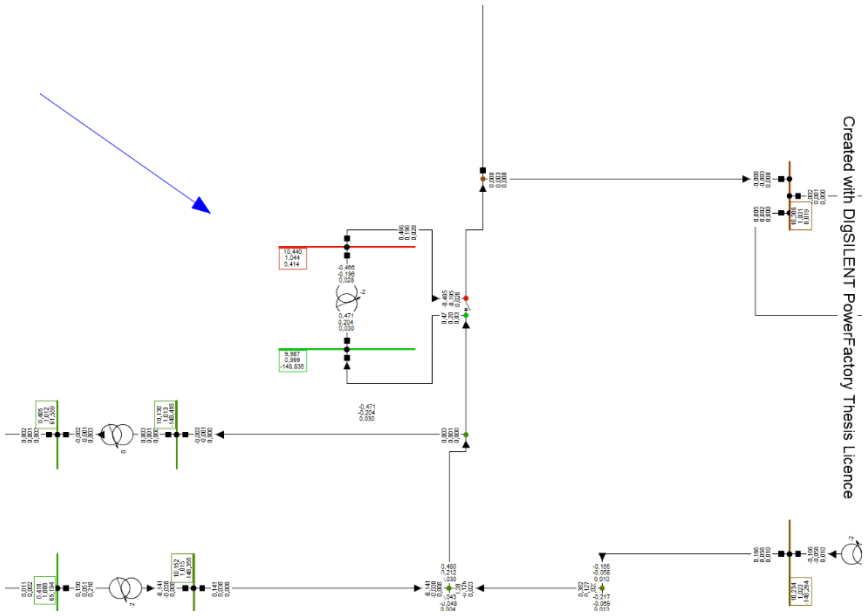


Figure 5. LVR installed along the feeder – activated.

LVR is able to decrease the voltage on one side of the feeder. However, if the feeder has many branches at the ends of which there is DG, LVR would not be able to help as much without a large number of them being installed at various points in the feeder.

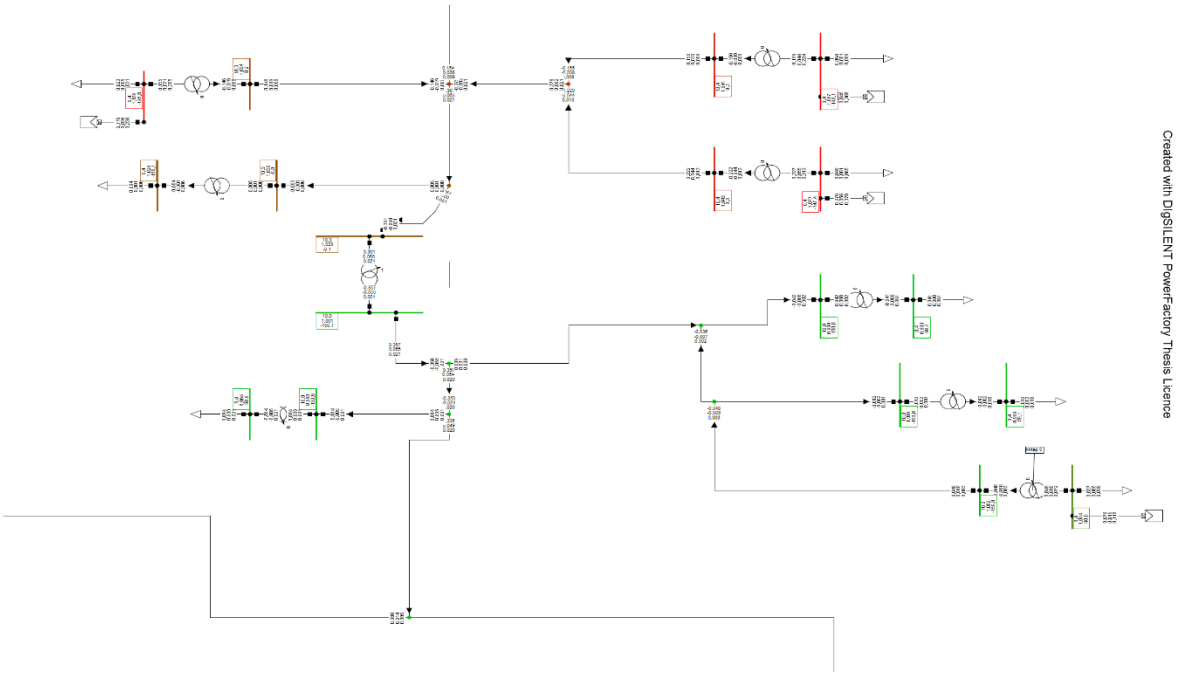


Figure 6. Impact of a single LVR installed in the feeder.

It suffices to zoom out on the feeder to have a better overview of LVR's impact, as seen in Figure 6. The upper part of the network (before the LVR) is experiencing overvoltages, while the lower part (after the LVR) has better voltage values.

In Figure 5 and Figure 6, LVRs are not operating alone, also the E-OLTCs of transformers where DGs inject are equipped with automatic tap changers.

So, LVRs are only able to decrease the voltages for a portion of the network, and when the network is large, a several LVRs will be needed. For this particular feeder, as seen in Figure 7, four LVRs are needed to secure plausible voltages for all buses in it. This may not be easy to do. LVRs are also suitable in cases when there are undervoltages. In any case, LVRs are not optimal for feeders with many long branches.

Even after installing four LVRs at strategic points in the feeder, there are still overvoltages caused in proximity of DG (busbars colored red). The difference is that in Figure 7, E-OLTCs of transformers where DGs are injecting are not operative. That is done to better understand the impact of LVRs individually.

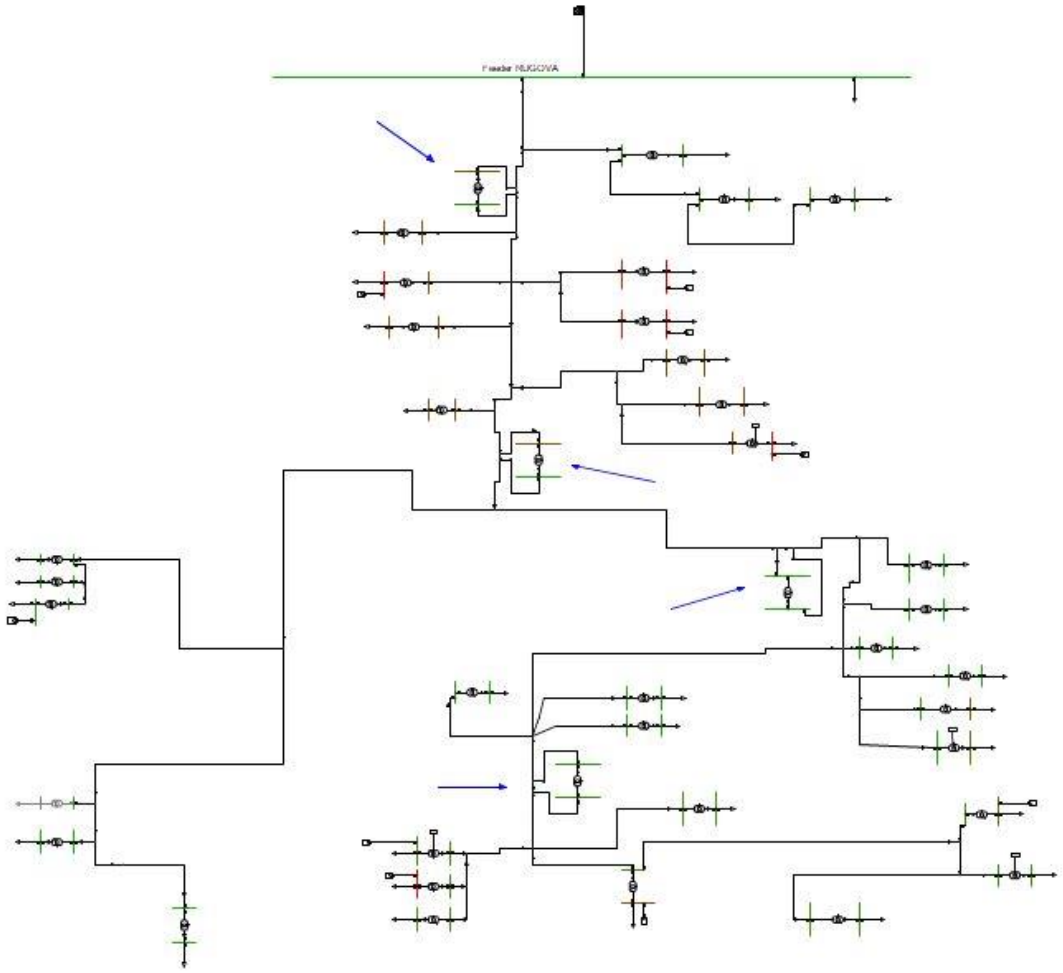


Figure 7. Feeder with four LVRs.

This means, LVRs are not suitable as a sole solution of these types of feeders. Therefore, in this study, they will only be considered in combination with E-OLTCs and not alone.

An informative graph of how LVRs are good with undervoltages is if the feeder's voltage profile.

As Figure 8 shows for long feeders such as is Rugova, if the loads are large and there is no distributed generation to somehow support undervoltages, the voltage drops by the end of the line will be very large. To counter this, several LVRs along the lines will aument the voltage up to plausible values.

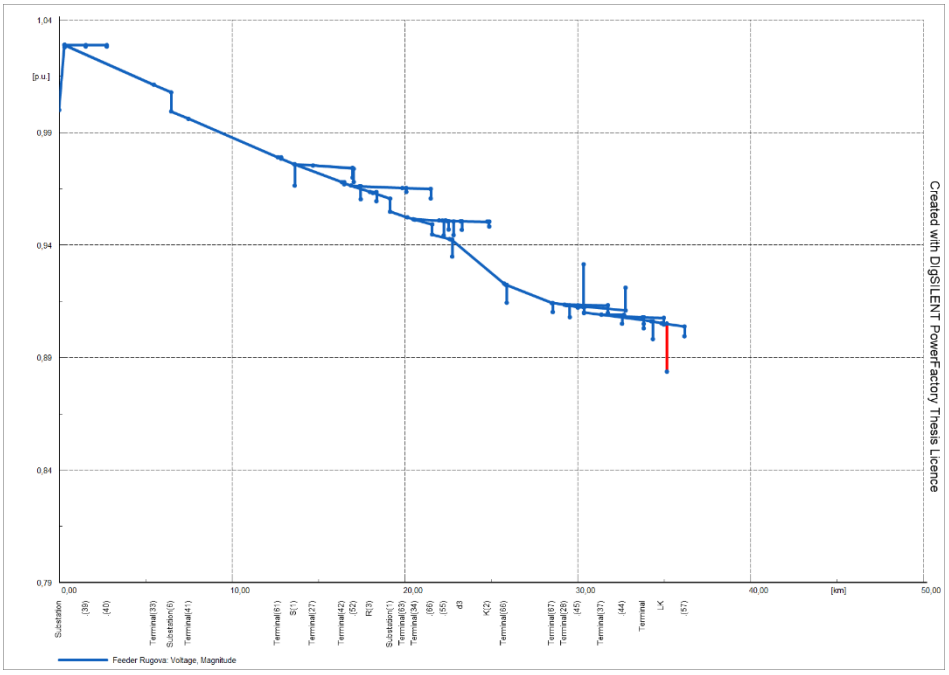


Figure 8. Feeder Rugova voltage profile, with large loads and small DGs [p.u].

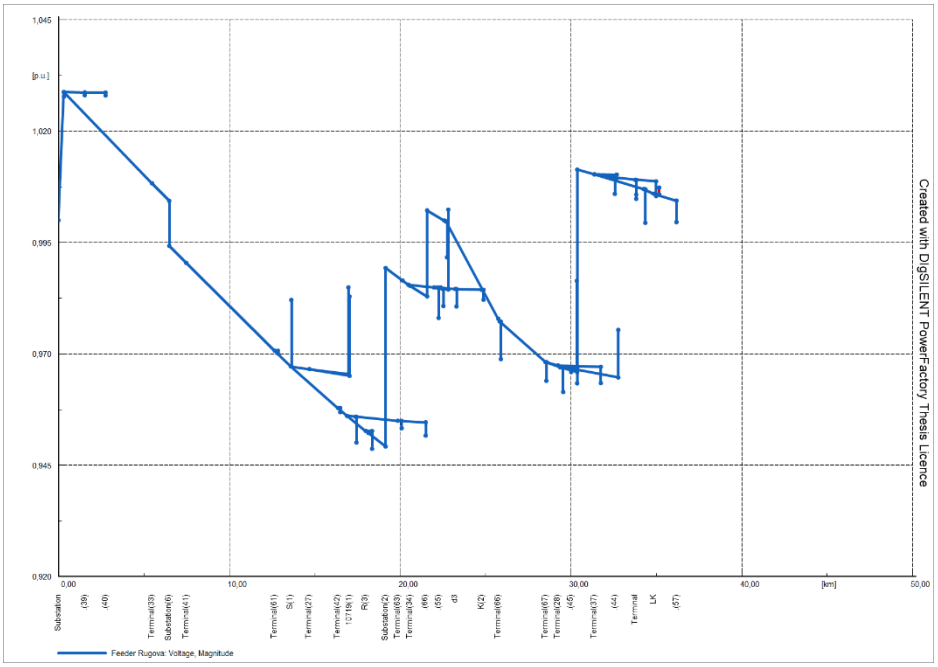


Figure 9. Feeder Rugova voltage profile when loads are large and small DGs – with LVRs installed [p.u]

Figure 9 shows quite an unusual voltage profile. However, it also proves that installing several LVRs along the line may help with undervoltages. In this case, there are 4 LVRs installed, whose impact can be identified with four instances of voltage jumps in the feeder's voltage profile.

As far as LVRs go, to illustrate better, model *DEABB 5153* line voltage regulator for medium voltage networks from ABB will be briefly discussed next. This LVR uses the so called 'booster/feeder' technology and uses mechanical switches. According to the manufacturer's catalog, energy losses are minimal [10].

Another important feature of LVRs is that they can be programmed and operate automatically at a fixed level of voltage or at a range of voltages depending on the power, without any manual input. Another advantage is that they can be monitored and controlled from distance at all times [11].
The design of the LVR and its enclosing is simple and can even be relocated, thus increasing the overall feasibility of LVRs [12].

LVRs are usually able to regulate the voltage in the range of ±10% Un, however, it can be slightly more than that. The following figures show the basic work principle of LVRs. Another specific of recent models of LVRs is that they operate completely free of oil, which means that there is no risk of fire [12].

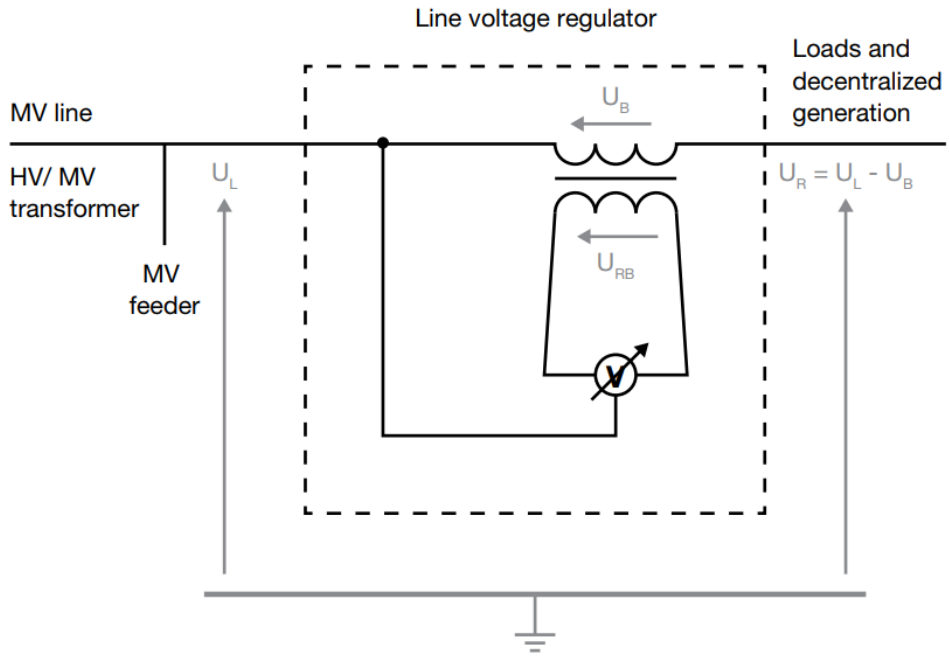


Figure 10. Line Voltage Regulator single line diagram

From Figure 10, on the left side of the LVR, there is the normal line voltage (U_L) which is unregulated. The regulator itself has two parts, the booster on the line's side and the booster on the regulator's side, with respective voltages U_B and U_{RB}. The voltage on the right side of the LVR will be the regulated voltage and it is expressed as $U_R = U_L - U_B$.
So, from the scheme we can see that the regulated voltage U_R will decrease as U_B increases.

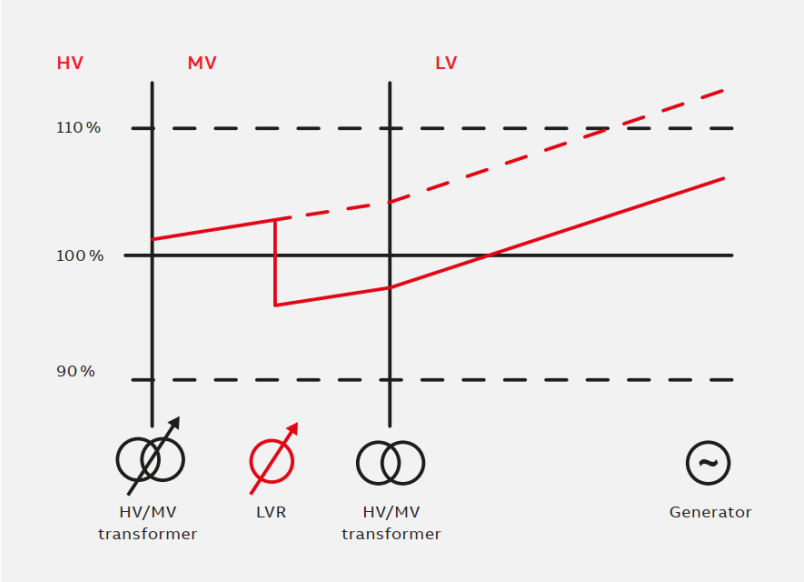


Figure 11. LVR regulation of voltage.

From Figure 11 we see the voltage tending to high values due to large DG, even surpassing the +10% Un limit. However, if the LVR is deployed, the voltage decreases within allowed limits.

Of course, besides lowering the voltage, the opposite can be done as well.

The voltage can be increased instead of being decreased. A LVR decreasing the voltage would be fit for feeders with high DG, on the other hand, a LVR increasing the voltage would be suitable for longitudinal feeders such as the one introduced earlier in this subchapter.

This LVR also supports a relatively large amount of power, up to 20 [MVA], that amount should be enough for almost any feeder in MV. According to recent catalogs from certain manufacturers, the losses are very small, less than 0.2%.

The technical characteristics of ABB's MV LVR are given on Table 1.

Rated power [MVA]	up to 20*
Frequency [Hz] / Phases	50 / 3
System voltage [kV]] *	up to voltage class 24 kV
Insulation class [kV, BIL/AC]	125/50
Number of steps	11
Total voltage regulation range	±10%
Step voltage	2% (±5 X 2%)
Number of switching operations (mechanical)	>1'000'000
Installation location	Outdoor

Installation type	Concrete substation (other solutions on request)
Grid connection	Gas-insulated MV-switchgear
Dimensions (L X W X H) [mm]	6x 2,5 X 3,3
Weight (approx.) [t]	38
Control modes	Fixed set-point load-dependent voltage set-point
Energy efficiency	Energy efficiency and losses depend on the P. P [MVA] 2.0 ; 4.0 ; 6.0 ; 8.0 ; n [%] 99,86 ; 99,88 ; 99,85 ; 99,83 ; Losses [kW] 2.9 ; 4.6 ; 8.7 ; 13.9 ;

Table 1. ABB's Catalog data for MV LVR model DEABB 5153.

2.2. Electronic On Load Tap Changer (E-OLTC)

The most vastly used method of voltage regulation in today's networks is by using electronic on load tap changers.
The operating principle of basic tap changers is shown in Figure 12.

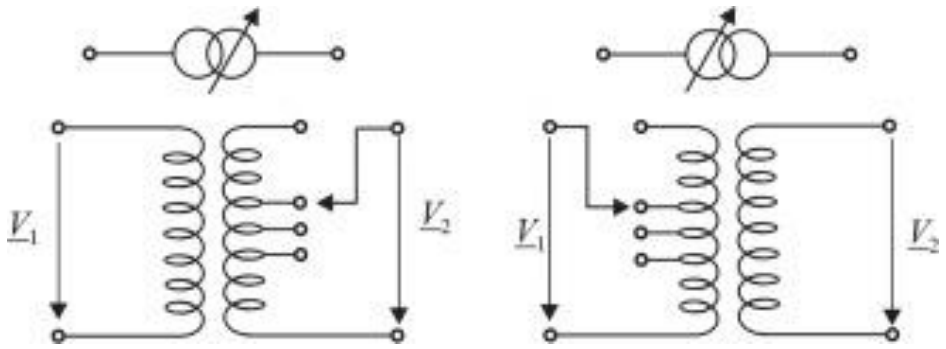


Figure 12. Drawing of Tap Changer's simplified work principle.

As mentioned in sub-section 1.3, what a tap changer does is change the effective number of turns in one of the transformer's windings. In other words, it changes the transformation ratio of a transformer.

Each tap position may add or remove a small percentage of the winding's nominal voltage. For example, an MV transformer usually has 5 tap positions, each step adding or removing 2.5% of the nominal voltage. That translates to a voltage regulating range of ±5% Un.

Almost all transformers installed at high voltages are equipped with E-OLTCs. That means, they can be controlled from the control room using SCADA systems, even

while energy is flowing through them.

In the lower levels of voltage, most of the transformers do not have the option of being controlled remotely. Instead, they must be off load and manipulated mechanically by technicians. That makes it very hard to change the settings often, rendering them practically useless tools to maintain the desired voltage levels at times of high volatility of voltage.
Mechanical tap-changers are also more expensive in terms of maintenance and operability [13].

Firstly, when tap-changers were being designed to operate on-load, only electronic switches were tried as a solution, while other components were kept unchanged. It turned out that changing the entire system and building fully electronic on-load tap-changers is a better approach than just replacing the switches [14].
So, E-OLTCs were engineered not very long ago, and it was done by taking advantage of the semi-conductor technology sophistication. The semi-conductor technology improvement made it possible to design equipment that is able to withstand charges, arcing, carbonization of contacts etc., due to the switches being solid state types.

That is how Electronic On Load Tap Changers came to be and started replacing classical off-load mechanical tap changers.
Differently from LVRs which were deemed unfit to independently regulate the voltages of feeders with high penetration of DGs, the E-OLTCs prove to be more suitable. That happens because usually PV systems are connected on the LV side, where they inject energy. That energy then goes through the LV/MV transformer and then flows in the medium voltage network. That means, if any overvoltages occur at the busbar where the PV system injects, that overvoltage can be mitigated by that transformer's tap changers before it reaches the other side to cause overvoltages in the medium network.
Having the possibility to remotely control the tap positions and regulate the voltage, combined with proper communication systems installed in these devices, made the transformers 'smart'. That means the transformer will offer other functions such as load shedding, power dispatch control of distributed generation, load measurement etc.
Next, a sophisticated type of a transformer with E-OLTC will be briefly discussed.
This particular type comes from a design, that takes into consideration issues that were overlooked in early designs, those are: **a)** the dielectric withstand analysis, **b)** steady-state voltage across the electronic switches, **c)** steady-state current through the electronic switches, **d)** analysis of the commutation process etc.

To make sure the E-OLTC is reliable, the protection systems must also be sophisticated.
The main protection systems of this device are: **a)** starting circuit and short-circuit protection (SCSCP), **b)** overcurrent and overvoltage protection, **c)** atmospheric discharge protection, **d)** voltage spike protection (VSP).
Figure 13 shows the single line diagram of an E-OLTC.
In the scheme above, the E-OLTC is designed in the primary side of the HV/MV transformer. That is purposely done because power losses are lower on the higher voltage side.
However, that also means that the OLTC device will be exposed to higher stresses from the insulation point of view. That is the reason why the aforementioned design and protection lists are important during the engineering of this device.

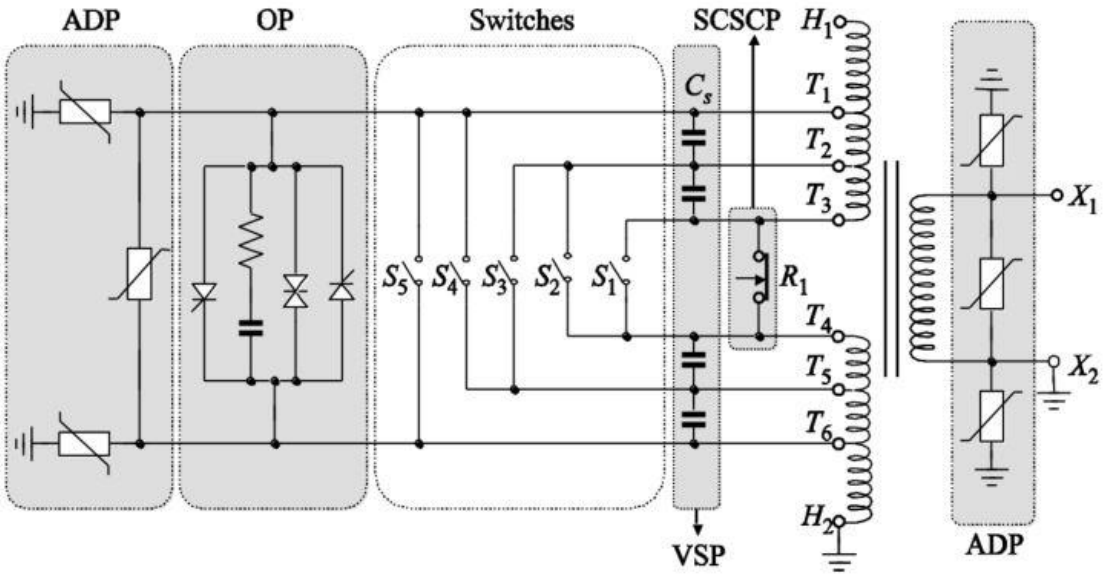


Figure 13. E-On Load Tap Changer scheme, transformer, bidirectional switch, and protection circuit

All in all, it is also possible for the tap changer to be installed on the LV side.

2.3. Comparing LVR and E-OLTC

E-OLTCs have shown in various studies that they are able to regulate the voltage when dealing with networks that have high levels of PV generation [15] [16].

According to studies where combining E-OLTCs and adjustable PV inverters is studied, results show that it will suffice as a measure to regulate the voltage [16].
However, this study assumes a fixed PV penetration level, which is not the case in

real life. That is the reason why this study does a 2-year Quasi-Dynamic analysis with varying PV outputs hourly.

Another problem is that the phases may not be balanced, which is what many studies do not get into, this thesis included. In studycases analyzed here, the system is assumed to be 3-phase balanced.

As for another study carried out by a German distribution system operator named EnBW Regional AG, it shows that the voltage was kept within the allowed range even when loads were fluctuating. That was made possible by installing an E-OLTC in the MV/LV transformer, where the PV injection was coming from [13].
However, E-OLTCs may not be sufficient every time, and help is needed from other tools. This thesis has a similar approach, but it is arguably more sophisticated, as it performs a Quasi-Dynamic analysis of 17520 load-flows with detailed hourly characteristics for every element of the feeder.
As inferred earlier, LVRs are better suited when dealing with longitudinal feeders, while E-OLTCs deal better with short feeders with many branches. LVRs perform very well when dealing with under-voltages.

With that being said, as it was predicted at the beginning of this thesis, the answer of voltage regulation in networks with high DG, probably lies in some sort of combination of E-OLTCs and LVRs.

Such combination will be thoroughly studied in the second studycase of this thesis. In that case, both E-OLTCs and LVRs will be installed, where E-OLTCs mainly deal with overvoltages directly in the busbars where DGs inject, and LVRs mainly deal with undervoltages throughout the main branch of the feeder.

In conclusion, a mix of the two, together with adjustable inverters in PV plants, is what will make sure that the system's voltage constraints are not violated. If even that does not prove to be enough, capacitor banks can also be added into the mix.

3 Machine learning approach on voltage regulation

3.1. Introduction to Machine Learning

Machine learning is a relatively new field in the natural sciences.
Generally speaking, ML recognizes patterns from some given inputs/outputs and then is able to predict the outputs for future inputs that have not been seen before. As ML pioneer Arthur Samuel put it: Machine Learning is the field that gives computers the ability to learn without being explicitly programmed.

ML is generally divided in two categories, Supervised Learning and Unsupervised Learning, depending on whether the program is told about the results (the former) or is left alone to group the outputs without knowing what the results should look like (the latter).
ML is based on mathematical concepts, and it can be applied virtually in any sector, from mere car sales to diagnosing cancer using X-ray scans [17].

The theory behind the mathematical concepts used in ML, such as linear and logistic regressions, gradient descent, neural networks, and others, may at times be complicated to grasp profoundly. However, in day-to-day applications of ML, models are programmed quite superficially thanks to high-level programming languages.

Various open-source libraries, of various programming languages, make it easier for the user to apply machine learning to their problem without having to dig too deep into the mathematics behind it. In recent years, Python has had the main word amongst ML programming languages, especially thanks to Tensorflow/Keras library.

ML possesses quite a range of concepts and tools. However, only the basics of ML and tools that will be directly used in voltage regulation in MV grids will be introduced here.
Keeping in mind that inputs are referred to as features and outputs as labels, the following standard notations that will be used later are:

$x^{(i)}$- the vector that includes all features of the i^{th} training example.
$x_j^{(i)}$- feature j *of* i^{th} training example.
m - the number of training examples
n - the number of features

When there are training examples with multiple features, we will also be dealing with a large number of vectors and matrices.
When coding such algorithms, the best approach is to use vectorized versions of expressions. Otherwise, many for-loops would be needed, which is less efficient in terms of code size and memory.

After the basics are discussed briefly, Multi-output classification Deep Neural Networks will be introduced as the chosen algorithm for our goal of voltage regulation using transformer tap changers and line voltage regulators.

3.1.1. Linear regression and gradient descent

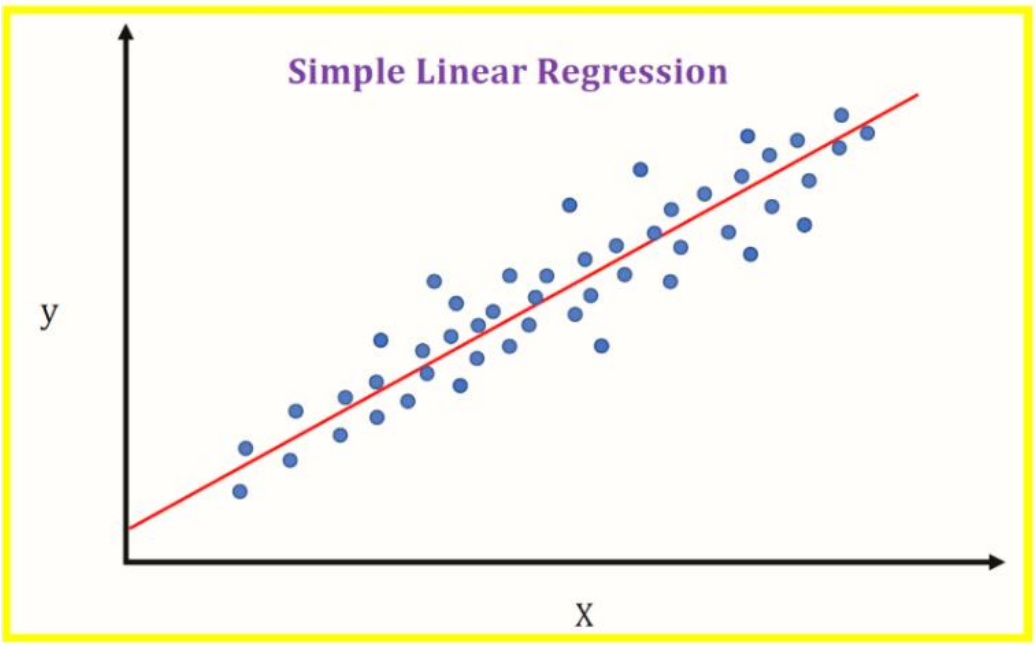


Figure 14. Linear Regression

The most basic ML algorithm, Linear Regression (LR) is shown in Figure 14.

The blue points in the graph show a set of a data that have specific X and y values[1]. The red line is what the linear regression algorithm comes up with. It shows the system's prediction for future values, built by training on this dataset.

[1] By convention, X (capital letter) denotes features/inputs while y (miniscule letter) denotes the outputs/labels.

The blue points can represent any 1-dimensional data. That means, 1 feature only. Next, in Figure 15 we see how the algorithm works from the inside.

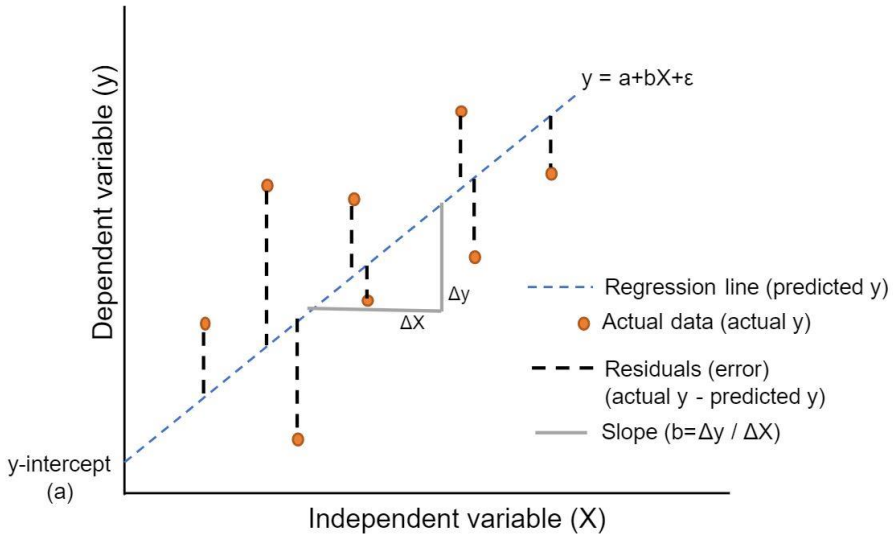


Figure 15. Mathematics behind LR.

The program starts with a hypothesis function of the form:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x,$$

which is none other than the generic line equation $y = ax + b$. In Figure 14 this line is shown in a blue, dashed line.

The way the program measures the accuracy of our hypothesis is by calculating the Cost Function (CF).
CF uses the least mean squares of the errors, which are the distances of the actual data points from the predicted line. These distances are shown in Figure 15 with black, dashed lines.

Cost function is calculated using the following formula:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

The better the line fits our data, the lower the cost function will be. In the ideal case of a perfect fit, the cost function would be zero.

Besides CF, other fitness indicators related to linear regression are Correlation Coefficient (r) and Coefficient of Determination (r-Squared).
'r' and 'r-Squared' coefficients range from -1 to 1 and 0-100%, respectively.

An r-Squared coefficient equal to 0 means that the predicting variable cannot explain the response variable at all. On the other hand, if r-Squared coefficient, the predictor variable explains the response variable perfectly.
The CF can be lowered using mathematical tools such as *gradient descent*.
So, the goal is to minimize $J(\theta_0, \theta_1)$ by changing θ_0 and θ_1.

When dealing with linear regression, the CF $J(\theta_0, \theta_1)$ has the shape of a bowl, as represented in Figure 16.

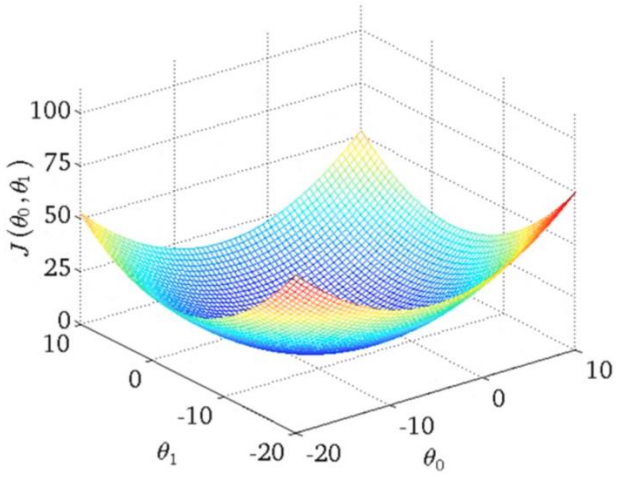


Figure 16. Cost function graphic representation.

When dealing with linear regression, it is easier to represent the CF graphically. From this point onwards, using mathematical tools such as gradient descent, we try to minimize $J(\theta_0, \theta_1)$.

Gradient descent is a mathematical tool that uses the derivative to find lower values of the CF. This step is repeated multiple times until the minimum is found. If the regression is linear, the minimum found will always be the global minimum. In other cases, we may find a local minimum but fail to find the global one.

A 2-D graphical representation of CF and gradient descent is shown below in Figure 17.

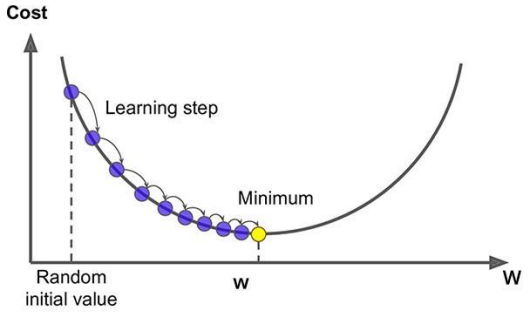


Figure 17. Gradient descent principle of work. W represents θ.

First, an initial value is chosen.

Then, the derivative of CF is found at that point. The derivative is also the slope of that same point. Hence, if the derivative is positive, it means that choosing another point to the right of it would give a higher CF, while choosing another point to the left of that point gives a lower CF, which is desired.

The contrary can also happen, if the derivative at that point is negative, it means that choosing another point to the right of it would give a lower CF, this is the case in Figure 17. This step is repeated many times until the minimum is found.

Using mathematical expressions, θ_j will be assigned new values on each step:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{ for j=0,1.}$$

Where $\boldsymbol{\alpha}$ is the Learning Rate.

The Learning Rate determines how fast the algorithm will get to zero, however, if it is too large, the algorithm may overshoot the minimum value. Thus, choosing the right learning rate is very important.

Besides constant Learning Rates, there are adjustable learning rates as well, that go on increasing/decreasing after a number of steps. This adaptive method of $\boldsymbol{\alpha}$ makes it possible for the algorithm to perform much better and faster.

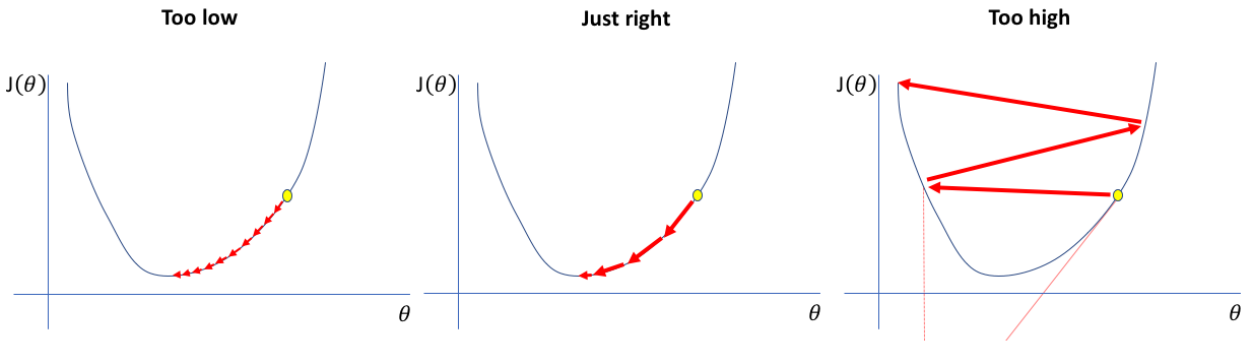


Figure 18. Choosing Alpha: Learning Rate of Gradient Descent

As mentioned before, a wrong value of $\boldsymbol{\alpha}$ may cause the algorithm to either overshoot or take too long to converge.

Let's discuss the choice of Learning Rate, represented in Figure 18.

In the first case, we have a very small $\boldsymbol{\alpha}$, it means the algorithm will need to update the θ_j values way too many times. That in turn will slow down the algorithm and convergence will not be achieved in a conveniently fast time.

In the second case, α is not too small nor too large, meaning the algorithm will not take too much time to converge, yet it will still allow our CF to converge to zero.

In the third case, α is too large. When the learning step is too high, the global minimum will be overshot, and the algorithm will not converge because the solution will be missed.

An illustrative case of gradient descent is shown in the figure below.

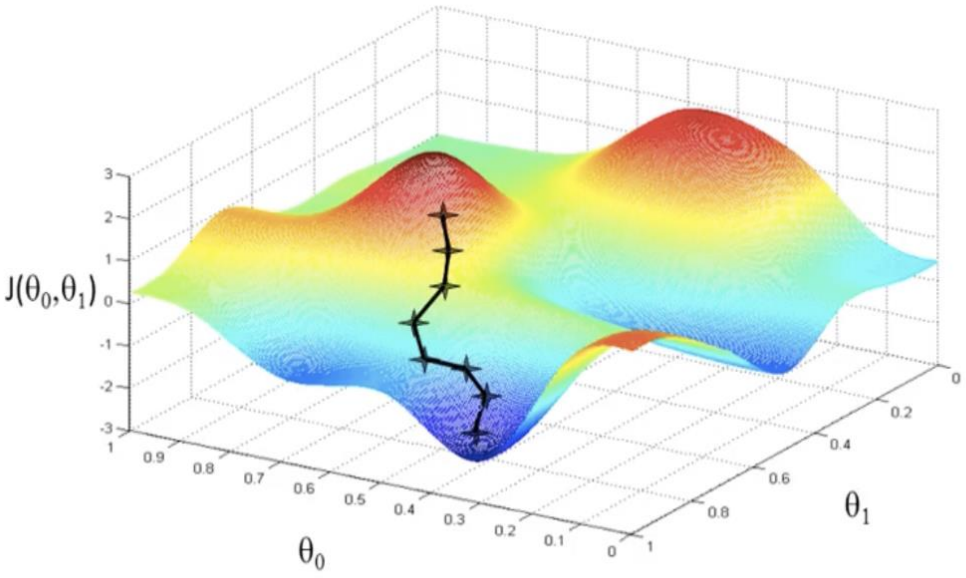


Figure 19. Gradient descent being applied to lower a Cost Function.

So far, we have seen *linear* regression, meaning our hypothesis was made of variables that are of exponent one, regardless of how many variables our hypothesis has. However, they can be of higher powers as it will be discussed in the following sections.

3.1.2. Polynomial regression

Regression does not have to be linear every time, it can be of higher orders as well.

Polynomial Regression (PR) comes in handy when the data we are working on is not characterized by linear dependency between the dataset's points. In day-to-day applications, polynomial regression of orders 2, 3 and up, is used more often than the linear regression.
This is true because that happens to be the nature of the phenomenons we encounter more often.

In PR, the hypothesis will be more sophisticated. The features will be: $\theta_j{}^n$, where j – number of features and n – exponent of the hypothesis. For example, if dealing with cubic regression, n=3.

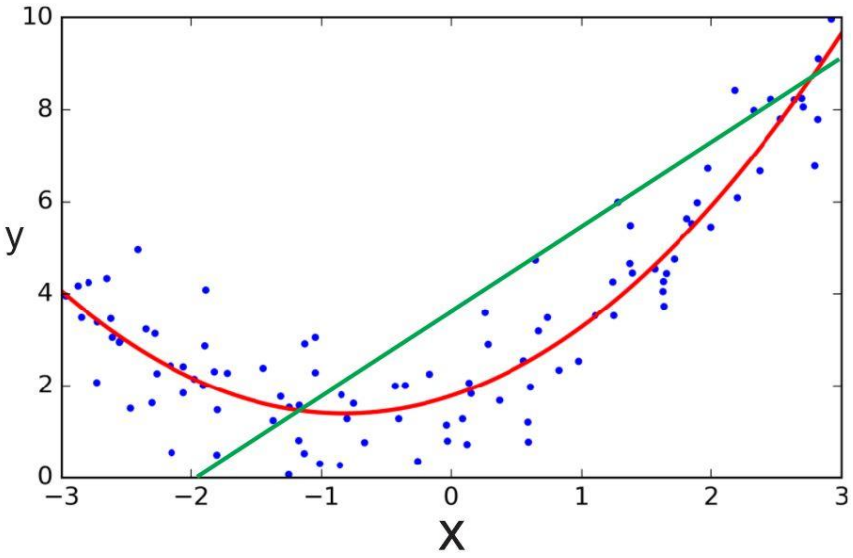


Figure 20. Linear Regression (green) vs Polynomial Regression (red) fitting a random dataset.

As seen in Figure 20, linear regression is not a good fit for datasets that have no linear dependency. In this particular case, the dataset values shown in blue dots, are better fitted using a polynomial hypothesis function (red curve) rather than a linear one (green line). Polynomial regressions can be of any order, but if the order is too high, there may be overfitting.

3.1.3. Bias/Variance tradeoff

Overfitting a particular set of training data is a problem encountered oftentimes, and it must kept in mind as something to avoided while modelling a ML algorithm.

So, although it is counter-intuitive, that means we do not want our CF to be exactly zero and r-Squared coefficient to be exactly one.

If the CF is zero, it means that the hypothesis fits the training data perfectly, but if we feed the machine another set of data, the results will most likely be poor. That is called overfitting.

If the opposite happens and the CF and r-Squared coefficient are large, we will have underfitting. That happens when a hypothesis has coefficients that do not fit any dataset.

Figure 21 shows these cases graphically.

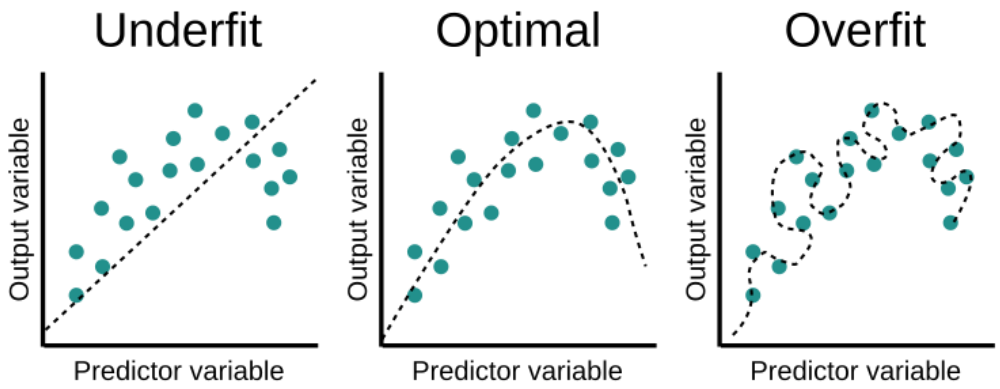


Figure 21. Regression model: a) Underfitting, b) Good fit, c) Overfitting.

In the first graph of Figure 21, we are dealing with underfitting. Our linear hypothesis is not accurate for this dataset. The CF will be too high and r-Squared coefficient will be too low. Both indicate that we have a model that is underfit. This case is also known as **High Bias**.

In the second graph, the CF and r-Squared will be better suited, although purposely not perfect.
This means that in the future, when this model is fed with a dataset unseen before, it will most likely perform well.

In the third graph, the cost function will be zero and r-Squared coefficient will be 100%. Both pointing to a perfect fit. Though it means the model found the exact hypothesis to describe our dataset, it most likely means that this model will perform poorly when fed with a new dataset that was not part of the training set. This is known as **High Variance**.

Evidentially, we need to find a middle ground between 'un-fitting' the training model, and overfitting it. This is a famous concept in ML, and it is called as the **Bias/Variance trade-off**.
Algorithms discussed thus far were simple cases of one-dimensional training sets. In real life cases, we will have to deal with multiple dimensions of datasets. These dimensions represent features.
In other words, our hypothesis will not have only θ_0 and θ_1 but it may have ${\theta_j}^n$, where j=0,1,2,,… and n=2,3,…
The higher the number of features, the harder it is to visualize the CF with respect to hypothesis coefficients.
For training examples with multiple features, cost function and gradient descent are the similar, but adapted.

3.2. Deep Neural Networks and activation functions – Utilization of machine learning tools in distribution networks

This subchapter will introduce the perceptron, activation functions and basic architectures of neural networks. Those are the prerequisites to Deep Neural Networks.

3.2.1. The perceptron

Before getting to our specific model, the basics of Neural Networks will be introduced and the potential choices of using particular algorithms will be explained.

Neural Network are a relatively new idea. Their name and inspiration comes from the human brain neural networks. The human brain has around 86^9 (eighty-six billion) neurons that are interconnected and fire electric signals to each other. [18]

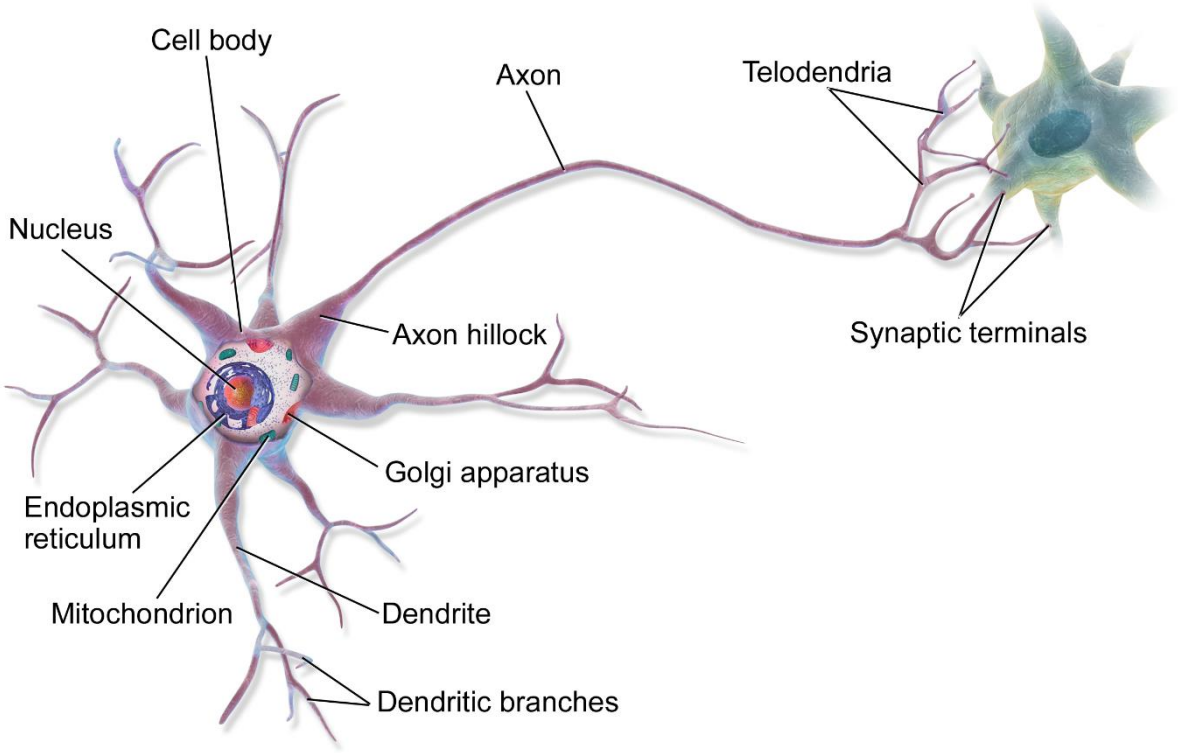


Figure 22. Brain neuron.

The electric impulses happen when the cell wall (membrane) changes its voltage. Each neuron is connected to thousands of other neurons. Neurons send and receive signals through the conductors (axons) to and from neighboring neurons. Whether

a neuron will 'fire' or not depends on the voltage threshold. If the threshold is reached, the neuron will fire. This happens to be the inspiration of Artificial Neural Networks (ANNs).

The basic artificial neuron is shown in Figure 23.

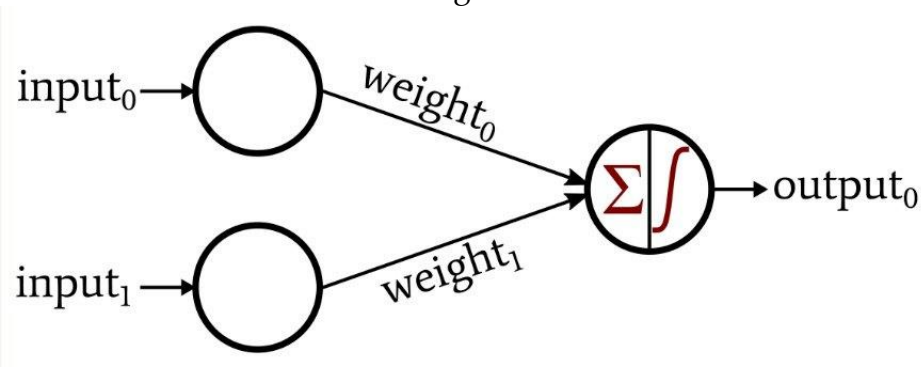


Figure 23. Basic Artificial Neuron topology.

Figure 23 shows a similarity in design between a natural neuron and an artificial one.

Here, the neuron is connected to two other neurons. Neurons connected to each other exchange impulses according to their voltages, which in this case are called weights. The weights are then accumulated and their aggregate determines whether the neuron receiving the impulses will now emit forward to other neurons an impulse of its own or not. The value that determines whether a neuron will have an impulse is called a threshold.
In algebraic terms:

$$\text{Output} = \begin{cases} 0 \; if \; \sum_j w_j x_j \leq threshold \\ 1 \; if \; \sum_j w_j x_j > threshold \end{cases}$$

So, a 0 signifies that the neuron will not be producing any impulse on the output, while a 1 implies that the neuron will 'fire'.

This basic single artificial neuron in ML is known as *the perceptron*.
The perceptron can be used to determine outcomes or outputs of some event or phenomenon when given inputs. Each input has its weight. The weights w_j determine the importance of input j.

The expression $\sum_j w_j x_j$ determines the output depending on whether the threshold is reached or not. This expression is better written as a multiplication of vectors or matrices $w_j \cdot x_j$.

In neural networks, *bias* is used as a parameter instead of the threshold. Bias and threshold are equal but of opposite signs. Bias can be used to re-write the output equation as follows:

$$\text{Output} = \begin{cases} 0 & if \quad w \cdot x + b \leq 0 \\ 1 & if \quad w \cdot x + b > 0 \end{cases}$$

So, the output of a perceptron is determined by the multiplication of the weights to their respective inputs, and then adding that to the bias.

The bias determines the nature of the perceptron, because if it is large, the perceptron will have an output equal to one. If the opposite happens, meaning the bias is a small or a negative number, the chances are higher for the output to be equal to zero.

Building a perceptron using a day-to-day example

Perceptrons are the building blocks of neural networks. Basically, ANNs are really just many perceptrons (neutrons) linked with each other in a similar fashion to the brain neurons, hence them being introduced first.

Let's take an illustrative example; You want to build a perceptron that determines whether you want to go out drinking with your colleagues on a Friday night.

Some inputs could be the following: your level of tiredness, the amount you like the colleagues who invited you, your bank balance at the time, your plans for the next morning etc. These inputs can be designed in any manner, here is one.

I) If you work really hard throughout the week and on Friday night you get way too tired to do things, that means that the weight of input 1 is large and negative as it lowers the probability for the threshold to be reached, so let's say it is (-3).
II) If when you go out drinking, you are not picky about who you are going out with, that means the weight of input 2 is small, let's say (1).
III) If your stomach has low tolerance and you get sick easily, rendering you unable to commit to your plans the next morning, it means that the weight of input 3 is also large and negative, let's say (-2).

Modeling a perceptron that tries to predict whether you will go out on a Friday night could look something like this:

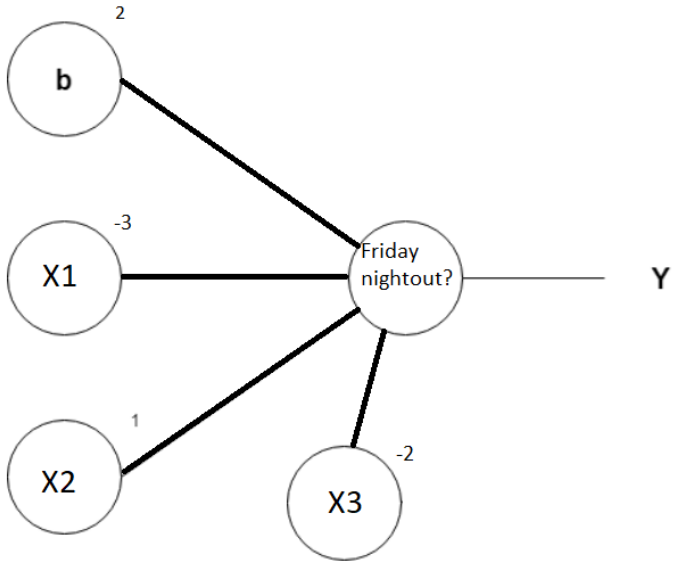


Figure 24. Perceptron trying to determine whether to go out or not.

In this case, the bias is 2, which is quite high, assuming that momentarily the person felt like going out because they have not been out in a long time. This is the biases job, to influence the outcome considering the fixed weights.

Let's assume the person to not be tired (X_1=0), to like the people they are going out with (X_2=1), and finally, to have plans early the next day (X_3=1).
The outcome of this result would be: $y = \sum_j w_j x_j + b = (-3)\cdot 0+1\cdot 1+(-2)\cdot 1+2 = 1$ which is > 0, meaning the perceptron thinks you will be going out.

It is noteworthy how bias plays a huge role in the outcome.
Every model, or in this case person, has different inputs, with different weights, and a different bias.

Depending on the circumstances, the inputs multiplied by their weights will determine whether the output will be a 1 (output larger than 0) or a 0 (output 0 or less), where 1 means going out, and 0 means staying home.

Many processes can be modelled and predicted in a similar fashion.
Basic logical functions such as AND and OR are used as primary examples of how perceptrons work. The following figure shows how the AND gate is realized using a perceptron.

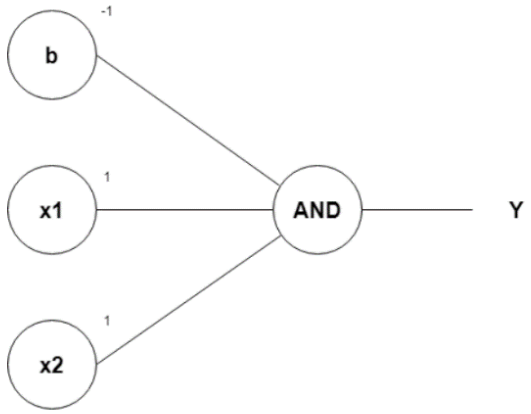


Figure 25. AND gate realized using a perceptron.

As seen in Figure 25, the bias is (-1), weights of inputs X_1 and X_2 are both 1.
Let's try some input values to check whether the outputs really correspond to an AND gate.

The equation is $\sum_j w_j x_j + b$.

If $X_1 = 1$, $X_2 = 1$, b=-1: (1·1) + (1·1) + (-1) = 1 which is > 0 so we get 1.

If $X_1 = 1$, $X_2 = 0$, b=-1: (1·1) + (1·0) + (-1) = 0 which is not >0 so we get 0.

If $X_1 = 0$, $X_2 = 1$, b=-1: (1·0) + (1·1) + (-1) = 0 which is not >0 so we get 0.

If $X_1 = 0$, $X_2 = 0$, b=-1: (1·0) + (1·0) + (-1) = -1 which is not >0 so we get 0.

Thus, using the previous perceptron equations we have verified that if those weights and that bias is used in a perceptron with two inputs, we will have built an AND gate. We can confirm our perceptron model from Table 2 below.

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

Table 2. Logical gate AND

That is pretty much all there is to a perceptron as the most basic unit in a neural network, on which we will expand next. A perceptron is the fundamental block of large neural networks which may contain tens of thousands or even more perceptrons.

The only sophistication of neural networks with respect to perceptrons is that, in the former, the outputs of a perceptron, happen to also be inputs of another perceptron.

3.2.2. The basic architecture of Neural Networks

In this subchapter, a first glance will be given at neural networks.
The following figure offers visual understanding on how neural networks are made out of perceptrons.

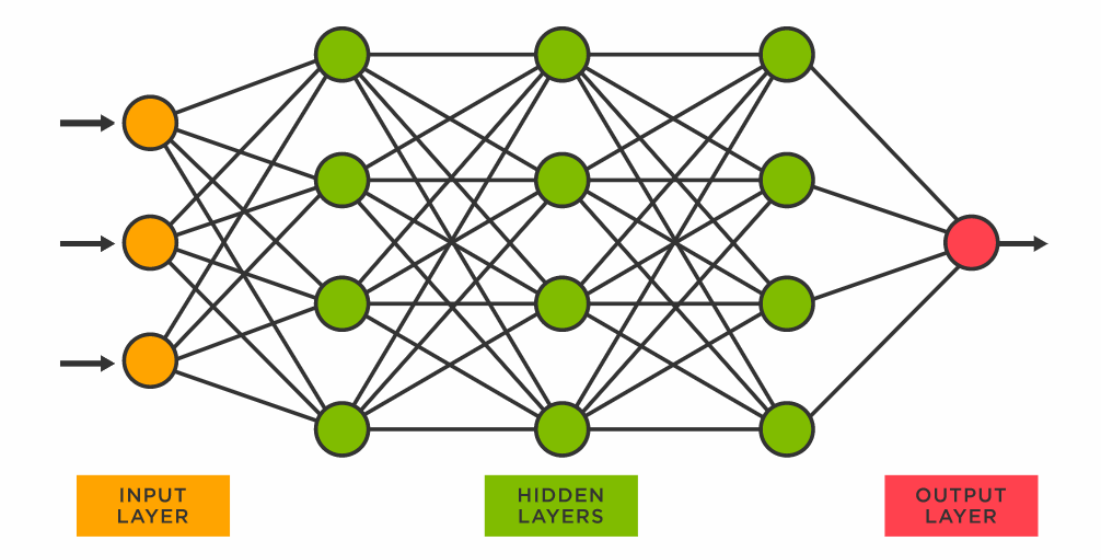


Figure 26. Neural networks with single output.

Figure 26 shows a generic neural network. All the circles represent perceptrons, which is what makes the neural network what it is, a vast network of neurons/perceptrons.

For clarification, neurons and perceptrons are the same thing, with the sole difference being their activation function. However, they are used interchangeably here.

Neural networks are made of layers, the first one is called the **input layer**, the last one is called the **output layer**, and any layer in between the input and the output layer is considered as part of the **hidden layer**.

The hidden layer may contain a single layer, it is not necessary to have multiple layers in-between the input and the output layers.

Neural Networks can be Deep Neural Networks or not based on whether there are multiple layers in the hidden layer or there is a single layer in the hidden layer, respectively.
As seen from this neural network, there are three neurons in the input layer, which is equal to the number of inputs as well. Now, instead of the Layer 1 neuron's outputs to be the final outputs, in the case of neural networks, they become inputs of the neurons placed in the next layer (Layer 2).
This goes on until the final layer is reached and the output is the final output instead of being an input for the next layer of neurons.

Forward propagation and **backward propagation** are the main concepts when it comes to how a neural network is trained from the technical point of view.
These concepts are rather difficult to derive mathematically. Therefore, since this thesis is more concerned about applying electric power system's problems to neural networks, we will not dive any deeper in them.
It is only important to remember the following generic steps: an input goes through the net with the initial weights and bias, reaches the output, the output value is compared to some reference values (cost function is checked), the algorithm starts going backwards towards the input and updates the weights depending on what the loss function shows, the cycle re-starts with updated weights and bias.
So, all the network needs to do is to minimize the loss/cost function by changing weights and biases.

Neural networks are largely more robust than single neurons when it comes to learning.

3.2.3. Activation functions

Some problems arise if we model neural networks using the activation function we have used for perceptrons, namely zeros and ones. If that is done in neural networks, the output of the entire NN will be highly correlated to any changes in the weights and/or biases in any part of the network.

In other words, a small change in weights and/or biases will cause a large change in the output. That is not desirable because the output should be very 'stable'.

There are many activation functions, some of them are briefly discussed and shown graphically in the following pages.

a) Sigmoid function

Instead of using zeros and ones as inputs, we will use a function called *sigmoid*, introduced next.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z=\sum_j w_j x_j + b$.

Therefore, the final expression of the output of a sigmoid neuron will be:

$$\sigma(z) = \frac{1}{1 + e^{-\sum_j w_j x_j + b}}$$

Sigmoid is also referred to as the *logistic function*.

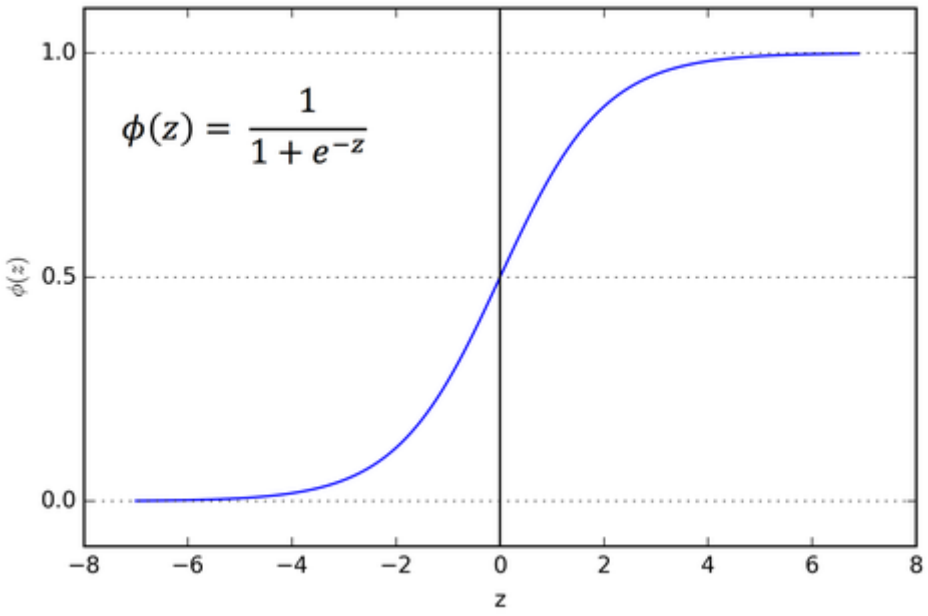


Figure 27. Sigmoid function.

By using the sigmoid function, we will make it possible for the inputs to take any value ranging between 0 and 1. That means, instead of making a steep change when trying to adjust the inputs, changes such as from 0 to 1 and vice-versa, now we can go smoothly.

We can change the input from 0.34 to 0.38 for example. That would not bewilder the output because the change is only 0.04, while previously the change was 1. In other words, we have increased the resolution of the input values.

The sigmoid function represented graphically in Figure 27, give a better understanding of its probabilistic nature. The sigmoid function has some semblance with the unit step function (Heaviside signal). The difference is around the ordinate.

The unit step function has discontinuity at value x=0 (ordinate), while sigmoid is smoothed out, continuous and derivable function.

Now, the difference between a neuron and a perceptron should be clearer. A neuron's activation function is sigmoid function while a perceptron's activation function is the step function. Of course, a neuron can have other activation functions as well, the statement of sigmoid being a neuron's activation function is valid only in explanatory differentiation from the perceptron.

Always referring to the sigmoid function's graphical representation in Figure 27, we will analyze its behavior.

We keep in mind that z is the output of a neuron, which is in direct proportion of the inputs, input weights and bias such as $z=\sum_j w_j x_j + b$.

If the input weights or the bias is large, z will be a large positive number. When plugging that z into the sigmoid function, it will look as follows:

$$\sigma(z) = \frac{1}{1 - e^{-\infty}}$$

Since $e^{-\infty} \approx 0$, we will have $\sigma(z) \approx 1$.

From this, we infer that large weights and biases will make that neuron 'fire'.

If we analyze the opposite extreme, where we have very low or even negative values for our input weights and bias, the expression will look as follows:

$$\sigma(z) = \frac{1}{1 - e^{+\infty}}$$

Since $e^{+\infty} \approx \infty$, we will have $\sigma(z) \approx 0$.

As mentioned before, the idea behind using a sigmoid function instead of a step function is to allow for small changes in weights and bias without causing large changes in outputs.
This makes it significantly easier for the algorithm to get to a desired output by changing the weights and bias.

b) Hyperbolic tangent activation function (Tanh)

The hyperbolic tangent (tanh) activation function assumes negative values as opposed to sigmoid. It also has a slightly more complicated mathematical expression:

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

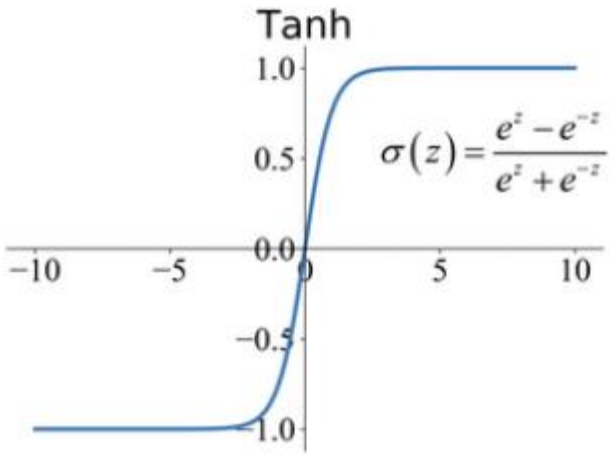


Figure 28. Hyperbolic tangent activation function.

c) Rectified linear unit activation function (ReLU)

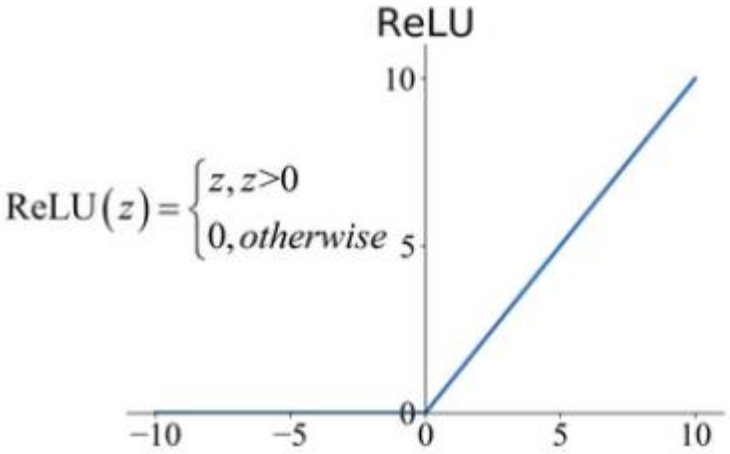


Figure 29. Rectified Linear Unit (ReLU) activation function.

The rectified linear unit, commonly referred to as simply ReLU is a very popular activation function. From the mathematical standpoint, it is simpler than *sigmoid* and *tanh*. On the negative side of the abscissa the function is zero, while on the positive side it is equal to z itself.

$$\mathrm{ReLU(z)} = \begin{cases} z \;\; if \;\; z > 0 \\ 0 \;\; if \;\; z \leq 0 \end{cases}$$

d) **Leaky rectified linear unit activation function (Leaky-ReLU)**

Leaky Rectified Linear Unit (Leaky ReLU) is a sophistication of the ReLU activation function seen in Figure 29.

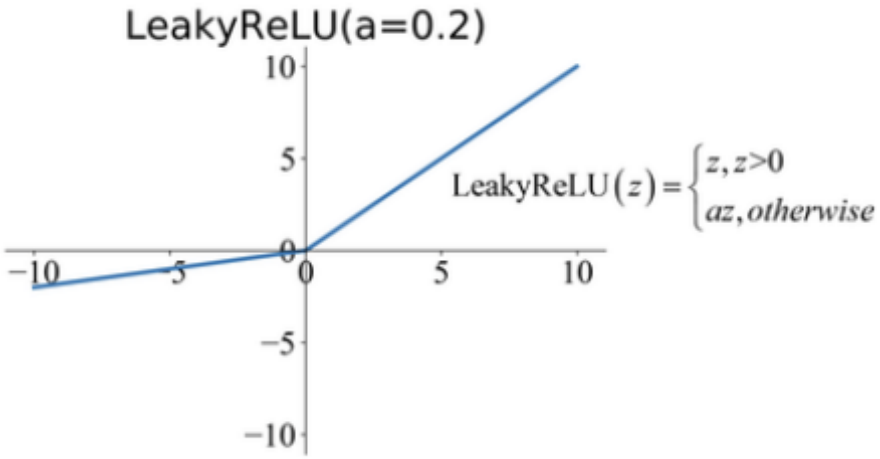


Figure 30. Leaky Rectified Linear Unit (Leaky ReLU) activation function.

The difference between the two types of ReLU activation functions is on the negative side of x-axis. The Leaky ReLU is not zero but instead it is equal to a constant *a*, multiplied by *z* itself, as depicted in Figure 30.
In mathematical terms:

$$\text{Leaky ReLU(z)} = \begin{cases} z & if \ z > 0 \\ a \cdot z & if \ z \leq 0 \end{cases}$$

In the graphical representation, constant *a* is set at 0.2, however, it can be changed accordingly.

Derivatives of Activation functions

One very important consideration when choosing the activation function is their derivative.

Earlier, we mentioned that the dependency of outputs on weights and bias is very important. In technical terms, that dependency is described by partial derivatives as follows:

$$\Delta output \approx \sum_j \frac{\partial(output)}{\partial(w_j)} \Delta w_j + \frac{\partial(output)}{\partial(b)} \Delta b$$

Without analyzing this expression too profoundly, we can see that the change in output (Δoutput) depends linearly on the changes in weights and changes in bias.

So, in order to easier control and improve our outputs, the machine will have to deal with derivatives.
That is the reason for activation function derivatives to be an important

consideration when choosing what activation function to use in a model.

In Figure 31, the derivatives of some popular activation functions are shown graphically.

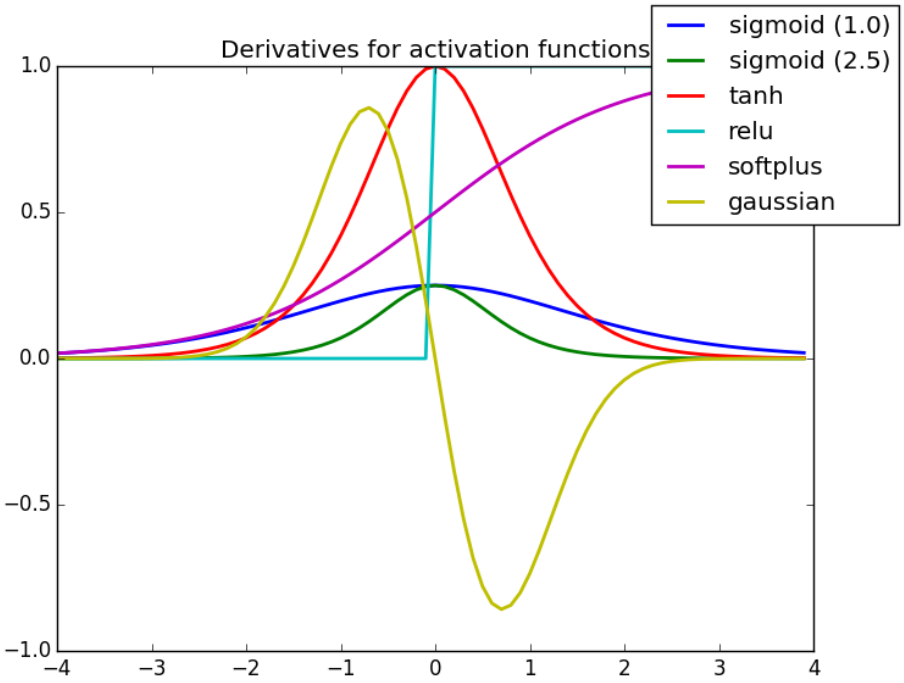


Figure 31. Derivatives of some popular activation functions.

The less computational power our machine needs to calculate the derivative of the activation function, the better pick that activation function is.

3.2.4. Choosing the right ML algorithm

As previously mentioned briefly, the most suitable Machine Learning algorithms to be applied to our problem are Artificial Neural Networks (ANNs).

There are tens of algorithms in neural networks, as illustrated in Figure 32.
Some of the most popular ones are Convolutional Neural Networks (CNNs), Deep Convolutional Network (DCN), Recurrent Neural Networks (RNNs), Hopfield Network (HN), Boltzmann Machine (BM), Support Vector Machine (SVM) etc.

Each of these has proven worthy in some specific use. For example, in image classification algorithms CNNs work best, in natural language processing RNNs work best, and so on.

Generic Deep Neural Networks (DNNs) are the choice of this study.
Specifically speaking, we can model our voltage regulation algorithm as a Multi-

class Multi-output (M-C/M-O) Deep Neural Network. However, building a M-C/M-O is still widely not supported from Python's relevant libraries, rendering this choice as too costly in terms of time and skills concerned for this thesis.

Therefore, the specific algorithm used will be a Multi-Output (binary classification) Deep Neural Network.

The number of outputs will be equal to $N_{nntap} \cdot N_{transformers}$.
So, the total number of outputs will be the total number of tap positions of all transformers of the analyzed feeder that are equipped with E-OLTCs.

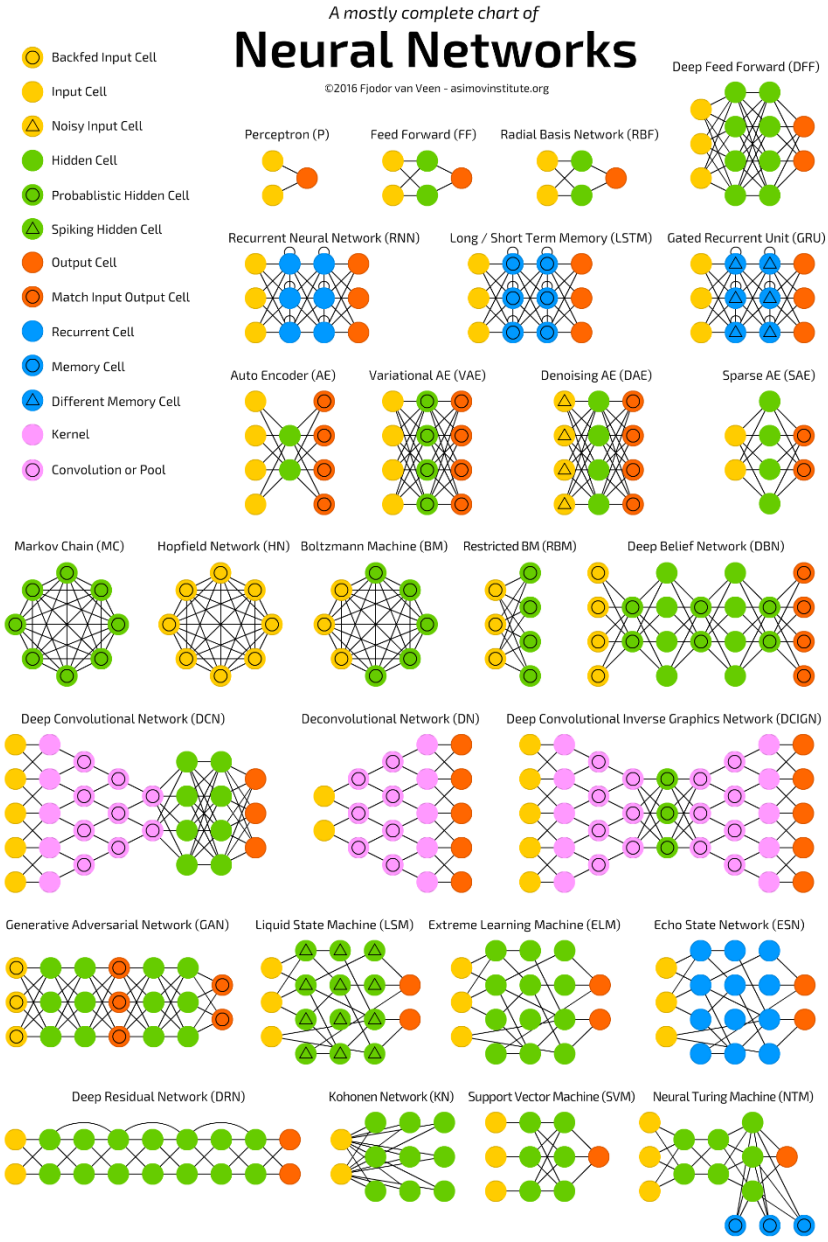


Figure 32. Neural Networks algorithms.

3.2.5. Building a Multi-Output Deep Neural Network for voltage regulation with E-OLTCs

We have introduced the basic neural network architecture earlier in 3.2.2 and said that Neural networks can contain tens of thousands of neurons or even more. We also mentioned what the input layer, hidden layer and output layer are.

In this subchapter we will dive deeper into the specific algorithm we will utilize for voltage regulation in MV networks with the help of E-OLTCs.
It is not mandatory for NNs to have only one output, there can be many outputs.
If there are many outputs, the algorithm will be a multi-output algorithm, which happens to be the one used in our model.
The algorithm we will use in regulating the voltage in medium voltage networks in grids with high penetration of distributed generation and electric vehicles, will be a Deep Neural Network (DNN) with multiple outputs (neurons).

Depending on how we build the algorithm and what activation function we use, the output can assume different values. If we have a single output and it takes values from 0 to 1, it means we are using a binary classification, e.g., if a photo fed to the algorithm shows a crosswalk or not. This particular NN, with a single output, is shown below in Figure 33.

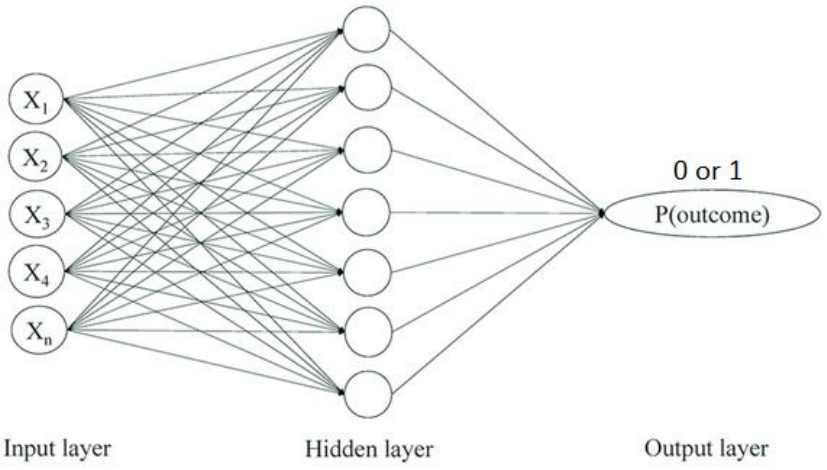


Figure 33.Single output NN with binary classification.

The output layer gives a single probability value that represents the chances of the input to be or not to be what we are testing it for.

For example, if the probability of outcome P(outcome) = 0.95, it means that the input we have feed to the algorithm has a 95% chance of being what we expected it to be.

If the NN has two outputs, they give the probabilities of each output being the

correct one. In our algorithm, we will use an output for each tap position of each transformer. Those outputs will give the probability of that tap position to be the correct one with respect to the inputs.

To illustrate this concept more clearly, a NN designed to find out whether a photo fed to the algorithm shows a dog or a cat is shown below.

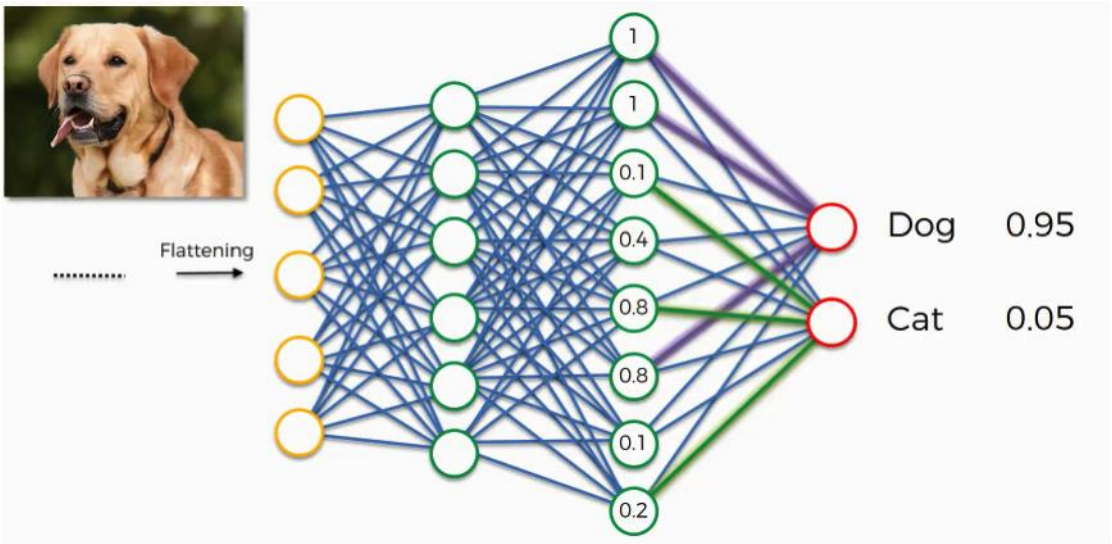


Figure 34. Neural Network determining: Cat or Dog.

Here, we feed an image to the NN, and on the output layer we have two neurons, the upper one gives the probability of the fed image to be a dog, while the lower neuron gives the probability of the fed image to be a cat. Outputs can be re-arranged in any manner.
In this particular case in Figure 34, the algorithm determines that there is a 95% chance the fed image is a dog, and rightfully so. Curiously, this particular model has reached a 97% level of accuracy.

The types of values shown on the last layer, depend directly on the activation function we choose.

It is important to clarify that in general there are two ways to program the output layer:

a) *Exclusive multi-output classification* Algorithm.

That means that the sum of all probability values of all outputs will always be equal to 1. That is natural in cases where the input's result can only be one of the choices/outputs, for example when trying to figure out if the photo is

showing a dog or a cat. If the photo shows a cat, it definitely is not showing a dog, and vice versa.

b) *Non-exclusive multi-output classification* Algorithm.

As the name infers, the categories in this case are not mutually exclusive. That means, a set of inputs can assume many categories/outputs simultaneously. This will be the case in our model, in order to not break the voltage constraints, a transformer's tap position can be set at more than one of the settings.
For example, let's say that at substation Feroleto, a transformer's tap position is (-1) and voltage at that busbar shows 0.97 [p.u], then, the transformer's tap position is set at (0) and the voltage at the busbar now shows 0.99 [p.u].
According to |ENTSO-E the voltage constraint of ±5% will not be broken in neither of the aforementioned tap positions. [8]
In this case, the sum of all probabilities in the output layer does not have to be 1.

The difference from the introductory topology of the NN we saw in Figure 26 and the one we will use in our model, is on the output layer. While in Figure 26 there was only one neuron in the output, in our model, there will be $5 \cdot N_{transformers}$ outputs.

As mentioned previously, we will train a multi-output classification NN, which means that we will have a fixed number of outputs, each with a result between 0 and 1. We will feed the model some inputs which will go through the net, to finally reach the output layer. The outputs represent none other than tap changing positions of MV transformers.

Transformers, depending on the date of production and the standards used by the manufacturer, may have different ranges of adjustable voltages.
For medium voltage transformers, the number of tap positions usually happens to be 5. That means, positions -2, -1, 0, 1, 2. The neutral position is at 0.

From a heuristic approach, it was deduced that the most accurate way to build the NN would be to choose a **non-exclusive** multi-output classification algorithm, which was explained earlier.
If we decided to run with a mutually exclusive multi-output classification algorithm, the training/results would be unrealistic because the transformer tap changers can, in fact, have more than one valid setting. Valid meaning no voltage constraint is breached.
To put it simply, multiple options can be right at the same time.

So far, for our task of voltage regulation, we have concluded the following:

i) We should use Deep Neural Networks.
ii) We should use Multi-Output networks.
iii) We should use a Non-exclusive algorithm.
iv) Our Classification should be Binary, using Sigmoid activation function.
v) The number of neurons in our output layer will be $5 \cdot N_{transformers}$.

These choices are valid in general terms and will be employed for our studycases.

The algorithm will measure its accuracy using Sigmoid as its activation function and Binary Cross-Entropy function as its loss function. In mathematical terms the binary cross-entropy loss functions is:

$$CE = -t_1 \log\big(f(s_1)\big) - (1 - t_1)\log\,(1 - f(s_1))$$

Where $f(s_1) = \frac{1}{1+e^{-s_1}}$ and t_1 means the output is positive.

The Binary Cross-Entropy loss function can also be expressed as:

$$CE = \begin{cases} -\log\big(f(s_1)\big) & \text{if } t_1 = 1 \\ -\log\big(1 - f(s_1)\big) & \text{if } t_1 = 0 \end{cases}$$

So, the loss function graphs we will see later are logarithmic.

Next, specific to the studycase parameters such as number of layers, neurons, outputs, epochs etc., will be discussed independently.
Figure 36 represents the first model at building a DNN for voltage regulation. This model is only concerned with a single transformer with 5 tap positions. The idea has evolved to building a DNN that predicts the taps of multiple transformers instead of a single one.

Each of the output neurons shows whether a specific tap position is the correct one or not. Our sophisticated model has 92 inputs and 65 outputs (13 TRs · 5 tap positions) for Sub-Urb feeder (CS-I). For the Rugova feeder (CS-II), the model has 88 inputs and 45 outputs (9 TRs · 5 tap positions).

It is difficult to draw our real NN in a single sheet, however, just to give an idea, our primitive model was illustrated using online tools [19].
In that NN, there are 40 neurons in the input layer, 75 neurons in the hidden layer (4 layers summed), and finally 5 neurons in the output layer. This should help build a better idea of how our DNN model works.

From a generic approach, Figure 35 is effectively the best representation of our Deep Neural Networks algorithm. The only difference is the number of neurons.

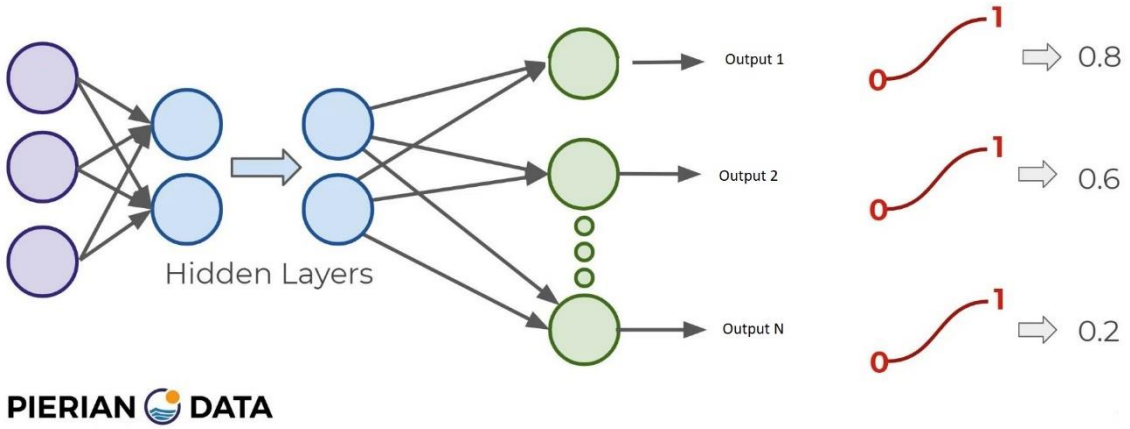


Figure 35. Multi-output Deep Neural Network with binary classification.

DNN terms such as multi-output, multi-class, and multi-label can be confusing. In this study, the intention is to predict multiple outputs (multiple tap positions), but for each output there can be only a probability value from 0 to 1 value as a result. If we were to build a neural network for a single transformer as previously described, we would have a multi-class algorithm as the one in Figure 36.

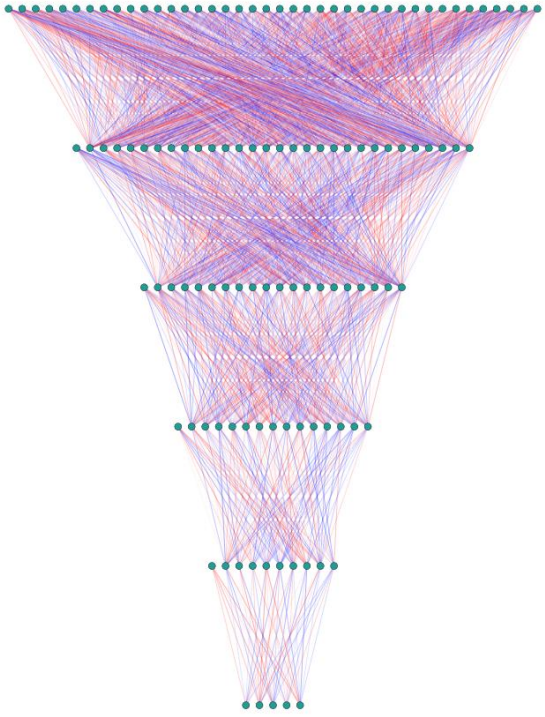


Figure 36. Neural Network architecture for voltage regulation, adapted for a single transformer.

5 outputs represent 5 possible tap positions of a single transformer. In this topology, the bias neurons can be spotted on the left side.

4 CASE STUDY I – 20 [kV] feeder Sub-Urb, Sicily.

4.1. Introduction

This section will briefly explain the general approach and methodology used in both of these studies.

The most difficult part of almost any machine learning project is obtaining and cleaning the data. It is estimated that around 80% of the time and work it takes to build a ML model goes towards data preparation and only 20% for the algorithm itself.

The quality of data is the number one factor that determines the quality of the learning model itself. That is why, the PowerFactory models were troubleshooted tens of times.

First, both feeders are modelled in PowerFactory from scratch. They are both 3-phase balanced systems.

Feeder Sub-Urb is based on the standard IEEE 33 bus MV network. It is called Sub-Urb as it represents a random sub-urban feeder with OHL. The DG of this feeder are connected on newly made LV busbars that serve only for the DGs and not loads. In the second studycase, the model is different. After some testing, it was not deemed very useful to carry out these studies in urban feeders supplied by cable, because usually there is not enough physical space to have large amounts of DGs such as to cause significant overvoltages.

Feeder Sub-Urb operates at 20 [kV]. Active power (P_L_x) and reactive power (Q_L_x) inputs for each load come directly from the standard model's public data. On the other hand, line parameters such as resistance and reactance come from various manufacturer's catalogs. Line lengths were chosen intuitively. Transformers were modelled based on the loads of the busbar where the transformers are connected.

The following chronological steps explain the approach of how this study will be

carried out for both feeders:

1. Collect realistic MV network data on all feeder elements, especially loads, transformers, lines.
2. Use standard programming methods and intuition to fill any missing data.
3. Use the collected and prepared data to model the feeder from scratch using PowerFactory.
4. Model photovoltaic systems at random points in the feeder, making sure their output is high enough to cause realistic overvoltages. Each PV system must have a different installed power.
5. Use GIS tools to extract irradiation data and use it to calculate the PVs outputs. Use exact coordinates for each feeder.
6. Create a 2-year characteristic of every PV output based on the irradiation data for that specific location. The data should be hourly and it will have 17520 values in total. These vectors would mostly consist of zeros, as the PVs do not generate at sunless hours.
7. Analyze daily and seasonal load profiles of residential users located in the feeder's geographic region. Then, use those load profiles to generate 2-year characteristics for each and every load, separately for both active and reactive power consumption. Again, the data should have 17520 values to represent hours of a 2-year period.
8. Adjust all relevant settings and run a few load flows then update any element parameter if necessary.
9. Analyze the power flow results, with special care at busbar voltages, line voltages drops, and most importantly, tap changer positions.
10. Run the power flow with and without ticking the Automatic Tap Changer option. Compare the results to see if they make sense.
11. Now that the feeder's correct modelling is verified, prepare for Quasi-Dynamic Analyses.
12. Run a Quasi-Dynamic analysis for a period of 2 years in hourly steps, that is 17520 load flows in total. Do it without employing the Automatic Tap Changer option.
13. Repeat step 12 but this time do it employing the Automatic Tap Changer option.
14. Save 2 year graphs of loads, tap positions, load profiles, voltage levels etc.
15. Export the Quasi-Dynamic analysis data to '.csv' or '.xlsx'. Focus especially on exporting the tap positions of each transformer with E-OLTC, for each step.
16. Use Python to import the data, clean it, re-organize it, delete what is not needed.
17. When the data is prepared, separate the features/inputs X (Loads, PV outputs

etc.) from labels/outputs y (Tap Positions). Export them afterwards.

18. Divide the data in two: the training set and the validation set.
19. Now comes the ML part. Create a Deep Neural network with a couple of hundred neurons in 4-5 layers. The output layer being equal to the number of dataset's outputs.
20. Run the DNN and based on results adjust its parameters by varying activation functions, number of epochs, ratio of train/test data, including early stopping, changing the architecture etc.
21. Reach a good result in both the training and testing datasets and save them.
22. Visualize results, calculate accuracy statistics and measure DNN's overall performance.
23. Now the model is ready to predict the tap changer positions of several transformers if it is feed new inputs, never seen before.

The above-mentioned steps are perhaps too detailed to paint the big picture, but the flowchart in Figure 37 might help.

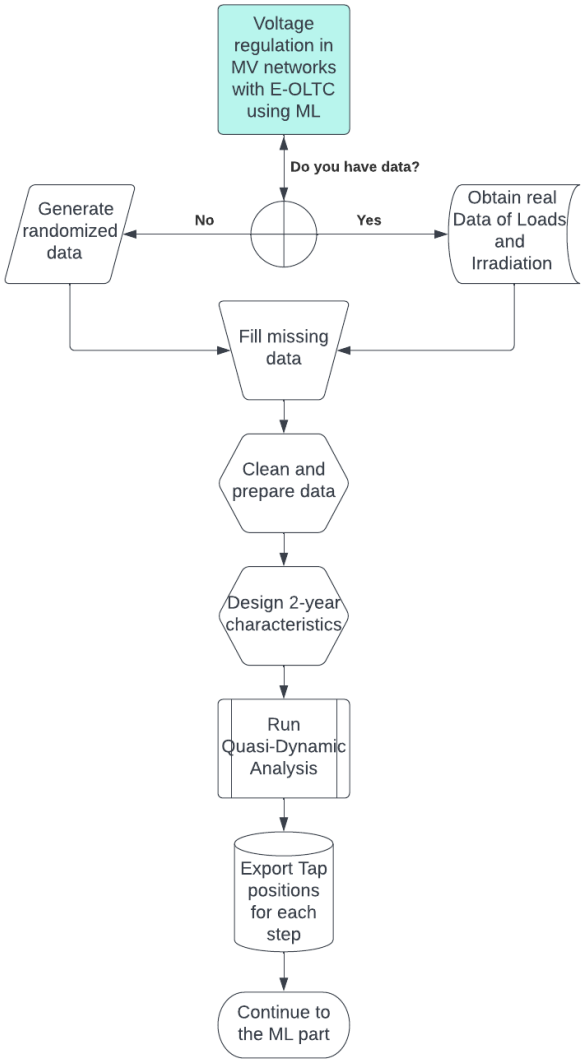


Figure 37. Simplified flowchart of the steps carried out for voltage regulation.

While this was a quick overview, we will see everything in detail while discussing the studycases individually.
Before getting into the study cases, we will take a look inside PowerFactory's automatic tap changer settings and options. These will be valid for both feeders so going through them once only is more efficient.

4.1.1. Tap changing settings in PowerFactory

PowerFactory has two different windows to change the tap changer's settings, one in the Transformer type (TypTr2) and another in the load-flow window of the transformer element (ElmTr2).

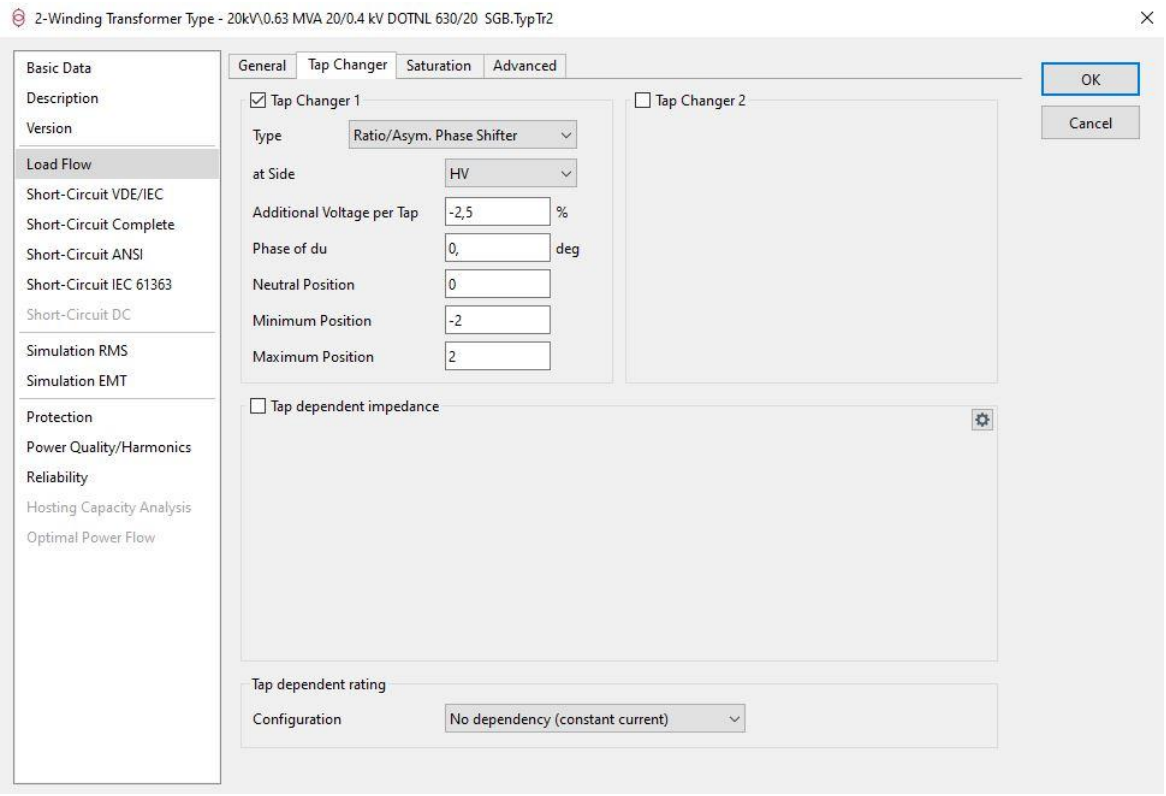


Figure 38. Tap Settings in TypTr2.

Figure 38 shows the settings inside the transformer type, in PowerFactory known as (*TypTr2).

From this window of settings, the user is able to decide the following attributes of the Tap Changer:

1) Type:
   -Ratio/Asym. Phase Shifter
   -Ideal Phase Shifter / Does not change the voltage magnitude
   -Sym. Phase Shifter / Does not change the voltage magnitude
2) Side at which the tap changer is installed:
   -High voltage side
   -Low voltage side
3) Additional voltage per tap: Can be any number in percentage, usually around 1.25 or 2.5
4) Phase of du: Can be any number of degrees
5) Neutral Position, Minimum Position, Maximum Position: The range of positions the transformer is able to operate on. Neutral Position setting means the transformer operates at its nominal transformation ratio.

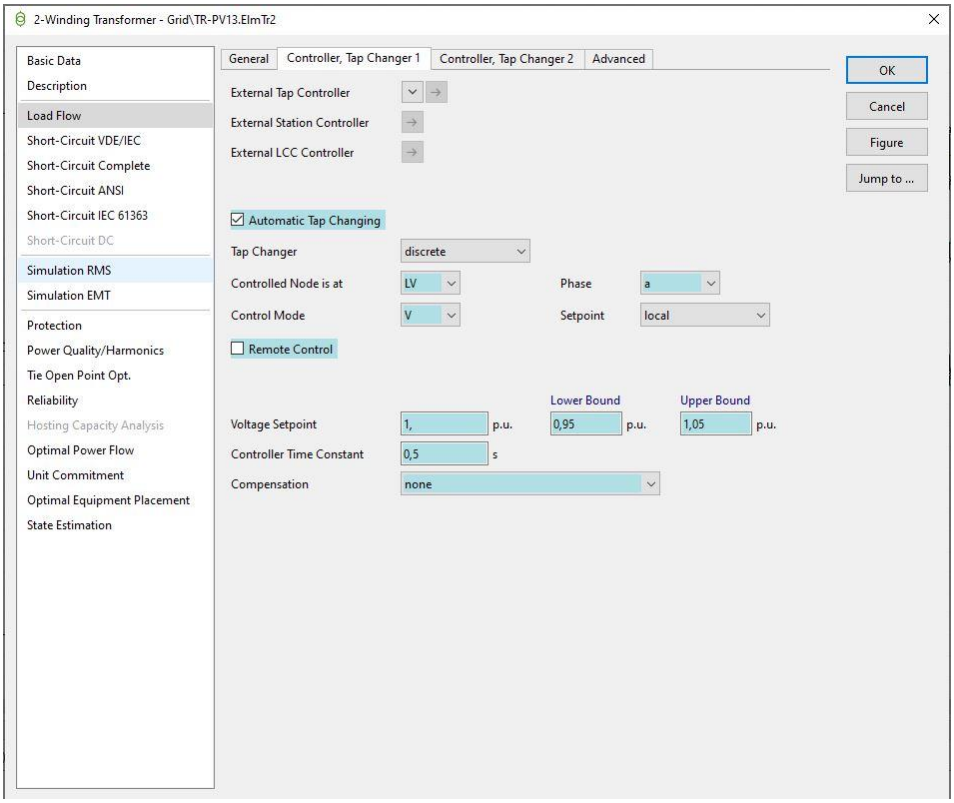


Figure 39. Tap Changer settings in ElmTr2 in PowerFactory.

Figure 39 is yet another window of settings for the automatic tap changer. In this case, it is related to the specific load-flow being run, rather than the transformer type.

In this window, the user is able to regulate the voltage setpoint at the busbar, voltage range (lower and upper bounds), control node of the E-OLTC etc.

4.1.2. Running a Quasi-Dynamic Simulation

A Quasi-Dynamic Analysis is essentially just a set of usual load flows done one after another in fixed time steps.

For our analysis, in order to increase the number of data, which in turn will increase the quality of our ML model, we will be working with 2 years of data.
Our dataset contains hourly data for a full two years. That translates to 17520 values for each of the inputs and outputs.

Before executing the Q-D Analysis, we must insert time characteristics to all the inputs we want to change with time.
We can think of these characteristics as vectors where every element represents the value of that input, at a given hour/step.

PV outputs should have their individual characteristics as well. This characteristic will be deterministic as it comes from irradiation data and is adapted to the installed power of each photovoltaic plant. That will be discussed in detail together with the studycases.
The main window of a Quasi-Dynamic Analysis is shown below.

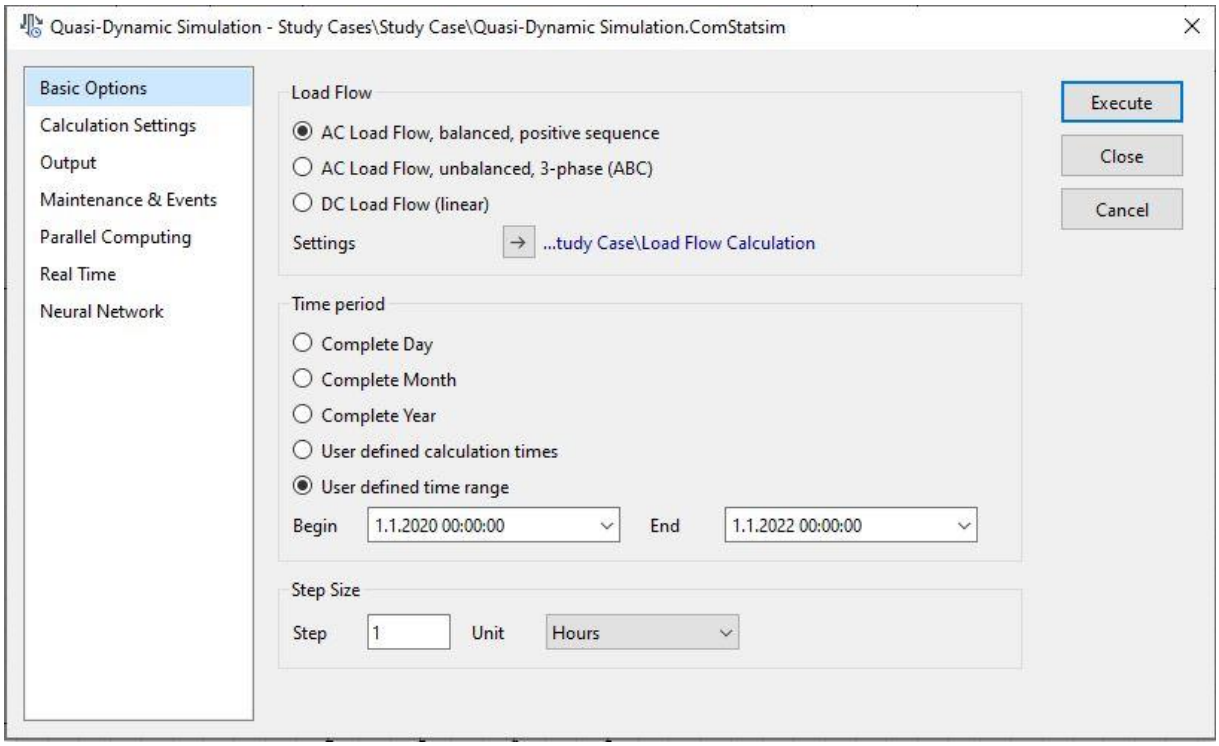


Figure 40. Running a Quasi-Dynamic Analysis main window.

In the main window of a Q-D Analysis the user will decide what type of Load Flow to run, and the settings of that load flow which may include the Power Flow Analysis method (default is Newton-Raphson), number of max iterations, reactive power control etc.
Besides the power flow type, the user will also define the time range and step size. As it can be seen in Figure 40, in our case, we have a 2-year period with a step size of 1 hour. However, PowerFactory may not perform well with characteristics that are longer than a full year, hence, under the hood, two 1-year Q-D analysis were run for each 2-year Q-D analysis reported here.

Figure 41 and Figure 42 are windows of settings that deal with exporting the data. After the analysis is complete and 17520 load flows have been run, the results must be saved and exported to a '.csv' or another format easy to work with.
Using these settings, the user will be able to select the time period of results being exported. This comes in handy if the user is interested in only a portion of the results rather than the entire timeframe. The user is also able to select what variables are to

be saved. PowerFactory has tens of variables to save for each element.
Finally, these saved data will be then used in the ML part, where they will make it possible for the models to be trained.

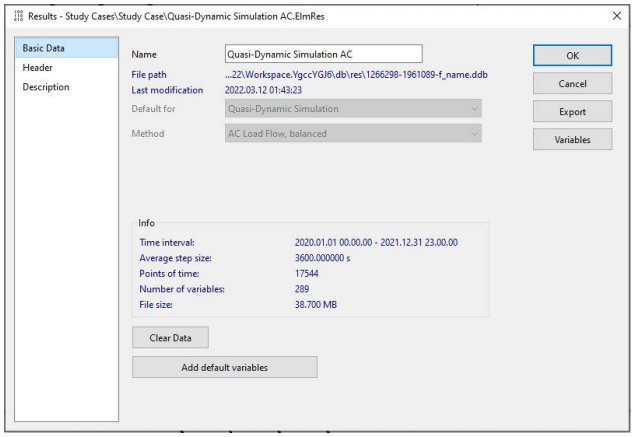


Figure 41. Exporting Quasi-Dynamic analysis results – Basic Data.

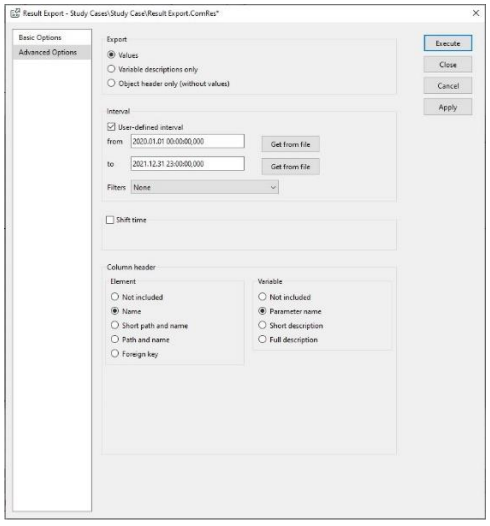


Figure 42. Exporting Quasi-Dynamic analysis results – Advanced Options.

4.2. Modelling feeder Sub-Urb in PowerFactory

The 20 [kV] feeder Sub-Urb is based on the standard IEEE 33 bus network. The topology and the loads come directly from IEEE, while line's electric parameters, line lengths, transformers etc., are modelled carefully without referring any previous networks.

Feeder Sub-Urb has 33 busbars, 33 loads, 13 PV systems and 46 transformers, of which 13 transformers are equipped with E-OLTC, because they are the ones connected to the PV systems.

The full topology of feeder Sub-Urb is shown in Figure 44.
Balancing the power flows of the feeder will be done by the External Grid connected at Busbar 1 which is modelled as a Slack busbar with voltage setpoint at 1.0 [p.u]. However, the voltage setpoint at the slack bus could be different from 1.0 [p.u], in these cases it usually tends to be higher in longitudinal feeders with significant voltage drops throughout the lines.

A voltage setpoint of 1.05 [p.u] is a common choice in certain cases. That is made possible by the HV/MV transformers in the substation. More specifically, utilizing tap changers that are usually installed on the HV side in order to decrease losses, although in terms of insulation it is worse.

Here is a zoomed-in portion of Sub-Urb Feeder first.

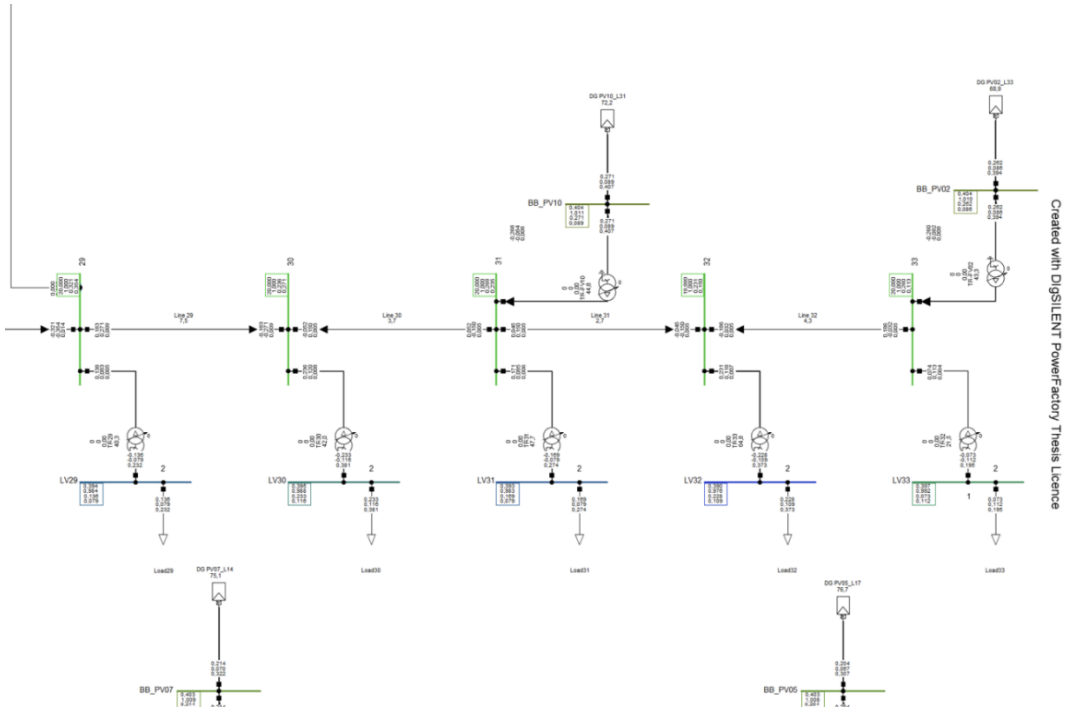


Figure 43. Feeder Sub-Urb 20 [kV] zoomed in a portion.

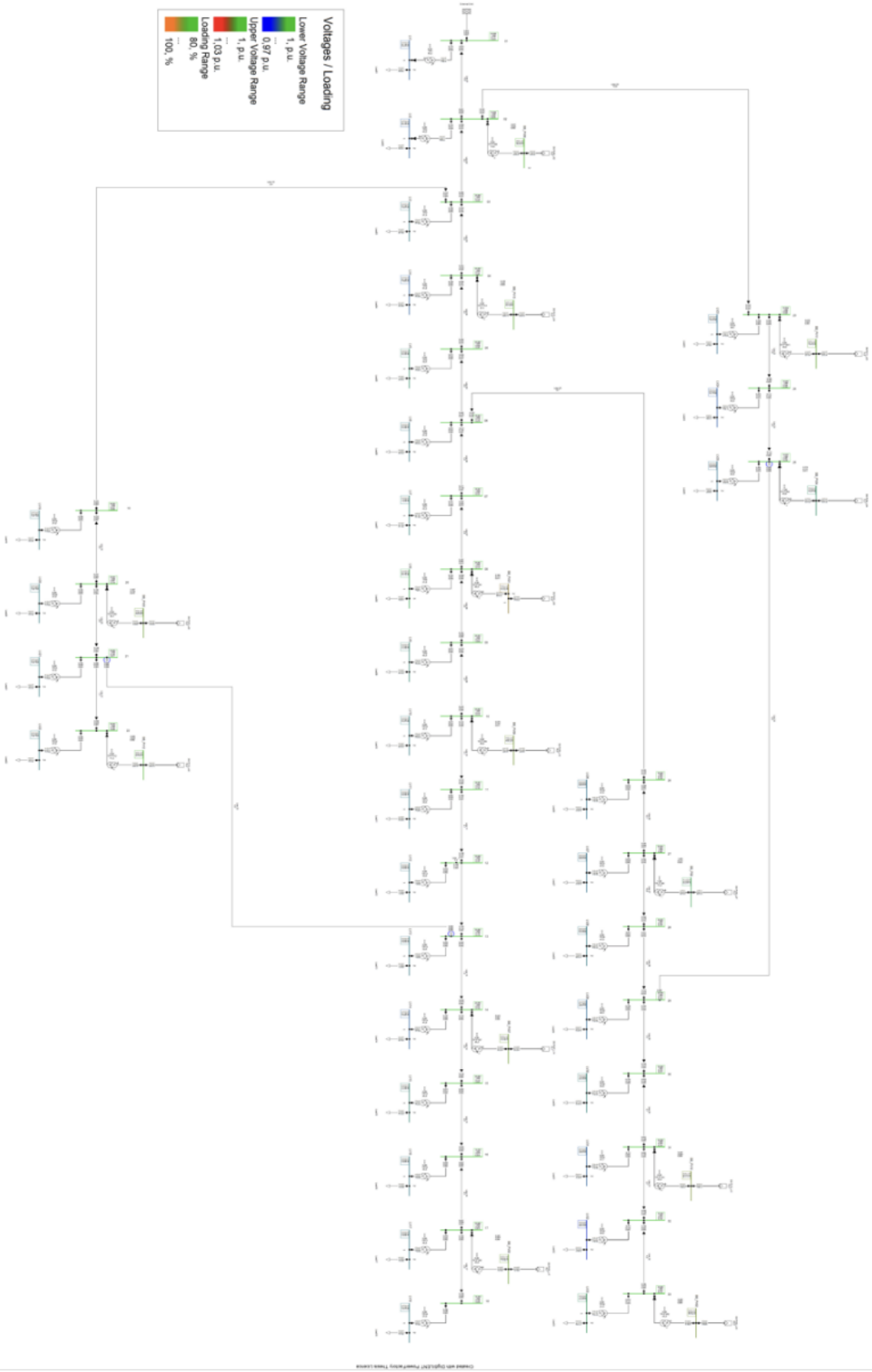


Figure 44. Feeder Sub-Urb 20 [kV] topology.

As Figure 43 shows, PV systems inject on LV side which is 400 [V].
In the LV busbar where energy is injected, there are 0.4/20 [kV]/[kV] transformers. This way, the energy is injected into the MV grid, may cause overvoltages in the MV side busbar but also potentially on the neighboring busbars as well.

Expectingly, with the use of E-OLTCs, the transformers will be able to change their tap position and lower the voltage on the MV side, avoiding overvoltages.
Important feeder data such as PVs, transformer, lines, and loads, will be given in the tables below.

Name	Act. Power [kW]	Pow.Fact.	Efficiency [%]	Tilt Angle [deg]	Inom [kA]	Low.V limit [p.u]	Up.V limit [p.u]
DG PV08_L27	1450	0,95	98,4	44	2,165	0,9	1,1
DG PV09_L10	790	0,95	98,4	44	1,155	0,9	1,1
DG PV03_L20	666	0,95	98,4	44	1,01	0,9	1,1
DG PV01_L08	500	0,95	98,4	44	0,866	0,9	1,1
DG PV04_L24	511	0,95	98,4	44	0,794	0,9	1,1
DG PV02_L33	355	0,95	98,4	44	0,577	0,9	1,1
DG PV10_L31	367	0,95	98,4	44	0,57	0,9	1,1
DG PV07_L14	290	0,95	98,4	44	0,433	0,9	1,1
DG PV05_L17	276	0,95	98,4	44	0,404	0,9	1,1
DG PV11_L23	235	0,95	98,4	44	0,354	0,9	1,1
DG PV13_L04	172	0,95	98,4	44	0,253	0,9	1,1
DG PV06_L02	106	0,95	98,4	44	0,159	0,9	1,1
DG PV12_L22	32	0,95	98,4	44	0,13	0,9	1,1

Table 3. Photovoltaic systems installed in Feeder Sub-Urb.

In Table 3, relevant data of PVs installed in the feeder are given.
It shows that each PV has a different installed power which also means they will have a different time characteristic.

Load Name	Act.Pow.[kW]	React.Pow.[kVAr]	I [kA]	Pow.Fact.
Load01	200	120	0,337	0,857
Load02	200	120	0,337	0,857
Load03	180	80	0,284	0,914
Load04	240	160	0,416	0,832
Load05	120	60	0,194	0,894
Load06	120	40	0,183	0,949
Load07	400	200	0,645	0,894
Load08	200	100	0,323	0,894
Load09	120	20	0,176	0,986
Load10	120	40	0,183	0,949
Load11	90	60	0,156	0,832

Load12	120	70	0,201	0,864
Load13	120	70	0,201	0,864
Load14	240	160	0,416	0,832
Load15	120	20	0,176	0,986
Load16	120	40	0,183	0,949
Load17	120	40	0,183	0,949
Load18	180	80	0,284	0,914
Load19	180	80	0,284	0,914
Load20	180	80	0,284	0,914
Load21	180	80	0,284	0,914
Load22	180	80	0,284	0,914
Load23	180	100	0,297	0,874
Load24	840	400	1,343	0,903
Load25	840	400	1,343	0,903
Load26	120	50	0,188	0,923
Load27	120	50	0,188	0,923
Load28	120	40	0,183	0,949
Load29	240	140	0,401	0,864
Load30	400	200	0,645	0,894
Load31	300	140	0,478	0,906
Load32	420	200	0,671	0,903
Load33	120	80	0,208	0,832

Table 4. Loads of Sub-Urb feeder 20 [kV]

Table 4 shows the loads of the feeder, their active power and reactive power, nominal current, and power factor.

Name	Type	Length [km]	Inom [kA]	X' [Ω/km]	R'(AC,20°C) [Ω/km]
Line 01	AL/FE_70/12	0,0412	0,235	0,3960	0,4430
Line 02	AL/FE_70/12	0,0412	0,235	0,3960	0,4430
Line 03	AL/FE_70/12	0,0412	0,235	0,3960	0,4430
Line 04	AL/FE_70/12	0,0412	0,235	0,3960	0,4430
Line 05	AL/FE_70/12	0,0528	0,235	0,3960	0,4430
Line 06	AL/FE_70/12	0,1320	0,235	0,3960	0,4430
Line 07	AL/FE_35/6	0,0528	0,145	0,4210	0,8200
Line 08	AL/FE_50/8	0,0412	0,170	0,4090	0,5900
Line 09	AL/FE_50/8	0,0198	0,170	0,4090	0,5900
Line 10	AL/FE_50/8	0,1320	0,170	0,4090	0,5900
Line 11	AL/FE_50/8	0,0528	0,170	0,4090	0,5900
Line 12	AL/FE_35/6	0,0412	0,145	0,4210	0,8200
Line 13	AL/FE_50/8	0,1320	0,170	0,4090	0,5900
Line 14	AL/FE_25/4	0,0198	0,125	0,4780	1,2000
Line 15	AL/FE_50/8	0,0528	0,170	0,4090	0,5900
Line 16	AL/FE_50/8	0,1320	0,170	0,4090	0,5900

Line 17	AL/FE_50/8	0,0528	0,170	0,4090	0,5900
Line 18	AL/FE_50/8	0,3630	0,170	0,4090	0,5900
Line 19	AL/FE_50/8	0,0198	0,170	0,4090	0,5900
Line 20	AL/FE_50/8	0,0198	0,170	0,4090	0,5900
Line 21	AL/FE_25/4	0,0198	0,125	0,4780	1,2000
Line 22	AL/FE_50/8	0,2310	0,170	0,4090	0,5900
Line 23	AL/FE_50/8	0,1155	0,170	0,4090	0,5900
Line 24	AL/FE_35/6	0,0412	0,145	0,4210	0,8200
Line 25	AL/FE_70/12	0,2310	0,235	0,3960	0,4430
Line 26	AL/FE_35/6	0,0528	0,145	0,4210	0,8200
Line 27	AL/FE_35/6	0,0412	0,145	0,4210	0,8200
Line 28	AL/FE_35/6	0,1320	0,145	0,4210	0,8200
Line 29	AL/FE_25/4	0,0198	0,125	0,4780	1,2000
Line 30	AL/FE_25/4	0,0198	0,125	0,4780	1,2000
Line 31	AL/FE_50/8	0,1155	0,170	0,4090	0,5900
Line 32	AL/FE_25/4	0,0248	0,125	0,4780	1,2000
Line 33	AL/FE_35/6	0,3630	0,145	0,4210	0,8200
Line 34	AL/FE_70/12	0,2310	0,235	0,3960	0,4430

Table 5. Feeder Sub-Urb 20 [kV] Lines data.

In Table 5 we see the data for lines. They all are of type AL/FE (Aluminum Conductor Steel Reinforced - ACSR), but of different cross sections.
Their resistance and reactance are given as well.
In the next table we will see the data of transformers used in this feeder.

Name	Auto.Tap	Sn [MVA]	Shc Volt. [%]	Cop.Los .[kW]	Ph.Shift [*30deg]	HV-rtd.Volt.[kV]	LV-Rtd.Volt.[kV]
TR-PV01	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV11	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV02	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV03	Yes	1,600	5,0	17,9	5	20,0	0,4
TR-PV04	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV05	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV06	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV07	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV08	Yes	1,600	5,0	17,9	5	20,0	0,4
TR-PV09	Yes	1,600	5,0	17,9	5	20,0	0,4
TR-PV10	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV12	Yes	0,630	4,0	8,4	5	20,0	0,4
TR-PV13	Yes	0,630	4,0	8,4	5	20,0	0,4
TR1	No	0,315	6,0	4,0	7	20,0	0,4
TR10	No	0,160	4,0	2,8	5	20,0	0,4
TR11	No	0,160	4,0	3,1	5	20,0	0,4

TR12	No	0,250	6,0	3,7	5	20,0	0,4
TR13	No	0,250	6,0	3,7	5	20,0	0,4
TR14	No	0,400	6,0	4,8	5	20,0	0,4
TR15	No	0,160	4,0	3,1	5	20,0	0,4
TR16	No	0,160	4,0	3,1	5	20,0	0,4
TR17	No	0,160	4,0	3,1	5	20,0	0,4
TR18	No	0,315	6,0	4,0	7	20,0	0,4
TR19	No	0,315	6,0	4,0	7	20,0	0,4
TR2	No	0,315	6,0	4,0	7	20,0	0,4
TR20	No	0,315	6,0	4,0	7	20,0	0,4
TR21	No	0,315	6,0	4,0	7	20,0	0,4
TR22	No	0,315	6,0	4,0	7	20,0	0,4
TR23	No	0,315	6,0	4,0	7	20,0	0,4
TR24	No	1,000	6,0	9,6	1	20,0	0,4
TR25	No	1,250	6,0	11,0	5	20,0	0,4
TR26	No	0,160	4,0	3,1	5	20,0	0,4
TR27	No	0,160	4,0	3,1	5	20,0	0,4
TR28	No	0,160	4,0	3,1	5	20,0	0,4
TR29	No	0,400	6,0	4,8	5	20,0	0,4
TR3	No	0,315	6,0	4,0	7	20,0	0,4
TR30	No	0,630	4,0	8,4	5	20,0	0,4
TR31	No	0,400	6,0	4,8	5	20,0	0,4
TR32	No	0,630	4,0	8,4	5	20,0	0,4
TR33	No	0,400	6,0	4,8	5	20,0	0,4
TR4	No	0,400	6,0	4,8	5	20,0	0,4
TR5	No	0,250	4,3	3,6	5	20,0	0,4
TR6	No	0,160	4,0	3,1	5	20,0	0,4
TR7	No	0,630	4,0	8,4	5	20,0	0,4
TR8	No	0,630	4,0	8,4	5	20,0	0,4
TR9	No	0,160	4,0	3,1	5	20,0	0,4

Table 6. Feeder Sub-Urb 20 [kV] Transformer data.

In Table 6 we have the transformer data of our feeder. The names of the transformers in the feeder tell whether that transformer is connected to a PV system or it is just an ordinary transformer.
As far we are concerned, the second column is the most important one because it shows whether that transformer is equipped with an E-OLTC or not. As expected, the transformers connected to PV systems are, while the ordinary transformers are not equipped with automatic tap changers.

Now that the feeder is modelled, we can move on to the next step which is creating the characteristics of loads and PVs.

4.3. Generating, extracting, and preparing the data

Feeder Sub-Urb has a peak PV production of 8.28 [MVA] while the peak aggregate load is 5.95 [MVA].
This study-case contains around 2,978,400 data values in total!

Since our PowerFactory model is complete, we will start to prepare the data.
The tasks that must be completed at this phase will be dealt with one by one in this subchapter. All works done here were using Python and PowerFactory almost exclusively.

4.3.1. Creating time characteristics for PVs

To obtain the correct outputs for the photovoltaic plants of this feeder, real irradiation data will be used.
The European Commission has developed a tool called PVGIS which gives access to their irradiation databases. [2] There are three databases in total. In our case, the chosen irradiation database is PVGIS-SARAH2, as it is also the system's default one.

Of course, not all PVs are in the same pinpoint, however, they are close enough between each other for us to consider that they are subject to the same irradiation.
The interface of PVGIS tool is shown below in Figure 45.

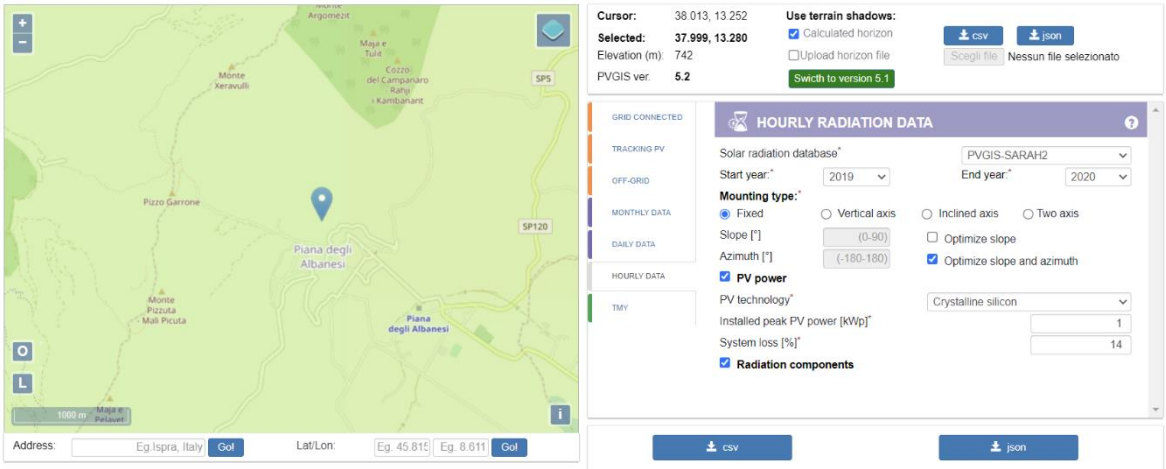


Figure 45. EU Commission's PVGIS tool – Sub-Urb feeder irradiation.

As seen from the map, feeder Sub-Urb is based in Piana degli Albanesi in Sicily. From the website's interface, we first regulate the settings according to our study then download the irradiation data.

[2] https://re.jrc.ec.europa.eu/pvg_tools/en/

The Mounting Type chosen is Fixed, as it is the cheapest and most common one. That means the panels will be fixed in a position determined by a slope, which is automatically optimized by the system using the given location's coordinates. The PV technology is Crystalline Silicon as it is the most common one. Its system losses are left to default. The database offers information on Direct Normal Irradiance data (DNI), Diffuse Horizontal Irradiation (DHI), and Global Horizontal Irradiance (GHI), however, it also offers data directly in terms of power, measured in kilowatt, which is much easier to work with.

As for the installed peak PV power [kWp], it is purposely left at 1 [kWp].

Since we have 13 PVs in Feeder Sub-Urb, each with a different installed peak power, it would be inefficient to download and then compile 13 sets of data. Because our PVs get the same sunshine at all times, we can use normalization to build a '0 to 1 scaled' irradiation characteristic for the entire period.

This the normalized PV characteristic that shows the irradiation represents a vector. Let's call this vector **Ipv** since it represents irradiation.

$$\mathbf{Ipv} = (0, 0, 0, \dots, 0.89, 0.91, 0.93, 0.88, \dots)_{1 \times 17520}$$

Ipv contains hourly values of active power outputs measured in [kW], for an arbitrary PV system that has 1 [kWp] installed power.

Now, we create another vector of our PVs installed peak power, using data from Table 3, column of Active Power [kW]. Let's call this vector **Ppv**.

$$\mathbf{Ppv} = (1450, 790, .., 32)_{1x13}$$

Ppv contains the active powers [kW] of each PV system in our feeder, 13 in total.
If we multiply vectors **Ipv** and **Ppv**, we will get a matrix of size 17520×13, where each column will represent the active power output of their respective PV system, and the rows would represent the time-steps.

$$\mathbf{Po} = (\mathbf{Ipv}' \times \mathbf{Ppv})_{17520x13}$$

Matrix **Po** has the characteristics for each PV system. It suffices to save this matrix and then access its relevant columns. Note that **Ipv** is transposed as the inner dimensions of two vectors being multiplied must match. The columns can be sorted by name.

	PV1	PV2	PV3	PV4	PV5	PV6	PV7	PV8	PV9	PV10	PV11	PV12	PV13
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0

...	...	...	...	...	...	...	...	...	...	...	...	...	...
17515	0	0	0	0	0	0	0	0	0	0	0	0	0
17516	0	0	0	0	0	0	0	0	0	0	0	0	0
17517	0	0	0	0	0	0	0	0	0	0	0	0	0
17518	0	0	0	0	0	0	0	0	0	0	0	0	0
17519	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7. PV characteristics - Sub-Urb Feeder.

At first glance, matrix **Po** in Table 7 can look wrong with all these zeros.
In fact, that is not the case. It just so happens that the hours seen in the table are night-time hours, meaning there is no irradiation, consequently, no PV production. To avoid confusion, Table 8 shows **Po** values of some random hours of our 2-year period.

	PV1	PV2	PV3	PV4	PV5	PV6	PV7	PV8	PV9	PV10	PV11	PV12	PV13
4440	0	0	0	0	0	0	0	0	0	0	0	0	0
4441	0	0	0	0	0	0	0	0	0	0	0	0	0
4442	0	0	0	0	0	0	0	0	0	0	0	0	0
4443	0	0	0	0	0	0	0	0	0	0	0	0	0
4444	3	2	4	3	2	1	2	9	5	2	1	1	1
4445	21	15	28	21	11	4	12	60	33	15	10	4	7
4446	79	56	105	80	43	17	46	228	124	58	37	14	27
4447	168	119	223	171	93	36	97	486	265	123	79	30	58
4448	246	175	328	252	136	52	143	714	389	181	116	44	85
4449	309	219	411	316	170	65	179	895	488	227	145	56	106
4450	343	244	457	351	189	73	199	995	542	252	161	62	118
4451	358	254	476	365	197	76	207	1037	565	262	168	64	123
4452	348	247	464	356	192	74	202	1010	550	256	164	63	120
4453	326	231	434	333	180	69	189	945	515	239	153	59	112
4454	280	199	373	286	154	59	162	811	442	205	131	50	96
4455	212	151	283	217	117	45	123	615	335	156	100	38	73
4456	126	89	167	128	69	27	73	364	199	92	59	23	43
4457	44	31	58	45	24	9	25	126	69	32	20	8	15
4458	14	10	19	14	8	3	8	41	22	10	7	3	5
4459	0	0	0	0	0	0	0	0	0	0	0	0	0
4460	0	0	0	0	0	0	0	0	0	0	0	0	0
4461	0	0	0	0	0	0	0	0	0	0	0	0	0
4462	0	0	0	0	0	0	0	0	0	0	0	0	0
4463	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8. PV outputs of a random period - Sub-Urb feeder.

In this table, a quick review of any of the columns indicates a power production curve that is in line with usual PV output in a full day.

After **Po** is saved to '.csv', the characteristic of any PV plant can be directly accessed from PowerFactory using *Po.csv* file with the relevant column as an address.
Another approach is to use Pandas library of Python and save each column of this

dataframe as an individual '.csv' file, then access those files from PowerFactory without needing to specify the column.
One such PV characteristic as the ones created is given in Figure 46.

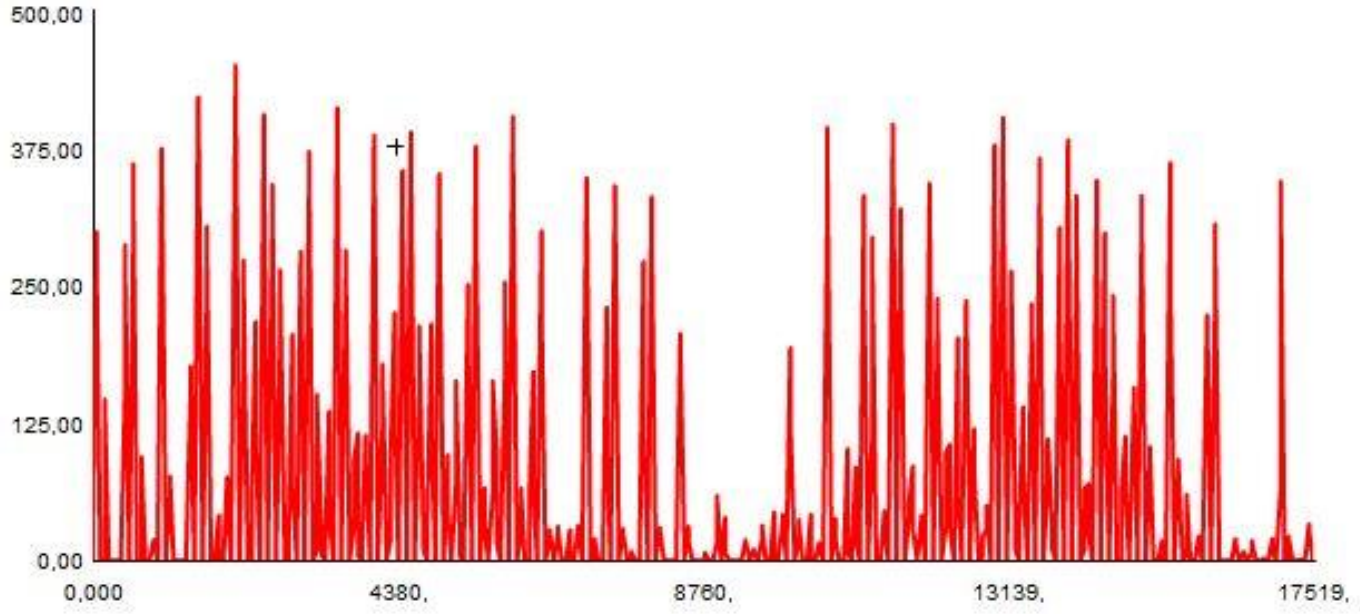


Figure 46. PV01 Active power output characteristic [kW].

In this case, the characteristic of PV1 was extracted from column 1 of **Po**.
Same goes for all PVs in the feeder which will be automatically used in the Quasi-Dynamic Simulation, a step for each load flow it runs.

4.3.2. Creating time characteristics for loads

The methods of creating time characteristics of loads do not differ too much in principle from the methods of creating time characteristics of PV outputs.
At this point, the only data we have are maximum values of loads (yearly peaks).

Obtaining several real load profiles from distribution companies is hard, therefore a little bit of improvisation is inevitable.
First, a yearly load profile of a residential consumer is secured and analyzed. If that load profile proves to be reliable and matches the type of loads connected to our feeder, which in this case are households mainly, then we can use that load profile for all of our loads, again, with normalization.

The load profile is first scaled from 0 to 1, so it becomes a *unit load profile*. If the vector of that load profile is of 2 years, even better. Let's call that vector of raw load profile data **Lr.**

$$\mathbf{Lr} = (L_{h1}, L_{h2}, L_{h3}, \dots, L_{h17520})_{1x17520}$$

Lr has 17520 values, each representing the load at a certain hour. If the highest values of **Lr** is Lrmax, then then the normalized vector, let's call it **Ln**, will be an elementwise division of **Lr** to Lrmax.

$$\mathbf{Ln} = \mathbf{Lr}/\text{Lrmax}$$

Ln is unit load profile as it is **Lr** normalized, that means its values will range from 0 to 1, and the matrix's dimensions are unchanged. **Ln** gives us an appropriate base to build load profiles for all other loads in the feeder. It is important to remember that we already have the peak values of each load.

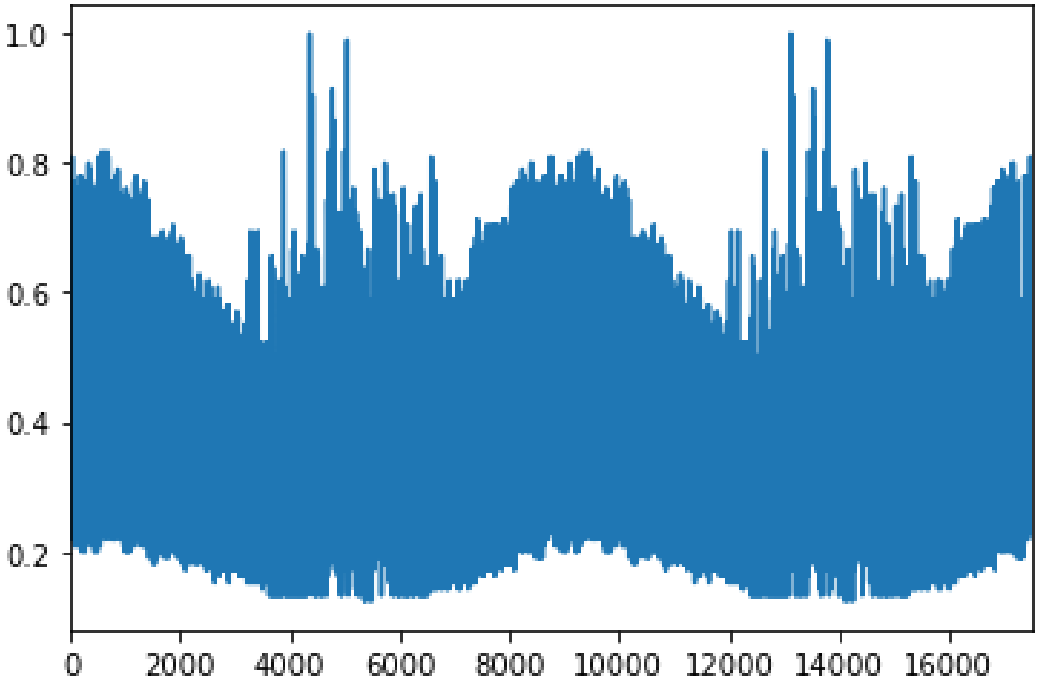


Figure 47. Unit Load Profile Ln.

Figure 47 shows the characteristic of the normalized or unit load profile, which is scaled from 0 to 1. Practically speaking, this will be the characteristic for all the loads, only the magnitude will change.
Let's build a new vector and call it **Lp**. **Lp** consists of peak values of loads in the feeder.

$$\mathbf{Lp} = (L_{p1}, L_{p2}, L_{p3}, \dots, L_{p33})_{1x33}$$

Lp contains the peak values of 33 loads of Sub-Urb Feeder.
If we now multiply the normalized load profiles vector **Ln**, with the vector of peaks **Lp**, we will get the matrix of all load characteristics of the feeder. Let's call this matrix **L.**

$$\mathbf{L} = (\mathbf{Ln}' \times \mathbf{Lp})_{17520x33}$$

Again, **Ln** has to be transposed to be compatible for a vectorial multiplication with **Lp**.

So far, we referred to the load but we did not specify which parameter it was really though intuitively it could have inferred Apparent Power [kVAr].
In fact, **Lp** is just for illustration as in reality our load data contains both Active Power (**Pp**) and Reactive Power (**Qp**) components. The principle does not change, it just means that the process has to be done twice for **Pp** and **Qp**, instead of just once for **Lp**.
That means, our matrix of loads **L** has in fact dimensions of 17520 × 66, where for each of the feeder's 33 loads, we have the values for active and reactive power.
We can now take a look at the final matrix of loads **L**.

	P1	P2	P3	P4	P5	P6	P7	P8	…	Q25	Q26	Q27	Q28	Q29	Q30	Q31	Q32	Q33
0	57	52	69	35	35	116	59	35	...	113	114	14	14	12	40	58	40	55
1	50	46	60	31	30	102	52	31	...	99	100	13	13	10	35	51	35	49
2	47	43	56	29	28	95	48	28	...	92	93	12	12	9	32	47	32	45
3	47	43	56	29	28	95	48	28	...	92	93	12	12	9	32	47	32	45
4	50	46	60	31	30	102	52	31	...	99	100	13	13	10	35	51	35	49
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17515	143	131	171	88	87	291	149	88	...	282	286	36	36	29	100	145	99	139
17516	136	124	163	84	83	276	141	83	...	268	271	34	34	28	95	138	94	132
17517	124	113	148	76	75	251	128	76	...	243	246	31	31	25	86	125	85	120
17518	100	92	120	62	61	204	104	61	...	197	200	25	25	20	70	101	69	97
17519	77	70	92	48	47	156	80	47	...	152	154	19	19	16	54	78	53	75

Table 9. Matrix of loads in Sub-Urb feeder

Load characteristics can be extracted in a similar fashion with last time where individual PV outputs were extracted from the big matrix, i.e., using columns. First, matrix **L** is exported in '.csv' format or any other supported format.

So, each load will have two characteristics, one for active power and one for reactive power. In PowerFactory, we must assign the relevant column of matrix **L** to the specific load and specific parameter. For example, for the active power characteristic of Load 23 connected at Busbar23, we access Load 23 settings window and refer the active power characteristic to column 23 of matrix **L**.

Alternatively, each column of **L** can be saved as an independent file and then accessed from PowerFactory's relevant Load/Parameter without having to specify the column.
One of the 66 characteristics is graphed in Figure 48.

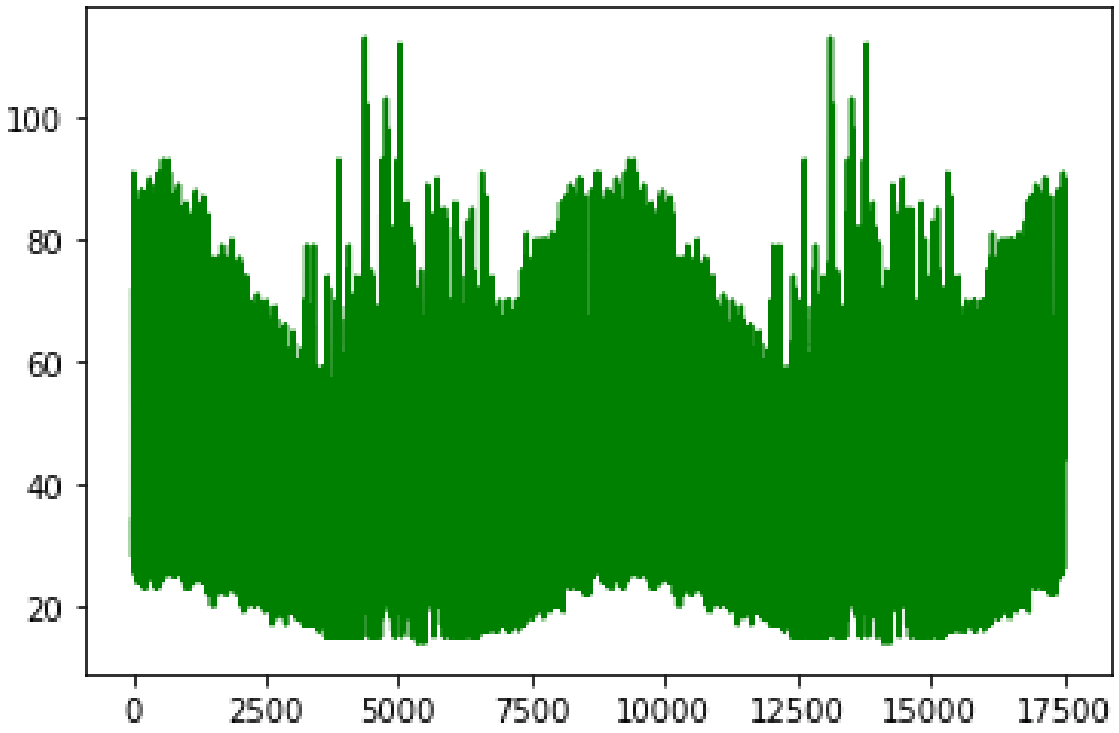


Figure 48. Active Power characteristic of Load 12 in Sub-Urb feeder [kW].

4.4. Running the Quasi-Dynamic simulation and exporting the results of Tap Positions

This step is probably the core one. All relevant prerequisites of this section were already introduced in 4.1.2. Therefore, here, the focus is on the graphical representation of parameters and the results of the Q-D simulation.

It is important to keep in mind that the Q-D simulation will be run twice, the first time without automatic tap changers and the second one with automatic tap changers. This will be done in order to compare the results between the two, which effectively would help understanding the effect of E-OLTCs in our network. A good comparison is the number of voltage constraints violations caused in each of the cases.

Building the characteristics for PVs and loads were discussed in the last two sections, but we have not seen these parameters visualized altogether as a whole, instead, here we will see the superposed curves of the loads and PVs.

Figure 49 shows the characteristics of all 13 PV systems in our network. This is none other than a graphical representation of Table 7 (matrix **Po**).

They are all different in magnitude because they installed peak power differs. However, as described at the beginning of this chapter, the irradiation is the same for all of them, as there are no vast sunshine differences in areas as small as the ones an MV feeder occupies.

A graphical representation that covers 2 whole years may have a resolution a bit too high to see clearly what is going on.
Figure 51 represents the same characteristics but it is zoomed in a random period of the year.

These curves show the PV outputs for a few days. From the graphics, one can infer that PV outputs are subject to meteorological changes such as clouds, rain etc.

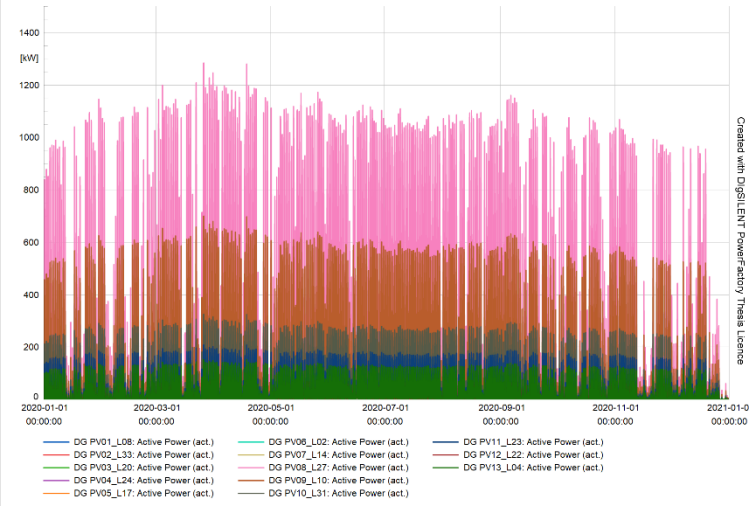


Figure 49. PV Outputs of Sub-Urb feeder (first year) [kW].

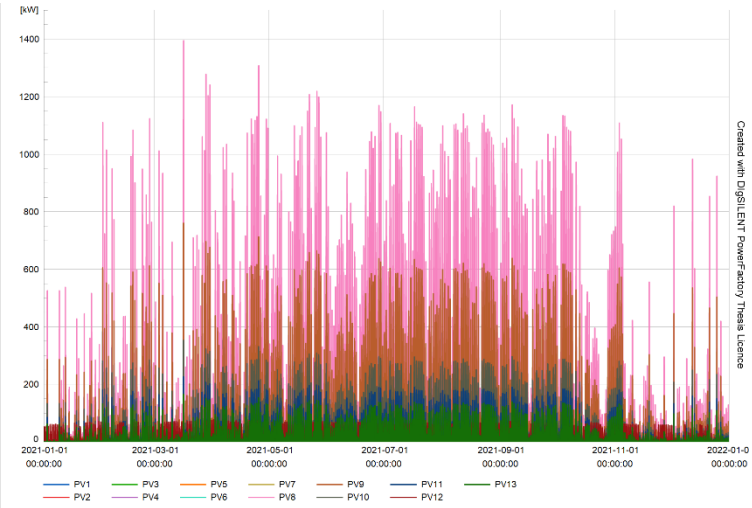


Figure 50. PV Outputs of Sub-Urb feeder (second year) [kW].

Since all PVs have the same characteristic, some of the PV outputs may not be visible in the graph, as they hide behind each other.

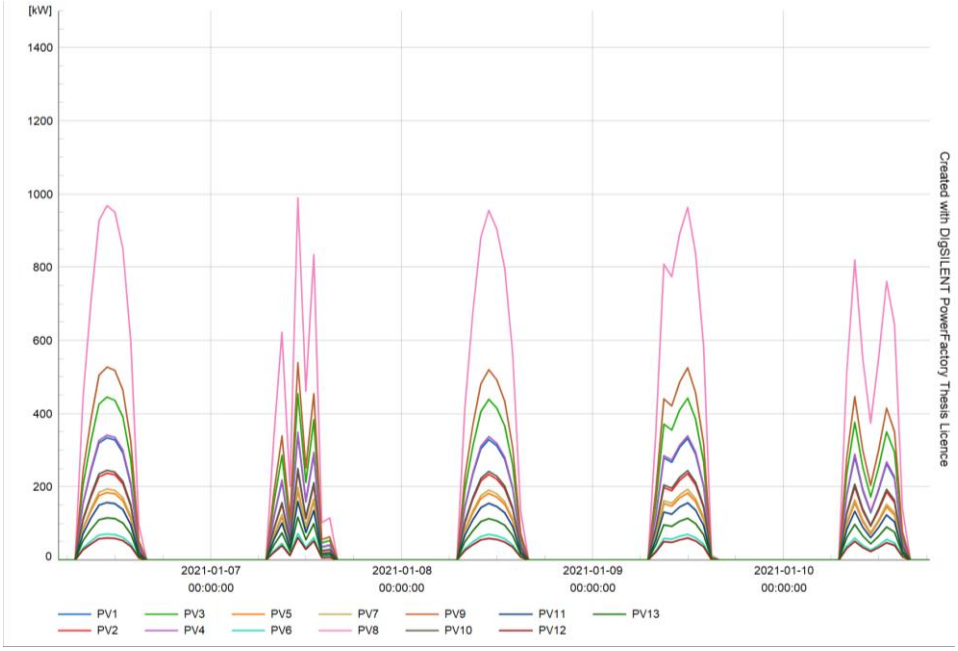


Figure 51. PV Output Sub-Urb feeder of random 5 days [kW].

Next, let's see the feeder's loads in Figure 52. As explained earlier, load characteristic does not change in the second year, it suffices that PV generation changes.

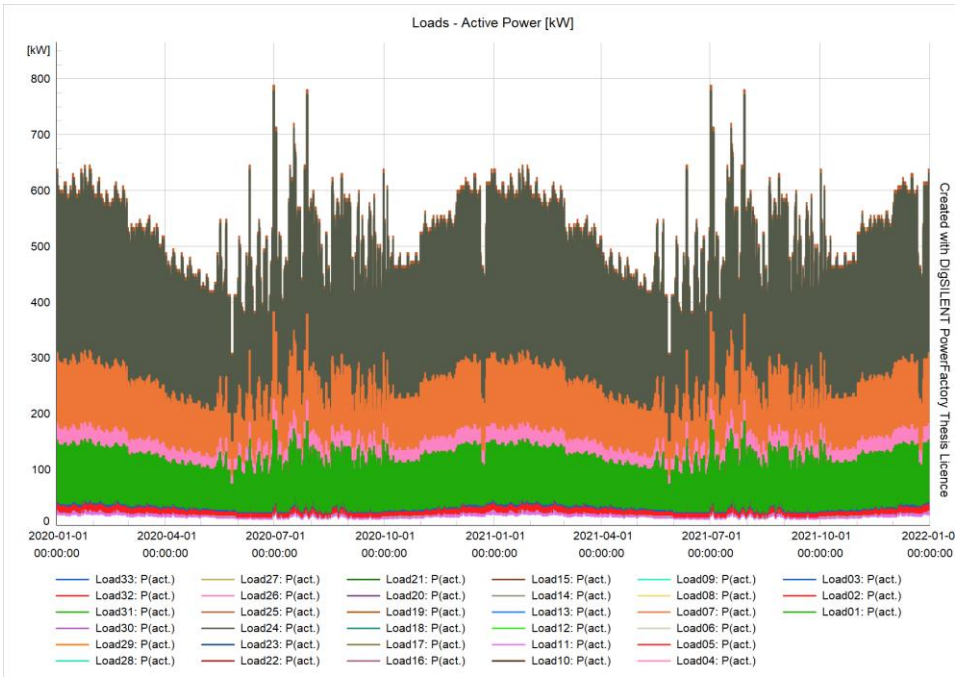


Figure 52. Loads (active power) of feeder Sub-Urb [kW].

Since matrix **L** contained both active and reactive powers of loads, this figure corresponds to the first half of Table 9 (matrix **L**). Figure 53 shows the same characteristic, zoomed into a short period of the simulation. The hourly steps are visible here.

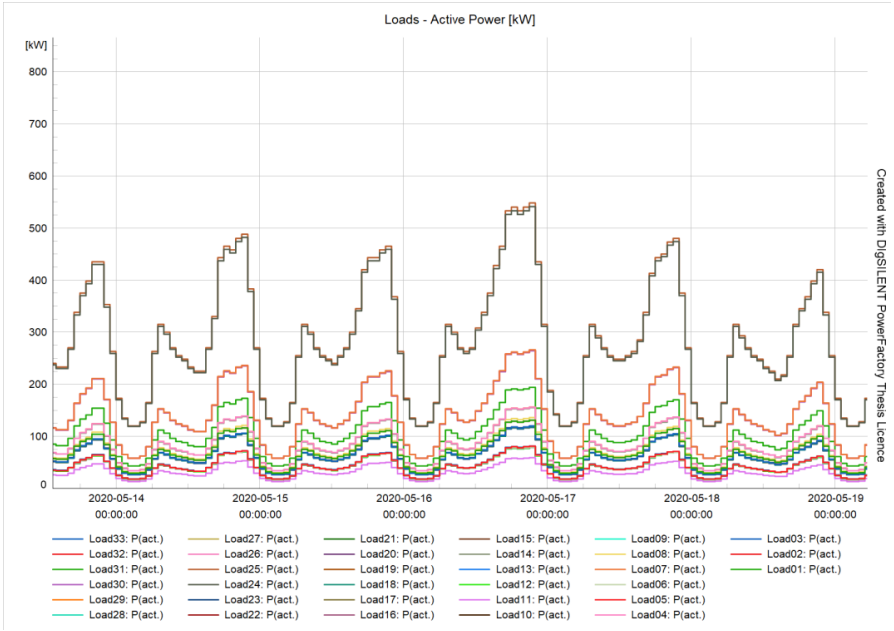


Figure 53. Loads (active power) in Sub-Urb feeder, zoomed in [kW].

Let's see both these load graphs again, this time for the reactive power instead. These characteristics correspond to the second half of Table 9. Matrix of loads in Sub-Urb feeder (matrix **L**).

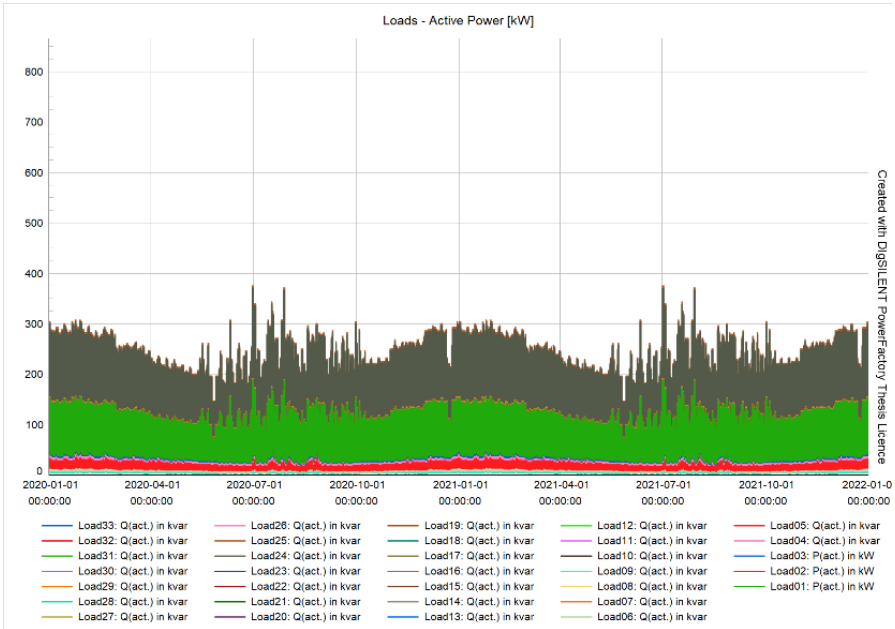


Figure 54. Loads (reactive) of Sub-Urb feeder [kVAr].

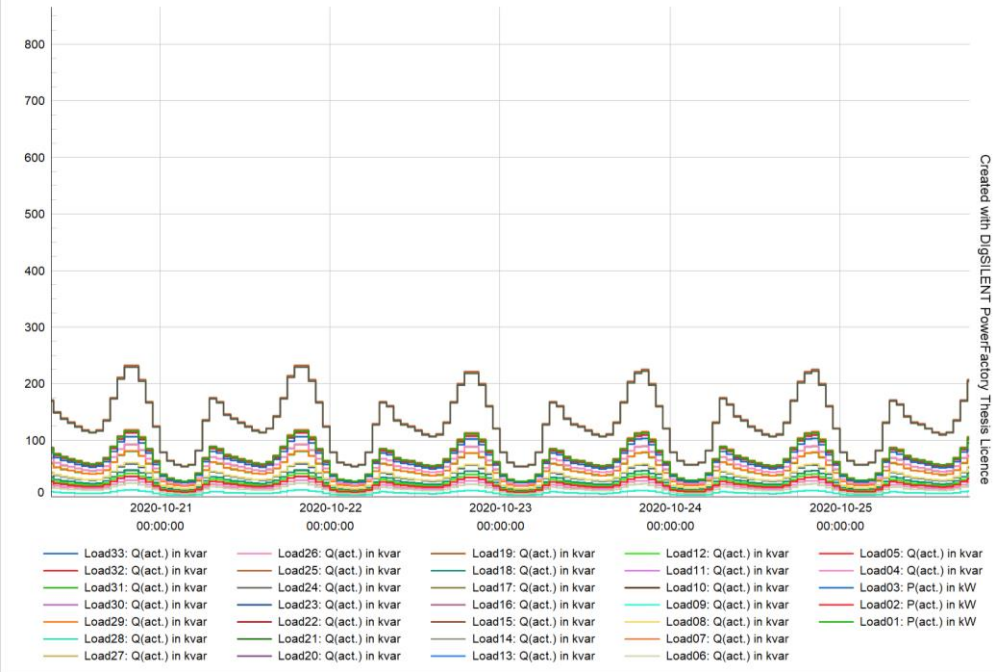


Figure 55. Loads (reactive) of Sub-Urb feeder, zoomed, [kVAr].

Not many differences, although each load has their specific power factor.

Characteristics of PV outputs and loads are the same regardless of whether the automatic tap changer is employed or not. As planned, from here we proceed with running two Q-D simulations, first without E-OLTCs then with E-OLTCs.

First let's take a look at the Q-D Simulation without employing the automatic tap changers (ATP).

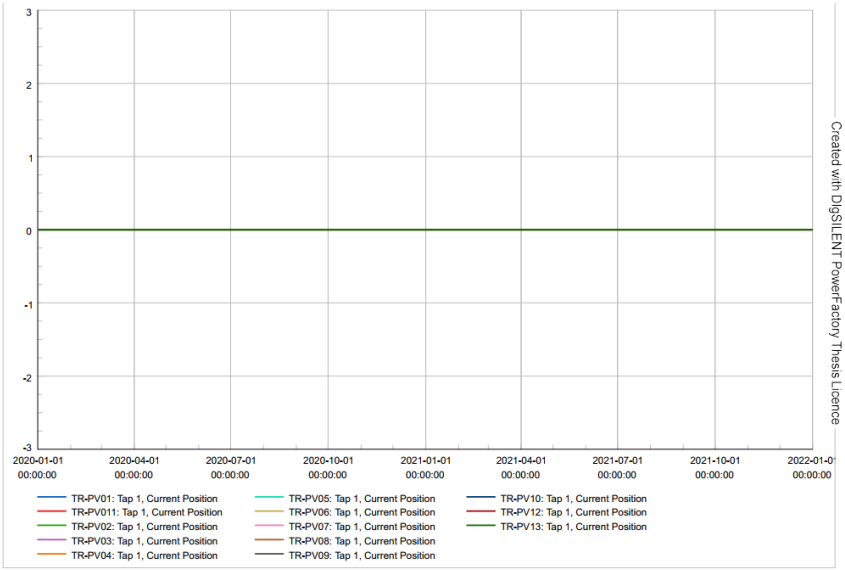


Figure 56. Q-D simulation without Automatic Tap changers - Tap positions

Since ATP is off, of course that taps of transformers will remain at their neutral position (0) for the whole duration of the analysis.

The simulation is done as explained, now the results of the simulation must be exported to a suitable format. After exporting the results from PowerFactory to a file named 'Sub-Urb_noEOLTC.csv', we will use Pandas library of Python to work on the saved data.

```
import pandas as pd

SU_raw_noEOLTC = pd.read_csv('QD_Sub-Urb_noEOLTC.csv', delimiter=';')
```

This file contains all sorts of parameters for all elements of the feeder, however, since this is our Q-D analysis without ATP, the only service we need here is to extract the number of voltage violations.

Therefore, we will delete the rest of the data, and only keep the 13 columns that show the voltage results at the busbars where PVs are connected. That is because we are considering that busbars connected directly where the energy is injected will be the most critical ones, and if they do not suffer overvoltages, the rest of the network can be considered as good to go.

```
SU_voltages_noEOLTC = SU_raw_noEOLTC.iloc[:,:-13]
```

We have now created a new dataframe named 'MI_voltages_noEOLTC' that has copied the last 13 columns of the previous dataframe.

Now all there is left is to count the over-voltage constraint violations. Under-voltages are not considered as they are not the problem when dealing with high DG penetration. The violations will be divided in two groups, violations over 5% and violations over 10%.

```
SU_noEOLTC_violations_5percent = (SU_voltages_noEOLTC>1.05).sum().sum()
print(SU_noEOLTC_violations_5percent)
21924

SU_noEOLTC_violations_10percent = (SU_voltages_noEOLTC>1.1).sum().sum()
print(SU_noEOLTC_violations_10percent)
0
```

Always according to ENTSO-E [8], Sub-Urb feeder experiences 21924 over-voltage violations in a 2-year timespan. All of them are between 5% and 10% over the nominal voltage, so 1.05 [p.u] to 1.1 [p.u]. On the other hand, Sub-Urb feeder has no

overvoltages that go beyond 10% of the nominal voltage, which is quite optimal already.

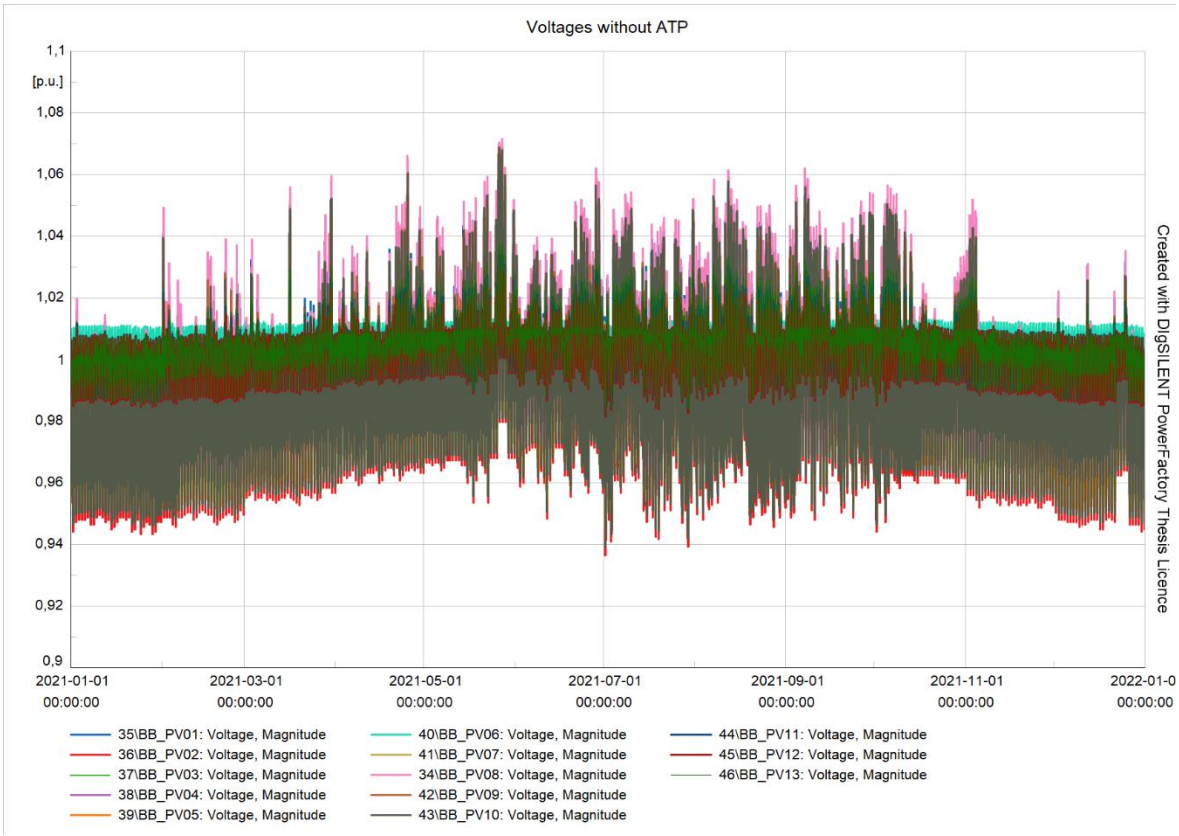


Figure 57. Voltages in Sub-Urb feeder without ATPs [p.u].

As Figure 57 shows, Sub-Urb feeder performs quite well although not perfect. That is all as far as Q-D analysis is concerned about the no ATP case.

Next comes the Q-D analysis with ATP, after which, the numbers of violations between the two cases will be compared.

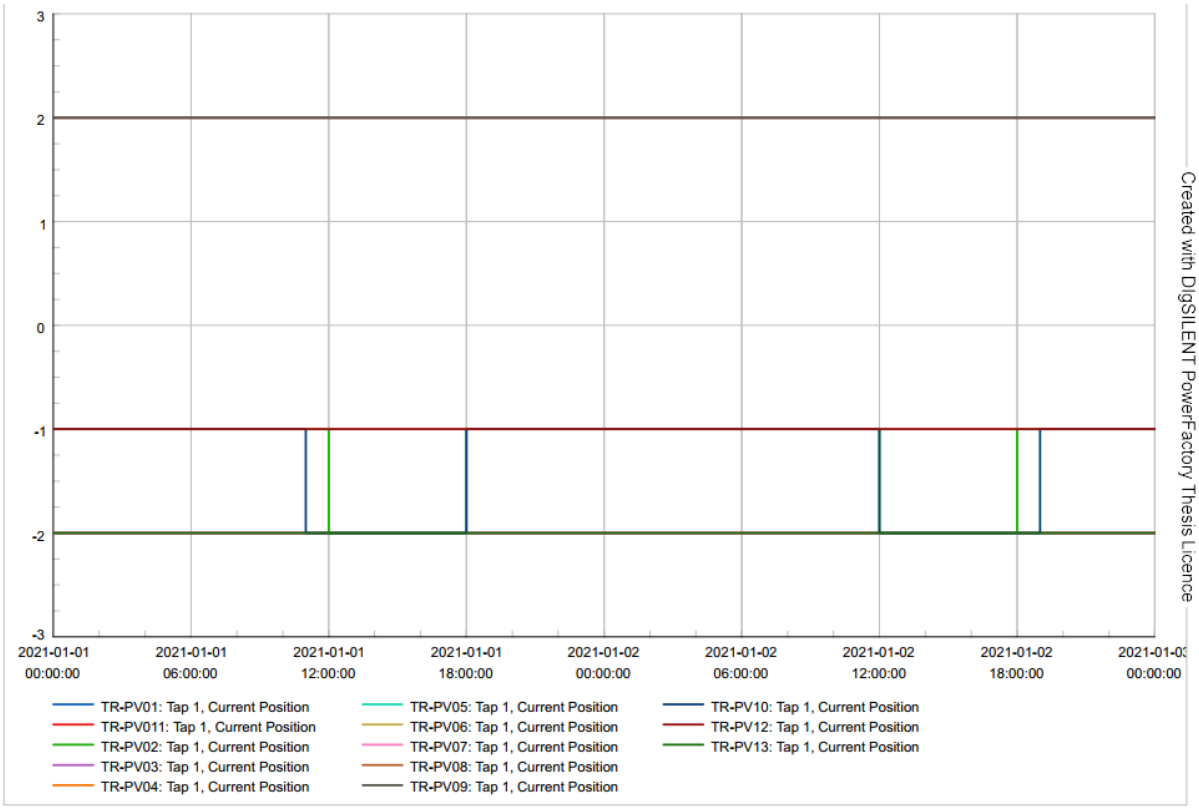


Figure 58. Q-D Analysis with ATP – Tap Positions.

Differently from Figure 56 where ATPs were off, Figure 58 shows that E-OLTCs are working as the state of the network changes. Since the chosen period is January, there aren't many jumps between tap positions.

This graphic is zoomed in to get a clearer view of what happens. For each step of the simulation, the tap position of each transformer will be saved to be later used in the ML algorithm.

Next we analyze the voltages in all busbars where PVs are connected.

Busbar names are correlated to the PV systems that are connected to them. As we see, busbar 1, due to its physical position in the grid, is almost always at a higher voltage than 1.0 [p.u], however, it never causes any overvoltage violations as it never reaches 1.05 [p.u].

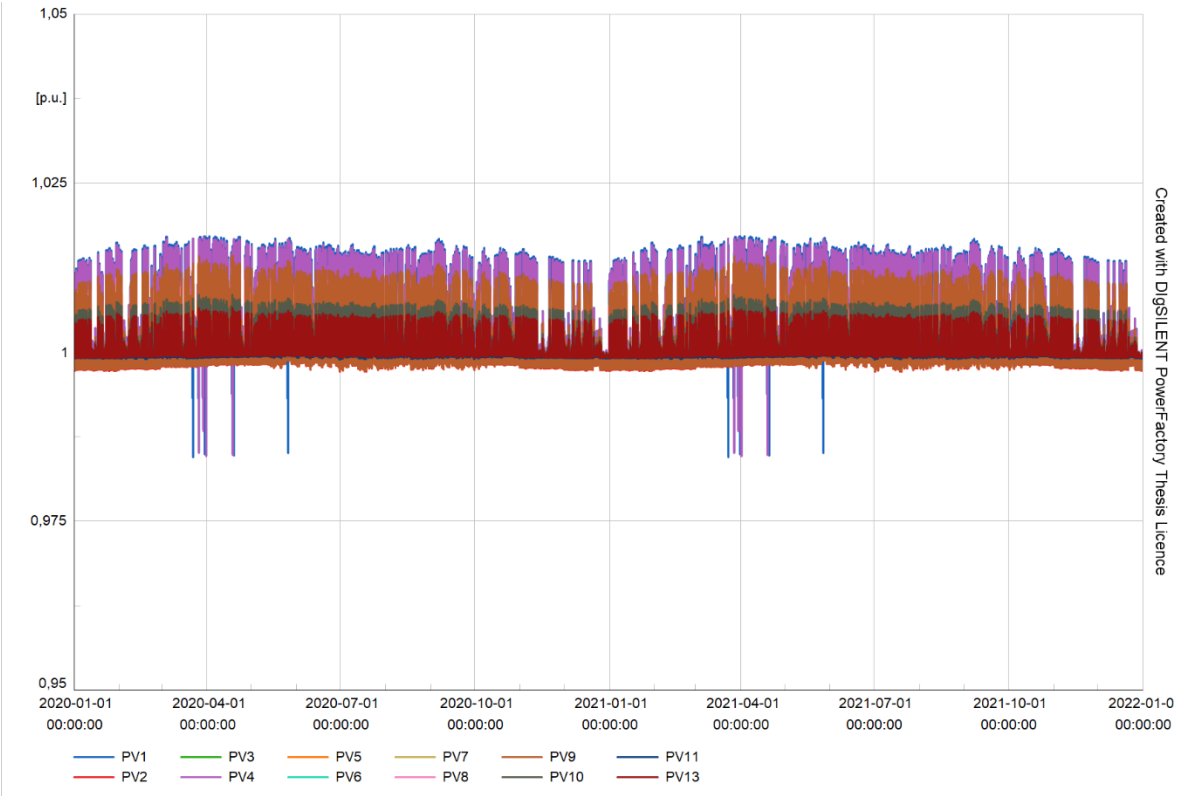


Figure 59. Voltages of feeder Sub-Urb with ATP [p.u].

As a matter of fact, Figure 59 shows that upon employing E-OLTCs, there will not be any voltage constraint violation, not even undervoltages. All the values are between 0.95 [p.u] and 1.05 [p.u]. So, although the feeder was doing quite well, it has improved even further. Since there are no under-voltages, LVRs will not be added to this feeder.

At this point, needless to say, E-OLTCs have shown great results, going from around 22 thousand instances of overvoltages, down to zero.
Similar to the previous case, we will use Pandas to count and verify the results.

```
import pandas as pd

SU_raw_withEOLTC = pd.read_csv('QD_SU_withEOLTC.csv', delimiter=';')

SU_voltages_withEOLTC = SU_raw_withEOLTC.iloc[:,:-13]

SU_withEOLTC_violations_5percent = (SU_voltages_withEOLTC>1.05).sum().sum()
print(SU_withEOLTC_violations_5percent)
0

SU_withEOLTC_violations_10percent = (SU_voltages_withEOLTC>1.1).sum().sum()
print(SU_withEOLTC_violations_10percent)
0
```

In conclusion, using E-OLTCs in Sub-Urb feeder makes 100% of voltage violations disappear. As expected, the results from the code confirm what Figure 59.

All there is left to do now is to use the exported data we got from running the Q-D simulation with ATP and continue with preparing it for the Machine Learning part.

4.5. Cleaning, re-organizing and filling the missing data points

Now that we have run the Q-D Simulation and extracted the files, we will need to prepare the files for the ML algorithm.
First, we create a dataframe from the Q-D simulation results.

```
import pandas as pd

SU_raw = pd.read_csv('QD_Sub-Urb_withEOLTC.csv', delimiter=';')
```

Let's check whether there is any missing data.

```
(SU_raw == 'NaN').sum().sum()
0
```

In Python, string 'NaN' means Not a Number, and it indicates missing data which may prove problematic later. In our dataset, there are no missing values as the results shows zero. In these cases, there is no need to fill any data. The 'sum()' method used is to first sum the number of NaN values across rows, then across columns, which gives us the total number of NaN values in the whole dataset.
The columns of our data aren't necessarily ordered as we need them, so let's sort them lexicographically.

```
SU_raw = SU_raw.reindex(sorted(SU_raw.columns), axis=1)
```

If when exporting the results we also opted to save the busbar voltages, those columns must be deleted as they are not inputs for our ML algorithm.
To reiterate, as inputs or features for the ML algorithm we will use:

- Loads (Active Power)
- Loads (Reactive Power)
- PV Outputs (Active Powers)
- PV Outputs (Reactive Powers)

While as outputs or labels for the ML algorithm we will use Tap Positions only.

As far as PV Outputs are concerned, the power factor is considered 0.95 in all cases, therefore Active and Reactive powers hold the same interrelation at all times.

A power factor of 0.95 was chosen as it is the worst case scenario and the limit allowed in many countries. In reality, modern inverters have a power factor range that goes up to 0.99.

4.6. Dividing and exporting the dataset features and labels

At this point our data is clean, complete, and only contains the columns of inputs and outputs that we will use directly in the algorithm.
It means, we will have 105 columns in total, as follows:

- 33 columns for active powers of loads
- 33 columns for reactive powers of loads
- 13 columns for PVs active power outputs
- 13 columns for PVs reactive power outputs
- 13 columns for Transformers Tap Positions

Table 10 represents the prepared dataframe of our full dataset, including both inputs and outputs. This dataframe has a size of 17520 × 105.

	PLoad01	QLoad01	…	PV01_P	PV01_Q	…	TR-PV01	…
0	57	34	…	0	0	…	-1	…
1	50	30	…	0	0	…	-1	…
2	47	28	…	0	0	…	-1	…
3	47	28	…	0	0	…	-1	…
4	50	30	…	0	0	…	-1	…
…	…	…	…	…	…	…	…	…

Table 10. Final data - inputs and outputs

Now, we have to divide the features (inputs) and labels (outputs) into separate '.csv' files. We already know that the last 13 columns are our columns of the outputs, therefore the following ensues:

```
X = SU_raw.iloc[:,:-13]
X.to_csv('X.csv')

y = SU_raw.iloc[:, -13:]
y.to_csv('y.csv')
```

However, there is one more issue. In our DNN algorithm, we will use binary classification for multiple outputs. On the outputs, we will have a neuron for each tap position of all transformers, so in this case we will have 5·13=65 neurons. This

means that we have to find a way to transform our outputs from integer values (-2, -1, 0, 1, 2) to some binary notation.

The way to do that is by assigning zeros and ones to each possible tap position. For example, if the tap position is at (-1), instead of the result being simply '-1', now it will have to be [0, 1, 0, 0, 0]. So, in a vector of 5 binaries, a one will be assigned to the correct position and zeros to the rest. This means, our outputs file 'y' will now become of size 17520×65, instead of 17520×13 as it was earlier.
This way, we will have 5 outputs for each transformer, and these outputs will be either zeros or ones. Figure 35 offers a general topology.
After completing this operation, 'y' will be overwritten.
The data are now split into inputs and outputs and saved in external files, ready to be used in the algorithm. This step concludes the long process of data preparation.

4.7. Coding the Deep Neural Network in Python

Now that we have our features file and labels file ready, we can start to build the Deep Neural Networks. In this section, the algorithm will be explained step by step. The first step always is importing the relevant libraries.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

These are the basic libraries we start with, however, more libraries will be needed later, so they will be imported during the process.
Let's continue by importing the files we have prepared earlier.

```
X = pd.read_csv('X.csv')
y = pd.read_csv('y.csv')
```

If the files we need are saved in a different folder, the directory must be adapted. Panda's 'read_csv' method uses comma as its default delimiter, however, if the files are saved on Excel for example, that must be adapted as well.

A machine learning algorithm will take examples of correct inputs and outputs and try to learn from them. However, the algorithm will have to make sure that it is not overfitting a certain dataset and will be failing new datasets. This was covered in detail in 3.1.3.

The way to avoid overfitting is by shuffling the dataset really well, i.e., randomly, and then dividing it in two parts, one for training, and the other for validation.

This means that our algorithm will train and improve itself using some data, and after it reaches an acceptable result with that data, it will test itself with new, unseen data, called testing data. In an optimal model, the results of the training set and the test set should be similar. The ratio of the testing set and the training set depends on the overall amount of training examples we have. In our case, since we have 17520 training examples, we can afford to leave only 20% of our data for the testing set. Now they will be split.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=74)
```

Here we needed to import function 'train_test_split', of module 'model_selection', from 'sklearn' library of Python. This module shuffles your dataset according to a random seed of your choice. Then splits it in a training set and a testing set. The arguments passed are the files we imported earlier, then the test size relative to the total number of training examples. In our dataset, 80% is used to train the model and 20% is used to test or to validate the model.

When we have all sorts of inputs such as powers, voltages, integers representing tap positions etc., there might be large disparity in magnitudes. For example, powers can be in the order of hundreds or thousands, while voltages in per unit are around 1, that discrepancy boggles the algorithm. To avoid that, we use scaling on the inputs as for outputs it is not needed. Yet again, there exists a high-level programming module that does it for the user, without having to code low-level functions. Let's see that next.

```
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
scaler.fit(X_train)


X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Module 'MinMaxScaler' will fit our training data only, where it will find a suitable range and scale all the data to it. The testing data is not scaled as we emulate the real-life case where we have no information about it. So, both the training and testing sets are scaled to the training scale, here called 'scaler'.

Our data is shuffled, split, and now scaled as well. Everything is ready for the Neural Network to be built.

The neural network is built as follows,

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(Dense(92, activation='relu'))
model.add(Dense(82, activation='relu'))
model.add(Dense(72, activation='relu'))

model.add(Dense(65, activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy')
```

Thanks to Tensorflow/Keras libraries, what once took quite some time to be coded, now is done in just a few lines, although this is only our primitive DNN model that will go on evolving.

Method 'Sequential' builds the Neural network, then, we add layers to it using method Dense.
Dense takes two mandatory arguments: the number of neurons and the activation function. We have discussed both in the previous chapter.
In the first layer, the number of neurons is 92, as per a rule of thumb stating that the number of neurons in the input layer should be pretty much the same as the number of total inputs. The activation function is ReLU of Figure 29.
Another empiric law is that the number of neurons should be reduced in half on every layer, but since our last layer has to have 65 neurons, we have to reduce less neurons on each layer.
The number of neurons in the output layer, however, cannot be different from the number of outputs. If they do not match, the program will indicate an error, which is the reason why we have 65 layers on our output layer. The activation function here is Sigmoid of Figure 27. Sigmoid will give high-resolution probability values from 0 to 1.

Since there are two layers in the hidden layer, this is considered a Deep Neural Network.
The final line compiles all the data of our DNN by optimizing the learning rate using an optimizer named 'adam' which is the best one, and by choosing the desired loss function, which in our case has to be 'binary_crossentropy' as we are dealing with outputs that must normally be either zero or one.
The modelling is done, now is time to train our model.

```
model.fit(x=X_train, y=y_train, epochs=250)
```

This is the most basic fitting option where the training inputs and outputs are passed as arguments together with the number of epochs. An epoch is a complete run of all inputs into the model. So, when all training examples go through the model by forward propagation, then they come back by backward propagation and update the weights and biases, that counts as an epoch. The more epochs, the more the model is trained. 250 epochs is just an arbitrary number.
For each epoch, the program returns the results of the loss function.

```
Epoch 1/250
438/438 [==============================] - 2s 2ms/step - loss: 0.0705
Epoch 2/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0300
Epoch 3/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0291
Epoch 4/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0287
Epoch 5/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0285
…

Epoch 246/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0253
Epoch 247/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0252
Epoch 248/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0252
Epoch 249/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0252
Epoch 250/250
438/438 [==============================] - 1s 2ms/step - loss: 0.0252
```

It is not practical to add results of 500 epochs as that takes many pages, however, even with the first few and the last few epochs, we see that the values of loss function as very low. A better idea is to graphically visualize the loss function throughout the epochs.

```
import matplotlib.pyplot as plt

loss =pd.DataFrame(model.history.history)
plt.plot(loss)
```

Figure 60 shows the loss function values as the number of epochs increase. This is not exactly a typical DNN loss function curve as perhaps the learning is happening a bit too fast.
That could be reasoned by the robustness of this DNNs, that is used for very complex problems. In this case, since all the parameters have clear numerical relations between each other, the machine is able to learn everything exceptionally fast.

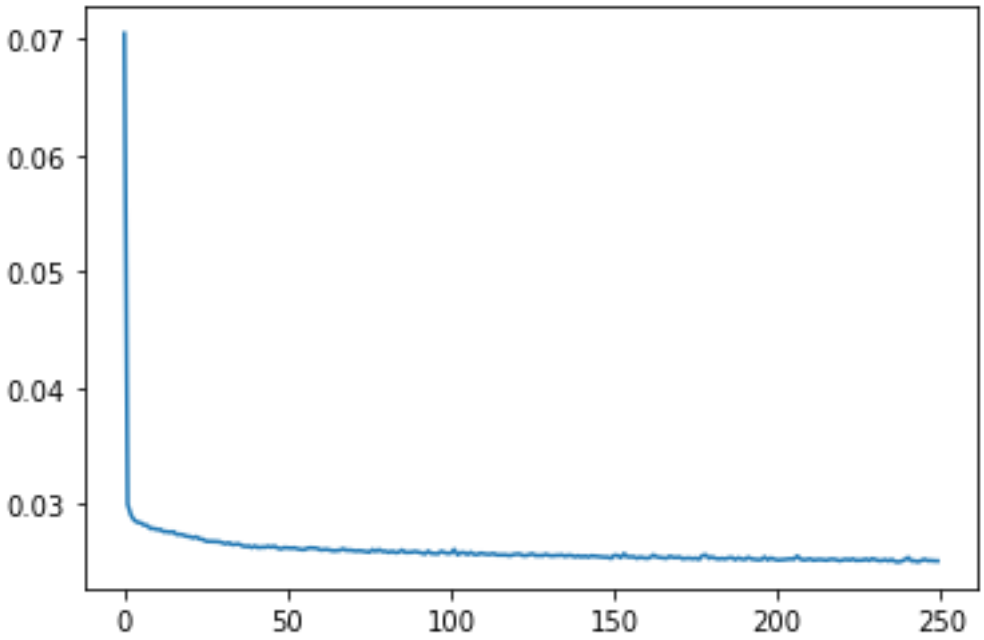


Figure 60. Loss function as Epochs increase.

Up until now, we trained the model using our training set, and we saw the loss function decreasing very fast in the first few epochs, after which not many changes occurred. Now, we put our model to test by having it predict our test data that it was not trained with, meaning that the model now will deal with unseen data.

```
model.evaluate(X_test, y_test, verbose=0)
0.027296315878629684

model.evaluate(X_train, y_train, verbose=0)
0.024843977764248848
```

The model's results when dealing with the training set and when dealing with the test set are close enough. That is an indication of a well-performing DNN.

This concludes our basic model. We can enhance our model by adding new tools or specifications to it. That will surely improve the results. To build the model, we could either delete the memory and restart from the top where we import the data, or simply adjust the parts of code that change, which is a better choice.

The first improvement we can intervene on, is the number of epochs. In our first model graphed in Figure 60, one can note that somewhere between the 50th and 100th epoch, the loss function reaches a value that is pretty much left unchanged for the rest of the epochs. In this case, almost 150 of our epochs are useless or even harmful, as they may cause overfitting. To avoid that, we could use some mechanism of early stopping which will stop the model from training when a loss function value does not change much for a certain number of epochs.

The second improvement is to add Dropout, which is a mechanism that turns off a portion of neurons to see whether that improves the results. Effectively, that changes the number of neurons in every layer. The value of Dropout of the layer may vary from 0 to 1, where 0 means no neurons are being turned off, while 1 means that 100% of the neurons are turned off. Years of trial and error have proven that the best value for Dropout is around 0.25 to 0.5, which is the choice here as well.

The third sophistication of our code is to add a validation set to the results, which means that our training set and our testing set will be compared to each other from the start. This is what makes early stopping possible, by comparing the loss function of the training set and the validation set (test set). The redacted code is as follows:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

X = pd.read_csv('X.csv') #importing the features (inputs)
y = pd.read_csv('y.csv') #importing the labels (outputs)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=74)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(Dense(92, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(82, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(72, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(65, activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy')

from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', mode='min',verbose=1,
patience=25)
```

```
model.fit(x=X_train, y=y_train, epochs=600,
validation_data=(X_test,y_test),callbacks=[early_stop])

losses = pd.DataFrame(model.history.history)
losses.plot()

model.evaluate(X_test, y_test, verbose=0)
0.028434421867132187

model.evaluate(X_train, y_train, verbose=0)
0.02784004434943199
```

The first part of the code is identical to the previous model, except for the training set which is now 30% instead of 20% in the previous model, that is because we can afford a larger test set if we have a lot of data. The difference is in adding the early stop function with a patience of 25, which means that the function will allow 25 epochs without improvement before stopping. If improvements happen continuously, the model will go up to 600 epochs, which is a very large number. Patience number can be changed to allow for fluctuations but 25 seems to be a good number. Inside the network modeling section of code, the only difference is adding the Dropouts. The activation functions and number of neurons are not changed as they were already optimal, although it is advisable to try different formations. On the other hand, the loss function cannot be changed even if we wanted to, that is because binary cross-entropy is the only one that applies to our problem.

Let's check the results of these improvements on the model.

```
Epoch 1/600
384/384 [==============================] - 4s 6ms/step - loss: 0.1026 -
val_loss: 0.0311
Epoch 2/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0335 -
val_loss: 0.0301
Epoch 3/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0312 -
val_loss: 0.0298
Epoch 4/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0306 -
val_loss: 0.0294
Epoch 5/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0301 -
val_loss: 0.0295
...
Epoch 59/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0285 -
val_loss: 0.0284
Epoch 60/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0286 -
val_loss: 0.0286
```

```
Epoch 61/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0284 -
val_loss: 0.0285
Epoch 62/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0285 -
val_loss: 0.0285
Epoch 63/600
384/384 [==============================] - 2s 5ms/step - loss: 0.0287 -
val_loss: 0.0284
Epoch 00063: early stopping
```

Even in the first epoch, the loss function decreases very steeply, for both the training set and the validation set. With the early stopping option we added in the model, the training only goes up to epoch 63. That means, the model has not seen any improvements since epoch 38. This will save time from the simulation and memory space for the machine.

Figure 61 shows the results graphically.
The ML model performing this well may be an indication that Sub-Urb feeder was already in a satisfactory state, generally speaking.

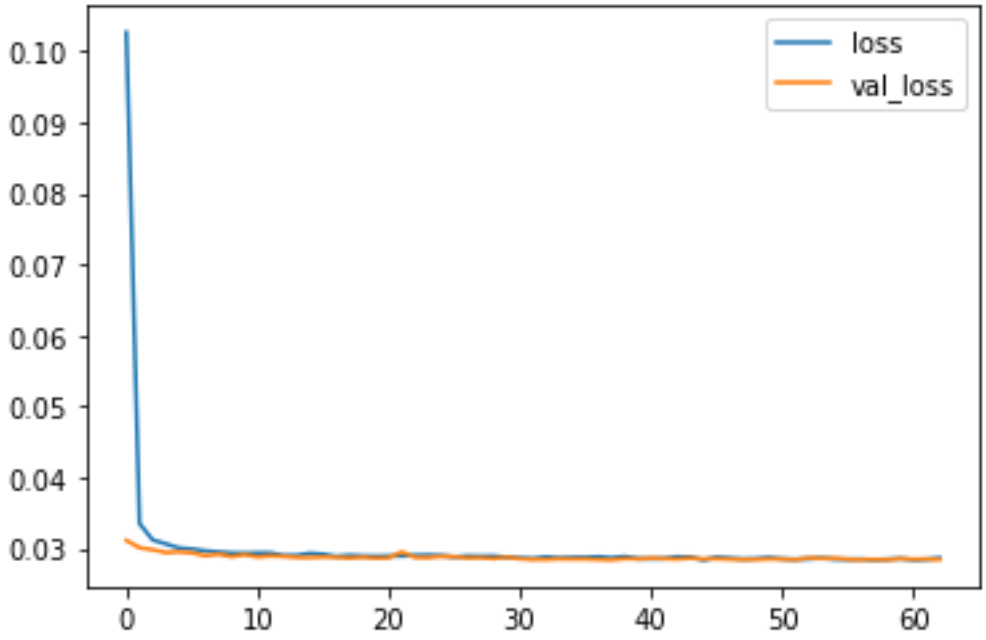


Figure 61. Loss function of the training set and validation set as epochs increase

If we go back to the results of the Quasi-Dynamic analysis when E-OLTCs were employed, we will note that there were no overvoltage constraints. Using this ML model with new inputs of PV generation, loads, electric vehicles both as loads or when injecting to the grid, we will always be able to find a combination of tap changer positions that guarantees no overvoltages.

The binary classification cross-entropy loss function may not be as informative as needed. Hence, the next step is to export the predictions of tap changers of the Deep Neural Networks. Then, these predictions will be plugged in PowerFactory to run another Quasi-Dynamic analysis in hourly steps, this time only 6-month long. The difference this time is that PowerFactory will not use its inter algorithms to figure out which tap position is the correct one with respect to voltage constraints violations. This time, a 5000 hour characteristic of each transformer with E-OLTC is built directly from the predictions of the Deep Neural Network, then registered in the sofware. So, this time we will input tap positions and monitor the voltages, instead of having the software choose the correct positions. Upon doing that, we can see in electrical engineering terms how well the DNN model is performing, without having to figure out what the loss function is telling us.

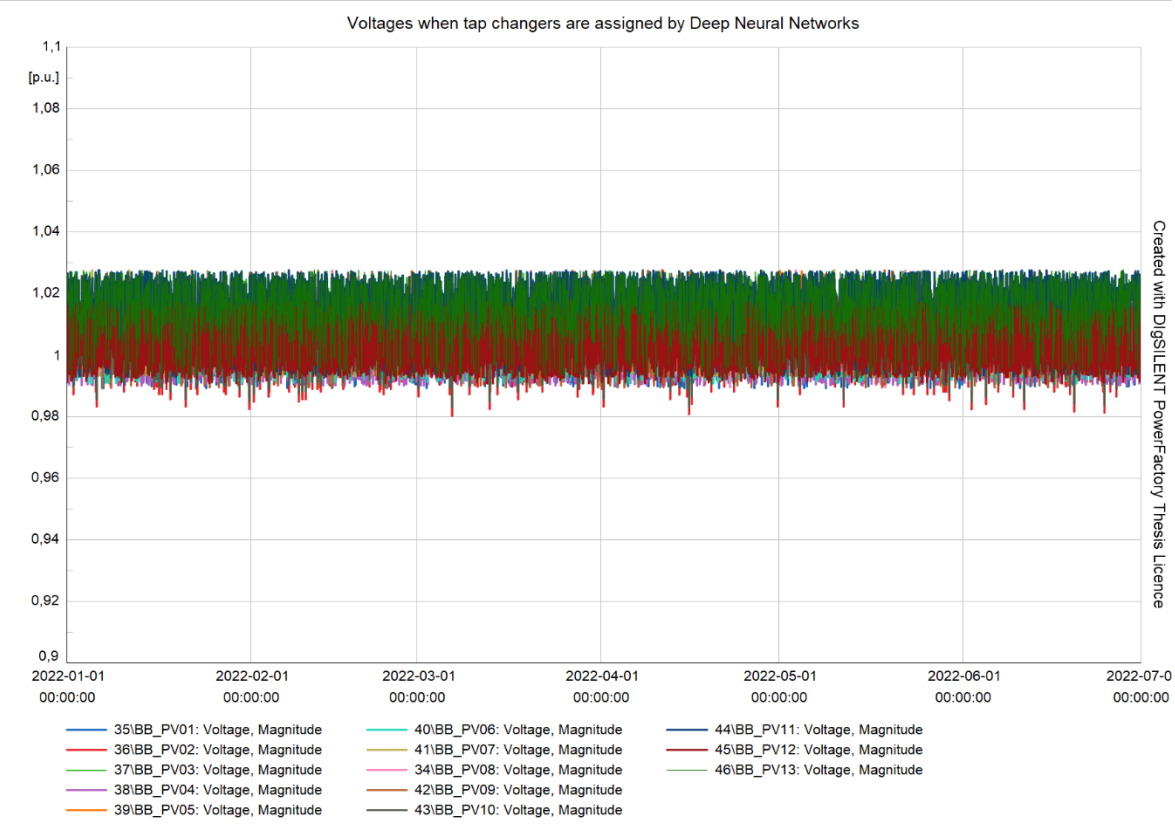


Figure 62. Voltages in Feeder Sub-Urb when tap positions are set according to DNN predictions [p.u].

Figure 62 supports the results loss function of Figure 61.

This graphical representation comes from a randomly shuffled dataset of inputs (this time taps are inputs as well, together with loads and PVs), and it proves that the DNN model's predictions on tap position are optimal eventhough it was fed a test set it has never seen before and was not trained on.

Of course, the fact that voltages do not go beyond the allowed limit but stay in the range of 0.98 [p.u] and 1.035 [p.u] is also thanks to the fact that this feeder was well designed even before using ATPs, since there were very small voltage contraint violations

Figure 59 (PowerFactory's calculations of taps) can be compared to Figure 62 (DNN's predictions of taps). The results show that, for this feeder, DNN performs as good as PowerFactory in choosing the right tap postions.

4.8. Additional results and visualizations

Depending on the task in question, several types of visualizations can be used to better understand the input data and the model's results. Since there are practically countless data to visualize, let's just take an arbitrary example where we try to figure out the times Transformer 10 has operated at tap position (-1).

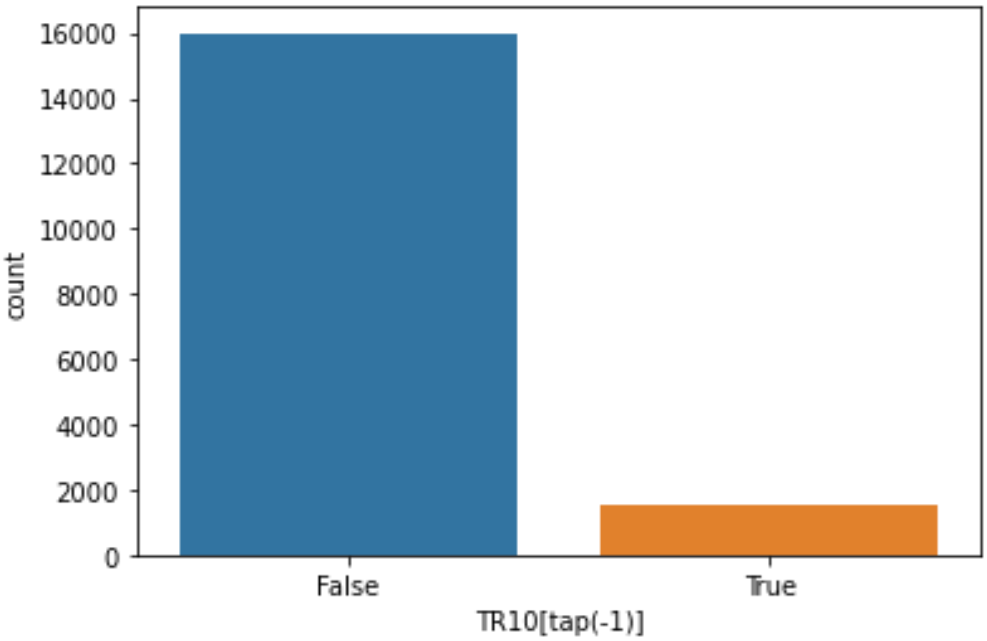


Figure 63. Countplot of TR10 operating at tap position (-1).

Figure 63 shows that Transformer 10 has operated at tap position (-1) only for a short time, which is understandable. The False bar shows all the instances of TR10 operating at all tap positions other than (-1). In similar fashion, we could use boxplots, scatterplots etc., to understand the interrelations of certain transformers and their tap positions. However, since our values are zeros and ones, there are not many benefits from doing so, apart from counting the number of voltage constraint violations which was already done.

This subchapter concludes our first studycase.

5 CASE STUDY II – 10 [kV] feeder Rugova, Kosovo

5.1. Introduction

Feeder Rugova is the second studycase of this thesis. The procedures in Rugova feeder are identical to the ones of Sub-Urb feeder. The only difference stands in the network model and parameters, which in terms of programming is irrelevant. In Case Study II (CS-II), the DGs are connected directly to LV busbars where loads are connected, while in Case Study I (CS-I) Feeder Sub-Urb, DGs had their own LV busbar. In order to avoid repetitiveness, most of the concepts that were introduced and explained in CS-I, will be omitted here in CS-II. In any case, the chapter layout has not changed, therefore, the reader can always address the equivalent subchapter in CS-I for detailed explanations.
Feeder Rugova is a real-life feeder in the Accursed mountains in the republic of Kosovo. Loads, line parameters, line lengths, transformers etc., are modelled according to the real ones. Data used to model them is curtesy of Kosovo's national DSO called Kosovo's Energy Distribution Services (KEDS).
Since the quality of an ML algorithm can only be as good as the data used to train it, the second studycase should yield even better results than the first one, because most of the data used for Rugova feeder is real and not randomly generated. Instead of reviewing the 24 steps basic steps of this research that were introduced in CS-I, let's simply take a new look at the flowchart of Figure 37, which shows the steps in less detail.

Running Quasi-Dynamic analysis basics were already discussed in detail in CS-I, therefore we can move forward to modelling Rugova feeder in PowerFactory.

5.2. Modelling Rugova feeder in PowerFactory

The 10 [kV] feeder Rugova is based on the real-life feeder with the same name, located in Kosovo. All data of elements and their parameters come from measurements and are exported directly from smartmeters in the feeder.
The full topology of feeder Rugova is shown in Figure 65.

Balancing the power flows of the feeder will be done by the at the HV/MV busbar where normally a transformer is connected. That busbar has an External Grid which is modelled as a Slack with voltage setpoint at 1.0 [p.u].

In practice the voltage setpoint at the slack bus is not always 1.0 [p.u]. For feeder Rugova especially, in wintertime when the loads reach their peaks, the HV/MV transformer's tap is usually set at a position that sets the voltage at 1.05 [p.u]. This makes it possible to keep the voltage levels within the allowed range.

In Rugova feeder, same as in Sub-Urb feeder, the PV systems inject in the LV side that is 400 [V]. This can be seen clearer in Figure 66. A LV/MV transformer of ratio 0.4/10 [kV]/[kV] then transforms the voltage to the MV, right for the energy to flow through the feeder, and in certain moments, even flow all the way up to the HV network busbar.

The latter case happens when the sum of loads in the feeder is exceeded by the PV production in the feeder. This particular case is shown in Figure 64, where the arrows on lines indicate the directions of power flows.

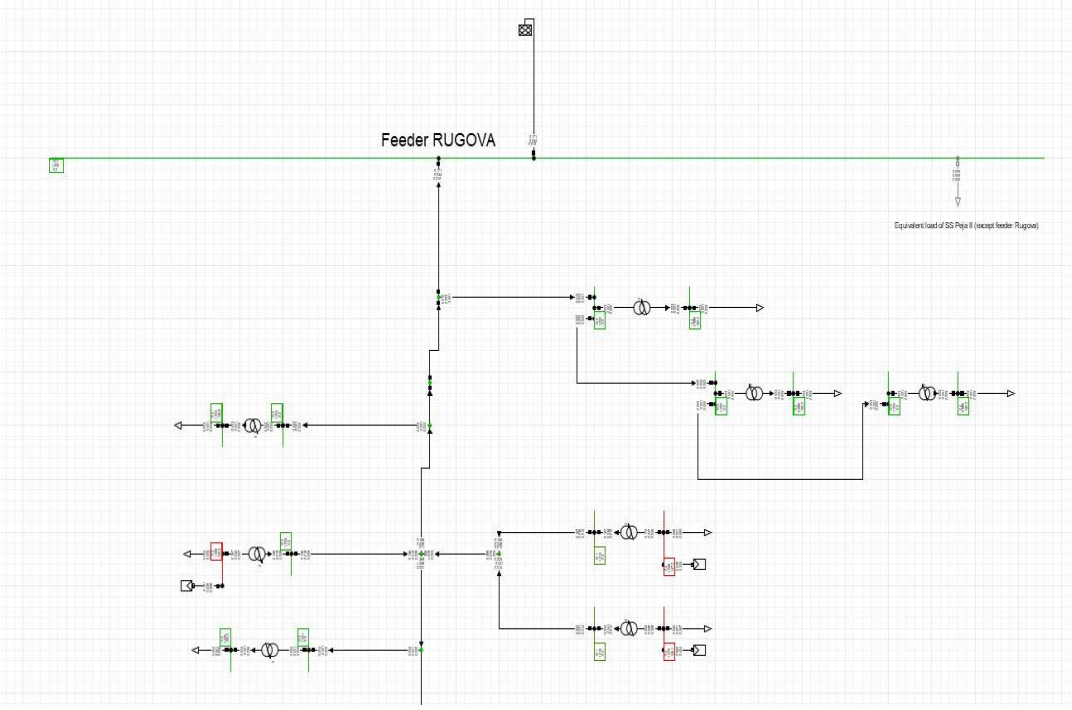


Figure 64. Feeder Rugova 20 [kV] upstream flow of energy.

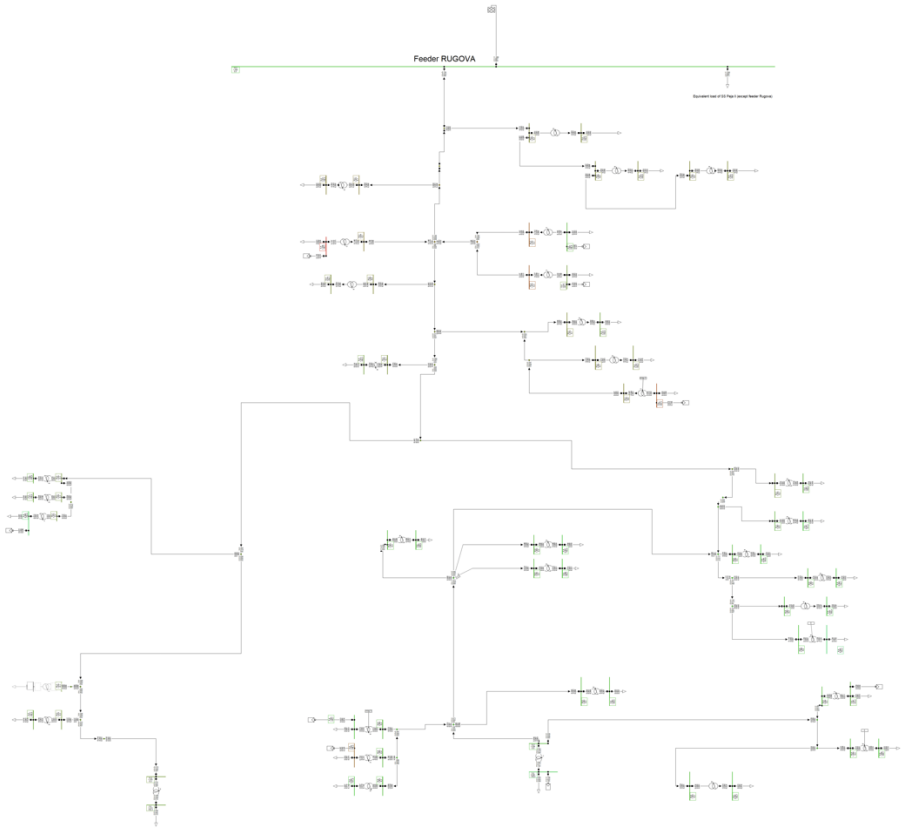


Figure 65. Feeder Rugova 20 [kV] topology.

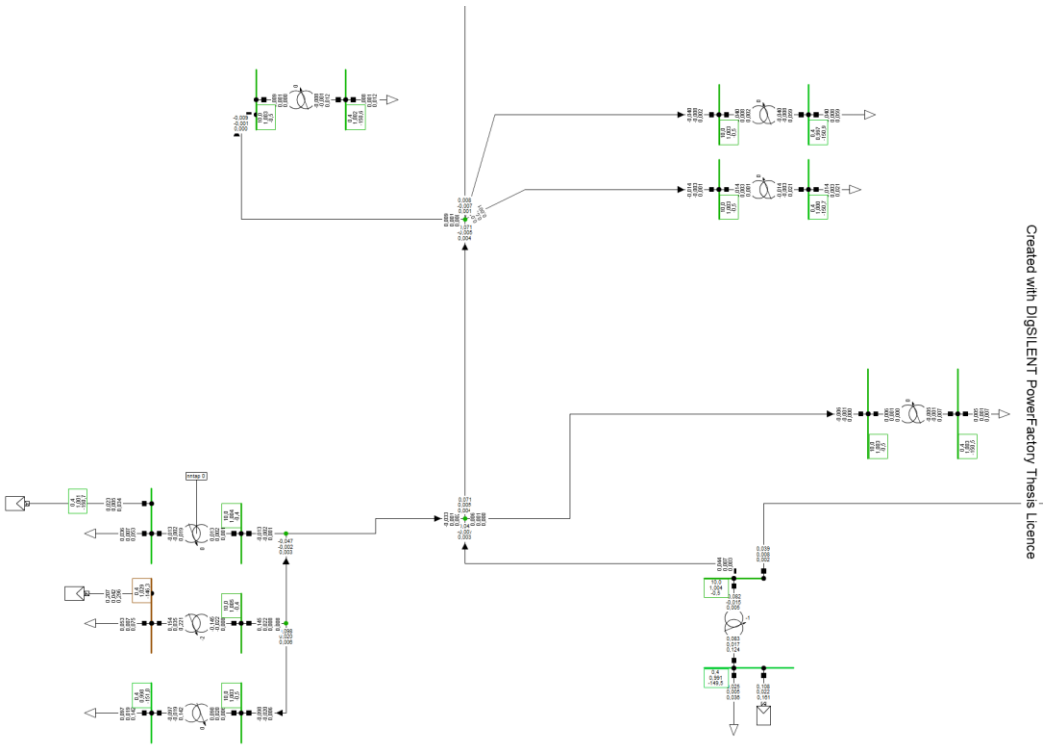


Figure 66. Feeder Rugova 20 [kV] topology of a random portion.

High production of PVs with respect to the load is the main cause of overvoltages in MV networks, so E-OLTCs will be put to use to try and minimize these overvoltages. Important feeder data pertaining lines, transformers, loads, PVs, will be given in upcoming tables.

PV Name	App.Pow. [kVA]	Pow.Fact.	Inom [kA]	Efficiency Factor [%]	Tilt Angle [deg]
PV System_01	124	0,95	0,179	95	30
PV System_02	304	0,95	0,439	95	30
PV System_03	54	0,95	0,078	95	30
PV System_04	178	0,95	0,257	95	30
PV System_05	40	0,95	0,058	95	30
PV System_06	332	0,95	0,479	95	30
PV System_07	450	0,95	0,65	95	30
PV System_08	372	0,95	0,537	95	30
PV System_09	56	0,95	0,081	95	30

Table 11. Photovoltaic systems installed in Feeder Rugova 10 [kV].

Table 11 gives the relevant data of PVs installed at various locations throughout the feeder. Each PV here has the same characteristic, but of different magnitudes.

Name	Length [km]	Inom [kA]	Cable / OHL 0/1	R'(AC.20°C) [Ω/km]	X' [Ω/km]
100389;AL/FE_35/6(4)	0,020	0,145	1,000	0,820	0,421
50051006 Rugova(1)	0,300	0,290	1,000	0,320	0,392
50051006 Rugova_a	0,300	0,290	1,000	0,320	0,392
100389;AL/FE_35/6(5)	1,215	0,145	1,000	0,820	0,421
100389;AL/FE_35/6(1)	1,215	0,145	1,000	0,820	0,421
Line(9)	0,187	0,125	1,000	1,200	0,478
50051006 Rugova_a(1)	5,178	0,290	1,000	0,320	0,392
50051006 Rugova	0,943	0,290	1,000	0,320	0,392
101180;AL/FE_35/6(87)	0,030	0,145	1,000	0,820	0,421
Line(20)	1,087	0,170	1,000	0,590	0,409
100417;	2,761	0,290	1,000	0,320	0,392
100389;AL/FE_35/6(3)	0,133	0,145	1,000	0,820	0,421
100417;(1)	0,531	0,290	1,000	0,320	0,392
101301;(1)	0,052	0,125	1,000	1,200	0,478
100384,000	1,635	0,125	1,000	1,200	0,478
101261;(1)	0,228	0,125	1,000	1,200	0,478
101261;	2,467	0,125	1,000	1,200	0,478
101301;	0,505	0,170	1,000	0,590	0,409
101248;AL/FE_35/6(97)	0,377	0,145	1,000	0,820	0,421
100413;	1,087	0,290	1,000	0,320	0,392
Line_a(1)	0,061	0,125	1,000	1,200	0,478

100428;AL/FE_35/6(11)	2,509	0,145	1,000	0,820	0,421
Line_a(3)	0,061	0,170	1,000	0,590	0,409
101048;	0,241	0,125	1,000	1,200	0,478
100486;	2,668	0,170	1,000	0,590	0,409
100447;	0,863	0,170	1,000	0,590	0,409
101070;(1)	0,029	0,125	1,000	1,200	0,478
Line(21)	0,091	0,125	1,000	1,200	0,478
Line_a(2)	2,664	0,170	1,000	0,590	0,409
Line(23)	0,742	0,170	1,000	0,590	0,409
Line_a(4)	0,061	0,125	1,000	1,200	0,478
101070;	3,064	0,125	1,000	1,200	0,478
100466;	1,545	0,125	1,000	1,200	0,478
100452;	0,045	0,125	1,000	1,200	0,478
100462;	0,077	0,125	1,000	1,200	0,478
100452;AL/FE_35/6(113)	0,172	0,145	1,000	0,820	0,421
100457,000	1,060	0,125	1,000	1,200	0,478
101242;	1,211	0,290	1,000	0,320	0,392
101242;(1)	0,198	0,290	1,000	0,320	0,392
100482;AL/FE_35/6	0,096	0,145	1,000	0,820	0,421
101077;AL/FE_35/6(70)	0,313	0,145	1,000	0,820	0,421
Line(27)	1,325	0,125	1,000	1,200	0,478
100683,000	0,595	0,125	1,000	1,200	0,478
101642;(2)	1,194	0,125	1,000	1,200	0,478
101642;	0,028	0,125	1,000	1,200	0,478
100446;	0,180	0,170	1,000	0,590	0,409
100893;AL/FE_35/6(44)	0,210	0,145	1,000	0,820	0,421
100893;AL/FE_35/6	0,072	0,145	1,000	0,820	0,421
100886;	0,360	0,125	1,000	1,200	0,478
100873;	0,267	0,170	1,000	0,590	0,409
100493;	0,114	0,170	1,000	0,590	0,409
101642;	0,028	0,125	1,000	1,200	0,478
101642;(1)	1,194	0,170	1,000	0,590	0,409
101642;	0,056	0,125	1,000	1,200	0,478
100495,000	0,863	0,290	1,000	0,320	0,392
100522;	0,652	0,125	1,000	1,200	0,478
Line(14)	1,003	0,125	1,000	1,200	0,478
100500;	0,183	0,125	1,000	1,200	0,478
Line(26)	1,304	0,170	1,000	0,590	0,409
100477;AL/FE_35/6(118)	0,015	0,145	1,000	0,820	0,421
100482;	2,882	0,170	1,000	0,590	0,409
Line(24)	1,501	0,290	1,000	0,320	0,392
101642;	1,194	0,170	1,000	0,590	0,409
100402;	2,287	0,125	1,000	1,200	0,478
100544;	2,357	0,125	1,000	1,200	0,478

Table 12. Feeder Rugova 10 [kV] lines data.

Table 12 contains line data. In this case, the names are unorderly, which is the case in real life as it is very hard to have specific names for all sections of lines in MV. The lines are all overhead-lines and not a single significant section of the feeder is in cables. Cables are used in very small sections in in-outs of transformers located in private property. Line resistances and reactances are included in the table as well. The table of loads is provided next table.

Load name	Active P. [kW]	Reactive P. [kVAr]	Inom [A]
01_L_50051006001 - Shtypeqi i Vogel	8	2	12
02_L_50051006002 - Llazi	100	0	144
03_L_50051006003 - Shtupeq i Madhe 1	98	19	145
04_L_50051006004 - Shtupeq i madhë 2	100	0	144
05_L_50051006005 - Llotov	27	5	40
06_L_50051006006 - Reka e Allages	57	11	83
07_L_50051006007 - Malaj	79	16	117
08_L_50051006008 - Pepaj	100	0	144
09_L_50051006009 - KOSHUTAN	38	8	57
10_L_50051006010 - SHKREL	66	13	97
11_L_50051006011 - DUGAIVË	163	0	235
12_L_50051006012 - Drelaj	102	20	151
13_L_50051006013 - Drelaj 2	38	7	55
14_L_50051006014 - Kuqisht 2	20	4	30
15_L_50051006015 - Kuqisht 1	30	6	44
16_L_50051006016 - Haxhaj	48	10	71
17_L_50051006017 - Stankaj	77	15	113
18_L_50051006018 - Boge	205	41	302
19_L_50051006019 - Guri i Kuq	85	17	125
20_L_50051006020 - UNIOR AQUA Drelaj	151	30	222
21_L_50051006021 - Bregu Kuqishtë	16	3	23
22_L_50051006022 - Fab. e Djathit	63	12	93
23_L_50051006023 - Rugova kamp	34	7	49
24_L_50051006024 - Magra Boge	46	9	68
25_L_50051006025 - Elkos Group R. K.Kur	11	2	16
26_L_50051006026 - Ipko Shtupeq i vogël	37	7	54
27_L_50051006027 - Laxha Kurtaj Stanka	87	0	126
28_L_50051006028 - Flamur Kelmendi	2	0	2
29_L_50051006029 - KOSHUTAN HAJLË	30	6	45
30_L_50051006030 -Shkreli 2	50	10	74
31_L_50051006031 Shtupeq i madh 3	8	2	12
32_L_50051006032 - Maja e zez	8	2	12
33_L_50051006033 maja e zez 2	8	2	12
34_L_50051006034 Fatmir kelmendi	46	9	68
35_L_50051006035 Restoran ERA	8	2	12
Equivalent load of SS Peja 2(w/o Rugova)	11,5	2,9	0,671

Table 13. Feeder Rugova 10 [kV] loads data.

In Table 13 we have the data of our feeder's loads. These loads correspond to the transformer they are connected to. Line resistances and reactances are included in the table as well.
As far as transformers go, there is one for each load. So, 35 generic transformers, of which 9 are also transformers where PV plants inject. Instead, in Sub-Urb feeder, all PVs were injecting in the LV side of transformers that had no loads connected.

Name	Loading [%]	Snom [MVA]	uk0 [%]	Automatic Tap Changer on/off
TRPV01_50051006006 - Reka e Allages	23,4	0,100	3,8	1
TRPV02_50051006026 - Ipko Shtupeq i vo	34,6	0,050	3,8	1
TRPV03_50051006012 - Drelaj	24,6	0,160	4,0	1
TRPV04_50051006030 -Shkreli 2	13,7	0,160	4,0	1
TRPV05_50051006017 - Stankaj	31,2	0,100	3,8	1
TRPV06_50051006027 - Laxha Kurtaj Sta	82,9	0,050	3,8	1
TRPV07_50051006004 - Shtupeq i madhë 2	22,6	0,160	4,0	1
TRPV08_50051006002 - Llazi	31,3	0,160	4,0	1
TRPV09_50051006034 Fatmir kelmendi	19,7	0,100	3,8	1
TR_50051006001 - Shtypeqi i Vogel	2,3	0,160	4,0	0
TR_50051006003 - Shtupeq i Madhe 1	22,8	0,160	4,0	0
TR_50051006005 - Llotov	6,6	0,160	4,0	0
TR_50051006007 - Malaj	29,2	0,100	3,8	0
TR_50051006008 - Pepaj	38,6	0,100	3,8	0
TR_50051006009 - KOSHUTAN	15,8	0,100	3,8	0
TR_50051006010 - SHKREL	17,3	0,160	4,0	0
TR_50051006011 - DUGAIVË	67,3	0,160	4,0	0
TR_50051006013 - Drelaj 2	14,4	0,100	3,8	0
TR_50051006014 - Kuqisht 2	8,4	0,100	3,8	0
TR_50051006015 - Kuqisht 1	12,8	0,100	3,8	0
TR_50051006016 - Haxhaj	12,4	0,160	4,0	0
TR_50051006018 - Boge	20,7	0,400	4,3	0
TR_50051006019 - Guri i Kuq	21,5	0,160	4,0	0
TR_50051006020 - UNIOR AQUA Drelaj		0,160	4,0	Out of Service
TR_50051006021 - Bregu Kuqishtë	4,5	0,160	4,0	0
TR_50051006022 - Fab. e Djathit	15,3	0,160	4,0	0
TR_50051006023 - Rugova kamp	14,0	0,100	3,8	0
TR_50051006024 - Magra Boge	12,4	0,160	4,0	0
TR_50051006025 - Elkos Group R. K.	3,3	0,160	4,0	0
TR_50051006028 - Flamur Kelmendi	25,9	0,050	3,8	0
TR_50051006029 - KOSHUTAN HAJLË	12,8	0,100	3,8	0
TR_50051006031 Shtupeq i madh 3	3,5	0,100	3,8	0
TR_50051006032 - Maja e zez	2,3	0,160	4,0	0
TR_50051006033 maja e zez 2	2,3	0,160	4,0	0
TR_50051006035 Restoran ERA	3,5	0,100	3,8	0

Table 14. Transformer data of Rugova feeder 10 [kV]

Each transformer is named with their unique national code, except for the ones connected to PVs which bear 'PV' in their name, listed in the first 9 rows.

For this study, the last column is the main one as it tells whether E-OLTCs are implemented or not. As expected, the ones that are connected to PV systems are equipped with automatic tap changers, while the others are not. The PowerFactory modelling part is now complete. The next step is to create the characteristics of PVs and loads.

5.3. Generating, extracting, and preparing the data

Feeder Rugova has a peak PV production of 1.91 [MVA] while the peak aggregate load is 2.08 [MVA].
Rugova study-case contains around 2,417,760 values in total!

These procedures were explained in detail in the previous study-case, therefore, here they will be significantly shorter and more result based.

5.3.1. Creating time characteristics for PVs

European Commission's PVGIS tool SARAH2 will be used to obtain the irradiance data of the area where Rugova feeder is located, as shown in Figure 67.

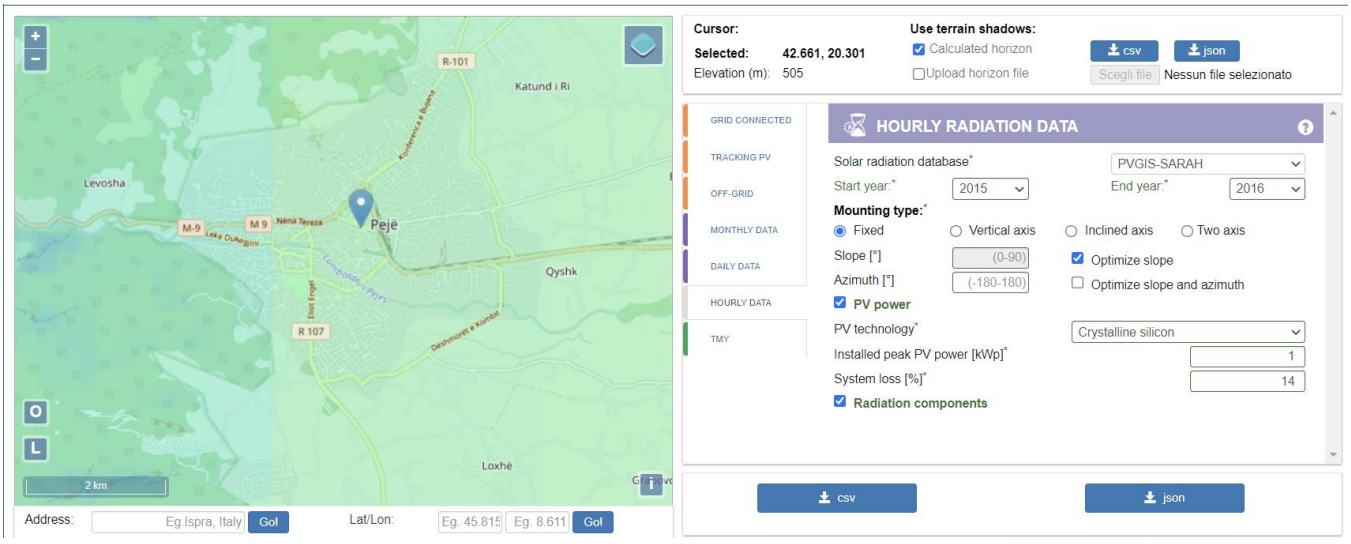


Figure 67. EU Commission's PVGIS tool – Rugova feeder irradiation.

The settings such as year of irradiation, mounting type, PV technology etc., must be changed accordingly. To create the characteristics of our PV plants that have various installed peak powers [kWp], we will first build the characteristic of a 1 [kWp] plant then find the irradiation of our plants using linearity.

After downloading the irradiation data for our fictious 'Unit PV plant', we will have a normalized vector, scaled from 0 to 1.

$$\mathbf{Ipv} = (0, 0, 0, \dots, 0.89, 0.91, 0.93, 0.88, \dots)_{1 \times 17520}$$

Ipv holds hourly values of active power output of our arbitrary plant.

Each of our PVs have their unique installed peak power according to Table 11, vector **Ppv** contains those values.

$$\mathbf{Ppv} = (124, 304, .., 96)_{1x9}$$

It is enough to multiply vectors **Ipv** and **Ppv** to get a dataframe or matrix that contains the characteristics of each PV plant. This a matrix will have a size 17520×9. The 9 columns represent the characteristics of feeder's 9 PVs. The rows are hourly steps of a period of 2 full years.

$$\mathbf{Po} = (\mathbf{Ipv}' \times \mathbf{Ppv})_{17520x9}$$

Ipv had to be transposed in order to make the vectors suitable for multiplication. Matrix **Po** shown in Table 15 contains all relevant characteristics. **Po** is then saved and manipulated using Pandas.

	PV01	PV02	PV03	PV04	PV05	PV06	PV07	PV08	PV09
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	11	27	5	16	4	31	42	34	5
8	14	34	6	20	4	39	52	42	6
9	14	34	6	21	4	39	52	42	6
10	16	38	7	23	5	44	59	47	6
11	15	36	6	22	5	41	55	44	6
12	14	32	6	20	4	38	50	41	5
13	10	23	4	14	3	27	36	29	4
14	4	9	2	6	1	11	14	12	2
15	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…

Table 15. PV characteristics - Rugova Feeder.

This is only a day of the 2-year characteristics of our 9 PV systems that are contained in matrix **Po**$_{(17520x9)}$.
The characteristic of a random PV of our feeder is represented in Figure 68.

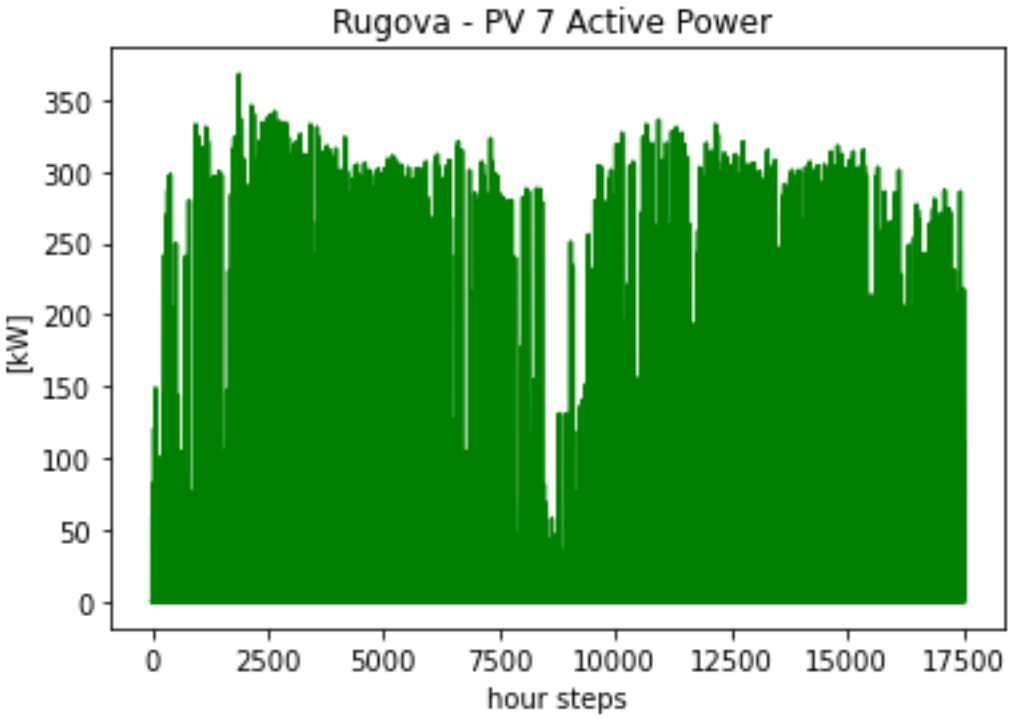


Figure 68. PV7 output characteristic – Rugova [kW].

From this point on, all there is left to do is go to PowerFactory and assign these characteristics to their respective PVs. That can either be done by accessing **Po** and then the relevant column, or by saving all the columns in separate '.csv' files and then accessing those files directly.

5.3.2. Creating time characteristics for loads

For the following procedures, Tables in the beginning of this chapter will be referred.

To create time characteristics of loads, first, a vector (**Lp**) of peak loads in the feeder will be built. Then, a vector (**Lr**) whose elements contain a reliable yearly characteristic will be built and normalized in a scale from 0 to 1 and renamed (**Ln**). Afterwards, the aforementioned vectors will multiply and create a matrix which contains all the load characteristics of our feeders, organized in columns. All these operations are shown below without further descriptions.

$$\mathbf{Lr} = (L_{h1}, L_{h2}, L_{h3}, \dots, L_{h17520})_{1x17520}$$

Lr has 17520 elements pertaining to load's hourly unit values. If max(**Lr)** = Lrmax, then:

$$\mathbf{Ln} = \mathbf{Lr}/\mathrm{Lrmax}$$

Ln will be the curve (characteristic) of all loads, but they each will have different magnitudes according to their peaks indicated by real measurements.
A generic **Lp** will be:

$$\mathbf{Lp} = (L_{p1}, L_{p2}\ , L_{p3}, \dots, L_{p33})_{1x35}$$

Lp holds the peak values of 35 loads found in feeder Rugova.
As advised, **Ln** and **Lp** are to be multiplied.

$$\mathbf{L} = (\mathbf{Ln}' \times\ \mathbf{Lp})_{17520x35}$$

Again, **Ln** is first transposed then for compatibility issues.

The loads matrix is now ready and it contains 70 columns, 35 for active powers of loads and 35 for reactive powers loads. The load matrix $\mathbf{L}_{(17520x70)}$ can be divided in two different matrixes, one pertaining to the active powers, the other to the reactive powers, $\mathbf{Lp}_{(17520x33)}$ and $\mathbf{Lq}_{(17520x33)}$, respectively.

L, as shown in Table 16, is a standard matrix, already seen several times throughout this study.

	P01_L	Q01_L	P02_L	Q02_L	…	P34_L	Q34_L	P35_L	Q35_L
0	2	1	37	1	…	15	3	2	1
1	2	1	32	1	…	13	3	2	1
2	2	1	30	1	…	12	2	2	1
3	2	1	30	1	…	12	2	2	1
4	2	1	32	1	…	13	3	2	1
…	…	…	…	…	…	…	…	…	…
17515	5	1	93	1	…	37	8	5	1
17516	5	1	89	1	…	35	7	5	1
17517	5	1	81	1	…	32	7	5	1
17518	4	1	66	1	…	26	5	4	1
17519	3	1	52	1	…	21	4	3	1

Table 16. Matrix of loads in Rugova feeder.

The characteristics are now ready to be accessed from PowerFactory, each column from its relevant load and parameter. 70 characteristics, for 35 loads with two parameters each (active power and reactive power).

If for example we want to add an active power characteristic to Load 18, we refer PowerFactory to L['P18_L'], which is the corresponding column of matrix **L**.

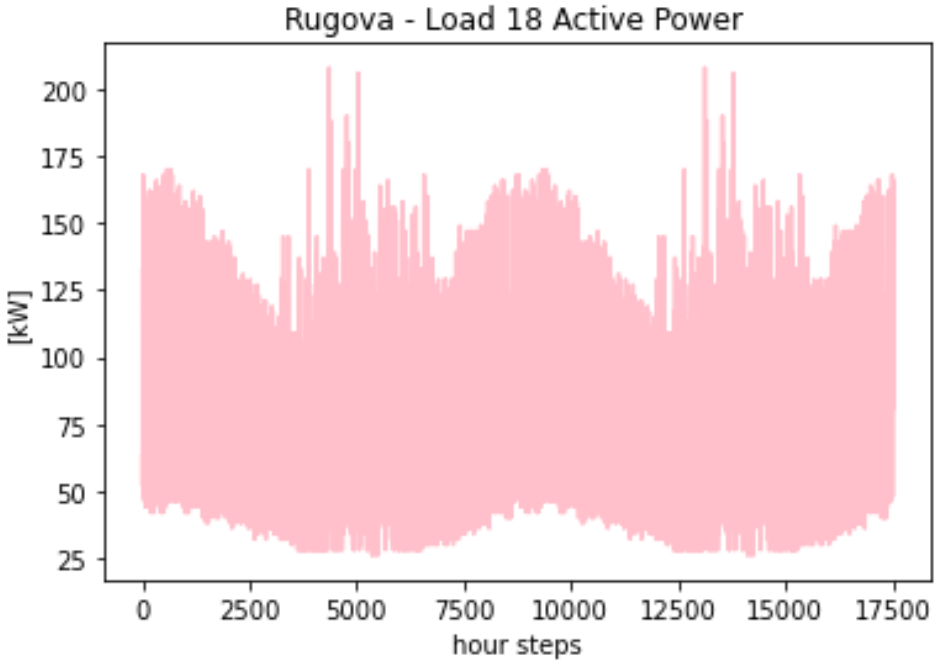


Figure 69. Active Power characteristic of Load 18 in Rugova feeder [kW].

Figure 69 shows the active power load characteristic of Load number 18.

5.4. Running the Quasi-Dynamic simulation and exporting the results of Tap Positions

Up to this point, the feeder has been modelled, the data has been collected and processed into characteristics for all PVs and loads. Next come the Quasi-Dynamic analysis which will allow us to collect large datasets of correct outputs when given certain inputs.
To reiterate, our outputs (labels) are the transformer tap positions. After running the Q-D simulation, we will have 17520 correct outputs which we will use to train our Deep Neural Network. Running the Q-D analysis was covered in CS-I 4.1.2, which is why on this section is on the graphical representation of the aggregate parameters and on the outcomes of the Q-D simulation.

Before proceeding with the simulations, a final check at our superposed (aggregated) characteristics is in order since in the previous two sections they were not included graphically. The PV characteristics of all 9 plants of our feeder are shown in Figure 70, which corresponds to Table 15 (matrix **Po**).

The peak power output of these PVs is significantly lower than the ones of feeder Sub-Urb, which is natural because the load is much smaller in Rugova, Rugova is located in a mountainous area in which it is difficult to find plane land to install the panels, and the fact that Rugova operates at 10 [kV].

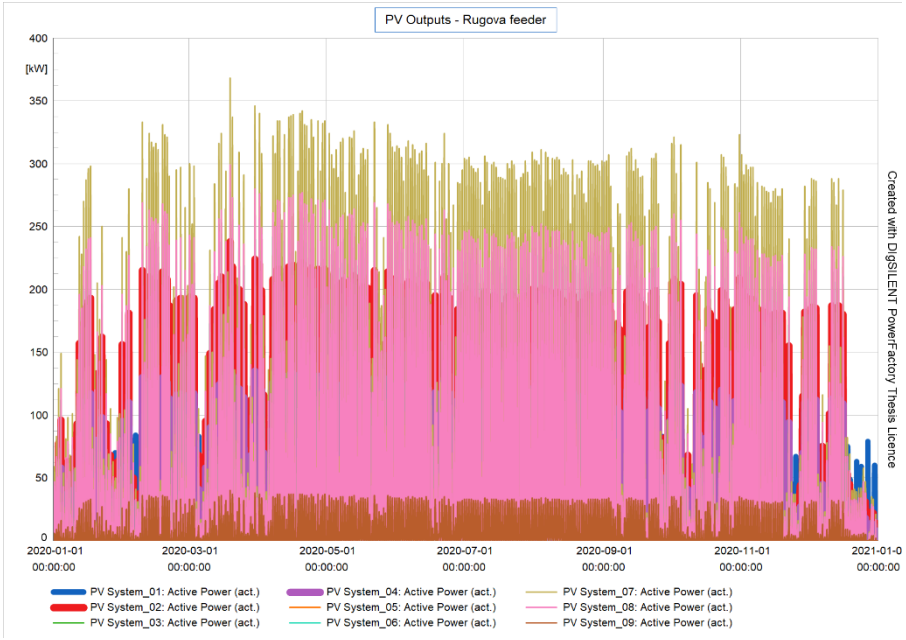


Figure 70. PV Outputs Rugova feeder (first year) [kW].

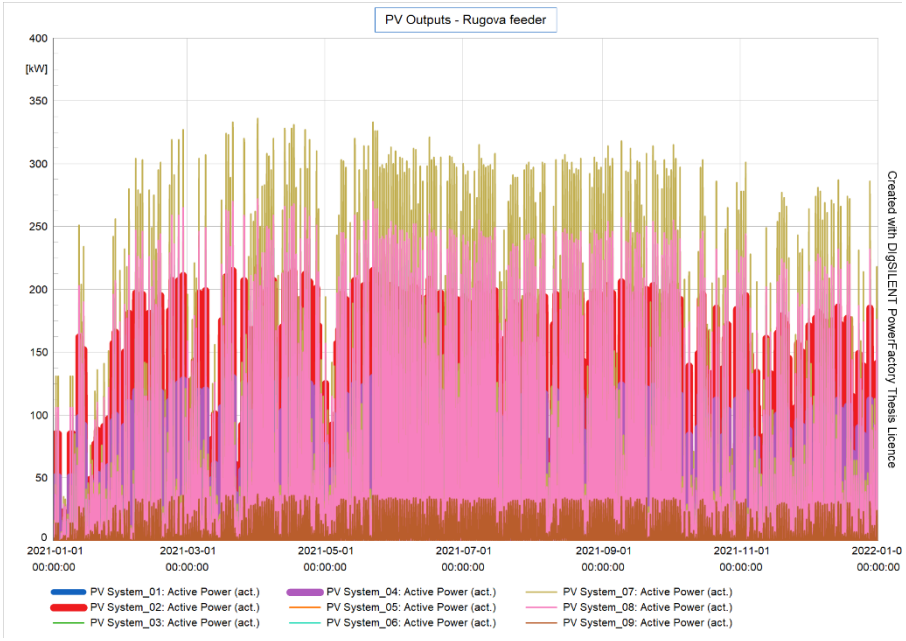


Figure 71. PV Outputs Rugova feeder (second year) [kW].

Figure 72 represents the aggregate PV production of the feeder as well, however, since the resolution is higher, the curves may be easier to understand when zoomed

in.

Curves of the PVs in the zoomed in version confirm that PV outputs are subject to clouds, rain, and other meteorological phenomenons.

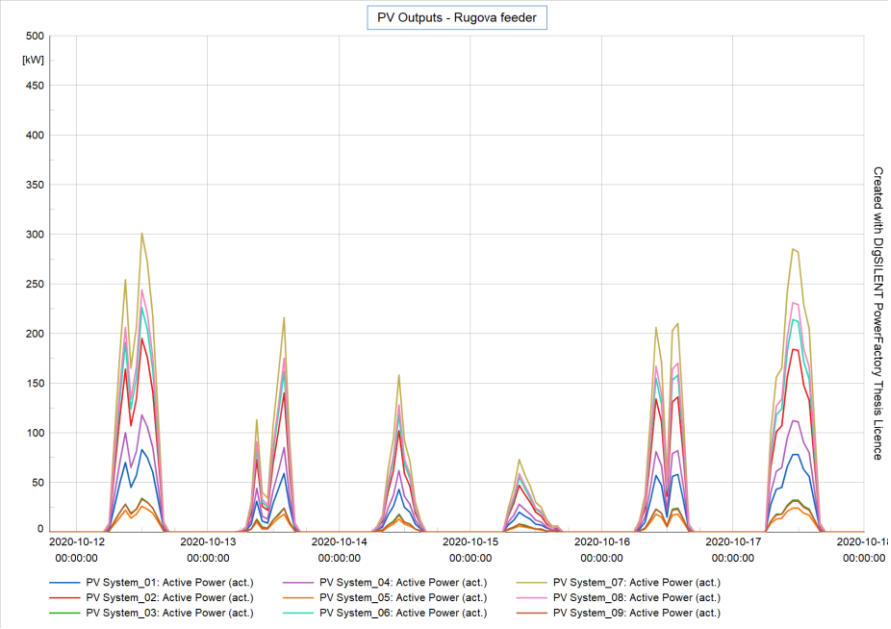


Figure 72. PV Output Rugova feeder zoomed.

Meanwhile Figure 73 depicts the aggregate loads (active component) of the feeder.

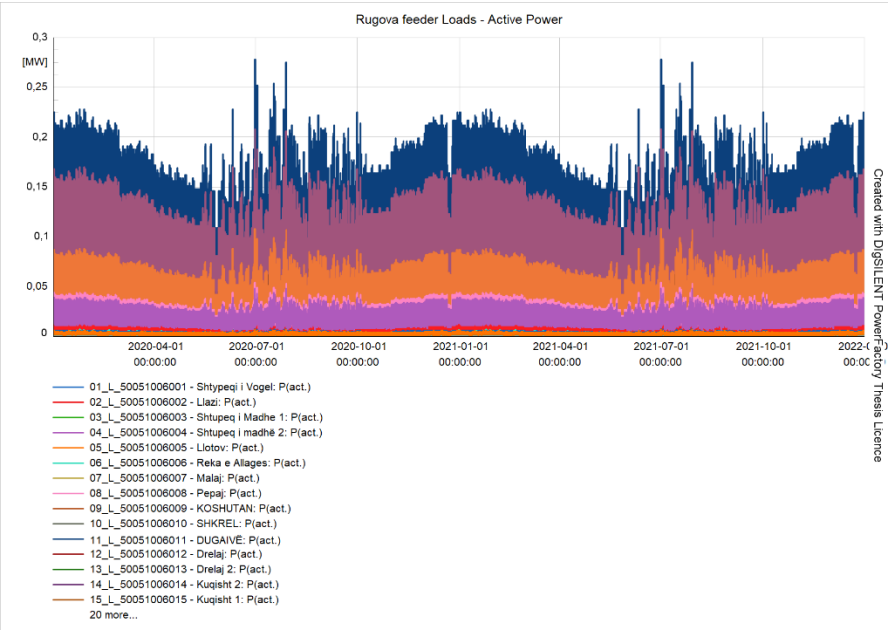


Figure 73. Loads (active power) of feeder Rugova.

Figure 74 depicts the same characteristic as before, just zoomed in.

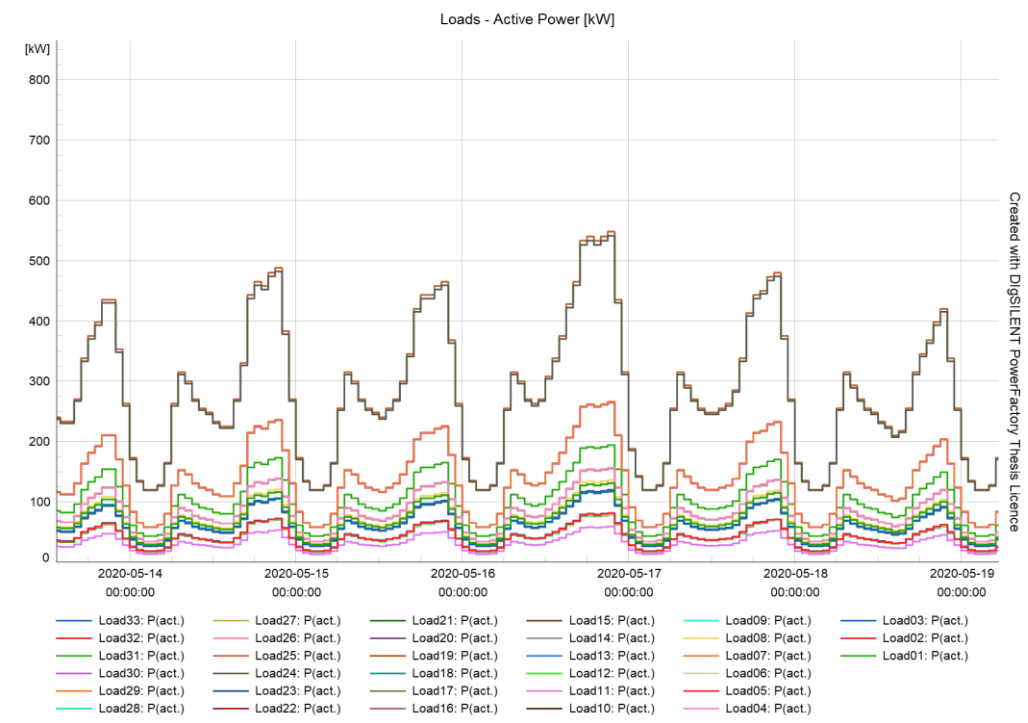


Figure 74. Loads (active power) in Rugova feeder, zoomed in. [kW]

Moving forward, Figure 75 depicts the reactive component of the feeder's loads.

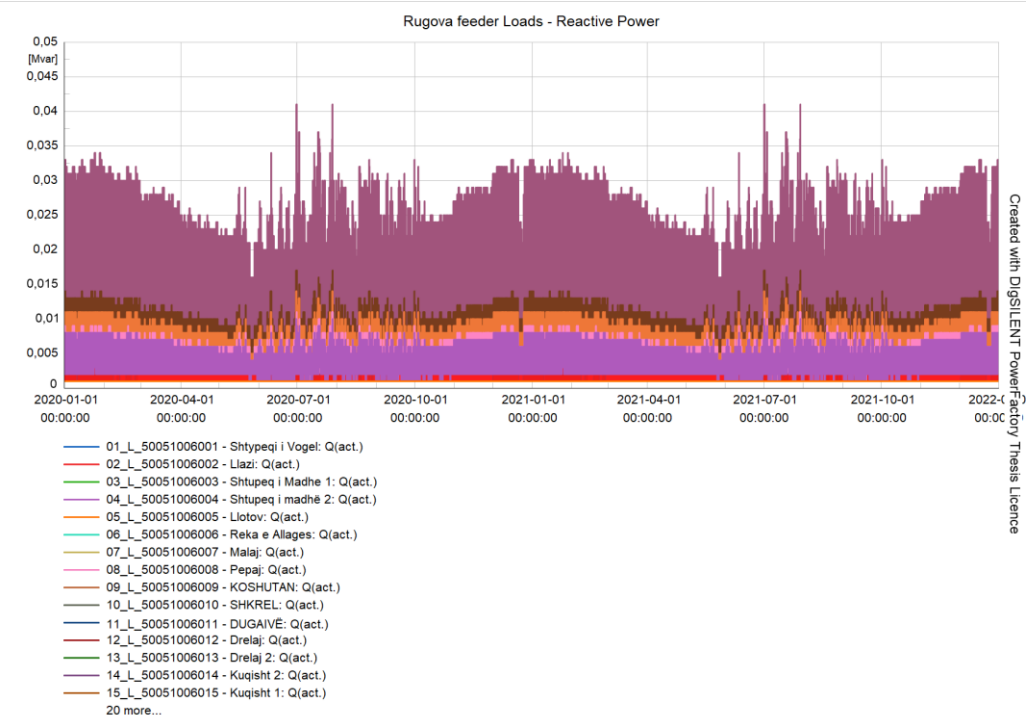


Figure 75. Loads (reactive) of Rugova feeder. [kVAr]

Figure 76 represents the zoomed in version of Figure 75, reactive loads.

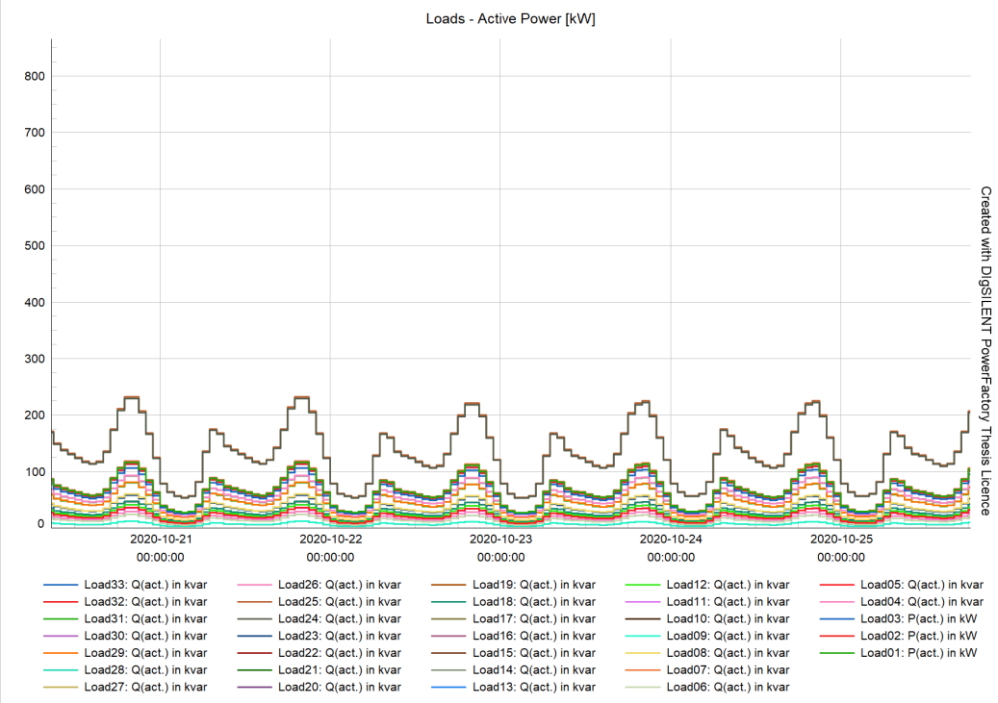


Figure 76. Loads (reactive) of Rugova feeder, zoomed in. [kVAr]

The figures of PV outputs and loads we have seen so far are the same regardless of whether the Q-D simulation is run with ATPs or without them. The same cannot be said about the voltages and naturally, E-OLTCs positions.

Let's first run the Q-D simulations without activating automatic tap changers.

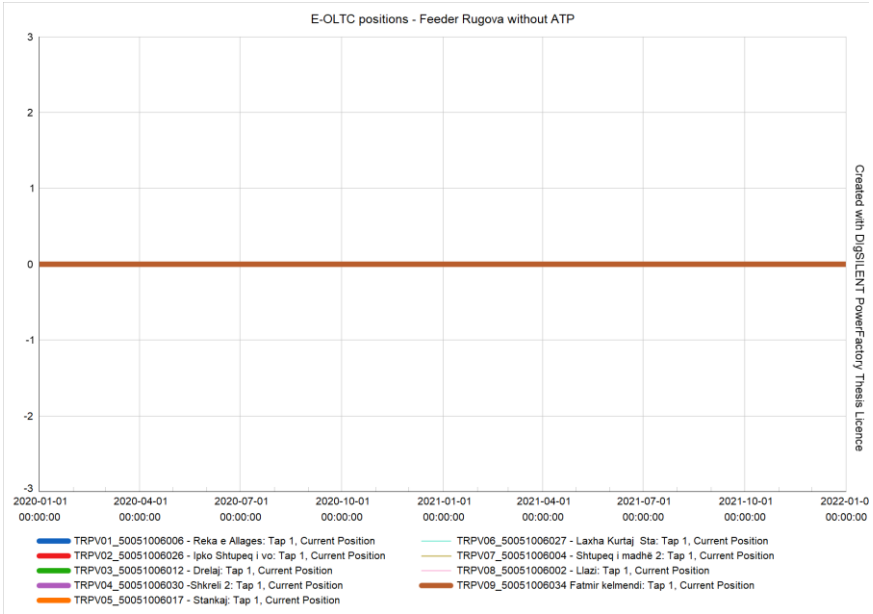


Figure 77. Q-D simulation without Automatic Tap changers - Tap positions

Figure 77 simply confirms that our settings in PowerFactory were correct, as the tap positions are at neutral (0) at all times because ATPs are not employed.

Same as for the Sub-Urb feeder but in much less detail, the number of voltage constraint violations will be counted using Pandas.

```
import pandas as pd

RU_raw_noEOLTC = pd.read_csv('QD_Rugova_noEOLTC.csv', delimiter=';')
```

'RU_raw' dataframe contains columns of many data we have no need of, therefore we will delete everything but keep the voltages.

```
RU_voltages_noEOLTC = RU_raw_noEOLTC.iloc[:,-18:-9]
```

Our new dataframe 'RU_voltages_noEOLTC' only contains 9 columns of the voltage values at the busbars where PVs inject. We will simply count the instances where the voltages exceed 5% and 10%.

```
ru_noEOLTC_violations_5percent = (RU_voltages_noEOLTC>1.05).sum().sum()
print(ru_noEOLTC_violations_5percent)
11174

ru_noEOLTC_violations_10percent = (RU_voltages_noEOLTC>1.1).sum().sum()
print(ru_noEOLTC_violations_10percent)
2404
```

Differently from Sub-Urb feeder, in Rugova feeder we have overvoltages that exceed 10% of the nominal voltage. We are always speaking of overvoltages, as they are a side effect of high penetration of DGs in MV networks.

Undervoltages are not of primary no concern here, especially since Kosovo's distribution code (Kodi i Shpërndarjes) allows for -10% Un at all times, and -15% Un for short periods [20]. However, we will later have an additional Q-D run of CS-II where E-OLTCs are combined with LVRs, which will especially take care of undervoltages.

As far as ENTSO-E is concerned [8], Rugova suffers 11,174 instances of overvoltages in a window of 2 years, of which, 2,404 are over 10% of the nominal voltage.

Figure 78 shows these counts graphically. Although the voltages date says two years, in fact it is a single year – mirrored for the reasons explained at the beginning of the chapter.

These results that stem from no ATP Q-D simulation, will later be compared to the Q-D simulations with ATPs.

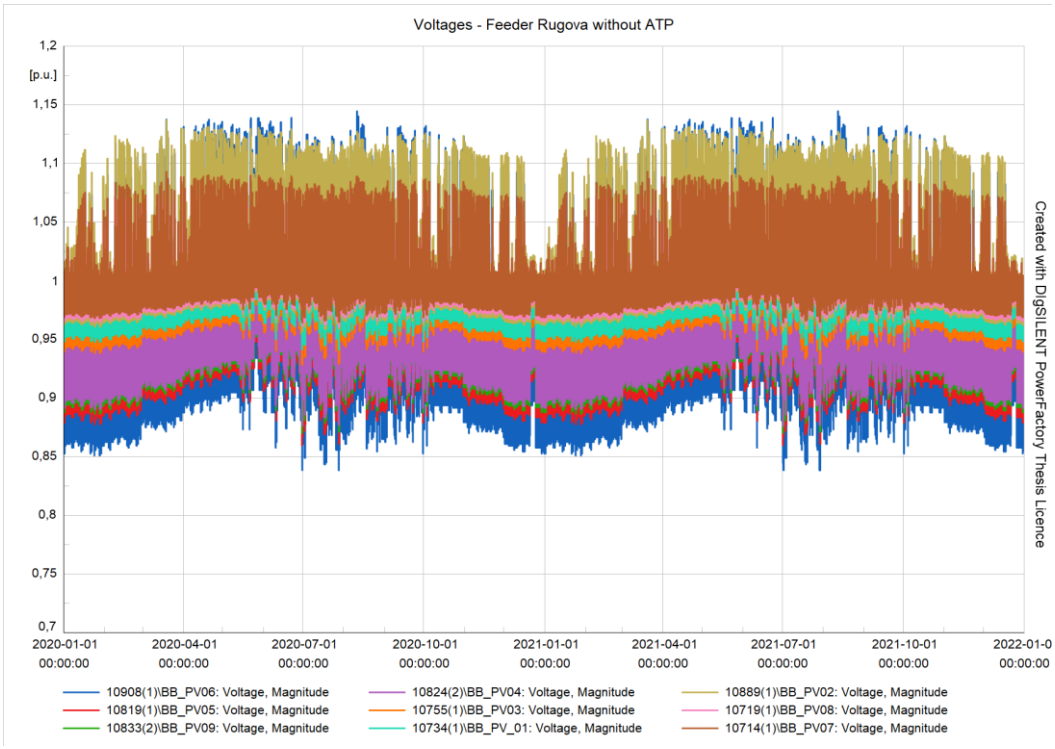


Figure 78. Voltages in Rugova feeder without ATPs [p.u].

Now, the Quasi-Dynamic simulation is run with automatic tap changers.

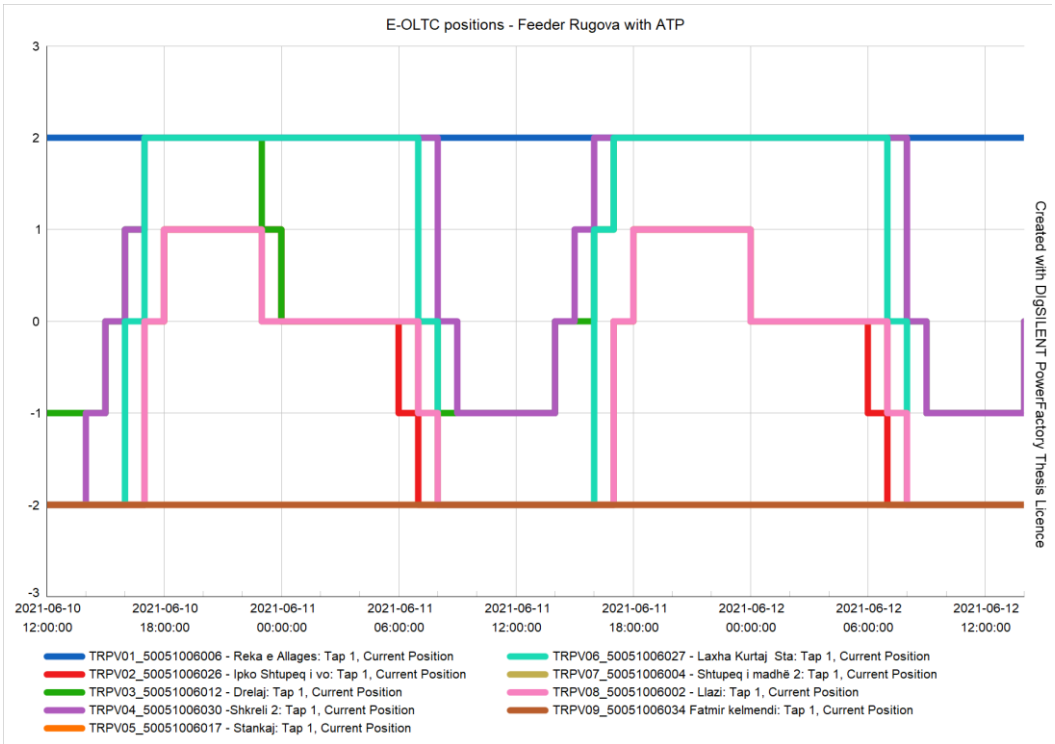


Figure 79. Q-D Analysis with ATP – Tap Positions, zoomed in.

In contrast to the previous case where tap positions were at their neutral position at all times, in the second case, when running a Q-D simulation with ATPs, one can clearly see the tap positions changing very often to appropriate the voltage in the feeder, as shown in Figure 79.

Figure 79 clearly shows that a transformer's tap position can range from its minimum tap position (-2) to its maximum tap position (+2). All that can happen within a single day, in function of PV production in different the hours of the day. Figure 80 follows the same parameter, but it covers the entire 2 years of the simulation, which makes it difficult to understand.

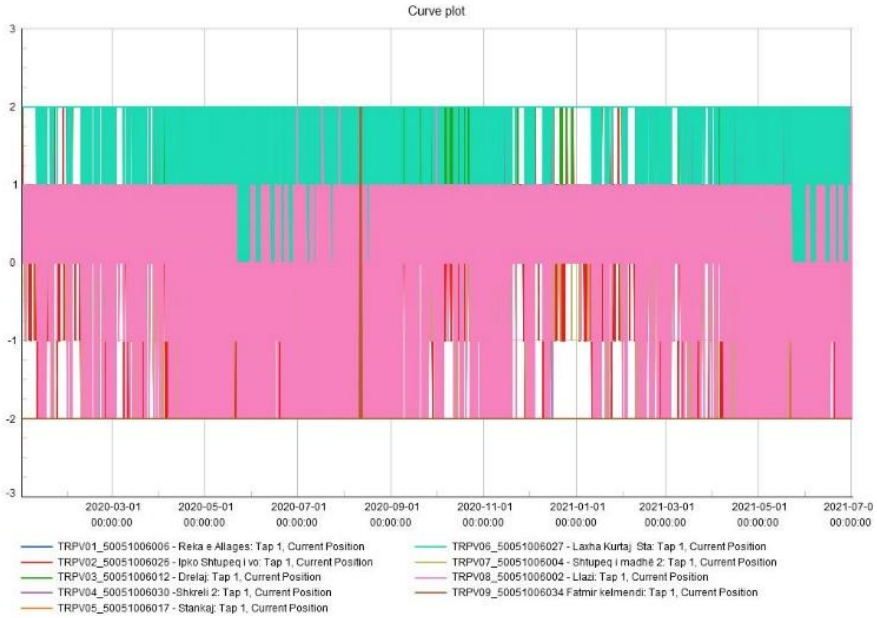


Figure 80. Q-D Analysis with ATP – Tap Positions.

Next, on Figure 81 we analyze the voltages in all busbars where PVs are connected, which are our busbars of interest since they are the most critical ones.

As the voltage results show, there are no more overvoltages of +10%Un. Meaning that all voltages are under 1.1 [p.u]. In the first case where E-OLTCs were not employed, we had 2,404 instances where voltage increased more than 1.1 [p.u].

There are undervoltages as well, and they will be addressed with re-modelling the feeder where four LVRs are installed at various parts of the feeder. The goal in this case is to combine LVRs and E-OLTCs. LVRs will improve the undervoltages throughout the line that are experienced in times of no PV production and large aggregate loads. On the other hand E-OLTCs will have the same function as in the previous analysis, which is decreasing the voltage next to high levels of distributed generation. Let's first confirm numerically the instances of overvoltages when we do employ ATPs.

```
import pandas as pd

RU_raw_withEOLTC = pd.read_csv('QD_Rugova_withEOLTC.csv', delimiter=';')

RU_voltages_withEOLTC = RU_raw_withEOLTC.iloc[:,-18:-9]

ru_withEOLTC_violations_5percent = (RU_voltages_withEOLTC>1.05).sum().sum()
print(ru_withEOLTC_violations_5percent)
2432

ru_withEOLTC_violations_10percent = (RU_voltages_withEOLTC>1.1).sum().sum()
print(ru_withEOLTC_violations_10percent)
0
```

Code has verified once again the effects of E-OLTCs on the grid. From 2,404 violations of over 1.1 [p.u] to zero, and around 80% less violations of over 1.05 [p.u].

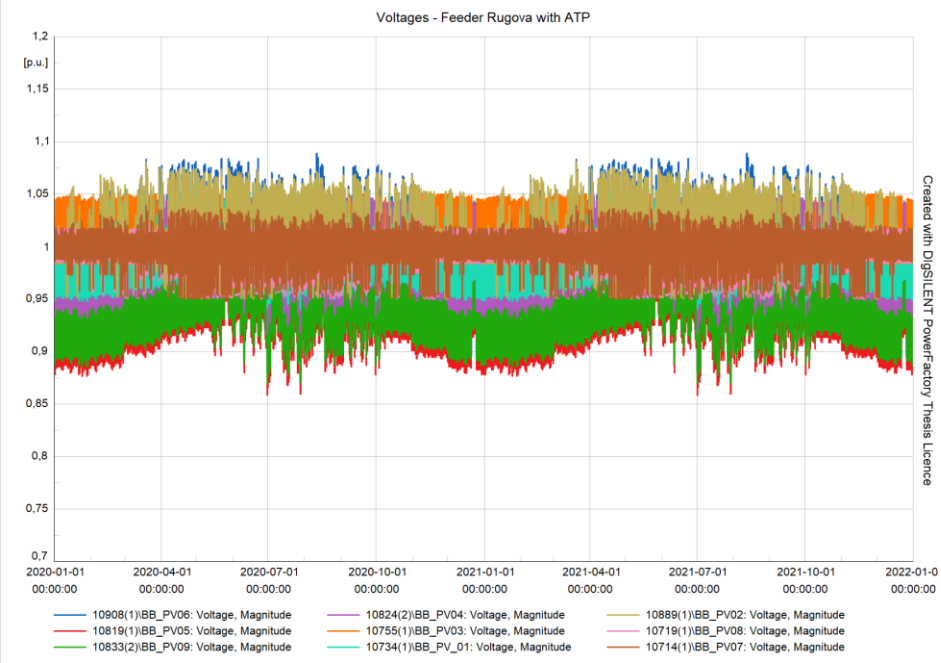


Figure 81. Voltages of feeder Rugova with ATP.

This concludes our Quasi Dynamic Analysis on our basic model.

5.4.1. Combining LVRs and E-OLTC

Now, let's take an extra step and do an additional analysis where the feeder will be re-modeled to include LVRs, as explained earlier.

In CS-I this was not performed as Feeder Sub-Urb did not have problems with undervoltages like Feeder Rugova does in CS-II. Running the Q-D simulations process, load characteristics, and PV characteristics do not change, hence most of the steps are omitted in this explanation.

To run the LVR and E-OLTC combo analysis, modifications must be done in the network because the LVRs must be modelled and then their tap positions must be saved for later use to train any eventual machine learning algorithm, which will not be done at this time.

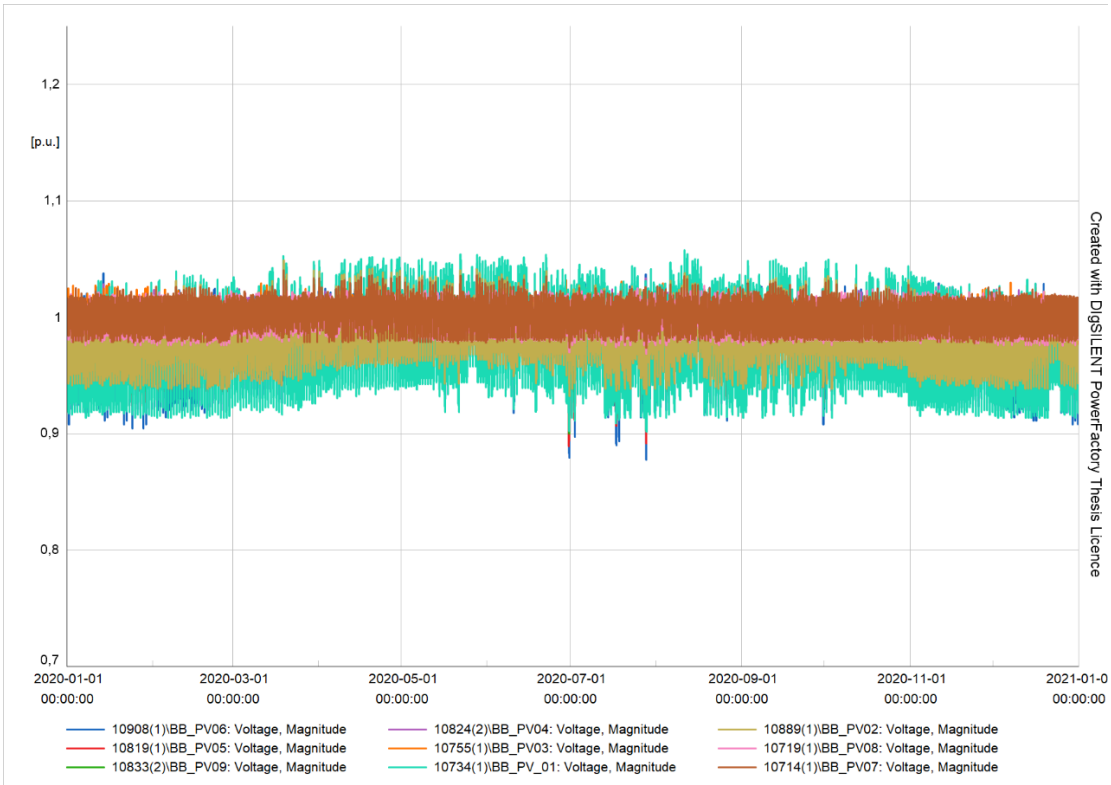


Figure 82. Voltages of Rugova when combining E-OLTCs and LVRs (first year) [p.u].

Figure 82 is the equivalent of Figure 81. The difference between the two is the addition of LVRs as means of augmenting undervoltages. This proves that for longitudinal feeders, the best approach is to combine E-OLTCs and LVRs. This combination makes it possible to regulate the voltage in such a manner that both overvoltages and undervoltages are taken care of.

This combo option is more expensive and harder to deploy. If installing this combination of LVRs and E-OLTCs is not feasible and a decision must be made to choose only one of the two, the right decision is E-OLTCs, as far as these types of feeders are concerned.

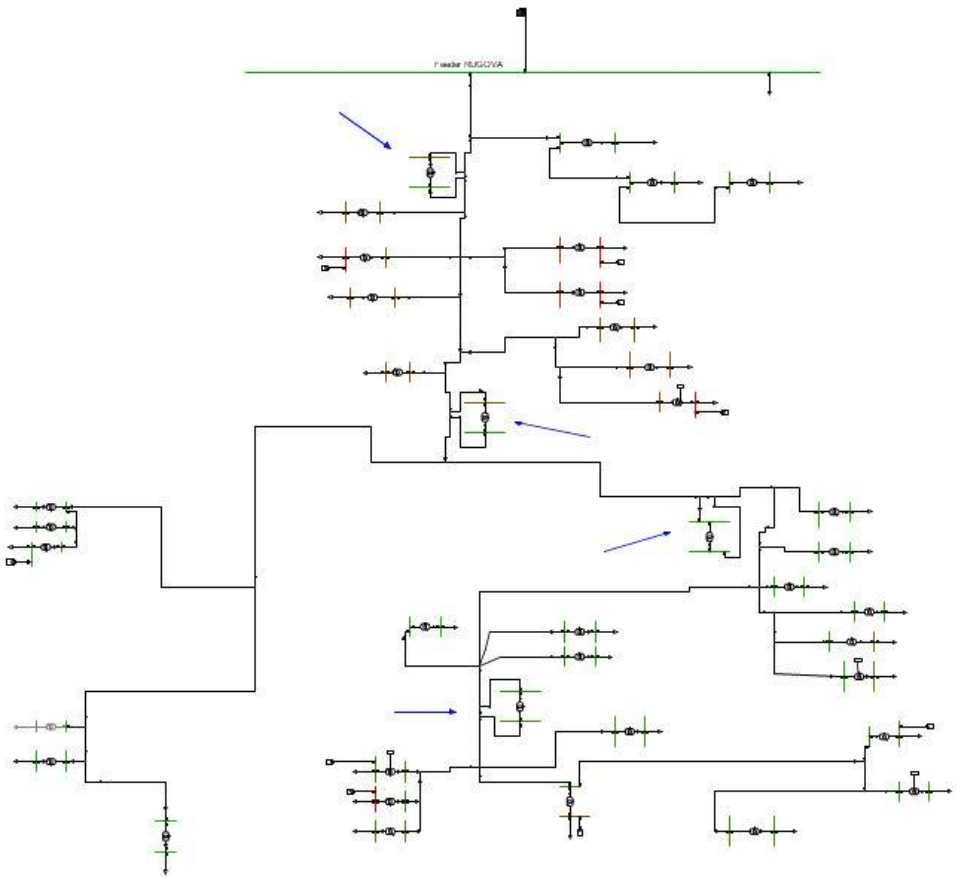


Figure 83. LVR and E-OLTC combination for voltage regulation in Rugova feeder.

If a DNN algorithm is to be trained with this new model of the feeder, the only difference would be to add 20 columns (4·5) in the labels/outputs. Since 4 LVRs are added and they each have 5 possible tap positions, 20 tap positions in total should be added in the outputs, translating to 20 extra neurons on the output layer. Nothing else besides the model and number of output neurons changes with respect to the previous case where LVRs were not included.

5.5. Cleaning, re-organizing, and filling the missing data points

The tap position results are exported to '.csv' or some other supported format and are ready to be cleaned for the ML phase. Similarly, to the first studycase but with fewer explanations, we will check for missing data, fill the missing data if there is any, re-organize and clean the needed data, and finally save it.

```
import pandas as pd

RU_raw = pd.read_csv('QD_Rugova_withEOLTC.csv', delimiter=';')
```

Checking for missing data:

```
(RU_raw == 'NaN').sum().sum()
0
```

There are zero missing data, meaning there is no need to fill any data and we can move forward with data preparation.
In case our columns are not sorted rightfully, we need to do that in a manner such as extracting columns will be easy to do with little code. That means our columns of loads, PVs, and outputs, are not mixed with each other, rather, go one after the other. Before saving the results from PowerFactory, it is always a good idea to make sure that the parameters are named in some order that makes it easy to identify. If that is done, we just need to sort the columns alphabetically as follows:

```
RU_raw = RU_raw.reindex(sorted(RU_raw.columns), axis=1)
```

At this point, our dataframe must contain only the outputs (labels), which are tap positions, and the inputs (features), which are the following:

- Loads (Active Power)
- Loads (Reactive Power)
- PV Outputs (Active Powers)
- PV Outputs (Reactive Powers)

Any other column in the dataframe must be deleted at this point, or the results of the ML algorithm may be unpleasant.

5.6. Dividing and exporting the dataset features and labels

This is the final step before starting the Machine Learning part. By now, the data should be analyzed, cleaned, sorted, and complete. For any ML algorithm, the features (inputs) and outputs (labels) need to be saved in such a manner that allows them to be imported in a non-contaminated way. The safest way, as done here, is to simply divide them earlier and save them in completely different files, in any of the supported formats.

For Rugova feeder, we will have the following layout of columns:

- 35 columns for active powers of loads
- 35 columns for reactive powers of loads

- 9 columns for PVs active power outputs
- 9 columns for PVs reactive power outputs
- 9 columns for Transformers Tap Positions

Table 17 represents the prepared dataframe of our full dataset, including both inputs and outputs. This dataframe has a size of 17520 × 105.

	PLoad01	QLoad01	…	PV01_P	PV01_Q	…	TR-PV01	…
0	57	34	…	0	0	…	-1	…
1	50	30	…	0	0	…	-1	…
2	47	28	…	0	0	…	-1	…
3	47	28	…	0	0	…	-1	…
4	50	30	…	0	0	…	-1	…
…	…	…	…	…	…	…	…	…

Table 17. Final data - inputs and outputs.

When dividing the features (inputs) and labels (outputs), we know that we can do that by dividing Table 17 across the minus 9th column. In other words, if we take out the last 9th columns and save it as outputs ('y.csv') and save the rest of the table as inputs ('X.csv').

```
X = RU_raw.iloc[:,:-9]
X.to_csv('X.csv')

y = RU_raw.iloc[:, -9:]
y.to_csv('y.csv')
```

As explained for Sub-Urb feeder, we need to modify the outputs now that they are saved independently. That happens because we will be using Sigmoid function in the last layer of neurons, which has binary outputs, so either zero or 1. So, for each transformer, we will use 5 neurons in the output layer, each one showing the probability of that tap position of that transformer to be the correct one.
So, as explained in 3.2.5, the number of outputs will be $N_{nntap} \cdot N_{transformers.}$ That is 5·9=45 neurons (outputs) in the output layer. This topology was explained in Figure 35.

The way to code it is by using binary values, for example, tap position (+1) is correct, the 5 neurons corresponding to the transformer in question will be [0,0,0,1,0].
With this change, there will be 45 outputs instead of the initial 5. After this step is complete, we can re-save 'y.csv' and rewrite the original file.

At this point, everything is ready to start to build the Neural Networks.

5.7. Coding the Deep Neural Networks in Python

Features and labels are saved in separate files and are ready for use. The tools and steps of our DNNs code will be the same as in the Sub-Urb feeder because our problem application is identical. For that reason, in this case, both models will be treated with less explaining.
First, importing the relevant libraries.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

Next, we use the features and labels that were saved on the previous section and make Pandas dataframes of them.

```
X = pd.read_csv('X.csv')
y = pd.read_csv('y.csv')
```

To avoid overfitting, the data must be shuffled and then divided in a training set and a test (validation) set. The ratio of the two this time will be 80/20.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=13)
```

These operations are done using function 'train_test_split' from 'sklearn' library, module 'model_selection'. The shuffling has to be random, which is why the user chooses a seed, in this case it is 13.
Since the inputs have different magnitudes, they must be scaled in a manner that makes it easier for the DNN to figure out the weights of neurons, which in turn gives better results. For this step, library 'sklearn' comes in handy again. This time, 'MinMaxScaler' function of module 'preprocessing'.

```
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
scaler.fit(X_train)


X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

It is important to only fit to the training dataset 'X_train', and not to the testing set

'X_test', as we supposedly do not have any information on it, as we will not have information on future sets we will have to predict. After fitting to the training set, we can now transform both the training set and the testing set.
Now comes the Network itself.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(Dense(88, activation='relu'))
model.add(Dense(75, activation='relu'))
model.add(Dense(58, activation='relu'))

model.add(Dense(45, activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy')
```

With the help of Tensorflow and Keras the basic Neural Network is built very fast. The number of neurons in all layers has of course changed from the previous study-case. In the first layer there are 88 neurons, which coincides with the number of inputs, although it is not mandatory. For the hidden layer, many 'constellations' have been tried, and the one shown above proved most successful. As for the last layer, it must have the same number of neurons as there are outputs.
CS-I already covered what 'Sequential' and 'Dense' do. The activation functions, the optimizer, and loss function have not changed as they fit this kind of problem best. ReLU activation function is offered graphically in Figure 29, next to the Sigmoid function.
The model is ready for training at this point.

```
model.fit(x=X_train, y=y_train, epochs=250)
```

So, the model will try to fit its weights and biases in order to reach optimal results as the ones fed to it. The number of epochs is an arbitrary number – 250. Depending on the problem, the number of epochs can be higher or lower. On each epoch, the entire dataset will be run once, and through backpropagation, the weights and biases will be updated for the next epoch.
The loss function value is reported after every epoch is run.

```
Epoch 1/250
438/438 [==============================] - 1s 682us/step - loss: 0.1918
Epoch 2/250
438/438 [==============================] - 0s 663us/step - loss: 0.1152
Epoch 3/250
438/438 [==============================] - 0s 678us/step - loss: 0.1094
Epoch 4/250
438/438 [==============================] - 0s 660us/step - loss: 0.1064
Epoch 5/250
438/438 [==============================] - 0s 654us/step - loss: 0.1045
…

Epoch 246/250
438/438 [==============================] - 0s 677us/step - loss: 0.0848
Epoch 247/250
438/438 [==============================] - 0s 678us/step - loss: 0.0846
Epoch 248/250
438/438 [==============================] - 0s 695us/step - loss: 0.0847
Epoch 249/250
438/438 [==============================] - 0s 679us/step - loss: 0.0845
Epoch 250/250
438/438 [==============================] - 0s 700us/step - loss: 0.0846
```

Only the first 5 and the last 5 epochs are shown for space-saving reasons.
In the first epoch, one can already note that the loss function decreases from 0.1918 to 0.1152, that happens due to the first update of weight and biases, which is usually the most beneficial one. Next, we graph the loss function with the number of epochs in the other axis, to better visualize the training procedure.

```
import matplotlib.pyplot as plt

loss =pd.DataFrame(model.history.history)
plt.plot(loss)
```

Figure 84 shows a steep decline of the loss function initially, afterwards it tends to be constant. The learning curve here is even more natural than in Sub-Urb feeder, understandable when considering the fact that Rugova has real data and not randomly generated. DNNs are robust enough even for feeder Rugova, and it is probably safe to say that they are robust enough for any MV feeder, in a largely feasible computational time and memory consumption.

Now is time to numerically evaluate the testing data and the training data. Their similarity in values is a sign of a well performing algorithm.

```
model.evaluate(X_test, y_test, verbose=0)
0.027296315878629684

model.evaluate(X_train, y_train, verbose=0)
0.024843977764248848
```

They are very similar in value, and even more importantly, low values. This tells that our algorithm is well trained but also that it has not overfit only a certain

dataset, rather it is performing well even on the test set that is used for validation.

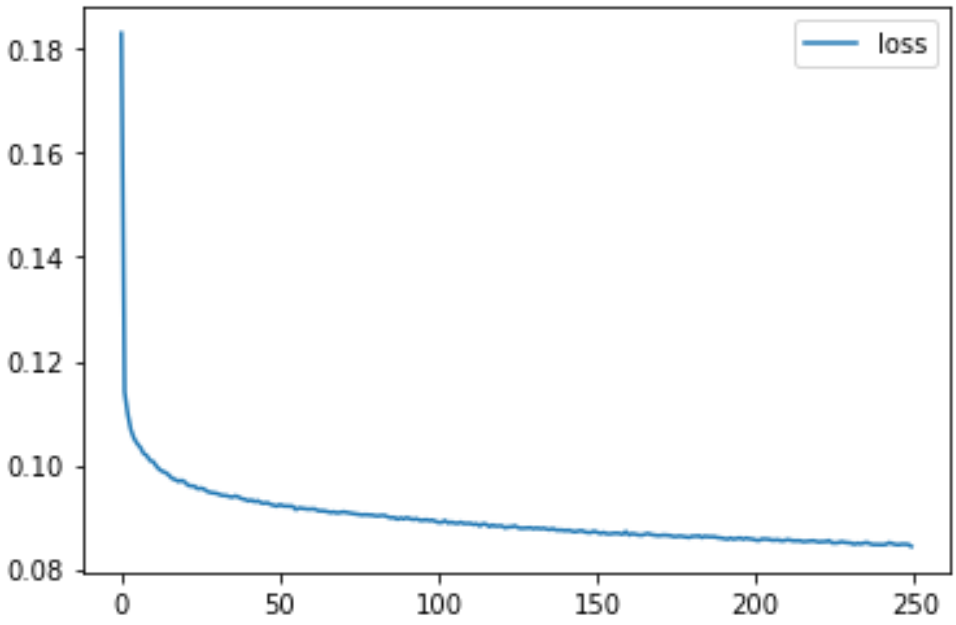


Figure 84. Loss function as Epochs increase.

Although this model can pass as a complete one, next, we will make some improvements in the model in order to achieve even better results. The modifications are the same ones as in CS-I, namely Dropouts and Early Stopping.

Dropouts try different and random topologies by switching random neurons on and off, then measuring the results of each configuration. The number of neurons that will be stopped is given in relative terms as a percentage of all neurons of that layer. On the other hand, Early Stopping keeps track of the loss function of both the training set and the validation set simultaneously, and if it notices that no improvements were made after a certain number of epochs (patience), it will stop the training. Early stopping will not only save time and memory, but it may also prevent overfitting. Now that the modifications are explained, here is the full improved model code.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

```
X = pd.read_csv('X.csv') #importing the features (inputs)
y = pd.read_csv('y.csv') #importing the labels (outputs)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=74)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()

model.add(Dense(88, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(75, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(45, activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy')

from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', mode='min',verbose=1,
patience=25)

model.fit(x=X_train, y=y_train, epochs=600,
validation_data=(X_test,y_test),callbacks=[early_stop])
```

And the results:

```
Epoch 1/600
384/384 [==============================] - 2s 4ms/step - loss: 0.2714 -
val_loss: 0.1404
Epoch 2/600
384/384 [==============================] - 1s 3ms/step - loss: 0.1551 -
val_loss: 0.1275
Epoch 3/600
384/384 [==============================] - 1s 3ms/step - loss: 0.1421 -
val_loss: 0.1223
Epoch 4/600
384/384 [==============================] - 1s 3ms/step - loss: 0.1348 -
val_loss: 0.1191
```

```
Epoch 5/600
384/384 [==============================] - 1s 3ms/step - loss: 0.1310 -
val_loss: 0.1173
...
Epoch 83/600
384/384 [==============================] - 1s 3ms/step - loss: 0.1125 -
val_loss: 0.1042
Epoch 84/600
384/384 [==============================] - 1s 3ms/step - loss: 0.1133 -
val_loss: 0.1041
Epoch 85/600
384/384 [==============================] - 1s 4ms/step - loss: 0.1125 -
val_loss: 0.1039
Epoch 86/600
384/384 [==============================] - 1s 4ms/step - loss: 0.1123 -
val_loss: 0.1035
Epoch 00086: early stopping
```

The sophisticated model has stopped training on epoch number 86. That means, the algorithm has not seen any significant improvements since epoch 61. In the figure showing the loss function, now we can include the validation set as well.

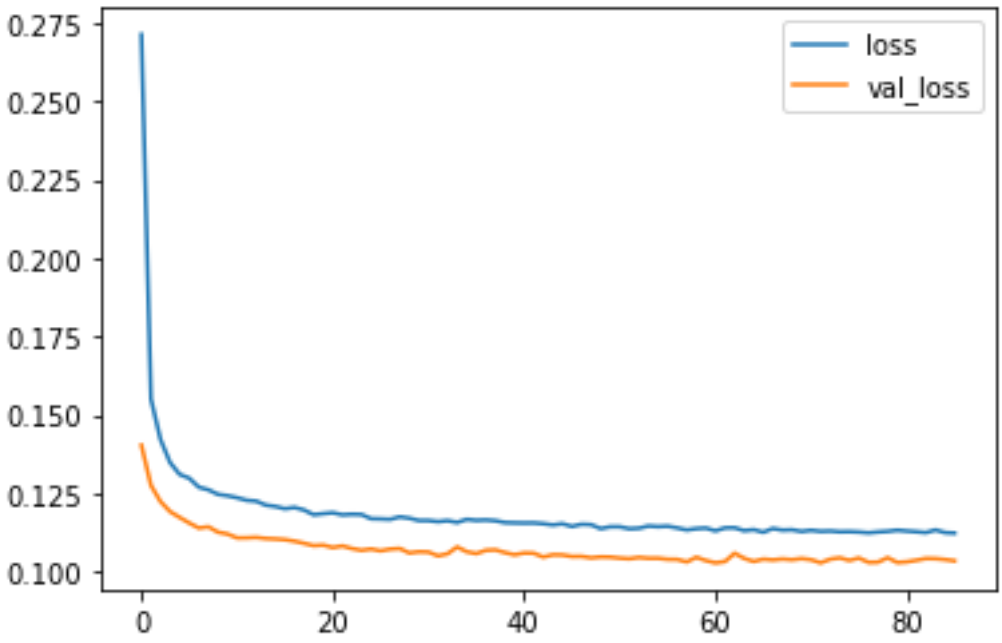


Figure 85. Loss function of the training set and validation set as epochs increase.

Figure 85 shows a loss function that decreases steeply in the very first few epochs. After some time, small noise is visible, however, the loss function of the training set and the validation set do not change drastically.

Since there is a large amount of randomization in these algorithms, the results will be slightly different every time they are run. Same goes for the number of epochs

when Early Stopping is employed. However, the final loss function values are pretty much the same.

A final interesting point to notice is that our primitive model of the DNN has yielded better results than our more sophisticated model. That could be for various reason that pertain to this specific dataset, but most importantly, it happens because the epochs go up to 250, instead of stopping at 86.

Similar to the last section of section 4.7, after the DNN model is finalized, it will be fed around 5000 training examples that contain inputs the model has not seen or trained on beforehand. The outputs or predictions of the DNN, which are the tap positions, will then be used in PowerFactory, thus inversing the process. So, now, a Quasi-Dynamic simulation of around 5000 hourls will be run with the usual inputs which are loads and PVs, and this time also with tap positions as inputs. Throughout the Q-D analysis, the voltages will be monitored, to evaluate the performance in electrical terms instead of data science terms (binary cross-entropy loss function).

Figure 86. Voltages of Rugova feeder when taps are controlled by the DNN model [p.u].

In the second feeder, as expected, the results show that the DNN does pretty much as good of a job as PowerFactory in determining the right tap positions. That

conclusion is reached by comparing Figure 86 and Figure 81, where the former shows the voltages in the feeder when DNN model does the predictions, while the latter represents the voltages of the feeder when PowerFactory adjusts the transformer's taps automatically.

It is important to keep in mind that feeder Rugova has had significant problems with both undervoltages and overvoltages before using E-OLTCs. That is why PowerFactory nor the DNN model is able to keep the voltage range between 0.95 [p.u] and 1.05 [p.u] at all times. However, always according to the local Distribution Code (Kodi i shpërndarjes) [20], the allowed voltage range in MV networks is between 0.9 [p.u] to 1.1 [p.u], with some exceptions when it is allowed to go down to 0.85 [p.u].

5.8. Additional results and visualizations

Due to our studycase type, not many worthy visualizations can be illustrated here. Just to take an example of utilizing libraries such as 'seaborn' or 'matplotlib', we can once again make a countplot of a certain transformers tap position being used or not throughout the 2-year simulation. For example, we analyze how often TR7 was on tap position (-2), and how often it was on the other 4 positions.

```
(y['TR7[tap(-2)]']==1).sum()
4136

(y['TR7[tap(-2)]']==0).sum()
13384
```

The count shows that there are 4,136 instances where TR7 operates at tap position (-2) and 13,384 instances where it operates at positions other than (-2), so positions [-1,0,1,2].

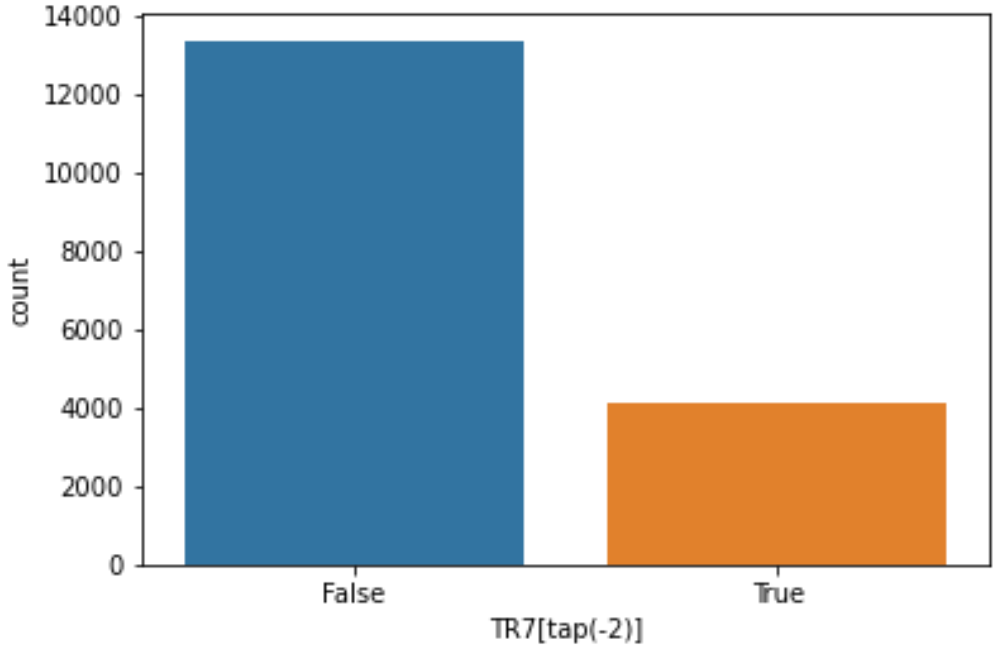


Figure 87. Countplot of TR7 operating at tap position (-2).

Results in Figure 87 are realistic, as they claim that TR7 operates at tap position (-2) around 24% of the time.

Seaborn library can be used for various types of countplots, scatterplots, etc. As previously said, they are not as helpful in studycases like this one, where the outputs are mainly zeros and ones.

That is all as far as our studycases are concerned.

6 Conclusion and future developments

The two studycases carried out in this study reach two conclusions. First one is the fact that E-OLTCs can have immense impact in voltage regulation of MV networks with high penetration of DGs, even without the help of some sort of reactive power control method. The second conclusion is that ML and DNNs can be used to find the correct tap positions of all transformers in a feeder, providing a new approach instead of the classical approach of software modelling and running load flows.

The results show that DNNs can learn fast and in such a manner that voltage constraint violations are almost eliminated or completely eliminated.

This study is focused mainly on tap changing transformers, especially those E-OLTCs of transformers directly connected to the LV busbars where photovoltaic systems inject energy. Line Voltage Regulators were deemed as unfit for voltage regulation in longitudinal feeders due to the fact that overvoltages are mostly encountered in the immediate busbars where DGs inject, and not the ones further along the line, which is where LVRs are installed usually. In cases of feeders with shorter side branches, or when the DGs are located close to the main branch of the feeder, LVRs may prove worthy.

Always depending on the feeder's topology, E-OLTCs may be effective with overvoltages, however, they may not help with undervoltages if the feeder already has large voltage drops at times when DGs are not producing, such as night-hours. In this case, a new Q-D simulation was run combining E-OLTCs and LVRs, and the results were improved. E-OLTCs fixed the overvoltage issues as they are connected directly to the LV busbars where DGs inject. On the other hand, LVRs fixed the undervoltage issues that occurred throughout the longitudinal feeders, as they are installed in the main branch of the feeder. That proves the initial hypothesis of this thesis which suggests that the best solution for voltage regulation is in fact a combination of multiple methods, such as LVRs and E-OLTCs.

To the best of the author's knowledge, the novice element of this study is the approach of performing Quasi-Dynamic analysis to obtain as many correct outputs as possible for tap positions, which later are used to train Deep Neural Networks.

In order to obtain as much data as possible, the Q-D simulations are run without including any outages or maintenance. In the future, upon finding relevant data of multiple years, outages and maintenance could be included for more realistic results.

The method of combining Q-D analysis and Deep Neural Networks showed promising signs as it took less than 100 epochs to reach optimal values of the loss functions, even with simplistic 4-layer DNN models. The trained model predicts tap positions of not just one, but of all transformers of the feeder that are equipped with automatic tap changing technology. That mostly includes transformers connected directly to PV systems but also any potential LVRs installed elsewhere in the feeder.

The first studycase shows that the loss function was already significantly lower than the loss function of the second studycase, that makes perfect sense provided that the first studycase had a much lower number of voltage constraints violations even before using ATPs.

The models were computationally cheap, both in terms of time and memory usage. As far a potential wide-scale deployment of this method is concerned, if we refer to multiple papers on this topic, communication could pose as the biggest problem. However, in recent years, manufacturers of PV inverters have developed cloud-based systems where reading the live metering, historical energy injection, controlling the live production of the plant, etc., is as easy as using a smartphone application to read the news.

An initial idea that DSOs could use for a potential pilot project is to obligate new PV producers to only inject energy through LV/MV transformers that are equipped with E-OLTCs, also obligate them to make the live meter readings accessible to the control room/dispatching center. The control room already has access to live meter readings of loads connected to each transformer in the feeder. If these two inputs are combined, it would emulate our studycases, and if a DNN model of the feeder is already trained and saved in the control room machines, the E-OLTCs throughout the grid could be controlled automatically with zero human inputs and with very high accuracy. The DNN model could be programmed to learn nonstop from new inputs/outputs as well.

As for the complexity of developing and installing a software in control room servers, it most likely is a negligible problem. That software will only have to be programmed to read the smartmeters of loads and PV productions in real time and adjusts the E-OLTCs according to the Deep Neural Network model that is already trained. The software could be sophisticated with a ‘wait time’ between E-OLTC

manipulations in order to lower frequent voltage fluctuations and wear and tear of E-OLTCs.
A control room of a HV/MV substation with 30-50 feeders should easily afford a DNN model for each of the feeders, in terms of time and random access memory (RAM) required. This definitely constitutes a cheaper and more efficient alternative to installing and paying for expensive software licenses, training staff to model networks run power flow analysis using that software, and perhaps even having to hire engineers instead of technicians in HV/MV substations where the control room is located.

As far as PV productions are concerned, in case access to live metering is not possible, another approach could be to have the model predict the PV outputs by using irradiation measurements data, which is sufficiently accurate and easier to do than what it may sound, even using free tools/databases.

In cases where the DSO has not installed smartmeters, hence has no digital live reading of the equivalent loads at transformers throughout the feeder, a new study similar to this one may be helpful. The only difference in that case would be the use of aggregate feeder load as an input, instead of using all loads independently. It is likely that the results would not be as good as in the actual case, that is because the physical positioning of loads in the feeder plays a significant role, and it would not be taken into consideration in this second case.

The aforementioned machine learning model could be enriched further by also imposing constraints on reactive power control devices and including other voltage regulating methods. Thus, instead of just E-OLTCs, combining multiple voltage regulation methods with machine learning.

7 Bibliography

[1] "Directive (EU) 2018/2001 of the European Parliament and of the Council of 11 December 2018 on the Promotion".

[2] J. Lopes, N. Hatziargyriou, J. Mutale, P. Djapic and N. Jenkins, "Integrating distributed generation into electric...".

[3] V. V. T. R. B. J. N. A. Woyte, "Voltage fluctuations on distribution level introduced by photovoltaic systems," *IEEE Transaction on Energy Conversion v.21, n.1,* 03 2006.

[4] J. R. A. F. Katiraei, "Solar PV Integration Challenges," *IEEE Power and Energy Magazine vol 9, n.3,* 04 2011.

[5] E. A. A. B. C. B. a. S. F. A. Bosisio, "A milp approach to plan an urban electric distribution with an H-shaped layout.," in *IEEE Eindhoven PowerTech,* 2015.

[6] A. B. C. B. a. E. A. A. Bosisio, "Urban distribution network planning with 2-step ladder topology considering joint nodes," in *IEEE Manchester PowerTech*, 2017.

[7] "https://www.iea.org/data-and-statistics/charts/evolution-of-solar-pv-module-cost-by-data-source-1970-2020".

[8] ENTSO-E, "Parameters related to voltage - ENTSO-E guidance document for national implementation for network codes on grid connection," 2016.

[9] K. Z. a. B. M. I. S. Abapour, "Dynamic Planning of Distributed Generation Units in Active Distribution Network," 2015.

[10] "https://search.abb.com/library/Download.aspx?DocumentID=DEABB%205153%2015%20en&LanguageCode=en&DocumentPartId=&Action=Launch".

[11] "https://new.abb.com/events/cired/line-voltage-regulator#:~:text=A%20line%20voltage%20regulator%20automatically, the%20nominal%20voltage%20Un.".

[12] "https://www.hitachienergy.com/offering/product-and-system/transformers/dry-type-transformers/line-voltage-regulators/line-voltage-regulator-for-medium-voltage-grids".

[13] G. R. C. M. a. P. Bauer, ", "Design of a Power-ElectronicAssisted OLTC for Grid Voltage Regulation",," in *IEEE*, 2015.

[14] R. A. a. X. T. J. H. Shuttleworth, "A Fast Response GTO Assisted Novel Tap-Changer," in *IEEE Transactions on Power Delivery,* 2001.

[15] A. S. T. B. C. T. M. P. K. F. a. T. D. Geibel, "Active, intelligent, low voltage networks - Concept, realisation and field test results," in *Electricity Distribution (CIRED 2013)*, Stockhol,, 2013.

[16] A. P. a. L. Chao, "Voltage Control in LV Networks: An initial investigation," in *Innovative Smart Grid Technologies Conference Europe IEEE PES*, Istanbul, 2014.

[17] R. &. S. J. &. K. R. Bansal, "Machine learning and its applications: A Review. 6. 1392-1398.," 2020.

[18] L. R. C. T. G. M. F. E. F. E. L. J. F. L. H.-H. Frederico A.C. Azevedo, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain".

[19] "www.alexlenail.me/NN-SVG/index.html.".

[20] "http://ero-ks.org/Vendimet/Shqip/2014/Kodi_i_Shperndarjes_2014.pdf".

[21] S. O. Duncan Glover, Power System Analysis and Design 6th edition, 2015.

[22] G. Latifi, Teknika e tensioneve të larta, Pristina, 1994.

[23] J. Momoh, Smart Grid Fundamentals of Design and Analysis, Jon Wiley and Sons, 2012.

[24] P. Kiameh, Grid Connected Solar Electric Systems The Earthscan Expert Handbook for Planning Design and Installation, McGraw and Hill, 2004.

[25] S. V. K. a. S. A. Khaparde, Transformer Engineering Design Technology and Diagnostics 2nd Edition.

[26] H. S. a. M. S. Sepasian, Electric Power System Planning Issues, Algorithms and Solutions.

[27] Bishop, Pattern Recognition And Machine Learning, 2006: Springer .

[28] S. B.-D. Shai Shalev-Shwartz, Understanding machine learning - from theory to algorithms, 2004: Cambdridge University Press.

[29] M. Nielsen, Neural networks and deep learning.

[30] S. G. Andreas C. Müller, Introduction to Machine Learning with Python A Guide for Data Scientists.

[31] Y. S. R. a. I. L. Tom Hope, Learning TensorFlow A Guide to Building Deep Learning Systems.

[32] S. P. Antonio Gulli, Deep Learning with Keras Implementing deep learning models and neural networks with the power of Python.

List of Figures

Figure 1. Classic network: Downstream power flow. 4
Figure 2. Modern network operation: Upstream power flows 5
Figure 3. Prizren-Zhupa feeder in the Sharr Mountains. 8
Figure 4. LVR installed along the feeder - inactive. 9
Figure 5. LVR installed along the feeder – activated. 9
Figure 6. Impact of a single LVR installed in the feeder. 10
Figure 7. Feeder with four LVRs. 11
Figure 8. Feeder Rugova voltage profile, with large loads and small DGs [p.u]. ... 12
Figure 9. Feeder Rugova voltage profile when loads are large and small DGs – with LVRs installed [p.u] 12
Figure 10. Line Voltage Regulator single line diagram 13
Figure 11. LVR regulation of voltage. 14
Figure 12. Drawing of Tap Changer's simplified work principle. 15
Figure 13. E-On Load Tap Changer scheme, transformer, bidirectional switch, and protection circuit 17
Figure 14. Linear Regression 20
Figure 15. Mathematics behind LR. 21
Figure 16. Cost function graphic representation. 22
Figure 17. Gradient descent principle of work. W represents θ. 22
Figure 18. Choosing Alpha: Learning Rate of Gradient Descent 23
Figure 19. Gradient descent being applied to lower a Cost Function. 24
Figure 20. Linear Regression (green) vs Polynomial Regression (red) fitting a random dataset. 25
Figure 21. Regression model: a) Underfitting, b) Good fit, c) Overfitting. 26
Figure 22. Brain neuron. 27

Figure 23. Basic Artificial Neuron topology. 28
Figure 24. Perceptron trying to determine whether to go out or not. 30
Figure 25. AND gate realized using a perceptron. 31
Figure 26. Neural networks with single output. 32
Figure 27. Sigmoid function. 34
Figure 28. Hyperbolic tangent activation function. 36
Figure 29. Rectified Linear Unit (ReLU) activation function. 36
Figure 30. Leaky Rectified Linear Unit (Leaky ReLU) activation function. 37
Figure 31. Derivatives of some popular activation functions. 38
Figure 32. Neural Networks algorithms. 39
Figure 33.Single output NN with binary classification. 40
Figure 34. Neural Network determining: Cat or Dog. 41
Figure 35. Multi-output Deep Neural Network with binary classification. 44
Figure 36. Neural Network architecture for voltage regulation, adapted for a single transformer. 44
Figure 37. Simplified flowchart of the steps carried out for voltage regulation. 48
Figure 38. Tap Settings in TypTr2. 49
Figure 39. Tap Changer settings in ElmTr2 in PowerFactory. 50
Figure 40. Running a Quasi-Dynamic Analysis main window. 51
Figure 41. Exporting Quasi-Dynamic analysis results – Basic Data. 52
Figure 42. Exporting Quasi-Dynamic analysis results – Advanced Options. 52
Figure 43. Feeder Sub-Urb 20 [kV] zoomed in a portion. 53
Figure 44. Feeder Sub-Urb 20 [kV] topology. 54
Figure 45. EU Commission's PVGIS tool – Sub-Urb feeder irradiation. 59
Figure 46. PV01 Active power output characteristic [kW]. 62
Figure 47. Unit Load Profile Ln. 63
Figure 48. Active Power characteristic of Load 12 in Sub-Urb feeder [kW]. 65
Figure 49. PV Outputs of Sub-Urb feeder (first year) [kW]. 66
Figure 50. PV Outputs of Sub-Urb feeder (second year) [kW]. 66
Figure 51. PV Output Sub-Urb feeder of random 5 days [kW]. 67

Figure 52. Loads (active power) of feeder Sub-Urb [kW]. 67
Figure 53. Loads (active power) in Sub-Urb feeder, zoomed in [kW]. 68
Figure 54. Loads (reactive) of Sub-Urb feeder [kVAr]. 68
Figure 55. Loads (reactive) of Sub-Urb feeder, zoomed, [kVAr]. 69
Figure 56. Q-D simulation without Automatic Tap changers - Tap positions 69
Figure 57. Voltages in Sub-Urb feeder without ATPs [p.u]. 71
Figure 58. Q-D Analysis with ATP – Tap Positions. 72
Figure 59. Voltages of feeder Sub-Urb with ATP [p.u]. 73
Figure 60. Loss function as Epochs increase. 80
Figure 61. Loss function of the training set and validation set as epochs increase. 83
Figure 62. Voltages in Feeder Sub-Urb when tap positions are set according to DNN predictions [p.u]. 84
Figure 63. Countplot of TR10 operating at tap position (-1). 85
Figure 64. Feeder Rugova 20 [kV] upstream flow of energy 88
Figure 65. Feeder Rugova 20 [kV] topology. 89
Figure 66. Feeder Rugova 20 [kV] topology of a random portion. 89
Figure 67. EU Commission's PVGIS tool – Rugova feeder irradiation. 94
Figure 68. PV7 output characteristic – Rugova [kW]. 96
Figure 69. Active Power characteristic of Load 18 in Rugova feeder [kW]. 98
Figure 70. PV Outputs Rugova feeder (first year) [kW]. 99
Figure 71. PV Outputs Rugova feeder (second year) [kW]. 99
Figure 72. PV Output Rugova feeder zoomed. 100
Figure 73. Loads (active power) of feeder Rugova. 100
Figure 74. Loads (active power) in Rugova feeder, zoomed in. [kW] 101
Figure 75. Loads (reactive) of Rugova feeder. [kVAr] 101
Figure 76. Loads (reactive) of Rugova feeder, zoomed in. [kVAr] 102
Figure 77. Q-D simulation without Automatic Tap changers - Tap positions 102
Figure 78. Voltages in Rugova feeder without ATPs [p.u]. 104
Figure 79. Q-D Analysis with ATP – Tap Positions, zoomed in. 104
Figure 80. Q-D Analysis with ATP – Tap Positions. 105

Figure 81. Voltages of feeder Rugova with ATP. 106

Figure 82. Voltages of Rugova when combining E-OLTCs and LVRs (first year) [p.u]. 107

Figure 83. LVR and E-OLTC combination for voltage regulation in Rugova feeder. 108

Figure 84. Loss function as Epochs increase. 114

Figure 85. Loss function of the training set and validation set as epochs increase. 116

Figure 86. Voltages of Rugova feeder when taps are controlled by the DNN model [p.u]. 117

Figure 87. Countplot of TR7 operating at tap position (-2). 119

List of Tables

Table 1. ABB's Catalog data for MV LVR model DEABB 5153. 15
Table 2. Logical gate AND 31
Table 3. Photovoltaic systems installed in Feeder Sub-Urb. 55
Table 4. Loads of Sub-Urb feeder 20 [kV] 56
Table 5. Feeder Sub-Urb 20 [kV] Lines data. 57
Table 6. Feeder Sub-Urb 20 [kV] Transformer data. 58
Table 7. PV characteristics - Sub-Urb Feeder. 61
Table 8. PV outputs of a random period - Sub-Urb feeder. 61
Table 9. Matrix of loads in Sub-Urb feeder 64
Table 10. Final data - inputs and outputs 75
Table 11. Photovoltaic systems installed in Feeder Rugova 10 [kV]. 90
Table 12. Feeder Rugova 10 [kV] lines data. 91
Table 13. Feeder Rugova 10 [kV] loads data. 92
Table 14. Transformer data of Rugova feeder 10 [kV] 93
Table 15. PV characteristics - Rugova Feeder. 95
Table 16. Matrix of loads in Rugova feeder. 97
Table 17. Final data - inputs and outputs. 110

Acknowledgments

First of all, I would like to thank my supervisor, Prof. Alessandro Bosisio, for mentoring me on my master's thesis at Politecnico di Milano, thus, continuing our collaboration which started when he first supported me on my bachelor's thesis at University of Pristina.

I would like to express my gratitude towards my parents, Sabri and Nexhmije, to my brother Shkëlqim and to my sister Fjollë, and last but not least, to my girlfriend and colleague, Vjosë. Without their financial and moral support, I would probably not be able to pursue a master's degree at Politecnico di Milano. I owe everything to them.

I am also thankful to my professors, friends, and work colleagues, for helping me become a better academic and engineer.