UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DEVELOPMENT OF A DEEP LEARNING BASED ATTACK DETECTION SYSTEM FOR SMART GRIDS

PABLO AZNAR DELGADO

JUNIO 2019

TRABAJO DE FIN DE MÁSTER

Título:	DESARROLLO DE UN SISTEMA DE DETECCIÓN DE ATAQUES EN SMART GRIDS BASADO EN DEEP LEARNING
Título (inglés):	DEVELOPMENT OF A DEEP LEARNING BASED ATTACK DETECTION SYSTEM FOR SMART GRIDS
Autor:	Pablo Aznar Delgado
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MÁSTER

DEVELOPMENT OF A DEEP LEARNING BASED ATTACK DETECTION SYSTEM FOR SMART GRIDS

JUNIO 2019

Resumen

La red eléctrica está evolucionando hacia las smart grids, las cuales utilizan flujos bidireccionales de energía e información para crear una red de distribución de energía automatizada y distribuida. Esta mejora requiere la incorporación de un gran número de elementos a la red, los cuales contribuyen a la aparición de nuevas vulnerabilidades y posibles nuevos ataques. Así, la seguridad se ha convertido en uno de los temas más importantes en las smart grids.

Por esta razón, existe la necesidad de detectar estos ataques y así reducir la pérdida de dinero que causan. Como consecuencia, el objetivo de este proyecto es desarrollar un sistema que detecte automáticamente estos ataques utilizando técnicas de Deep Learning. Para ello, se llevarán a cabo las siguientes tareas:

- En primer lugar, es necesario estudiar el estado del arte actual de diferentes áreas: los tipos de ataques que las smart grids pueden sufrir, diferentes herramientas que nos permitan obtener datos de ellas y las técnicas y herramientas de Deep Learning que serán necesarias.
- Como no es posible obtener datos reales de las redes inteligentes porque son confidenciales, se utilizarán técnicas de simulación basada en agentes. En estas simulaciones se realizarán diferentes ataques y se generarán datos sintéticos.
- Una vez obtenidos los datos, se procesarán y se analizarán utilizando técnicas de Big Data. Además, se aplicarán algoritmos de Deep Learning para poder hacer inferencias sobre el comportamiento de la red.
- Finalmente, los datos obtenidos de la red neuronal se procesarán para detectar los ataques.

Este sistema permite automatizar el proceso de detección de ataques en smart grids, evitando en cierta medida la pérdida de dinero que supone el fraude eléctrico.

Palabras clave: smart grids, ataques, detección, deep learning

Abstract

The electrical power grid is evolving into a more modern electric grid, called smart grid. Smart grids use two-way flows of electricity and information to create an automated and distributed advanced energy delivery network. This improvement has required the addition of a large number of new elements, which contribute to the appearance of new vulnerabilities and therefore, possible new attacks. Thus, security has become one of the most important issues in terms of distribution and generation of energy.

For this reason, there is a need to detect these attacks and thus, reduce the loss of money that they cause. Consequently, the objective of this project is to develop a system that automatically detects these attacks using Deep Learning techniques. For that purpose, the following tasks will be carried out:

- First, it is necessary to study the current state of the art of different areas: the types of attacks that smart grids can suffer, different tools that will allow us to obtain smart grids data and the deep learning techniques and tools that will be necessary.
- As it is not possible to obtain real data from smart grids because it is confidential, Agent-Based Model Simulation will be used in order to simulate them. In this simulation different attacks studied previously will take place and synthetic data will be generated.
- Once the data is obtained, it will be processed and analyzed using big data techniques. In addition, deep learning algorithms will be applied to them in order to make inferences about the behavior of the grid.
- Finally, the data obtained from the neural network will be processed in order to detect the attacks.

This system provides the possibility of automating the process of detecting attacks on smart grids, avoiding the great loss of money to companies caused by electrical fraud.

Keywords: Smart Grids, attacks, detection, deep learning

Agradecimientos

Me gustaría agradecer a las personas que han hecho posible que pudiera acabar tanto este proyecto como la carrera.

A mis padres y a mi hermano, por apoyarme, comprenderme y animarme durante todos estos años.

A mi abuelo, por ser una gran inspiración y la razón por la que decidí comenzar esta carrera.

A Guillermo, Javier y Alba, con los que he compartido gran cantidad de momentos y con los que sé que siempre podré contar.

A mis compañeros del laboratorio, por el gran ambiente de trabajo que crean y por ayudarme en todas las dudas que iban surgiendo.

A mi tutor Carlos A. Iglesias, por darme la oportunidad de realizar este proyecto y por su ayuda durante la realización del mismo.

Contents

List of Figures

CHAPTER 1

Introduction

This chapter introduces the context where this project takes place. In addition, the objectives of the project are explained. Finally, the structure of the document is described.

1.1 Context

The current power grid is being modernized and smart grids are emerging. They have a large number of new components, such as new sensing and metering technologies, high-power converters, modern communications infrastructure, modern energy management systems, etc. All these components provide the possibility of collecting a large amount of data that was not possible to collect before [23].

All this new amount of information can be used for different purposes: better forecast of energy consumption, economic savings to the final costumer, better integration of renewable energy, integration of electric cars in the grid, etc. However, in spite of the advantages provided by all the data originated by smart grids, new vulnerabilities and new possible attacks emerge.

Therefore, security has become a very important aspect of smart grids due to the great amount of information handled. For this reason, a new need arises, tools that allow these attacks to be detected in order to save the great economic loss that fraud entails for electric companies.

In addition, nowadays deep learning techniques are gaining in popularity [46]. Deep artificial neural networks have won numerous contests in pattern recognition and machine learning.

Deep learning allows computational models composed of multiple processing layers to learn different representations of data with multiple levels of abstraction [32]. In addition, deep learning uses the backpropagation algorithm to discover and learn the structure of a dataset. This algorithm indicates how the internal parameters of a machine have to be changed, as these parameters are used to compute the representation in each layer from the representation in the previous layer. Deep learning methods have drastically improved the state of the art in different fields: speech recognition, visual object recognition, processing images, video, and audio, etc.

Consequently, there is a new need which is to detect the attacks that can suffer smart grids due to the large amount of information that they handle. In addition, deep learning techniques are becoming increasingly popular because of the good performance they offer in a variety of fields. For this reason, this project called *Develoment of a Deep Learning based Attack Detection System for Smart Grids* is carried out.

1.2 Project goals

The main objective of this project is to develop a system that automatically detects different attacks that smart grids may suffer using deep learning techniques.

For its development, this main objective has been divided into the following goals:

- First, it is necessary to study the current state of the art of different areas:

  On the one hand, regarding to smart grids, it is going to be studied how they work, the different vulnerabilities and types of attacks that they can suffer and different tools that allow obtaining data from them.

  On the other hand, deep learning techniques that will allow us to analyze the data for the detection of anomalies will be studied.

- As it is not possible to obtain real data from smart grids due to its confidentiality, Agent-Based Model Simulation will be used in order to obtain them. This will provide us with the possibility of simulating a grid in which different attacks studied previously will take place. As a result of the simulation, synthetic data will be generated, to which deep learning techniques will be applied.

- Once the smart grid data has been obtained, it will be processed and analyzed using big data techniques. This processing is done to prepare the data and, in this way, have the appropriate format to apply them deep learning algorithms through a neural network. This will allow us to make inferences about the behaviour of the grid.

- Finally, the data obtained from the neural network will be analyzed, and thus, it will be possible to detect smart grid attacks.

1.3 Structure of this document

In this section, a brief overview of the chapters included in this document is provided. The structure is the following one:

Chapter 2 studies the current state of the art in the different areas of the project: smart grids, deep learning, big data, and agent-based modelling simulation.

Chapter 3 provides a description of the main technologies that are used in this project and justifies the use of them.

Chapter 4 explains the architecture of this project. A global vision of the architecture and the different modules that compose it are presented.

Chapter 5 presents different study cases that can be applied to this project.

Chapter 6 describes the conclusions, the achieved goals, the problems encountered during the development of this project and the future work.

CHAPTER 2

State of Art

In this chapter, a study of the current state of the art of the different areas of the project is made. Firstly, the smart grids and their vulnerabilities are explained, as well as, the attacks they may suffer. Secondly, Deep Learning and Big data are described. Finally, Agent-Based Modelling Simulation and different tools are introduced.

2.1 Smart Grids

The electrical power grid is evolving into a more modern electric grid. Traditional power grids are generally used to carry power from a few central generators to a large number of users or customers. In contrast, smart grids use two-way flows of electricity and information to create an automated and distributed advanced energy delivery network [17].

The smart grid is a modern, efficient and reliable electric power grid. This achieved by automated control, high-power converters, modern communications infrastructure, sensing and metering technologies, and modern energy management techniques such as demand optimization [23]. Furthermore, smart grids are implemented over the existing electrical network, that is, the current components are not replaced, but new ones are added to monitor the entire network.

Smart grids are electricity networks that wisely integrate the behavior and actions of all users connected to them, to deliver in an efficient, sustainable, economical, and secure way electricity supply [58].

The main advantages provided by smart grids are mainly due to their ability to improve reliability performance and encouraging greater efficiency decisions by the customers and the utility provider. The new components that are added to the electricity grid such as Smart Meters and Information Communication Technologies (ICT) provide the opportunity to achieve energy savings, exploit renewable energy resources and favor customers' participation in the energy market.

These new ICT infrastructures, that allow a more efficient network operation, offer the possibility of setting electricity prices in a more dynamic and reactive way. This helps to balance supply and demand in real time. Therefore, both companies and consumers benefit from this as electricity markets operate more efficiently, reducing peak demand and spot price volatility [51].

Among the new infrastructure installed are the advanced metering infrastructures (AMI). The AMI are the smart meters and the communication network that allows two-way communication between the provider and the consumer's smart meter. Thanks to this, the electric companies have information in real time of the power consumption of their clients and therefore, they can offer new services as the dynamic pricing [26].

All this leads to the consumer becoming more aware of their consumption due to smart meters. As a result, consumers will regulate their consumption by, for example, program-

ming the operating time of the different household electrical appliances at the times when electricity is cheapest. Therefore, the demand curve will be leveled and energy will be saved, as the consumer will avoid the most expensive hours of the day. So, this also means economic savings for both, final consumer and electric companies. It also means economic savings for companies because the load peaks generated by users during the day are a problem as this demands the existence of quick and available power reserves, which is very expensive [45].

In addition, a better forecast of energy consumption will be possible due to all the data collected by the different sensors installed in the network and their monitoring. This makes easier to detect fraud, which entails great economic losses for electric companies and also to integrate renewable energy because it is very unpredictable. With all that data, it would be possible to predict in a more accurate way the output of renewable energies that depend on the weather forecast.

Another benefit of smart grids is that they are very efficient. This is because the losses in the transport of energy are minimized since the failures will affect the minimum possible. It will be possible to re-establish the network more quickly in the event of failures or service interruptions, considering that these will be detected more easily and it will be possible to act before them. This is possible also as a result of the data collected from the different sensors and monitoring.

Furthermore, the smart grids encourage the use of electric cars as these are well integrated into the network because they can consume energy or inject it into the grid. This, linked to promoting renewable energies and the energy saved, contributes to a cleaner and greener energy.

Electricity generation is linked to consumption, so electric supply involves the transmission and distribution of electricity as well as retail activities. Furthermore, in smart grids, local storage systems, electric vehicles and, distributed generation are gaining popularity. Therefore, automated agents, grids, and markets have to be able to integrate these new systems into the network in a way that guarantees supply [44].

Sensors distributed across different network elements and their monitoring generate a large amount of data. This amount of data flowing through smart grids is increasing rapidly every day and many of the benefits of the smart grids mentioned above depend on this data. The data has to be treated in order to draw different conclusions from it and make it useful and that is the reason why smart grids are becoming an interesting research area for data scientists.

The data collected from smart grids can be divided into three different types, which are generation data, transmission/distribution data, and consumer data [50].

- ***Power generation data.*** The power can be generated from water, coal, tides, wind, nuclear, etc. This data can be useful for knowing the dynamic state behavior of wind turbines, fault analysis in a coal-based power plant, etc.
- ***Power transmission and distribution data.*** This data is useful to analyze the power system state estimation and, in this way, guarantee the stability of the network and avoid blackouts.
- ***Power consumption data.*** Electricity will be consumed by consumers from residential, commercial and industrial areas, as well as, transportation, emergency and governmental services, etc. This data is collected by smart meters and have different utilities, and some of the most important ones are load forecasting, real-time pricing, load control, metering information and energy analysis, etc.

Security has become one of the most important issues in smart grids due to the large amount of data that they handle. This has originated new vulnerabilities and, as a consequence, new attacks that can suffer smart grids. In the following subsections, the main ones are presented.

2.1.1 Vulnerabilities of Smart Grids

Smart grids are complex networks made up of millions of devices connected to each other. These networks improve conventional networks but as a consequence make them more vulnerable to different types of attacks. Regarding to [2], [41] and [12], smart grids have different vulnerabilities that might allow attackers to access the network. These vulnerabilities are:

1. ***Leaking of customer data.*** A large amount of private information is collected from customers. This information can be used to find out when someone is at home or what devices are being used.
2. ***Uncontrolled expansion of the number of intelligent devices.*** In order to manage the electricity supply and network demand, different intelligent devices are used, which may be the target of cyber attacks to the network.

3. ***Physical access vulnerabilities.*** Smart grids add a large number of new components to the traditional power grids, which make them vulnerable to physical access.

4. ***Outdated IT systems.*** Power systems coexist with relatively short-lived IT systems. This could lead to outdated systems in service.

5. ***Intercepted communications.*** The communication between the devices of control systems is vulnerable to data spoofing.

6. ***Using Internet Protocol (IP) and commercial off-the-shelf hardware and software.*** Using IP standards offer a great number of benefits, but also make the grid vulnerable to IP attacks.

According to [2], attackers can be grouped into: (i) Non-malicious attackers who try to break the security of the systems for fun. (ii) Consumers who carry out the attack as an act of vandalism, in order to harm another consumer or the electric company. (iii) Terrorists who see the attack as a way to harm a large number of people. (iv) Employees dissatisfied with the company. (v) Competitors attacking each other.

In the following subsection, the attacks that could be originated as a consequence of these vulnerabilities are presented.

2.1.2 Attacks on Smart Grids

Cyber-attacks are a very important issue in the world of cyber-security, which results in governments and organizations having to spend a great amount of time and money to detect and deal with them. In addition attacks on Smart Grids, through smart meters, AMI devices, etc. jeopardize the integrity, confidentiality, and privacy of the information that is handled. Furthermore, the limited computational resources increase the number of risks of a cyber attack [16].

As a consequence of the vulnerabilities explained in Sect. 2.1.1, smart grids can suffer from different types of attacks, such as disruption attacks, destruction attacks, thefts, extortions and repurpose attacks [15], [2].

1. ***Disruption attacks.*** They consist of interrupting a service, commonly through DDOS attacks, and they could compromise the availability of the network. The target of this type of attack usually is communication equipment. These interruptions can

be from outside or inside to inside assets, from inside to outside targets or attacks on certain user groups.

2. ***Destruction attacks.*** They interrupt service as well as disruption attacks. However, when the attack is finished, the target cannot be reusable. In consequence, the target, which is usually industrial equipment, has to be reset or even replaced for a new one. For instance, the disconnection of households through smart meters can lead to a destabilization of the network, causing the possibility of ending up with several elements of the grid damaged. Furthermore, the equipment destined for the management of the grid or critical electrical nodes can also be attacked. Moreover, the data that is originated in different sensors of the grid could be modified so this can originate problems in the control center, which can arise blackouts.

3. ***Theft.*** It consists of stealing information from the grid, which can lead to several problems. It is the most common attack and it can prejudice different assets. Furthermore, it can ruin the credibility of users or the reputation of providers and manufacturers. Thefts can be achieved by modifying smart meters readings, manipulating billing, etc. This could be accomplished through False Data Injection (FDI) attacks, which consist of sending false packets in order to change the grid data. Moreover, the stolen data from the historical consumption or live data can be sold to competitors, being information with great value because many conclusions can be drawn from them. Thus, the energy market can be manipulated regarding this stolen data.

4. ***Extortion schemes.*** They can be threats of either destruction or DDOS attacks. On the other hand, they can also be crypto-locker malware, which can result in a great loss of data.

5. ***Repurpose attacks.*** They consist of changing the behaviour of a specific asset of the grid. Malware spreading could lead to infected hosts acting as fake servers or proxies and they can be used for distributed computing.

In Table 2.1, the relationship between the different attacks and the vulnerabilities that cause them can be observed.

Attack	**Vulnerability**
Disruption Attack	Uncontrolled expansion of intelligent devices
	Outdated IT systems
	Using IP
Destruction Attack	Physical access vulnerabilities
Theft	Leaking of customer data
	Uncontrolled expansion of intelligent devices
	Intercepted communications
	Using IP
Extortion Schemes	Leaking of customer data
Repurpose Attacks	Outdated IT systems
	Uncontrolled expansion of intelligent devices
	Using IP

Table 2.1: Attacks and Vulnerabilities

2.2 Deep learning

Machine learning is the capability of AI systems to extract patterns from raw data in order to acquire their own knowledge. The introduction of machine learning gave the AI systems the ability to solve problems involving knowledge of the real world and make decisions that appear subjective [21].

Traditional machine learning techniques were limited in processing raw data. Therefore, it was very difficult to transform raw data into a form in which the machine learning system could detect or classify input patterns. Representation learning solves this problem and it allows a machine to automatically discover the representations needed for detection or classification. For this project, Deep Learning techniques are going to be used, which are representation-learning methods with multiple levels of representation [32].

Deep learning is a specific kind of machine learning, which allows computational models to learn representations of data with multiple levels of abstraction. Using the backpropagation algorithm, deep learning is able to discover complex structures in very large datasets. This algorithm indicates how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer [32].

Deep learning architectures are inspired by the human brain because conventional digital computers work in a totally different way. Moreover, the brain is a highly complex, nonlinear, and parallel computer. It is able to organize the neurons in order to perform computations many times faster than the fastest computer today. On the other hand, neural networks model the behaviour of the brain, so they can be defined as a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use [25].

According to [25], neural networks offer the following benefits:

1. **Non-linearity.** The nature of a neuron can be linear or non-linear. Consequently, a neural network is non-linear.

2. **Input-Output Mapping.** Supervised learning involves modification of some parameters of the neural network. The network is fed with a random example and, depending on the output, different parameters will be modified to get closer to the desired output. These parameters are the different weights that have been assigned.

3. **Adaptivity.** Neural networks have a great capacity to adapt their parameters according to the changes that occur around them. A network trained to operate in a specific environment can be easily retrained if a minor change has been produced in that environment. Moreover, the more adaptive is a system, the more robust is its performance.

4. **Evidential Response.** A neural network provides information about the pattern selected and also the confidence of that decision.

5. **Contextual Information.** Every neuron is influenced by the rest of the neurons of the network. So, knowledge is characterized by the structure of the neural network.

6. **Fault Tolerance.** The performance of a neural network does not vary significantly depending on the conditions in which it is running.

7. **VLSI Implementability.** A neural network can be implemented using VLSI (Very-Large-Scale-Integrated) technology due to its parallel nature.

8. **Uniformity of Analysis and Design.** Neural networks, as information processors, use the same notation in all domains.

9. **Neurobiological Analogy.** A neural network design is very similar to the brain. It makes possible to process data in parallel in a fast and powerful way.

In the following subsections, different types of neural networks (feed-forward, recurrent networks, restricted Boltzmann machines, and autoencoders) are presented.

2.2.1 Feed-forward neural network

Feed-forward neural networks [53], trained with a back-propagation learning algorithm, are one of the most popular neural networks. They can be divided into two types: single-layer and multi-layer. In Fig. 2.1, a multi-layer neural network divided into different layers which are composed of neurons is represented.

The input nodes represent the data that is introduced into the network and it could be of different types. Moreover, the hidden nodes layer is composed of neurons which are not connected to each other but to adjacent layers. This layer is called hidden because it does not interact with the external environment. Finally, the output layer put together the data produced by the previous layers and generates the desired output [7]. If the neural network was single-layer, it would only consist of the neurons in the output layer.

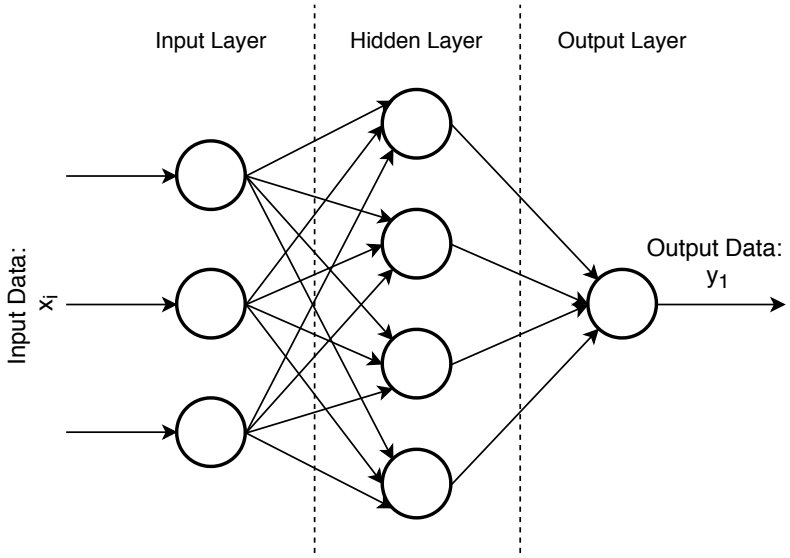

Figure 2.1: Feed-forward neural network

2.2.2 Recurrent neural network

Recurrent Neural Networks (RNN) [10] have at least one feedback connection so the activations can flow round in a loop, as can be seen in Fig. 2.2.

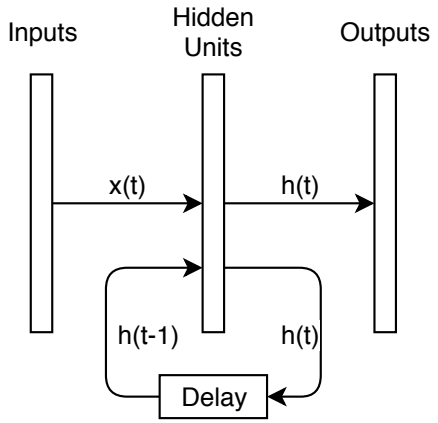

Figure 2.2: Recurrent Neural network

One of the most used RNN is the multi-layer perceptron, explained in Sect. 2.2.1, with the addition of loops. This results in the neuronal network having memory, where it stores information about the previous input. That is, the behaviour of hidden neurons might not just be determined by the activations in previous hidden layers, but also by the activations at earlier times. Indeed, a neuron's activation might be determined in part by its own activation at an earlier time [37].

2.2.3 Restricted Boltzmann Machines

Boltzmann Machines (BMs) [18] are bidirectionally connected networks of stochastic processing units and they can be interpreted as neural network models. Furthermore, Restricted Boltzmann Machines (RBMs) have restrictions on the network topology and they are parameterized generative models that represent probability distributions.

BMs has two types of units, visible and hidden neurons.

- Visible neurons. They form the first layer and correspond to the components of an observation.
- Hidden neurons. They model the dependencies between the components of the observations.

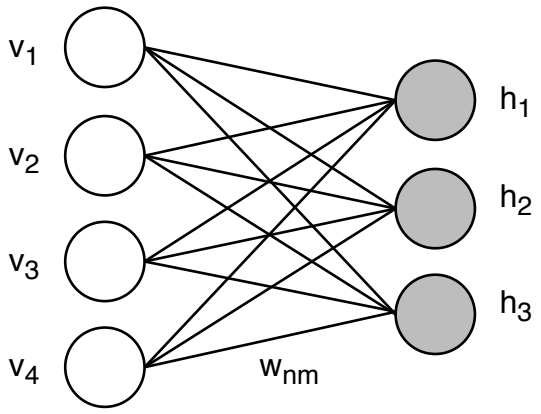

Figure 2.3: Restricted Boltzmann Machine

A representation of an RBM is shown in Fig. 2.3. It is composed of m visible units $V = (V_1, ..., V_m)$ that represent observable data and n hidden units $H = (H_1, ..., H_n)$ to capture dependencies between observed variables.

2.2.4 Autoencoders

An autoencoder [59] is a feed-forward neural network, explained in Sect. 2.2.1, in which the desired output is the input itself.

For this purpose, autoencoders do not perform the identity function by mapping the input directly on the output, but they have hidden layers which may be of smaller or larger

dimensions than the input. Therefore, by modifying the input dimensions, the autoencoder performs the reconstruction of the data.

An autoencoder consists of two parts [52]:

- *Encoder* f_θ*:* maps an input vector to a latent representation $y^{(i)} = f_{\theta'}(x^{(i)}) = s(Wx^{(i)} + b)$ with s being a non-linear activation function.
- *Decoder* $g_{\theta'}$*:* its objective is to reconstruct the input $x^{(i)}$ from $y^{(i)}$. For that purpose, $x^{(i)} = g_{\theta'}(y^{(i)}) = s(W'y^{(i)} + b')$ is applied.

The objective of training is to determine $\hat{\theta} = \{W, b\}$ and $\hat{\theta}' = \{W', b'\}$ in order to minimize the reconstruction error of input vectors $x^{(i)}$.

$$\hat{\theta}, \hat{\theta}' = \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} (L(x^{(i)}, g_{\theta'}(f_\theta(x^{(i)})))) \tag{2.1}$$

In Equation 2.1, it is shown the function that has to be minimized, where L is a loss function, and θ and θ' can be optimized by gradient descent methods.

Autoencoders learn representations of inputs by retaining useful features in the encoding phase which help to reconstruct that input, discarding useless or noisy features.

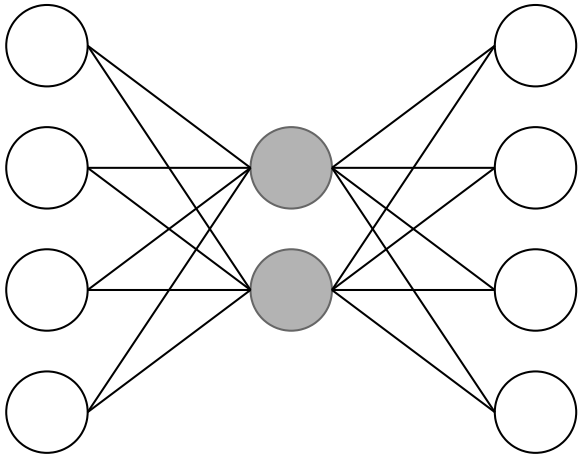

Figure 2.4: Autoencoder

2.3 Big Data

Big Data [11] are high-volume, high-velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery, and process optimization. In other words, a data set is Big Data if it is possible to capture, curate, analyze and visualize that data with the current technologies.

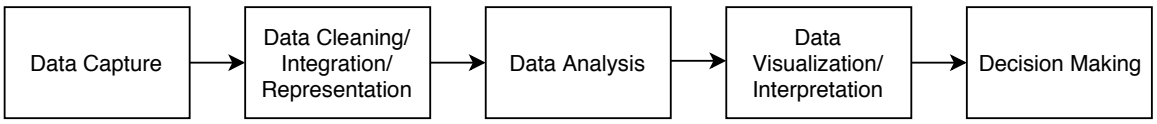

Figure 2.5: Data analysis process

In Fig. 2.5, it is shown the whole process of data analysis, from its capture to the decision making. Each step is explained below [11]:

1. **Data Capture**

   Nowadays, the way of capturing and storing data has changed, due to the exponential increase of the data created every day. The storing capacity has to increase as well as the I/O speed, that it is poor. New storage technologies, such as SSDs (Solid-State Drive), with higher I/O speed than HDDs (Hard Disk Drives), partially solves this problem but it is not enough. Therefore, the design of storage subsystems for Big Data has to be changed.

   Thus, in order to improve the performance, a good solution is optimizing data access using techniques as data replication, migration, distribution, and access parallelism.

2. **Data Curation**

   The objective of data curation is data discovery and recovery, ensure data quality, provide added value, and the possibility of reusing and preserving data over time. It involves a number of sub-fields including authentication, archiving, management, preservation, retrieval, and representation.

3. **Data Analysis**

   The most important challenge in Big Data analysis tasks is scalability. In recent years, the analysis algorithms have improved so that they are now faster and can deal with more amount of data. However, the speed of the CPUs is increasing, but not enough, which leads to the development of parallel computing.

4. **Data Visualization/Interpretation**

   This step of the process is very important because after the analysis is carried out, the data must be presented in such a way that conclusions can be drawn from them. The use of graphs will help to present the data in an intuitive and effective way.

5. **Decision Making**

   This is the last step of the process and after the data is captured, curated, analyzed and visualized, different conclusions have to be drawn in order to make decisions.

In order to perform the whole process explained above, different techniques are used. These techniques, which are very varied, are optimization methods, statistics, machine learning, data mining, and visualization approaches.

First, optimization methods usually require a great amount of memory and time. In addition, they are very useful to solve problems of very different fields, and sometimes it is required to do it in real-time. For that purpose, data reduction and parallelization are needed.

Moreover, statistics make possible to use of the correlations and causal relationships between different objectives. Statistics is the science that allows us to collect, organize and interpret data. Furthermore, new techniques are emerging because the classic ones do not suit well with Big Data.

Additionally, Data Mining techniques involve machine learning and statistics and they are very useful to obtain valuable information from data, including analysis, classification, regression and association rule learning.

Lastly, visualization approaches are used to present the data in an understandable way and interpret it in order to draw conclusions and make decisions.

2.4 Agent-Based Modelling and Simulation

Agent-based Modelling and Simulation (ABMS) [34] is an approach to modelling systems comprised of autonomous, interacting agents. These agents can be defined by the following properties:

- ***Autonomous and self-directed.*** An agent is independent within its environment interacting with the rest of the agents. The agent's behaviour relates its perception of the environment to its decisions and actions.
- ***Modular or self-contained.*** An agent is an identifiable, discrete individual with a set of characteristics or attributes, behaviors, and decision-making capability.
- ***Social, interacting with other agents.*** Agents have mechanisms of interaction between each other so that collisions are avoided, they communicate, etc.

In addition, agents live in an environment with which they interact as they do with other agents. The behaviour of an agent can be based on an objective or specific goal and therefore, they can have the ability to learn and adapt that behaviour based on their experiences.

The objective of an Agent-Based Modelling and Simulation (ABMS) system is to understand the interactions in a given Complex Adaptive System (CAS). Furthermore, an agent-based model allows researchers to experiment how a simulated CAS behaves under certain conditions. Therefore, this gives the opportunity to analyze experiments that are impracticable in the real world, due to their possible consequences or their economic cost. In addition, ABMS facilitates the generation of theories, as well as, their validation and it should continuously improve its methodological foundations [44].

2.4.1 ABMS Tools

Firstly, in order to study and analyze smart grids, Agent-Based Modelling and Simulation (ABMS) has been widely used because of the heterogeneity of electricity systems, its bi-directional interactions and the feedback loops between agents and institutions that it offers.

For that reason, in order to obtain synthetic data, ABMS has been chosen. It allows us to simulate a smart grid and a False Data Injection or Topology attack. Once the simulation

is ended, we will be able to obtain all the data from the simulation for further processing and apply deep learning to them.

Consequently, different ABMS tools that may be useful for this project have been analyzed, mainly Mosaik and MESA.

- **Mosaik** [47] is a simulation compositor for Smart Grid simulations written in Python. It aims to provide support for scenario specification, simulation composition and execution, and scenario result analysis. This tool is explained in detail in Sect. 3.2.1.
- **MESA** [35] is an open-source, Apache 2.0 licensed Python package, that allows users to create agent-based models using built-in core components or customized implementations. In addition, it has a browser-based interface to visualize and analyze the results.

  Mesa's architecture is divided into different modules that can be grouped into three categories:

  - *Modelling:* this module is composed of a Model class that stores model-level parameters being the container of the rest of the components. This module also consists of an Agent class, a scheduler that synchronizes the different agents and the space or network in which the agents are located.
  - *Analysis:* it involves data collectors that record data from the models and a batch runner that automatizes multiple runs.
  - *Visualization:* these components use a server interface to visualize the results in a browser.

In addition, among the ABMS tools analyzed, Mosaik is a better option than MESA because it is specifically oriented to the simulation of smart grids. Furthermore, as Mosaik is composed of different simulators, this will allow us to implement a new one which will simulate the mentioned attack.

In [16], different frameworks for attack modelling in smart grids are analyzed: **ASTORIA**, **NeSSI2**, and **SCORE**.

ASTORIA (Attack Simulation TOolset for smart gRid InfrAstructures) [57] is a framework that uses Mosaik with ns3 in order to simulate smart grids with ICT equipment. It allows users to simulate cyber attacks integrating Mosaik's simulators with ns3. In addition, it is written in python, available for Linux but it does not have a stable version.

NeSSi2 [16] represents a distributed DDoS attack against a real like simulated smart grid topology. It simulates a UDP flooding DDoS attack that consists of sending a large number of UDP packages to the victim. It is written in Java and available only for Windows OS.

SCORE (Smart-Grid Common Open Research Emulator) [54], based on CORE, integrates the power and communication network. In order to simulate an attack, two Linux virtual machines are used. One with Ubuntu that runs the smart grid simulation and the other with Kali Linux to execute the cyber attacks. In addition, it is written in Python but it is no longer supported.

Despite having the option of using these tools, it has been decided to use Mosaik as mentioned above. This is because some of these tools are no longer supported, are not stable, are not available for Linux, or only allow simulating one type of attack such as DDOS. In addition, Mosaik is also used by tools as ASTORIA so we can deduce that it is a good option.

CHAPTER 3

Enabling Technologies

In this chapter, the technologies used in this project are introduced. Firstly, the Python libraries related to data processing are explained. Secondly, Mosaik is introduced, which is an ABMS tool that allows us to obtain synthetic data related to smart grids. Thirdly, the different machine learning libraries used are described which are TensorFlow, Scikit-learn and Keras. Finally, the mathematical model ARIMA is introduced.

3.1 Data managing libraries

This section presents the different Python libraries used in this project in relation to data managing.

3.1.1 Pandas

Pandas [36] is a Python library of rich data structures and tools for working with structured data sets. The library provides integrated, intuitive routines for performing common data manipulations and analysis on such data sets.

The two primary data structures of Pandas are (i) Series: one-dimensional ndarray with axis labels and (ii) DataFrames: two-dimensional size-mutable, potentially heterogeneous tabular data structures with labeled axes.

The main features offered by Pandas are the following:

- Handling of missing data (NaN).
- Size mutability: insertion and deletion of columns from DataFrames.
- Automatic and explicit data alignment.
- Split-apply-combine operations on data sets.
- Intuitive merging and joining data sets.
- High-performance time series functionality.
- Robust IO tools for loading data from files.

3.1.2 Numpy

Numpy [38] is the fundamental package for scientific computing with Python. This library provides the possibility to use N-dimensional array objects, as well as, sophisticated functions. In addition, it also supports the use of linear algebra, Fourier transform, and random number capabilities.

The main two objects that NumPy provides are: (i) N-dimensional arrays which are a homogeneous collection of objects indexed using N integers, defined by their shape and the kind of item the array is composed of, and (ii) Universal function objects (ufunc).

3.1.3 Matplotlib

Matplotlib [5] is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments. The Matplotlib library is divided into three parts: 1. *Pylab interface:* to create plots. 2. *Matplotlib frontend:* a set of classes to lift, to create and to manage figures, text, lines, plots, etc. 3. *Matplotlib backend:* Vector graphics, PNG, etc.

3.1.4 h5py

The h5Py [13] package is a Pythonic interface to the HDF5 binary data format. It allows users to store huge amounts of numerical data, and easily manipulate that data from NumPy.

The HDF5 technology suite [19] consists of a data model, a library, and a file format for storing and managing data. The main characteristics of the HDF5 technology are:

- Large variety of datatypes supported.
- Flexible and efficient I/O.
- High-volume and complex data.
- It is portable and extensible.

The HDF5 data model defines HDF5 information sets, also called infosets, which are containers for annotated associations of array variables, groups, and types. So, an HDF5 file contains HDF5 datasets, HDF5 groups, and HDF5 datatype objects. Furthermore, the HDF5 data model defines link mechanisms for creating associations between HDF5 information items. Finally, the HDF5 data model defines a facility to annotate HDF5 information items using HDF5 attributes.

3.2 ABMS

In this section, Mosaik and Mavewrig which are ABMS tools that allow users to simulate smart grids are presented.

3.2.1 Mosaik

Mosaik [47] is a simulation compositor for Smart Grid simulations written in Python. It aims to provide support for scenario specification, simulation composition and execution, and scenario result analysis.

Mosaik allows users to simulate a smart grid scenario using existing simulators. Moreover, it is in charge of the synchronization of their processes and of their exchange of data. Its main objective is to integrate different existing and technologically heterogeneous simulation models into a Smart Grid simulation.

The aim is to compose different, existing and technologically heterogeneous simulation models into an overall Smart Grid simulation. In order to do this, Mosaik has the following characteristics: (i) It provides an API that allows the simulators to communicate with Mosaik. (ii) It implements different handlers for the different simulator processes. (iii) It permits to execute a simulation in which the scenario involves different simulators. (iv) It is in charge of synchronizing the different simulators and their exchange of data.

The main components of a Mosaik simulation are:

- *Mosaik-web* [1]: it allows users to visualize the progress of the simulation in a browser, also showing different graphs with the evolution of different parameters.
- *Mosaik-pypower* [2]: it is an adapter for the PyPower load analysis library. PyPower [33] is a python implementation of MATPOWER and it allows users to calculate power flows and operation costs. Moreover, it uses the bus-branch model to represent power grids. This model defines a number of nodes connected through branches. These nodes are divided into:
  - The reference bus: there is one reference bus in every grid and it has constant voltage magnitude and angle. The active and reactive power of the node are also calculated.

[1] https://bitbucket.org/mosaik/mosaik-web
[2] https://bitbucket.org/mosaik/mosaik-pypower

– PQ buses: Pypower will calculate the voltage and angle of these nodes, whereas the active and reactive power are given.

Moreover, in order to define the structure of a grid, it supports JSON and Excel file formats.

As can be seen in the code below, in the JSON are defined the buses, branches, and transformers. In addition, it can be declared new branch and transformer types.

Listing 3.1: PyPower JSON

```
{
    "bus": [
        ["Grid", "REF", 110.0],
        ["Bus0", "PQ",  20.0],
        ["Bus1", "PQ",  20.0],
    ],
    "trafo": [
        ["Trafo1", "Grid", "Bus0", "TRAFO_23", true, 0]
    ],
    "branch": [
        ["B_0", "Bus0", "Bus1", "SPAM_200", 5.0, true]
    ],
   "branch_types": {
      "SPAM_200": [0.1337, 0.0815, 0, 404]
   },
   "trafo_types": {
      "TRAFO_23": [23, 100, 800, 100, 0.0123, 1.234, {"-1":
          0.9, "0": 1, "1": 1.1}]
   }
}
```

- *Mosaik-householdsim* [3] *and Mosaik-csv* [4]: these simulators allow users to simulate households using CSV datasets to read their load profiles. In addition, Mosaik-csv is a simulator that can also model the profiles of renewable energy, like photovoltaic panels.
- *Mosaik-hdf5* [5]: it stores all the data obtained from the simulation in a HDF5 file.

In addition, Mosaik provides an API which connects the simulators to Mosaik. This

[3]https://bitbucket.org/mosaik/mosaik-householdsim
[4]https://bitbucket.org/mosaik/mosaik-csv
[5]https://bitbucket.org/mosaik/mosaik-hdf5

will facilitate the addition and creation of new simulators.

In order to store the data generated by the simulation, Mosaik uses an HDF5 database. Mosaik-hdf5[6] is the Mosaik module that generates the HDF5 database with all the data of the simulation. Once the simulation has ended, the HDF5 file is generated.

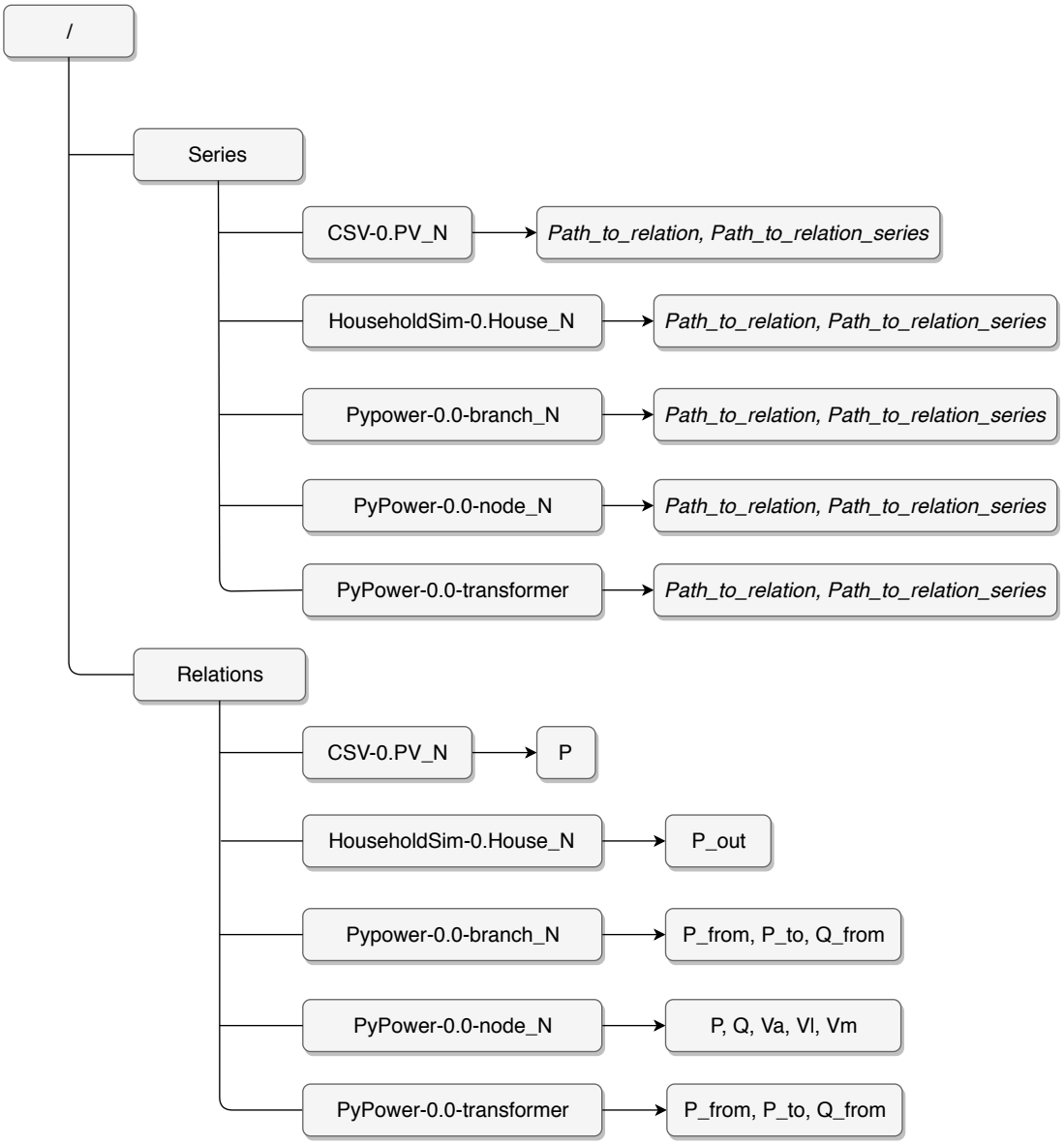

Figure 3.1: Data Structure

After performing a simulation, we obtain the HDF5 database and visualizing it using HDFView[7], which is a visual tool for browsing and editing HDF5 files, allows us to see the database structure shown in Fig. 3.1.

[6]https://bitbucket.org/mosaik/mosaik-hdf5

[7]https://support.hdfgroup.org/products/java/hdfview/

The following groups can be observed:

- *Relations group.* It contains a dataset for every entity, which indicates the relationship of each entity with the rest of them. Those datasets store tuples (*path_to_relation*, *path_to_relation_series*). As can be seen in Fig. 3.2, the Relations group contains one data set for each element of the grid, which in the selected case *CSV-0.PV_0* (photovoltaic panel), it is connected to the *node_b7*. This group allows us to know the topology of the grid.

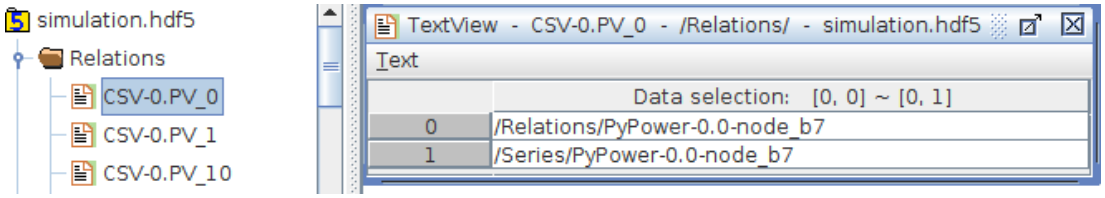

Figure 3.2: Relations group

- *Series group.* It contains one group for every entity. Furthermore, each group contains one dataset for every attribute of the entity. These datasets are arrays with the different values of the attributes. As can be seen in Fig.3.3, the Series group contains one group for every entity of the grid, and each group is composed of datasets of the attributes of the entity.

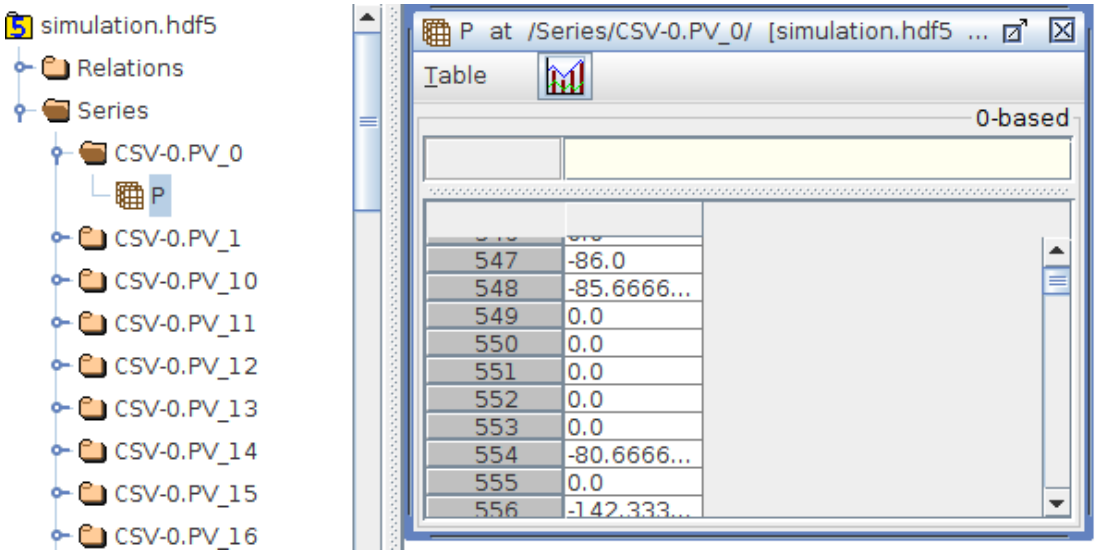

Figure 3.3: Series group

Each group has different entities which are the following:

- *Photovoltaic panels*, named as $CSV - 0.PV\_N$. They generate power and inject it to the grid. Their attribute is the power generated (P). This attribute is generated by the simulator, reading the different power levels generated from a CSV file.
- Households, named as $HouseholdSim - 0.House\_N$. Their attribute is the power they consume (P_out). The HouseholdSim simulator reads from a file the load profiles of each house.
- Nodes, named as $PyPower - 0.0 - node\_aN$. They are the main nodes of the grid to which the rest of the entities are connected. Their attributes are are the active and reactive power, and the different values of voltage (P, Q, Va, Vl and Vm). The simulator PyPower has as input a JSON file, that allows it to generate the topology of the grid. In addition, this simulator manages the different values of its attributes. Moreover, PyPower not only manages the network nodes but also manages the different branches of the grid and its transformer.
- Branches, named as $PyPower - 0.0 - branch\_N$. They connect the main nodes of the grid and their attributes are the active and reactive power that flows through them (P_from, P_to, and Q_from).
- Transformer, named as $PyPower - 0.0 - transformer$. It provides power to the grid and its attributes are the active and reactive power, and the different values of voltage (P, Q, Va, Vl and Vm).

The name of the entities is formed by the tuple (name-of-the-simulator, name-of-the-entity_N), where N is the id of that specific entity.

3.2.2 Maverig

Maverig [48] is a tool which main purpose is to provide Mosaik with a graphical user interface. This user interface should allow the creation of smart grids scenarios, in which users do not have to write Python code, but will do everything through that interface. In addition, it allows the execution of simulations in the interface itself. The main objective of the system is, therefore, the creation of compact and simple scenarios and the control and visualization of the simulation through a homogeneous user interface.

Therefore, Maverig has two different main modes:

- ***Composition Mode.*** This mode allows users to create smart grids scenarios. For this purpose, Maverig provides different elements that allow the creation of such scenario: Reference Bus, PQBus Node, Transformers, Lines, Photovoltaic panels, Households, etc.
- ***Simulation Mode.*** This mode allows users to run simulations in the scenarios created in the Composition Mode while observing the most significant parameters.

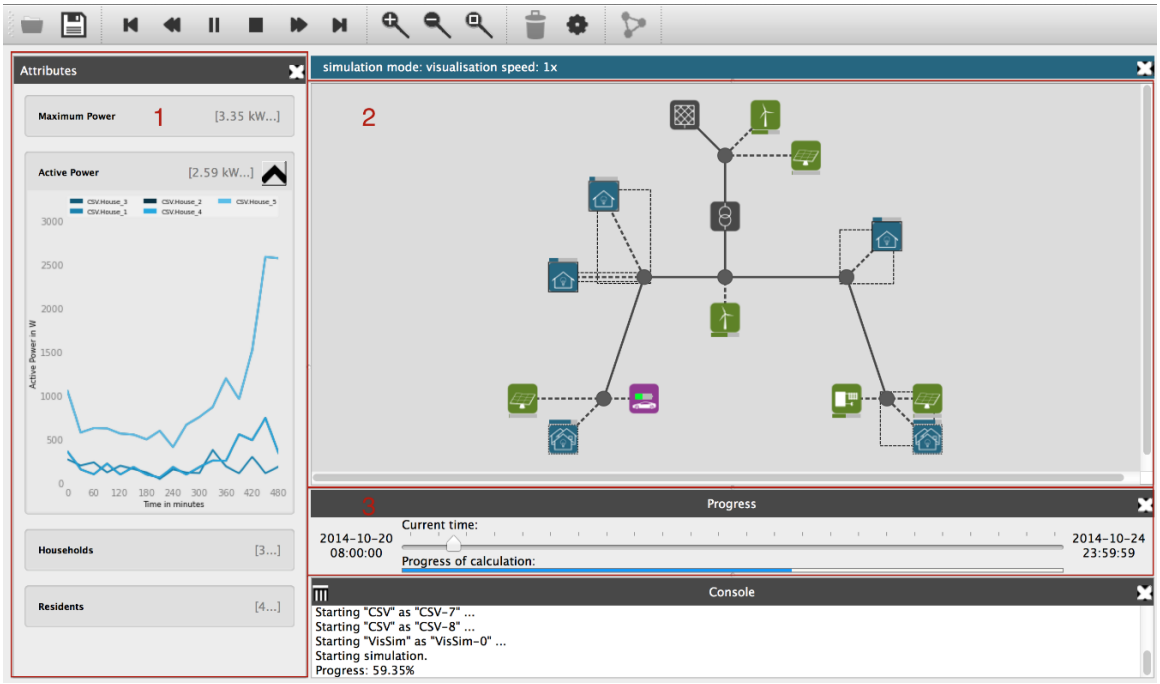

Figure 3.4: Maverig [48]

In Fig. 3.4, the Maverig's graphical interface is represented. This graphical interface allows users to control different parameters: the scenario that has been created, the different parameters that take the elements of the network and their evolution over time, information on the progress of the simulation, and so on.

The use of this tool was considered because it would have simplified the use of Mosaik. This is because, for example, the creation of a scenario can be done dragging and dropping the elements instead of using a python script. However, as there was very few published information about this software, we contacted the developers and they recommended us to use Mosaik because Maverig was no longer maintained. Furthermore, complex scenarios can be realized better with Mosaik using python scripts.

3.3 Machine Learning Technologies

In this section, the different machine learning technologies used in this project are presented: Scikit-learn, TensorFlow and Keras.

3.3.1 Scikit-learn

Scikit-learn [42] is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. It provides simple and efficient tools for data mining and data analysis being built on NumPy, SciPy, and matplotlib. Furthermore, Scikit-learn is open source and distributed with BSD license and therefore, accessible to everybody and reusable in various contexts. It allows users to perform different tasks:

- Classification: identifying to which category an object belongs to.
- Regression: predicting a continuous-valued attribute associated with an object.
- Clustering: automatic grouping of similar objects into sets.
- Dimensionality reduction: reducing the number of random variables to consider.
- Model selection: comparing, validating and choosing parameters and models.
- Preprocessing: feature extraction and normalization.

The API provided by the Scikit-learn library is designed following these principles [9]:

(i) Consistency. All objects share a consistent interface composed of a set of methods. In addition, this interface is documented. (ii) Inspection. Parameters are stored and exposed as public attributes. (iii) Non-proliferation of classes. The only objects represented by customs classes are the learning algorithms. This makes scikit-learn easy to use. Furthermore, Datasets are represented as NumPy arrays or SciPy matrices and hyper-parameters by standard Python string or numbers. (iv) Composition. Machine learning tasks are usually a sequence or combination of transformations that are made to data. Moreover, some learning algorithms are also viewed as meta-algorithms parametrized on other algorithms. (v) Sensible defaults. Scikit-learn assigns default values when an operation requires a user-defined value.

3.3.2 Tensorflow

Tensorflow [1] is an open-source software library for machine learning developed by Google. Python was the first client language supported by Tensorflow but nowadays more languages can use it, like for example C++, Java, Go, R and C#.

Furthermore, DistBelief [14] is the distributed system for training neural networks that Google has used since 2011. It was developed by Google Brain as a proprietary machine learning system based on deep learning neuronal networks. In February 2017, Google Brain improved DistBelief and Tensorflow was released, which is the second generation system.

TensorFlow is able to use a large number of powerful servers in order to achieve fast training. In addition, once the training is completed, it is capable of running the trained models on different platforms. These platforms could be mobile devices or even a cluster of servers. Moreover, Tensorflow is flexible enough to support new machine learning models incentivizing, in this way, experimentation and research [1].

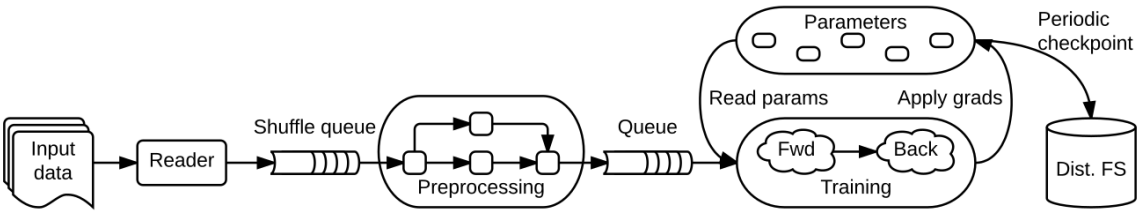

Figure 3.5: A schematic TensorFlow dataflow graph for a training pipeline [1]

A data-flow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training and checkpoint state is shown in Fig.3.5. Tensorflow allows executing parallel and independent computations in different devices at the same type, due to the exchange of data of the different components that compose the data flow [1].

There are differences between batch data flow systems and Tensorflow. One of the most important ones is that it supports multiple concurrent executions. Furthermore, individual vertices may have a mutable state that can be shared between different executions of the graph. [1]

A TensorFlow graph is composed of the following elements [56]:

- **Tensors:** they are a generalization of vectors and matrices to potentially higher dimensions, represented as n-dimensional arrays. The shape of the tensor may be partially known and every element of it has the same known data type. In operations

with tensors, if the shape of its input is known, usually their result is a tensor of known shape. However, in some other cases, the shape of a tensor is only possible to know at graph execution time.

- **Operations:** they are computations on tensors. An operation is defined as a node in a TensorFlow graph with zero or more tensors as input and zero or more tensors as output.

  Every operation has a specific type and different attributes that define their behaviour. Moreover, they can be polymorphic and with different times of compilation, meaning that the expected types of the inputs and outputs are settled by the attributes. There are different types of operations[1]:

  - **Stateful operations: variables.** They use a buffer in order to keep track of the parameters of the model while it is being trained.
  - **Stateful operations: queues.** TensorFlow includes FIFOQueue and other types of queues that dequeue tensors in random and priority orders.

TensorFlow offers an API which allows to the client to specify which subgraph should be executed. It is possible to use the API multiple times on the same graph and each use is called a step.

In addition, TensorFlow is very flexible because it allows to specify different model architectures in user-level code due to the possibility to add mutable state and coordination via queues [1].

Moreover, Tensorflow has distributed execution. Each operation resides on a particular device, which is responsible for executing a kernel for each operation assigned to it. Furthermore, it allows multiple kernels to be registered for a single operation.

In order to place the operations in different devices, Tensorflow takes into account implicit or explicit constraints in the graph. The user may specify partial device preferences, but there is also a placement algorithm. Therefore, Tensorflow is very flexible in the sense of mapping the different operations to the devices.

Consequently, Tensorflow is able to execute large subgraphs many times with low latency [1].

In order to explain the Tensorflow's Python API, an example, taken from [20], is going to be presented. This example is the classification of handwritten digits in the MNIST[8]

[8] http://yann.lecun.com/exdb/mnist/

dataset. For that purpose, first the TensorFlow library has to be imported and the MNIST dataset read into memory.

Then, a computational graph has to be created. These graphs are used to attach to them, new operation nodes.

Listing 3.2: Tensorflow and data import

```
import tensorflow as tf
mnist = mnist\_data.read("/tmp/mnist", one_hot=True)
graph = tf.Graph()
with graph.as_default()
```

Once the graph is created, operations have to be created. For that purpose, placeholders are defined. They are special variables that must be replaced with concrete tensors upon graph execution. For each placeholder, a shape and data type are specified. Moreover, the first dimension of the shape is declared as *None* because, in this way, it can be fed by a tensor of variable size in that dimension. The other dimension (784) is the number of pixels of the image.

Listing 3.3: Placeholder definitions

```
examples = tf.placeholder(tf.float32, [None, 784])
labels = tf.placeholder(tf.float32, [None, 10])
```

The affine transformation $Y = X\Delta W + b$ has to be made. X is a matrix containing the pixels of n images, W is a weight matrix, b a bias vector and Y is a new matrix containing the logits of the model for each example and each possible digit. In order to transform these *logits* into a valid probability distribution, the softmax function is used.

Next, the objective function is computed, producing the loss of the model. For that purpose, the *mean cross entropy*, between the probability distributions and the labels, is calculated.

The *GradientDescentOptimizer* is used to update the weights of the model. It is initialized with the learning rate and provides an operation *minimize* for the *loss* tensor.

Listing 3.4: Objective function and Gradient Descent Optimizer

```
weights = tf.Variable(tf.random_uniform([784, 10]))
bias = tf.Variable(tf.constant(0.1, shape=[10]))
logits = tf.matmul(examples, weights) + bias
estimates = tf.nn.softmax(logits)
cross_entropy = -tf.reduce_sum(labels * tf.log(estimates), [1])
loss = tf.reduce_mean(cross_entropy)
gdo=tf.train.GradientDescentOptimizer(0.5)
optimizer = gdo.minimize(loss)
```

Finally, the algorithm can be trained. For this, a session environment is initialized with the graph as the parameter of its constructor. Then, the variables of the graph are initialized and a number of iterations performed. These iterations consist of training the neural network with examples and labels of the MNIST dataset. When these iterations are over, the loss will be small.

Listing 3.5: TensorFlow Session

```
with tf.Session(graph=graph) as session:
    tf.initialize_all_variables().run()
    for step in range(1000):
        x, y = mnist.train.next_batch(100)
        _, loss_value = session.run([optimizer, loss], feed_dict={examples:
            x, labels: y})
        print("Loss at step {0}: {1}".format(step, loss_value))
```

This example illustrates the use of TensorFlow's Python API.

3.3.3 Keras

Keras [22] is a high-level neural network API, written in Python, that runs on top of TensorFlow, CNTK or Theano. The author of Keras, François Chollet, said: *"The library was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing research"*. Its main characteristics are:

- Modularity: a model is a sequence or graph of independent modules that combined together allows users to build neural networks.

- Minimalism: each module is kept short and self-describing.
- Easy extensibility: Keras can be extended with new functionalities. Furthermore, new modules are very simple to add and it also provides modules with examples of different areas.
- It supports convolutional networks and recurrent networks, as well as combinations of both.
- User friendliness: for Keras the most important thing is the user experience and therefore, it follows best practices for reducing cognitive load. It is an API designed for human beings, not machines.
- It runs seamlessly on CPU and GPU.

The API provided by Keras is consistent and simple in a way that minimizes the actions that the user has to perform and it also provides feedback on the user's errors. In addition, as Keras integrates lower-level deep learning languages and allows users to implement anything possible in the base language (TensorFlow), it can be said that its flexibility is not reduced due to the ease of use and learn.

3.4 Mathematical models

In this section the mathematical model ARIMA is explained, as well as a python library that allows its implementation.

3.4.1 ARIMA

ARIMA (Auto-Regressive Integrated Moving Average) models were introduced by Box and Jenkins and have become one of the most popular approaches to forecasting. They are a type of stochastic process used to analyze time series behaviour. In addition, ARIMA models make it possible to predict future values of a time series based on previous values, its own lags and the lagged forecast errors [39].

ARIMA models include three different models: AR (auto-regressive) models, MA (moving average) models, and ARMA models that combine the two previous models. These models (AR, MA, and ARMA) can be used when the time series is stationary. Therefore, when the time series is not stationary, the letter I (integrated) is used, which indicates that the data have been transformed into stationary. In order to build the model, ARIMA analyzes the previous observations to find the correlations between them. After the analysis, the model will use what it has learned to predict future values [24].

$$\begin{aligned} Y_t = \theta_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} \\ +\varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - ... - \theta_q \varepsilon_{q-1} \end{aligned} \tag{3.1}$$

The representation of the ARIMA model equation is shown in Equation 3.1, where ϕ_i are the autorregressive parameters, θ_j are the moving average parameters, and ε_t is the error term at time t. The general way to represent an arima model is *ARIMA(p,d,q)*, where the different parameters are:

- p: order of the auto regressive part.
- d: degree of first differencing involved.
- q: order of moving average part.

ARIMA modeling consists of three steps: model identification, parameter estimation, and diagnostic checking. Model identification corresponds to checking if the time series have

some theoretical autocorrelation properties. Then the *(p,d,q)* parameters are calculated and finally, the results are verified [28].

3.4.1.1 Statsmodels

Statsmodel [49] is a library for statistical and econometric analysis in Python. It provides classes and functions for the estimation of many statistical models, as well as for conducting statistical tests, and statistical data exploration. This package is distributed under the open source Modified BSD license. The library is intended to be used by applied statisticians and econometricians and, also, Python users working with disciplines related to statistical models. Statsmodels is designed to be user-friendly and easily extensible by developers from any discipline.

During its development, most of the results obtained with the different models implemented by the library have been verified with at least one other statistical package: R, Stata or SAS. In addition, some statistical methods are tested with Monte Carlo Studies.

This library has been used to implement the ARIMA model using the *statsmodels.tsa.arima_model.ARIMA* class.

CHAPTER 4

Architecture

In this chapter, the architecture of this project is explained. Firstly, a global vision about the project architecture is presented, identifying all the modules that compose it. Finally, all the modules are described in detail, explaining all their components.

4.1 Architecture

In order to achieve the objective of this project, which is the detection of attacks in smart grids, the global system is presented. Its representation and the different modules that compose it: Multi-Agent System, Data Preprocessing Module, Autoencoder, and Anomaly Detection Module are shown in Fig. 4.1.

First, the data is obtained from the *Multi-Agent System*, Sect. 4.1.1, and then it is treated in the *Preprocessing Module*, Sect. 4.1.2. In this module, the data is cleaned, some features are generated and finally, it is normalized. The next step is to apply deep learning algorithms to the data, Sect. 4.1.3. For this purpose, the *model* is created using an *Autoencoder*, which is trained and tested. Finally, in the *Anomaly Detection Module*, Sect. 4.1.4 the results obtained from the autoencoder are analyzed and treated in order to classify the data between normal or attacked.

In the following sections, these modules are explained in detail.

4.1.1 Multi-Agent System

Due to the impossibility of obtaining real data related to smart grids since they are confidential, ABMS is used in order to generate synthetic data.

Therefore, this module is in charge of generating the synthetic data to which the deep learning algorithms are going to be applied. For that purpose, Mosaik, Sect. 3.2.1, which is a Multi-Agent System, is used. Furthermore, it is a simulation compositor for Smart Grid simulations written in Python.

In order to simulate False Data Injection Attacks, a new simulator for Mosaik has been implemented. This new attack simulator allows us to modify the value of power consumption of a configurable number of houses. Therefore, it provides us the possibility of setting that power consumption value to zero or to a specific percentage in a certain moment of the simulation.

Furthermore, in order to simulate attacks, several components of Mosaik had to be configured (*executable file and HouseholdSim simulator*) and the attack simulator had to be created (*attack.py*):

- *Executable file.* This is the main file of Mosaik, in which the simulators that will partic-

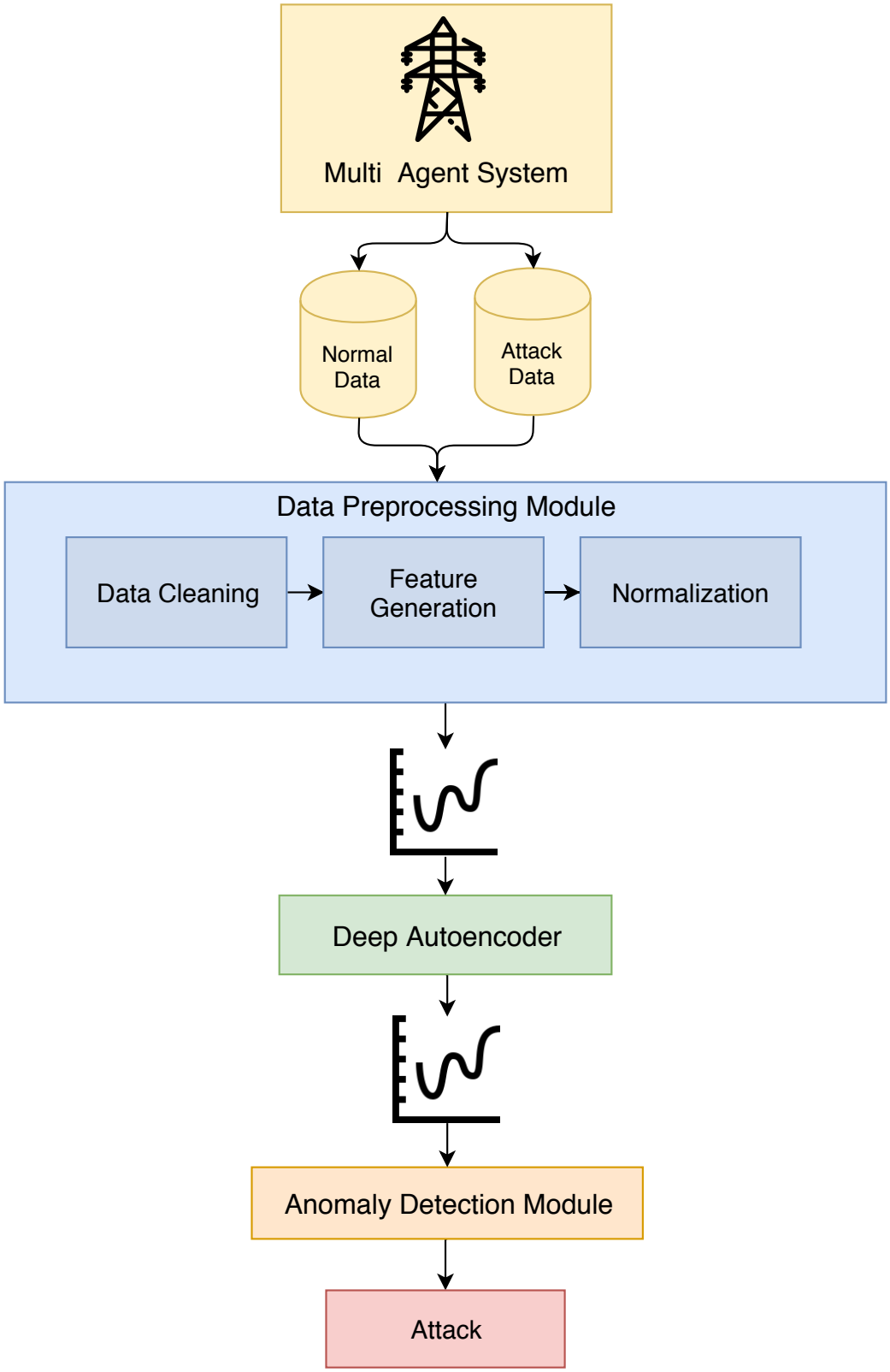

Figure 4.1: Architecture for Anomaly Detection

ipate in the simulation are defined, as well as the different connections between them. For this reason, it has had to be modified to include the definition and initialization of the attack simulator and its connection with the house simulator.

The attack simulator has to be initialized in this file as follows:

Listing 4.1: Initialization of the simulator

```
attacks = attacksim.Attack.create(38, target_attr='P_out')
```

When initializing the simulator, two attributes must be set. The first attribute is the number of houses that are going to suffer an attack, which in the case of the example shown above is 38. If the number of attacks is higher than the number of houses, the simulation does not start and an error is thrown. The other attribute is the *target attribute*, that is to say, the value that is going to be modified by the attack. This target attribute is *P_out*, which is the value that indicates the house power consumption.

In addition, the assignment of attacks to houses is done randomly, in this way, in different simulations not always the same house is attacked. This assignment is made in the following way:

Listing 4.2: Connection of simulators

```
world.connect(house, attacked_houses[house], ('P_out','P_out_val'),
    async_requests=True)
```

First, the house and the instance of the corresponding attack have to be indicated. The next two attributes are the parameters of the two simulators that are going to be linked. *P_out* is the value of power consumption of the house and *P_out_val* is the new power consumption value generated by the attack simulator.

- *Attack.py.* This is the new implemented simulator that is initialized in the executable file. The duration of the attack is configurable, allowing users to execute the attack during the whole simulation or during an specific time frame. In addition, the data flow between the house simulator and the attack simulator is bi-directional. First, the house power consumption value is received so it can be modified according to the attack chosen to simulate. Once it is modified, that new value is returned to the house simulator to replace the original value.

  The *set_data* function provided by the Mosaik API has been used to send a value from

one simulator to another. Therefore, it has been used to send data from HouseHoldSim to AttackSim and vice-versa. The data that is sent is an object mapping source entity IDs to objects which in turn map destination entity IDs to objects of attributes and values. In addition, it has to follow the following structure:

Listing 4.3: Data format

```
{"src_full_id": {"dest_full_id": {"attr1": "val1", "attr2": "val2"}}}
```

Where *src_full_id* is the identifier of the source simulator, *dest_full_id* is the identifier of the destination simulator and then the attribute with its value that is wanted to be sent. This data will be received as input of the destination simulator.

In this way, the power consumption value of the houses can be set to a specific power value, or to a specific percentage of the original value.

- *HouseholdSim simulator.* Initially, data was sent from the attack simulator to the original HouseHoldSim but was never received, which was a problem. After investigating why this data was not received, it was concluded that the simulator was prepared to send data to other simulators but not to receive it, as this was not necessary. For this reason, this simulator had to be configured in order to be able to receive data from other simulators.

  Therefore, the configurations that have been made to this simulator include sending data to the attack simulator and the possibility of receiving data in the event that an attack takes place. The data that is received, in the format explained above, replaces the original power consumption values generated by the simulator itself.

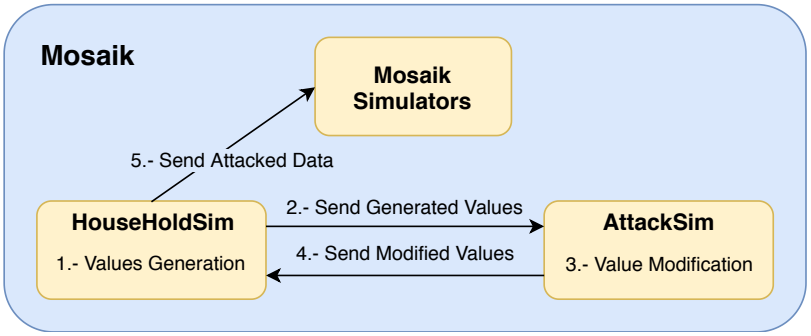

Figure 4.2: Attack Implementation

In Fig. 4.2, the interaction of the new implemented attack simulator with the other simulators is shown. As can be seen, first, HouseHoldSim generates the power consumption

values of the houses. In case of an attack occurs, this simulator will send the data to the attack simulator which will modify these data depending on the attack chosen for its simulation. Once the values have been modified, they are sent back to HouseholdSim, which is in charge of sending them to the rest of the simulators of Mosaik. In addition, this diagram shows the reason why HouseHoldSim had to be configured to receive data, as well as the implementation of the attack simulator that receives and sends data.

Using this Multi-Agent System, different datasets will be generated: one dataset without attacks, that is to say, a normal behaviour of the grid, and other datasets where different attacks have been simulated. This will allow us to train and test the neural network. Normal data will be necessary to train the network, because, in this way, it will learn the normal behaviour of the smart grid. Therefore, when feeding the neural network with the dataset with attacks, it will be able to detect them. This process is explained in detail in Sect. 4.1.3.

Mosaik generates an HDF5 database with all the information of the simulation. For that reason, the h5py library [13] is used, which is a python package that provides an interface to the HDF5 binary data format. Each field of the database has 44640 values. These 44640 values are the result of generating data every minute for a month (60 minutes * 24 hours * 31 days).

In addition, as explained in Sect. 3.2.1, Mosaik generates different values of each component of the grid: (i) Transformer and nodes of the grid, active and reactive power values and their potential difference (*P, Q, Va, Vl and Vm).* (ii) Branches, active and reactive power values (*P_from, P_to and Q_from*). (iii) Photovoltaic panels, power generation values(P). (iv) Households, power consumption values (P_out).

All these data are related and influence each other. Among all these data, to detect attacks has been chosen to study the evolution of the power consumption values of the houses, which are the values that are modified by the attack. These values change every 15 minutes, but they are stored in the HDF5 database every minute. Thus, the same value is stored 15 times and, therefore, the data has to be cleaned to remove all those unnecessary values. Furthermore, in order to feed the autoencoder, some preprocessing has to be done to the data so that they have an adequate format.

4.1.2 Data Preprocessing Module

This module is in charge of preprocessing the data generated by the ABMS module, explained in Sect. 4.1.1, so that they are in a suitable format to introduce them in the autoencoder. This process is divided into three different steps, which are data cleaning, feature generation, and normalization.

1. *Data cleaning.* First, the data has to be cleaned in order to eliminate the values that are not necessary. As explained before in Sect. 4.1.1, Mosaik model works with a minute time step, but the residual load profiles of HouseHoldSim have a resolution of 15 minutes. However, this is not a problem for HouseHoldSim because it repeats the same value 15 times until the power consumption value changes. This results in the same value stored 15 times, so the purpose of this step is to clean all those unnecessary repeated values.

2. *Feature generation.* Once the data has been cleaned, it has to be reorganized and also new features have to be generated in order to help the autoencoder to learn the existing relations between the data.

   The reorganization consists of creating an overlapping sliding window [4] with the power consumption values. The main advantage of using an overlapping sliding window versus a non-overlapping one is that, when it comes to detecting an attack (anomaly), it will be detected earlier. In a non-overlapping window, the anomaly can be identified only when the time is greater or equal to the time length of the window, whereas in an overlapping window this problem does not exist and the attack could be detected every time step (15 minutes). A representation of the mentioned overlapping sliding window is shown in Fig. 4.3.

   Furthermore, after reorganizing the data, new features are generated to help the anomaly detection process. These features are explained in detail in Table 4.1. In addition, the temporal contextual features (*day*, *hour* and *minute*) are created because power consumption depends on temporal seasonality [3]. Whereas the other features provide more necessary information about the sliding window.

3. *Normalization.* All the features of the dataset have different scales so, in order to all of them have the same weight, they have to be normalized. For that purpose, the *MinMaxScaler()* function of the *scikitlearn* library has been used. This function, showed in Equation 4.1, scales and translates each feature in a range between zero and one.

0	15	30	45	60	75	90
152.660	249.480	260.340	165.640	164.730	228.500	229.100
249.480	260.340	165.640	164.730	228.500	229.100	261.720
260.340	165.640	164.730	228.500	229.100	261.720	182.620
165.640	164.730	228.500	229.100	261.720	182.620	87.240
164.730	228.500	229.100	261.720	182.620	87.240	152.710
228.500	229.100	261.720	182.620	87.240	152.710	153.770

Figure 4.3: Overlapping Sliding Window

$$x_{scaled} = \frac{x_i - min(x))}{max(x) - min(x))} \tag{4.1}$$

In addition, scaling the data between 0 and 1 allowed us to use activation functions like the hyperbolic tangent (tanh) and the sigmoid function, in order to try more methods to optimize the neural network model. Furthermore, it makes the training faster.

4. *Labelling.* This step will only be executed when training and verifying the performance of the neural network. This is because to calculate the F-Score it is necessary to compare the predicted values of the classification in attacks or normal behaviour with the label that that indicates whether or not such attack has actually occurred. Therefore, in this step, a new column will be added to the dataframes created previously in which it will be indicated with a 0 if it is normal behaviour or with a 1 if there has been an attack.

   When performing the simulation with Mosaik, the parameters that indicate when the attack starts and its duration are established by the user. Therefore, this information will be used to know which labels have to be assigned to each data entry. This dataframe column will not be used during training since for the operation of the autoencoder is not necessary. It will be only used to calculate the precision, recall, and F-Score of the neural network, that is to say, to evaluate its performance. Therefore, the final user of the system, once the neural network is trained, does not have to execute this step of the preprocessing.

Feature	Description
Day	Current day of the first window value
Hour	Current hour of the first window value
Minute	Current minute of the first window value
P_n	Power consumption window values
$\bar{x}$	Mean of the window values
s	Standard deviation of the window values
$\bar{x}_i$ - $\bar{x}_{i-1}$	Difference between the mean of the window values and the mean of the previous window values
$\bar{x}_i$ - $\bar{x}_{i+1}$	Difference between the mean of the window values and the mean of the next window values
P_n - P_1	Difference between the last and first value of the window
Q1	First quartile of the window values
Q2	Median of the window values
Q3	Third quartile of the window values
IQR	Interquartile range of the window values

Table 4.1: Generated Features

Once these steps have been executed and the data has been preprocessed, a Pandas DataFrame will be generated with all the data and features, becoming the input of the autoencoder. These DataFrames will be stored in a pickle (*.pkl* file) in order to preprocess the data only once and to be able to load them faster.

4.1.3 Deep Autoencoder

Regarding to [6], autoencoders are a technique used to estimate the state of a smart grid. This technique is a type of unsupervised learning, which does not require labeled training datasets. In addition, the operation of an autoencoder is based on the assumption that data can be reduced to a lower dimensional subspace. In that subspace, normal cases and anomalies seem very different. In this way, using autoencoders and applying dimension reduction techniques, it will be possible to detect anomalies [40].

An autoencoder learns from reducing or increasing the dimension of its input data in order to try to reconstruct that data. In this process, the hidden layers are in charge of prioritizing the properties that have more weight when reconstructing the output [43].

Depending on whether the hidden layers have more or fewer dimensions than the input data, the autoencoder will have different properties. On the one hand, if they have fewer dimensions than the input, the autoencoder will detect the most outstanding properties of the data. On the other hand, if they have more dimensions than the input, the autoencoder will be robust against noise or missing inputs. Moreover, autoencoders are not designed to perfectly reconstruct the input because in this way patterns could not be detected in the data.

An autoencoder consists of two parts: an encoder that maps the original input to a hidden layer with an encoder function and a decoder that produces a reconstruction. The representation of an autoencoder, where its different parts are identified, is shown in Fig. 4.4.

In order to implement the autoencoder, Keras has been used, which is a Python library. It is a high-level neural network API capable of running on top of TensorFlow, explained in Sect. 3.3.3.

The Python code implemented in order to create the autoencoder model, using Keras, is shown in Lst. 4.4. The input dimension will be 20, which is the total number of features explained in Table 4.1, and in the hidden layer, the dimension is going to be reduced to 8. Finally, in the output layer, it will be reconstructed the input so, its dimension is the same as the input, which is 20.

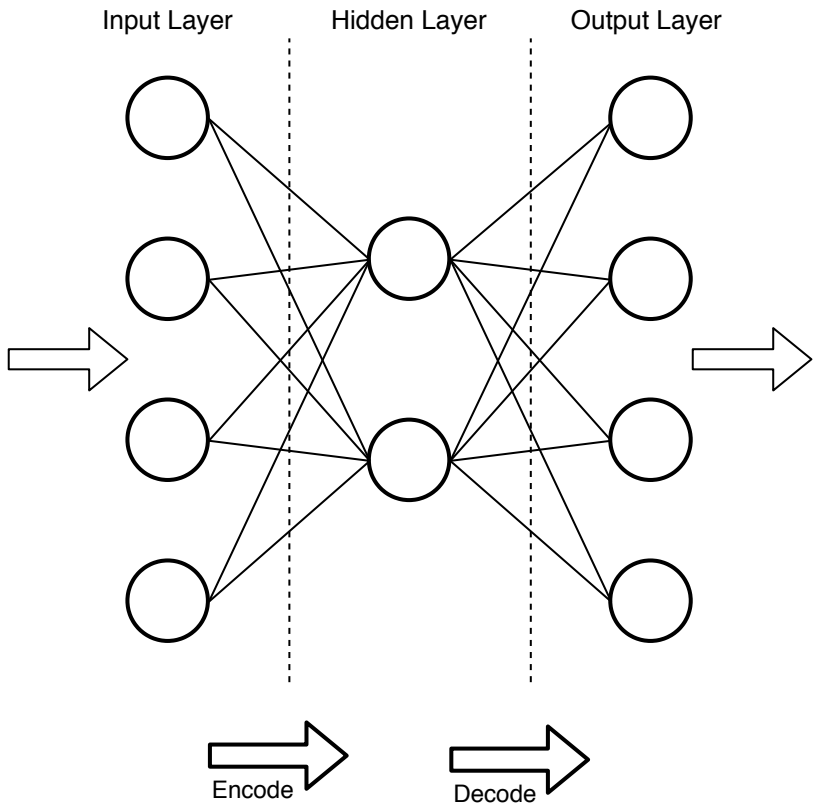

Figure 4.4: Autoencoder

Listing 4.4: Model Creation

```
def create_model(time_window_size, metric):
        model = Sequential()
        model.add(Conv1D(filters=8, kernel_size=5, padding='same',
            activation='tanh', input_shape=(time_window_size, 1)))
        model.add(GlobalMaxPool1D())
        model.add(Dense(units=time_window_size, activation='tanh'))
        model.compile(optimizer='adam', loss='mean_squared_error', metrics=[
            metric])
        return model
```

The hidden layer is defined using the *Conv1D* Keras layer. This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs [27]. The input dimension is defined in the *input_shape* argument, which is set to *time_window_size* that is the dimension of the input data.

In addition, the output layer is defined using the *Dense* Keras layer, which is a regular

densely-connected layer. Its dimension is determined in the *units* argument, which is set to *time_window_size* that is the input dimension.

The objective of the autoencoder is to reconstruct the input in the output after reducing its dimensions and for that purpose, the metric that is going to be minimized is the *Mean Squared Error* function. Being x_i the input dataset and $\hat{x}_i$ the output, the function to minimize is shown in Equation 4.2.

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(\hat{x}_i - x_i)^2 \tag{4.2}$$

Finally, the optimizer selected to train the neural network is the *adam* optimizer [30]. *Adam* is an algorithm for first-order gradient-based optimization stochastic objective functions, based on adaptive estimates of lower-order moments. It is computationally efficient, has little memory requirements and it is invariant to diagonal re-scaling of the gradients. In addition, the selected activation function is the *hyperbolic tangent*, shown in Fig. 4.5, which advantage is that negative inputs will result as strongly negative and zero inputs will result near zero. Furthermore, being the parameters b the bias, w the weight and φ the activation function, the mathematical expression is shown in Equation 4.3.

The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero.

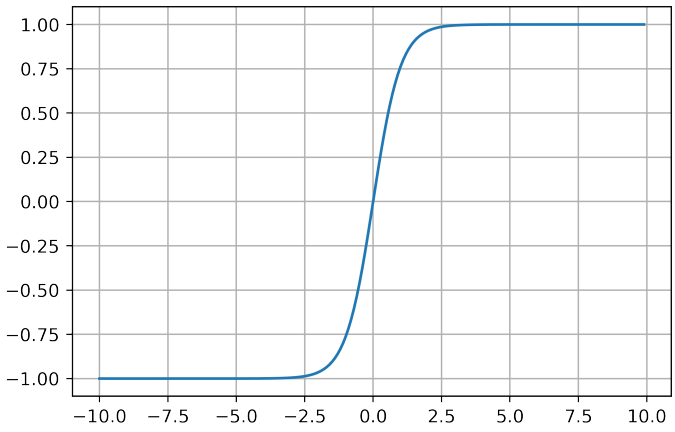

Figure 4.5: Hyperbolic tangent

$$y_k = \varphi(\sum_{j=1}^{m} w_{kj}x_j + b_k) \tag{4.3}$$

Therefore, the aim of the autoencoder is to reproduce the input, after reducing its dimension, in the output with a certain error. That error, the reconstruction error, will be calculated and used in the Anomaly Detection Module, explained in Sect. 4.1.4, in order to classify the data between normal or attacked.

For that purpose, the neural network has to be trained with normal data (which has not attacks) and then, tested with attack data. That is the reason why the Multi-Agent System, introduced in Sect. 4.1.1, generates the different explained datasets. Therefore, to train the autoencoder, the following steps have been followed:

- First, the model has to be created. For this purpose, the function explained in Lst. 4.4 is used.
- Once the model has been created, the *Keras fit()* function is used to train it. This function has the following parameters: (i) x and y are the input datasets, (ii) *batch_size* number of samples per gradient update. The chosen value is 32 which is the default value assigned by *Keras*, (iii) *epochs*, number of iterations over the entire dataset during training. The chosen value is 100, (iv) *verbose*, it indicates how the user wants to see the progress of the training process for each epoch. The chosen value is 1, which shows a progress bar, (v) *callbacks*, the callback used is the *ModelCheckpoint* of *Keras* that saves the model after every epoch. In this way, if the training is interrupted, the achieved results are saved into a file in order to recover the results that had been already obtained.

Listing 4.5: Model Training

```
self.model.fit(x=input_timeseries_dataset, y=dataset,
               batch_size=batch_size, epochs=epochs,
               verbose=self.VERBOSE,
               validation_split=validation_split,
               callbacks=[checkpoint])
```

- After training the model, the next step is saving it into a file, as well as the different weights calculated as a result of the training. Therefore, when the model is going to be used to make predictions, it will have to be loaded, as well as the calculated weights, from the generated files. In this way, the model only has to be trained once since the training is one of the most time-consuming tasks and thus, a great deal of time will be saved.

Listing 4.6: Saving Model

```
    open(architecture_file_path, 'w').write(self.model.to_json())
    self.model.save_weights(weight_file_path)
```

Once the training process is done, the autoencoder will have learned the correlations between the different features of the normal dataset, that is to say, it will have learned the normal behaviour of the grid. The *predict* function, showed in Lst. 4.7, is used to test the autoencoder and generate the output. This function uses the *predict* function of *Keras*, which generate the output predictions for the input samples and it makes the computation in batches. The parameters used are the ones set by default, which the most important one is the *batch_size* with a value of 32. In addition, before introducing the data in the autoencoder, it is necessary to resize them so that they have the appropriate dimensions.

Listing 4.7: Predict function

```
def predict(self, input_data):
    input_dataset = np.expand_dims(input_data, axis=2)
    predicted_dataset = self.model.predict(x=input_dataset)
    return predicted_dataset
```

Therefore, if the autoencoder is tested with attack data (anomalous data), it will not be able to perform the reconstruction of the data correctly since it has learned a normal behaviour of the grid. This means that the reconstruction error of anomalous data will be higher than the one of normal data. Thus, the Anomaly Detection Module will process the data obtained from the autoencoder in order to calculate the reconstruction error and therefore, be able to infer whether there has been an attack or not. This processing will be mainly the comparison between the reconstruction error calculated with normal data and with anomalous data.

4.1.4 Anomaly Detection Module

This module is in charge of classifying the data between normal or anomalous from the data obtained from the autoencoder. Therefore, in order to detect attacks, the autoencoder reconstruction error will be calculated and then analyzed.

As explained in Sect. 4.1.3, the autoencoder learns the existing correlations between the different features of the data. Thus, the autoencoder has to be trained with normal data, so that it learns the normal behaviour of the grid. Once trained, it has to be tested. This test consists of two parts, first, it is tested using normal data and then with attack data, calculating the reconstruction error in every test. That reconstruction error, when tested with attack data, will be much higher than the one obtained when tested with normal data. For this reason, it is important to calculate the reconstruction error of normal data because, in this way, the different reconstruction errors can be compared. This is because the neural network does not know how to reconstruct an anomalous behaviour since it has learned the normal behaviour of the grid during training.

Furthermore, the absolute value of the error is not as interesting as its relation with the error in a known normal situation. For this reason, the error is normalized and, in this way, its comparison with the error obtained in a normal situation is easier. This normalization consists in dividing the actual error obtained, by the error computed during training, that is to say, the error computed with normal data. This error normalization is shown in Equation 4.4.

$$e_{norm} = \frac{e_{test}}{e_{train}} \tag{4.4}$$

Regarding to [8], in order to classify the data between attack or normal behaviour, the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) are used. Therefore, the reconstruction error previously discussed will be the MAE and the RSME. In Equation 4.5 is shown the formula for calculating the MAE and in Equation 4.6 the one used for RSME.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|(x_i - \hat{x}_i)| \tag{4.5}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2} \tag{4.6}$$

Both the MAE and the RMSE are calculated for the training and test datasets. In addition, in order to compare them in an easier way, they are normalized as it is explained before.

Therefore, the normalized training reconstruction error will be one, as it is divided by itself. Furthermore, the normalized reconstruction error when testing the neural network with normal data (without attacks) will be a value very close to one, as the network has learned how the grid behaves in a normal situation. However, the normalized error when testing with attacked data will be much higher than one, because the network does not know how to reconstruct that data. Table 4.2 shows a summary of the different values that the normalization error can take.

	Normal	Attack
Training	1	-
Test	$\sim$1	$>>$1

Table 4.2: Normalized Error

This calculation of the reconstruction error and its normalization allow us to know in which houses there has been an attack in case it is calculated for the whole month of data. The data correspond to the power consumption values of the houses during a month as this is what has been generated using Mosaik. For this purpose, a threshold has to be calculated in order to be able to divide between normal houses and attacked houses. For the calculation of this threshold, an initial value greater that one is set, since, as explained before, the error has to be much greater than one. This threshold will be increased, testing different values, to finally choose the one that offers the best result. Therefore, the data has to be labelled, as explained before in Sect. 4.1.2, in order to calculate the optimal value of the threshold and thus, be able to know which houses have been attacked. If the data were not labelled, the threshold could not be calculated because there would be no way of knowing whether the autoencoder has calculated the error correctly. Once this threshold is calculated, the data no longer needs to be labelled.

The results of a simulation in which four houses suffered an attack are shown in Fig. 4.6. The results show that houses number 9, 20, 22 and 27 have a much bigger error than the rest of them, so this means that they are the four attacked houses. Comparing these results with the labels that indicate whether there has been an attack, it can be seen that the classification has been done correctly.

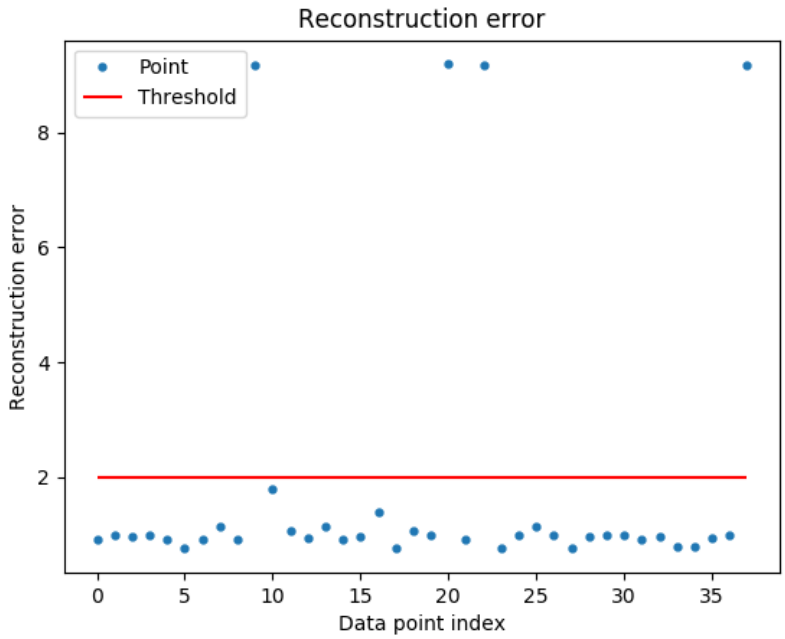

Figure 4.6: Reconstruction Error

However, in order to know the exact moment in which the attack takes place, different operations have been applied to the error. In this way, it can be processed in order to make inferences about it. For this purpose, instead of calculating the reconstruction error for the entire simulation course, as it was done to know in which house had occurred an attack, it is calculated for each entry (row) of the DataFrame that contains the different features, explained in Sect. 4.1.2.

Therefore, once the reconstruction error has been calculated for each DataFrame entry, it has to be processed in three stages: calculation of an accumulator window, calculation of the optimal threshold and filtering of the results.

1. ***Accumulator window.*** First, a window is created with the values of the reconstruction error. This window consists of modifying each value by the sum of all the components of the window. An example of how the values are modified by the window is represented in Table 4.3. The original values of the table are random values as it is an example and a value of 3 for the window is suppose. Therefore, the modified values will be the sum of each original value with the two previous ones.

Original	0.8	1.1	0.85	3.4	4.5	3.8	1.2	1.4	1.1
Modified	0.8	1.9	2.75	5.35	8.75	9.15	9.5	6.4	3.7

Table 4.3: Window error

This accumulator window is useful to improve the detection of attacks. This is because the window takes into account not only each value individually but also those near it. The smart grid data provided by Mosaik is taken every 15 minutes, and it is normal for an attack to have a longer duration. For this reason, by taking into account the influence of the values that are close to each value, the detection of attacks could be improved.

2. ***Getting the optimal threshold.*** Using the new error values obtained from the previous step in which an accumulator window is applied to the reconstruction error, it is necessary to calculate the optimal threshold that classifies the data between normal or anomalous. As explained before, in order to calculate the optimal threshold, first it is set to a value close to one, and then, by means of small increments, the different results are evaluated. In this way, it can be found out which value provides the best results. The obtained optimal threshold will allow us to classify the data in the same way that it was done in Fig. 5.6.

   Therefore, this error processing step classifies the data between normal and attacked. But when analyzing the results obtained, it was concluded that this classification could still be improved. This is due to the fact that in certain attacks, the classification is not completely precise and that is why the next step, in which the results are filtered, is necessary.

3. ***Filter the results.*** After calculating the new error values when applying the accumulator window and calculating also the optimal threshold, the classification performed can be improved by analyzing the results.

   In an example case, where an attack is detected at 9 a.m., 10 a.m. and 11 a.m., but the values between them have not been detected as attacks, the attack could be considered to have lasted the entire time slot between 9 a.m. and 11 o'clock in the morning. Therefore, in order to improve the accuracy of attack detection, if in a time slot (e.g. 10 hours) the 30% of the values have been detected as attacks, we can infer that the attack has lasted all that time slot.

   In addition, it has been concluded that this course of action is appropriate since supply disruption attacks or, specifically, False Data Injection attacks are not usually carried

out intermittently or do not have a very short duration. These attacks usually last a long time because the longer the attack lasts, the more money the attacker is going to save.

Moreover, once the results were filtered the classification improved. It was thought that this step could affect the performance of that classification since values detected as normal were being changed by attacks. However, the classification of non-attacked points did not get worse and therefore, the conclusion reached when analyzing the results was satisfactory.

Furthermore, an additional modification to those mentioned above was made so that higher-value errors were given more importance than lower-value errors. This modification consisted in multiplying the error by a function, shown in Fig. 4.7, to then calculate the accumulator window. In this way, through the multiplication of the value by the function, the influence that values have on those of their environment would be increased. This is because a reconstruction error close to the threshold is not the same as one with a much higher value due to, the larger the error, the easier it will be to detect the error. Therefore, when calculating the accumulator window, multiplying the data by this function should improve the results, since it takes into account the environment in which each value is located.

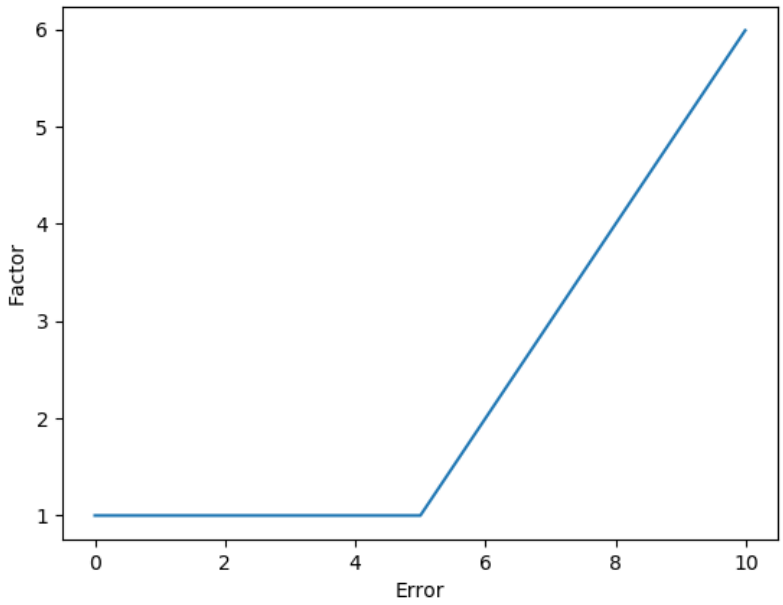

Figure 4.7: Error Modification

In a hypothetical case, like the one represented in the figure, where the optimal threshold was 5, the values below that threshold which are considered normal would not be modified, that is to say, they would be multiplied by one. Furthermore, values above the threshold, which are considered attacks, would be multiplied by the corresponding factor m. However, in view of the results, this modification of the error was dismissed as it did not produce any improvement in the classification accuracy.

In conclusion, in the anomaly detection module, the results obtained from the autoencoder are treated in order to classify between normal and attacked data. First, the reconstruction error is calculated using the output and the input of the autoencoder. Then, with the obtained reconstruction error, an accumulator window is calculated to take into account the influence of the values close to each value. Once the accumulator window is obtained, the optimal threshold is calculated which allows us to carry out the classification. Finally, the results of the classification are filtered in order to improve it after having analyzed the previous results. In addition, other error transformations were tested which were discarded as they did not have any positive effect on the results.

CHAPTER 5

Case study

In this chapter, the evaluation of the detection of different attacks by the system developed in this project is presented.

5.1 Topology Attack

The attack chosen for its implementation is the disconnection of a network element, specifically a house. This is a type of disruption attack known as a Topology Attack [29].

On one hand, the possible origins of these attacks are the following:

- *Act of vandalism.* This attack could be originated as an act of vandalism in order to harm a specific consumer, cutting off its electricity supply, or the electric company damaging it economically or its reputation.
- *False Data Injection Attack.* Another possible origin of this attack could be the modification of the value of power consumed by the user, which is a False Data Injection attack. This could be done with the aim of defrauding economically the electrical company [15].

On the other hand, these attacks could have different consequences:

- *Blackout in the attacked house.* This would harm its inhabitants and the reputation of the company.
- *Destabilization of the network.* If this attack was made on a large number of smart meters at the same time, it could cause a destabilization of the network because it could bias the power system state estimation [31].
- *Damage of network elements.* They could cause the possibility of ending up with several elements of the grid damaged, which will result in equipment having to be reset or even replaced by a new one [15].
- *Mislead the control center.* In the event that these attacks are successful, they could mislead the control center to take erroneous actions, or at least, make the control center distrusts the state estimation.

A simplified smart grid scenario is represented in Fig. 5.1. In this scenario, the electrical company manages the smart meters at the house of the customers through the AMI head-end. This head-end is in charge of controlling the meters via the Data Concentrator Unit (DCU). In addition, the DCU concentrates all the traffic of a Neighborhood Area Network (NAN). Moreover, the Outage Management System (OMS) allows the electric company to analyze and look for user reports [55].

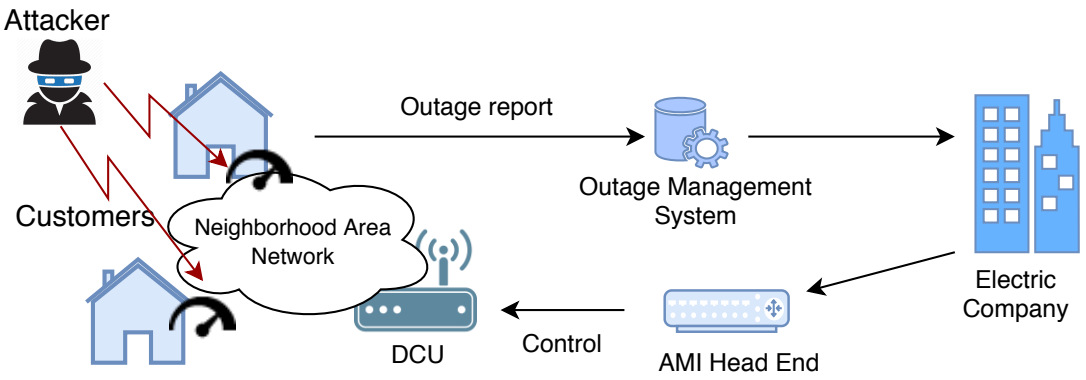

Figure 5.1: Simplified scenario

Furthermore, a graphical representation of the whole scenario chosen for study, which is a residential area, is shown in Fig.5.2. This area is composed of different elements: households, photovoltaic panels, PQ buses, and the power grid's transformer node. Households consume energy, whereas the photovoltaic panels associated with them are generators. Furthermore, the power grid's transformer node supplies energy to all the residential area and the PQ buses (P and Q are the active and reactive power), also called Load Buses, balance the active and reactive power of the grid.

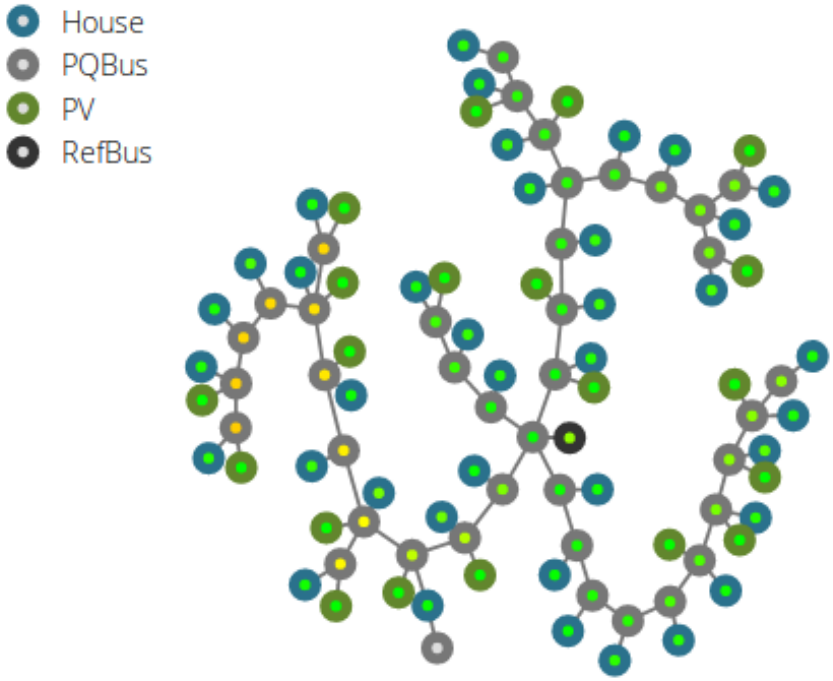

Figure 5.2: Attack Scenario [47]

This representation is the one used by Mosaik, which is the ABMS tool used for simulating smart grids, explained in Sect. 3.2.1. The different elements are represented by nodes of different colors. The blue nodes are houses, the green ones are photovoltaic panels, the grey ones are PQ Buses and the black one is the power grid's transformer node.

Moreover, the attackers will be associated to the houses that will be disconnected and it would depend on if it is an individual attack, that affects only one house or a large-scale attack that affects to a large number of them.

The scenario to be studied is a residential area. This area has 38 houses, 20 of which have solar panels. The number of houses attacked will vary depending on the attack being studied.

Therefore, Mosaik is used with the new attack simulator implemented to generate different datasets. A dataset with a normal behavior of the grid is necessary, as explained before, in order to train the autoencoder. In addition, several datasets with different False Data Injection attacks are also generated in order to test the system. These datasets store the power consumption values of the 38 houses over the course of the simulation, which is one month (31 days).

The power consumption values of a house during a month, generated by Mosaik, are represented in Fig. 5.3. These values represent a normal behavior of the grid, that is to say, the house has not suffered any attack.

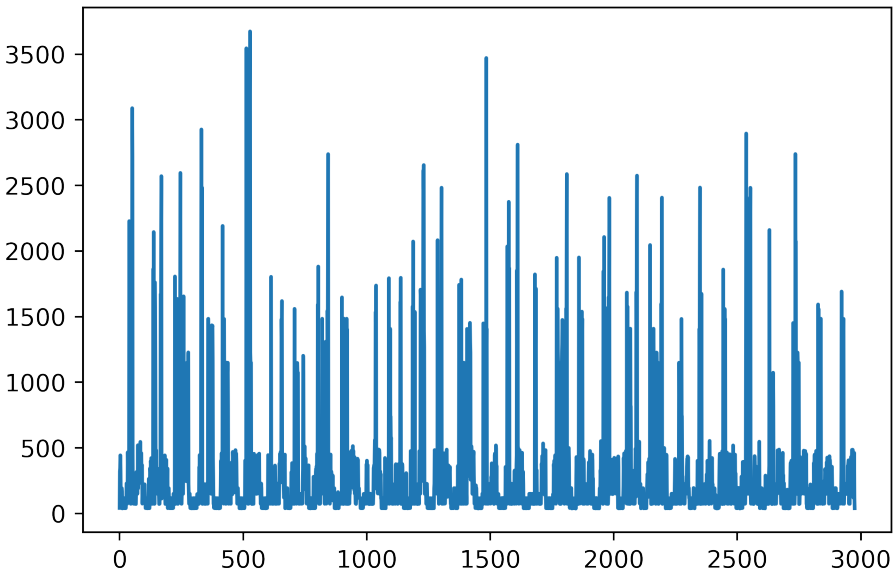

Figure 5.3: Power consumption values of a house

In the following subsections, the process of knowing in which house there has been an attack in all the month that the simulation lasts is first explained. Then, the process to know in which moment the attack has taken place is also explained.

5.1.1 Detection of which house has suffered an attack

For this case study, the attack that has been decided to study is the one depicted in Fig. 5.4. This attack consists in that at a certain moment of the simulation, the value of the consumed power becomes 30% of the real power consumption value. Therefore, it is a False Data Injection attack. Specifically, in this case, 4 of the 38 houses of the scenario has suffered an attack.

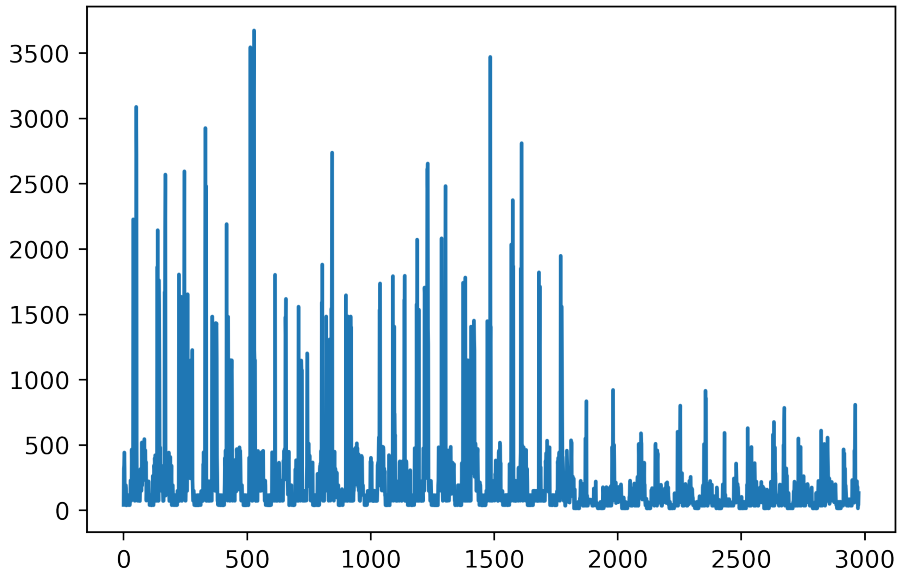

Figure 5.4: Power consumption values of an attacked house

Once the different datasets (normal and attacked) have been generated with Mosaik, they have to be preprocessed. This preprocessing consists of cleaning the unnecessary data, generating features and finally normalizing the data. In Fig. 5.5, the DataFrame obtained is represented. Furthermore, a DataFrame is generated for each house and, in this way, the data will be organized for saving time when analyzing them.

Concretely, a *.pkl* file will be generated with the data of all the houses in a normal scenario and then, a *.pkl* file for each house in a scenario in which an attack has taken place. In this way, we will facilitate the training process, having one single file as the input of the autoencoder. Furthermore, when detecting an anomaly, it will be better for the

	Day	Hour	Minute	0	15	30	45	60	75	90	...	mean	diff_mean-1	diff_mean+1	std	diff_values
0	1.0	0.0	0.0	136.76	330.67	230.49	268.72	471.48	137.80	62.75	...	212.63500	212.63500	-9.10625	132.659187	-74.35
1	1.0	0.0	15.0	330.67	230.49	268.72	471.48	137.80	62.75	62.41	...	203.52875	-9.10625	-9.81625	139.859930	-266.76
2	1.0	0.0	30.0	230.49	268.72	471.48	137.80	62.75	62.41	63.91	...	193.71250	-9.81625	2.38000	133.188628	21.65
3	1.0	0.0	45.0	268.72	471.48	137.80	62.75	62.41	63.91	252.14	...	196.09250	2.38000	-25.65625	133.992248	-19.19
4	1.0	1.0	0.0	471.48	137.80	62.75	62.41	63.91	252.14	249.53	...	170.43625	-25.65625	-41.49375	137.240395	-408.01
5	1.0	1.0	15.0	137.80	62.75	62.41	63.91	252.14	249.53	63.47	...	128.94250	-41.49375	0.40500	76.838641	1.73

Figure 5.5: Features

autoencoder having each house in separated files.

After the DataFrames have been created, they are stored in these *.pkl* files. In this way, a lot of time is saved because when doing different tests, the DataFrames are only generated once (which takes a lot of time) and then loaded directly from the file (which does not consume much time).

Once all the dataframes have been generated, the autoencoder is trained with the one that contains a normal grid behavior. After training, which takes a bit of time, the autoencoder is fed with the DataFrames that contains the attacks. Therefore, the reconstruction errors (RMSE) of each DataFrame is calculated and normalized regarding the normal behavior one.

In Fig. 5.6, the normalized reconstruction error of each house that composes the scenario and, in which four of them have suffered an attack, is represented.

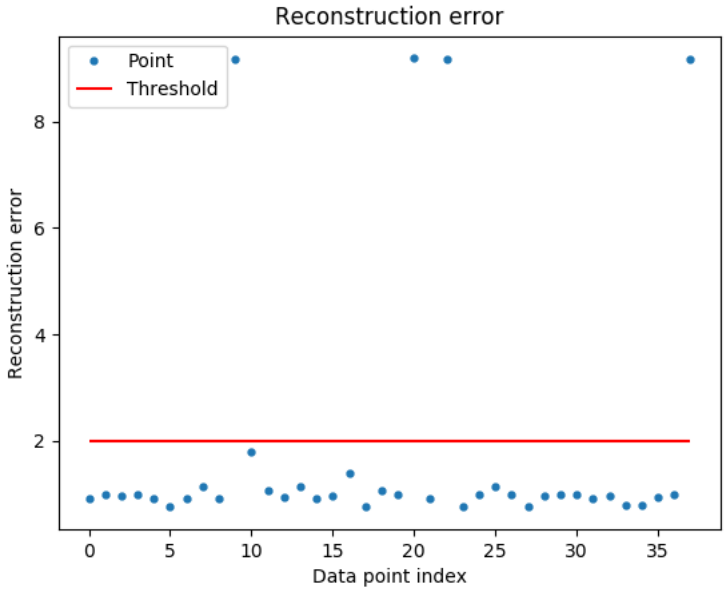

Figure 5.6: Reconstruction Error

As can be seen, the houses 9, 20, 22 and 27 have a much bigger error than the rest of them, which means that they are the four attacked houses. After the simulation was carried out, it was analyzed to find out which houses had suffered an attack and they were the ones detected by the autoencoder. In this way, by using a threshold, it is possible to know which houses have suffered an attack. However, it is not possible to know when the attack occurred because the reconstruction error is calculated throughout the whole month that the simulation lasts.

In order to detect when the attack has taken place, it is necessary to calculate the reconstruction error for each DataFrame entry and not for the entire DataFrame as has been done in this case study. This process is explained in depth in the following section, Sect. 5.1.2.

The result obtained for other False Data Injection Attacks, such as for example setting the consumption to 0, is similar to the one presented. That is, the reconstruction error obtained for houses that suffer attacks is much greater than those that do not suffer them and therefore, they can be identified through a threshold.

Therefore, it can be concluded that the autoencoder has a good performance for this purpose since it detects without problems the houses that have suffered an attack. In addition, other attacks were tested, such as the reduction of power consumption values to 0%, 10% and 20%, in which the results were also satisfactory.

5.1.2 Detection of when the attack has occurred

In this subsection, the case study that consists of detecting when the attack has occurred is presented. First, in Sect. 5.1.2.1 the False Data Injection attack that sets the power consumption value to 0 is presented. Then, Sect. 5.1.2.2, Sect. 5.1.2.3 and Sect. 5.1.2.4, explain the attacks that sets the power consumption value to 30%, 10% and 20% of the original values, respectively.

The data generation process and the pre-processing of the data is the same as explained above in Sect. 5.1.1. First, Mosaik generates the different datasets. Then, they are pre-processed in order to generate the DataFrames. The main difference regarding what is explained in Sect. 5.1.1, is that when calculating the reconstruction error, it is not calculated for the whole month of the simulation, but for each DataFrame entry. In this way, it can be detected if in a certain moment there has been an attack.

5.1.2.1 Attack 0%

In this section, an attack that consists of the total disconnection of a house since the power consumption values become zero is going to be studied. Nevertheless, this attack is theoretical, it does not usually occur in real life as there would always be a residue of noise in the signal.

In Fig. 5.7, the power consumption values of an attacked house during the whole month of the simulation are represented. As can be seen, at a certain moment the values of the signal become zero.

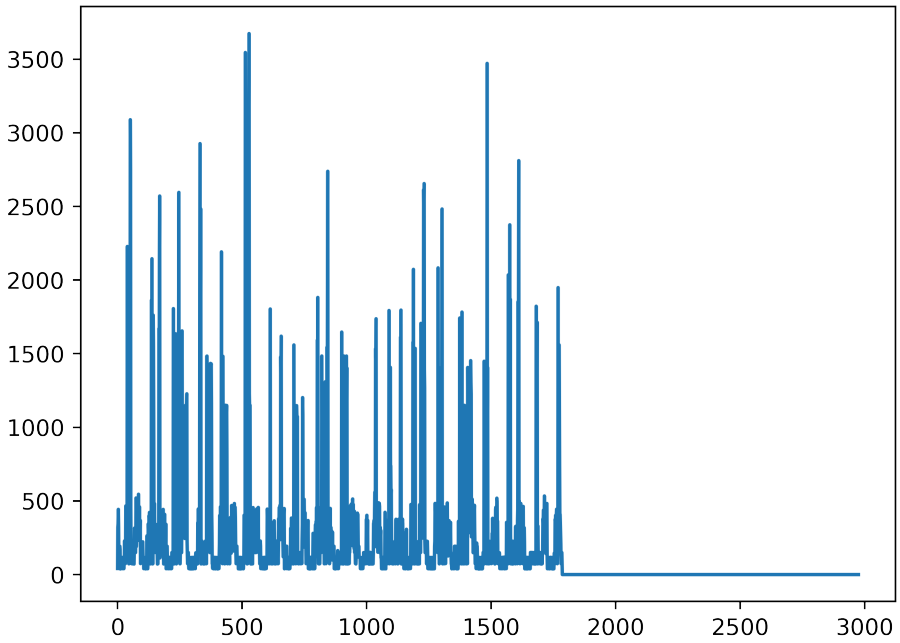

Figure 5.7: Attack - 0

Once the autoencoder is trained with normal data, it is tested. For that purpose, it is fed with the DataFrames that have the attacked data. With the results obtained, the reconstruction error (RMSE) is calculated and normalized regarding to the one obtained with normal data. In Fig. 5.8, it is represented the error obtained from one of the 38 houses that compose the scenario.

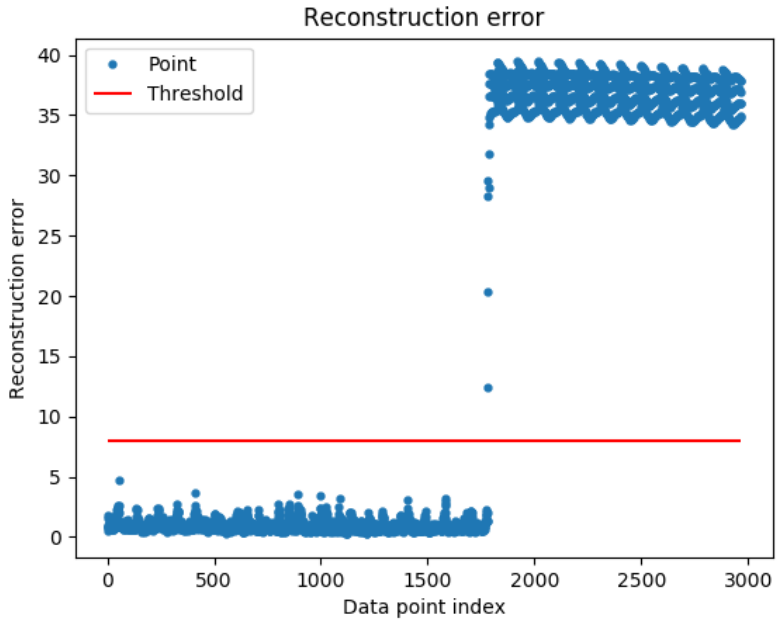

Figure 5.8: Reconstruction Error - 0

As it can be seen, there is no problem in identifying attacks because the error is much higher than the one obtained in a normal situation. Therefore, by calculating a threshold it will be possible to identify which DataFrame entries are considered attacks. In this way, it can be known when the attack occurs.

	Precision	Recall	F1-score	Support
0	1.00	0.99	0.99	66082
1	0.98	1.00	0.99	43771
micro avg	0.99	0.99	0.99	109853
macro avg	0.99	0.99	0.99	109853
weighted avg	0.99	0.99	0.99	109853

Table 5.1: Classification Report - 0

Once the classification is done, the classification report is calculated. Table 5.1 shows the classification report obtained. As can be seen, the classification is almost perfect, with an F1-score of 0.99, which was the expected result after seeing the representation of the reconstruction error in Fig. 5.8.

5.1.2.2 Attack 30%

In this section, an attack that consists of a False Data Injection attack that sets the power consumption values to a 30% of the original values is studied.

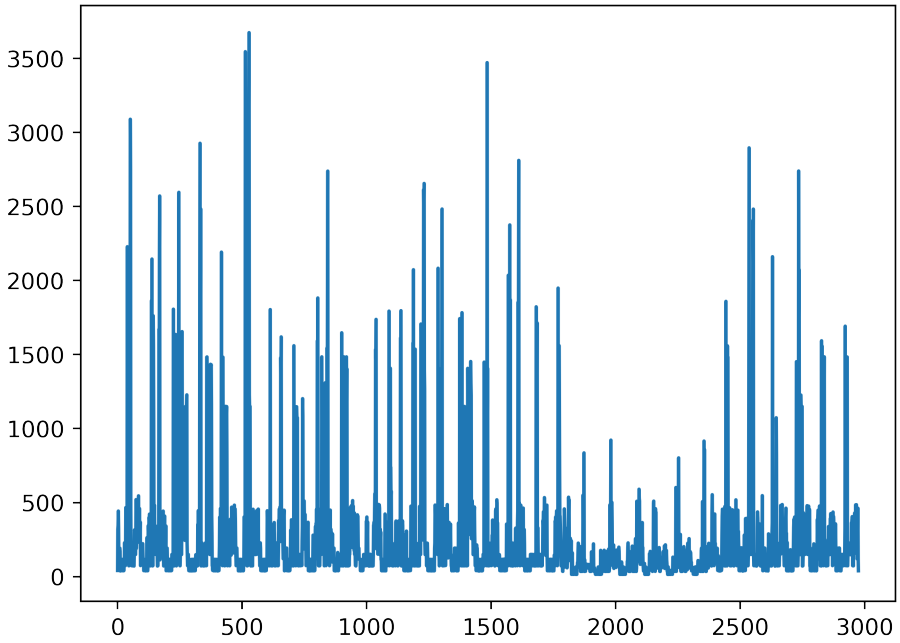

Figure 5.9: Attack - 30%

In Fig. 5.9, the power consumption values of a house that has suffered this attack are represented. As can be seen, in a certain moment of the simulation the values follow the same trend as the original ones but with a value of 30% of them and after a period of time the attack ends and the values are the original ones again.

Following the process explained before, the autoencoder is fed with the attacked data and with the results obtained the reconstruction error (RMSE) is calculated. In Fig. 5.10, the obtained reconstruction error is presented. If the graph is analyzed from a visual perspective, it can be inferred that there has been an attack on a time frame of the simulation. However, the main problem is that using a threshold, all the attacked points will not be detected as attacks. This is due to the fact that, observing Fig. 5.10, there are many values in the attacked time frame that are very similar to the values when no attack has occurred. These values and the trend that they follow are very similar to those that exist during a normal behaviour of the grid.

Therefore, in order to perform the classification between normal or attacked data, the calculation of a threshold is not sufficient to detect which values are considered attacks.

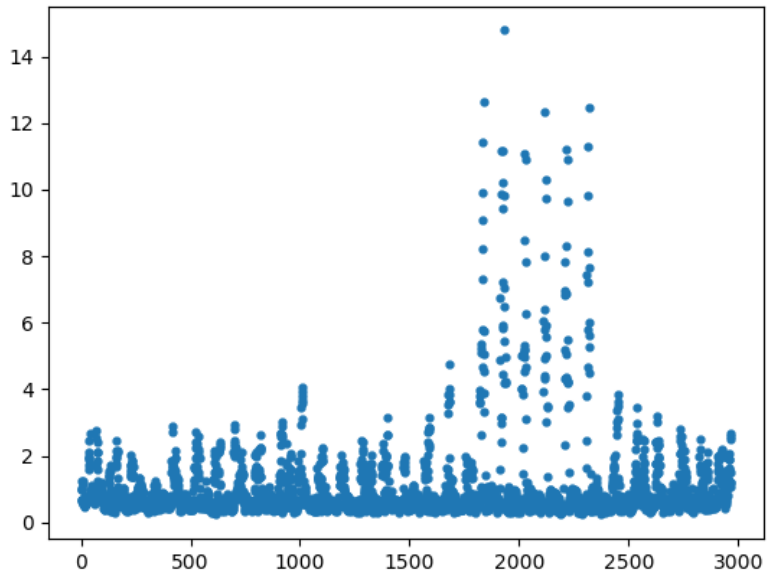

Figure 5.10: Reconstruction Error - 30%

For this reason, certain transformations have to be applied to the reconstruction error following the methodology explained in Sect. 4.1.4. These transformations are the following: calculation of an accumulator window, calculation of the optimal threshold, and the filtering of the results.

First, the accumulator window has to be calculated. This window consists in that each value of the reconstruction error is modified by the sum of the previous values that are inside that window. In this way, each value and those near it are taken into account. This improves the detection of the attack since an attack does not usually last 15 minutes (which is the time between each value generated by Mosaik) but usually have a longer duration. Therefore, in this way, each value is taken into account and the influence that the values of its environment have on it, too.

Once the accumulator window is determined, the optimal threshold that allows us to perform the classification has to be calculated. For this purpose, a value greater than one is set for it and it is increased gradually. Each time the threshold is increased, the result obtained is observed, so that, after testing the different threshold values, the one that provides the best result can be chosen. This optimal threshold calculation allows us to perform the classification between normal data and attacked data.

The results obtained were analyzed and it was observed that there was still room for

improvement. Therefore, it was concluded that if an attack was detected at 10 a.m., 11 a.m. and 12 a.m. but not in the intermediate values, it could be inferred that the attack lasted for that entire time slot. In addition, false data injection attacks do not usually last 15 minutes, as these are mainly carried out by attackers that want to reduce the amount of the electricity bill. Moreover, these attacks are usually of long duration. Consequently, the results were transformed regarding to the analysis carried out.

	Precision	Recall	F1-score	Support
0	0.89	0.98	0.94	90174
1	0.89	0.53	0.66	22648
micro avg	0.89	0.89	0.89	112822
macro avg	0.89	0.76	0.80	112822
weighted avg	0.89	0.89	0.88	112822

Table 5.2: Classification Report - 30%

Once the explained transformations are carried out, the classification report is calculated. It is represented in Table 5.2. If the results are analyzed, it can be inferred that they are good results. The worst result obtained is the recall of the attacks, but this is due to what was explained before. Looking at the graph of the reconstruction error obtained in Fig. 5.10, almost 80% of the values are very close to one, which makes very difficult to make the classification. Nevertheless, thanks to the transformations that have been made, it has been possible to improve the results. However, the value of the precision is quite high, and therefore, a good value of F1-Score is obtained. In addition, if the attack instead of having a certain duration was until the end of the simulation as in the previous section, the results are very similar.

5.1.2.3 Attack 10%

In this section, the attack that consists of setting the power consumption values to 10% of the original ones is studied.

In Fig. 5.11, the power consumption values of a house that has suffered this attack are represented. As it can be observed, in a certain moment of the simulation the values become the 10% of the original values.

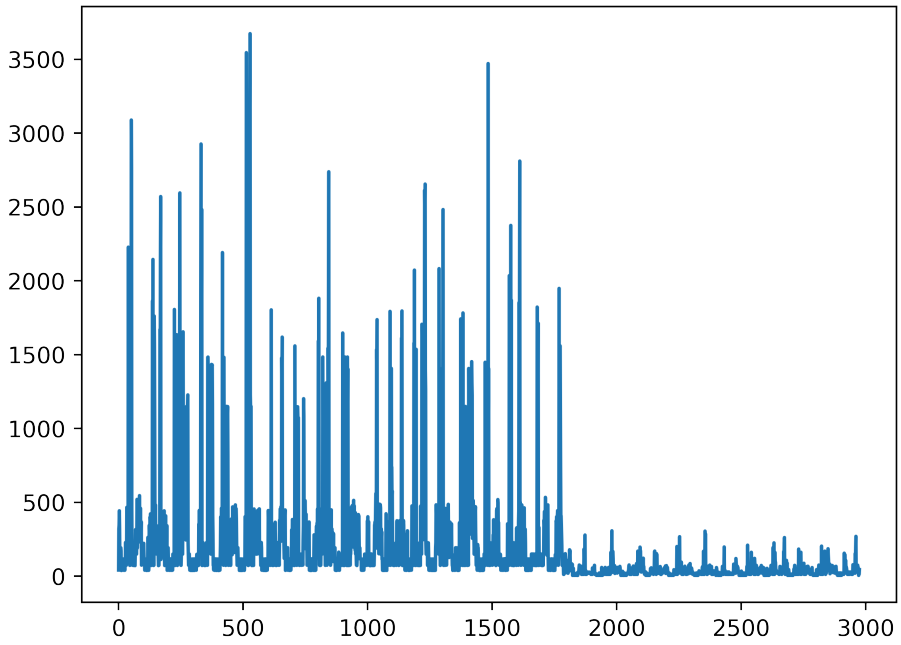

Figure 5.11: Attack - 10%

Following the data processing explained above, the data is introduced into the autoencoder and with the results that are obtained, the reconstruction error is calculated. This reconstruction error is represented in Fig. 5.12. As can be seen, now the attack is easier to detect because the number of points with an error higher than one is greater than in the previous case. Consequently, it can be deduced that the result will be better than the previous case.

Therefore, if the reconstruction error is treated and the transformations explained above are made, the classification report can be obtained. This classification report is represented in Table. 5.3. The results obtained from the classification are almost perfect and very similar to those obtained in Sect. 5.1.2.1, where the attack studied was total the disconnection of the houses. Both the precision and recall values are very high, which means that the F-score value is also high.

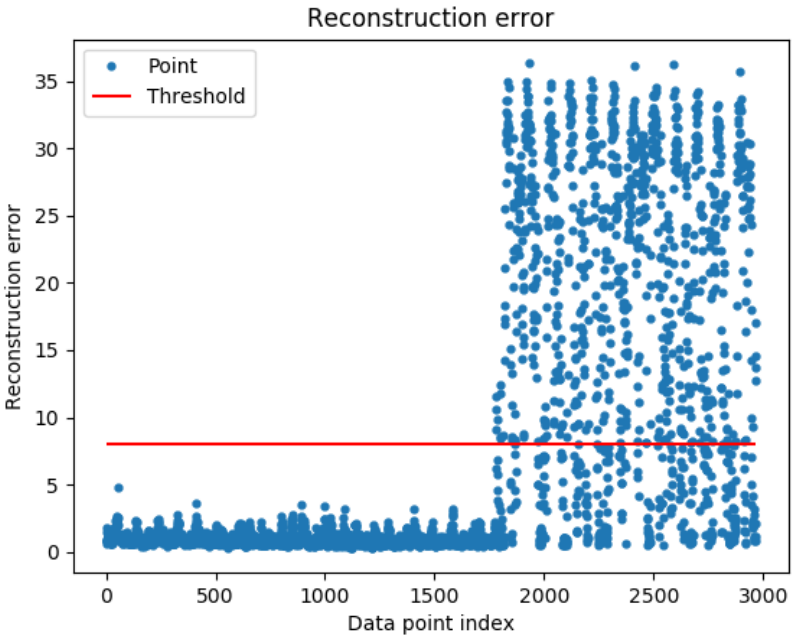

Figure 5.12: Reconstruction Error - 10%

	Precision	Recall	F1-score	Support
0	0.97	0.98	0.98	67868
1	0.97	0.96	0.96	44954
micro avg	0.97	0.97	0.97	112822
macro avg	0.97	0.97	0.97	112822
weighted avg	0.97	0.97	0.97	112822

Table 5.3: Classification Report 10%

5.1.2.4 Attack 20%

In this section, the attack that consists of setting the power consumption values to 20% of the original ones is studied.

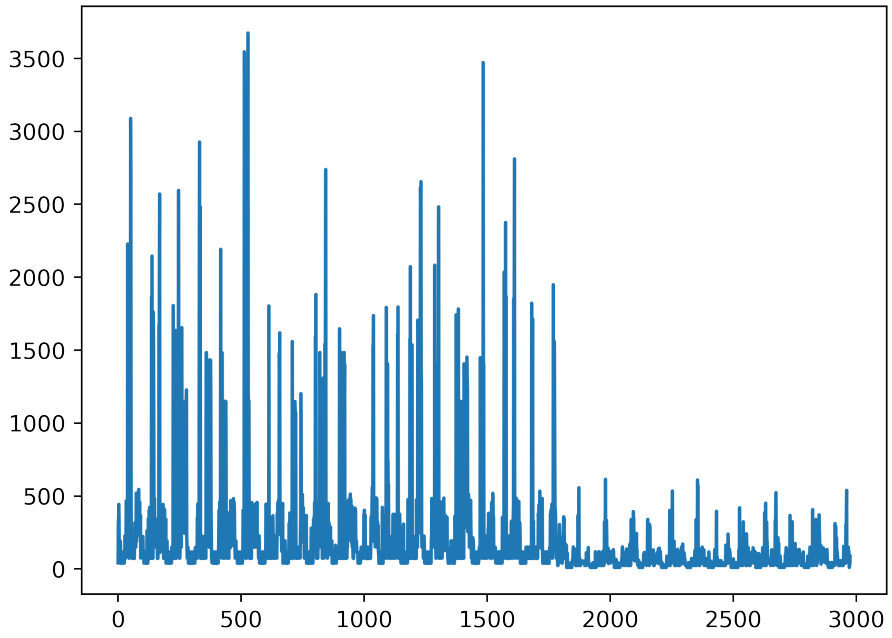

Figure 5.13: Attack - 20%

In Fig. 5.13 are represented the power consumption values of a house that has suffered this attack. As it can be observed, in a certain moment of the simulation the values become the 20% of the original values.

After preparing the data and feeding it into the autoencoder, the reconstruction error, represented in Fig.5.14, is calculated and the necessary transformations are carried out. In this way, the classification report represented in Table 5.4 is obtained.

The results are better than those obtained in the attack of the 30% despite the fact that the total F1-score is lower. This is because the number of attack entries is higher as the duration of the attack increases. But the F1-score obtained for the attacks is higher. In addition, the final values obtained from F1-score represent the good performance of the autoencoder.

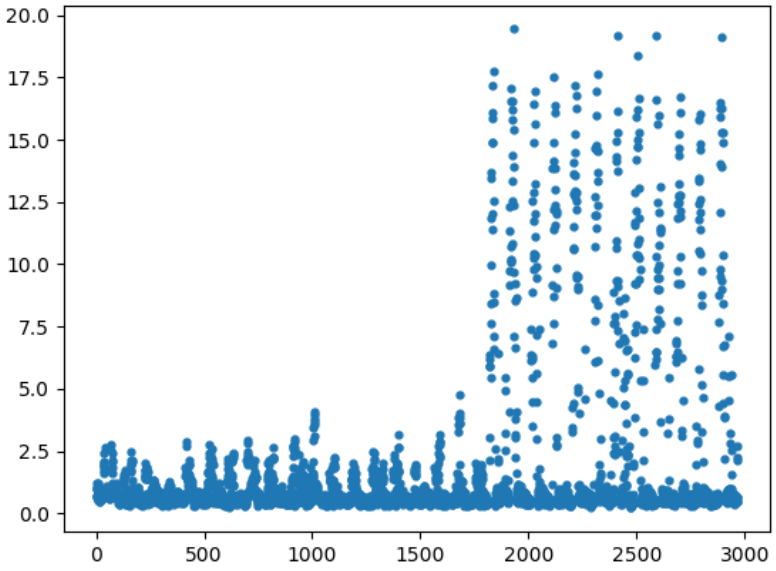

Figure 5.14: Reconstruction Error - 20%

	Precision	Recall	F1-score	Support
0	0.80	0.98	0.88	67868
1	0.96	0.64	0.76	44954
micro avg	0.84	0.84	0.84	112822
macro avg	0.88	0.81	0.82	112822
weighted avg	0.86	0.84	0.84	112822

Table 5.4: Classification Report - 20%

5.2 Attack Detection using ARIMA

In this case study, the performance of ARIMA models, explained in Sect. 3.4.1, in detecting attacks on smart grids is studied. Therefore, both the autoencoder and the anomaly detection module, explained in Sect. 4.1, are replaced by the mathematical model ARIMA. This module has not been included in the architecture of the main system as it has been developed in order to compare its performance with the autoencoder.

The attack chosen for its study is the False Data Injection attack that reduces the power consumption values to 30%, as it is represented in Fig. 5.4.

In order to apply the ARIMA model to the smart grid data, the DataFrames generated with all the features, that have been explained in Sect. 4.1.2, are not necessary. Therefore, the data used by the ARIMA model are only those that form the time series of power consumption values.

Using ARIMA it is intended to make a prediction of the following values of power consumed in the current instant so that when comparing them with the real values it can be deduced if there has been an attack or not.

First, an attempt was made to make a prediction for all the values that compose the following day (96 values), in order to compare that prediction with the actual power consumption values. In this way, the error obtained in the prediction could be analyzed and thus be able to infer if there has been an attack or not. However, when making the prediction of a day, the values obtained were constant, that is to say, they converged to the mean and therefore, it was decided to change the method of prediction.

Then, it was decided to predict only the next value of the time series. Once that value was predicted, it was inserted into the training set in order to train the model again and, in this way, be able to predict the next values of the time series recursively. The result obtained was the same as when trying to predict an entire day, the values were constant and converged to the average. This is because when the *predict* statsmodel function, represented in Lst. 5.1, has *start_index* and *end_index* as parameters, it does the same as it has been just explained. This function introduces the value that has just been predicted in the training set in order to retrain the model and predict the next value.

Listing 5.1: ARIMA Model

```
forecast = model_fit.predict(start=start_index, end=end_index)
```

Consequently, the method used to make the prediction had to be changed again. This time instead of introducing in the training set the prediction that had just been made, the real value of the time series was introduced. Once the actual value has been introduced into the training set, the model is retrained in order to predict the following values. For this purpose, the code that can be observed in Lst. 5.2 was used.

Listing 5.2: ARIMA Model

```
def arima_func(data,arima_order):
    # prepare training dataset
    train_size = int(len(data) * 0.60)
    train, test = data[0:train_size], data[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit(trend='nc', disp=0)
        yhat = model_fit.forecast()[0]
        print("Real", test[t], "predictions", yhat)
        predictions.append(yhat)
        history.append(test[t])

    return test, predictions
```

In this code, the data is first divided into two sets: the training set and the test set. Once the data is divided, the model is trained using the training set, and then the next value is predicted. This prediction is then inserted into the training set and the model is retrained to predict the next value of the test set. The for loop is used to iterate between the different values that compose the test set and thus be able to predict them.

When creating the model, it is necessary to establish the *(p,d,q)* parameters of ARIMA. In order to find the combination of these parameters that minimize the error between the prediction and the real values, a grid search has been carried out. In other words, different combinations of these parameters have been tested and the one that provides the lowest reconstruction error has been chosen. In addition, the error chosen for its minimization during the grid search was the RMSE.

The obtained values for parameters *(p, d, q)* as a result of the grid search are *(5,1,1)* respectively. Therefore, this combination will be the *arima-order* used to make the predictions.

First, the prediction of the power consumption values of a house that has not suffered any attack, that is to say, during a normal behavior of the grid, was made. In Fig. 5.15, the actual values as well as the predictions are represented. This graph allows us to compare the obtained predicted results with the actual ones and, as can be seen, although the predictions are not precise in the maximum power consumption values, they follow the trend of the real values. Therefore, the accuracy of the prediction is quite good.

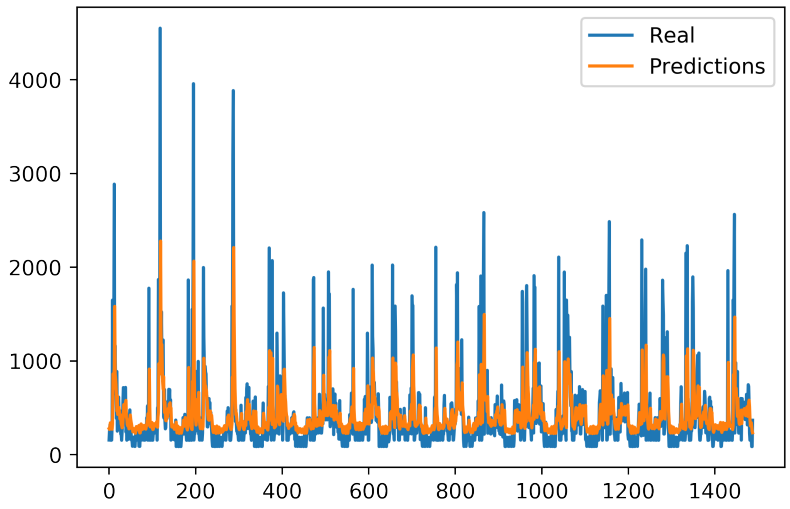

Figure 5.15: ARIMA predictions normal data

Once the behaviour of the model was analyzed in data without attacks, the model was tested with attacked data. This attack, as explained before, is a False Data Injection Attack, that sets the power consumption values to 30% of the original value. In Fig. 5.16, it can be observed the obtained predicted results. This graph allows us to compare the real data values with values of the predictions, and analyzing it allows us to draw different conclusions. When it comes to detecting the attack, this prediction has a problem and it is that the model ends up learning the behavior of the attack. This is because once a value has been predicted, the corresponding actual value is inserted into the training set in order to retrain the model. In this way, the following values can be predicted. However, when the attack begins the predicted values deviate from the real values.

Therefore, in order to analyze the evolution of the reconstruction error obtained in the prediction, its rolling mean was calculated. This allowed us to study the average error inside a window. The chosen window is formed by 96 values since it is the number of values that form a day. This moving average was calculated, assuming that the average of the error while the attack was originated was going to be higher since the model had not learned the behaviour of the attack yet.

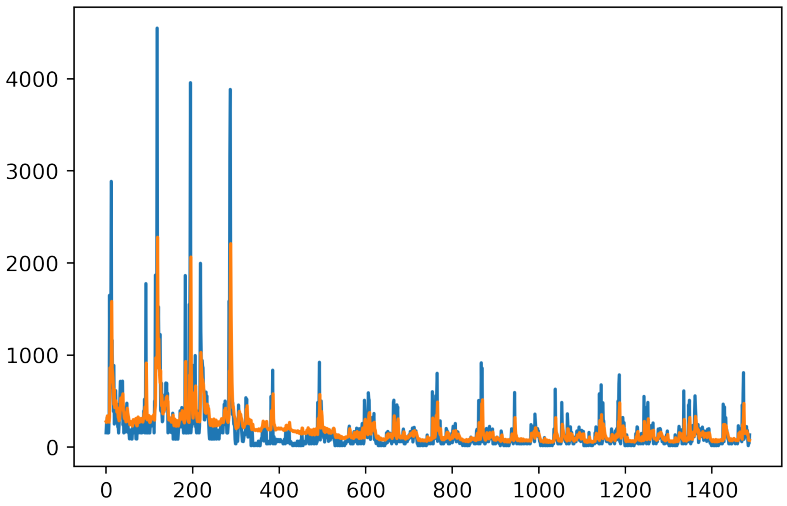

Figure 5.16: ARIMA predictions attacked data

In Fig. 5.17, the rolling mean of the error obtained in the predictions of a house in a normal behavior can be observed. The values are in a range between 0 and 60.

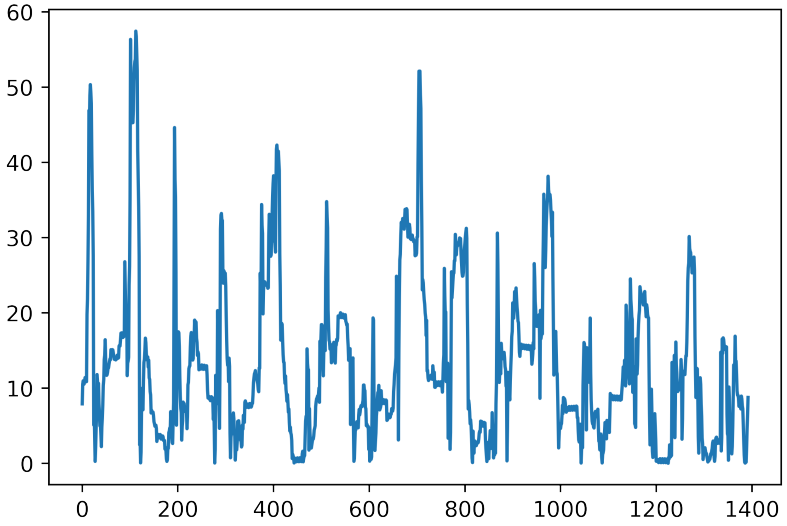

Figure 5.17: Rolling mean error normal data

Whereas in Fig. 5.18, the rolling mean of the error obtained in the predictions of an attacked house is represented. The attacked house is the same as the one represented in Fig. 5.17 and as can be seen the initial values of the time series coincide until the moment when the attack begins. However, at a certain moment, the rolling mean reaches values close to 125, which is much higher than the maximum obtained when there was no attack.

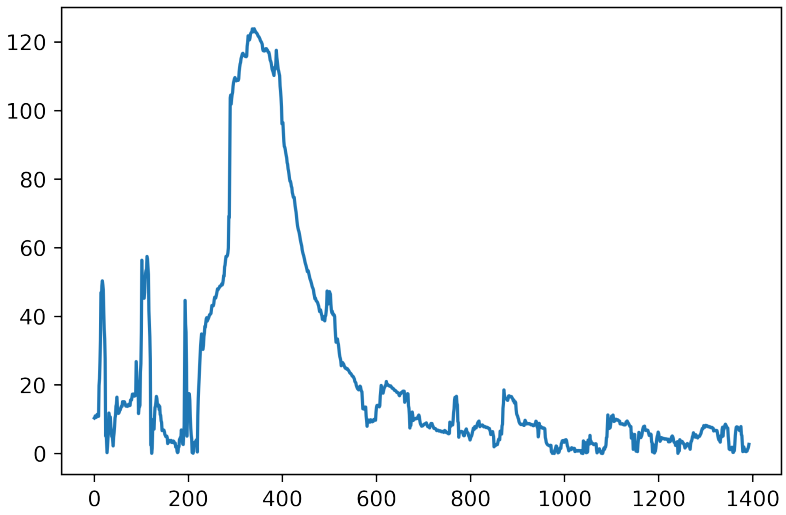

Figure 5.18: Rolling mean error attacked data

By comparing Fig. 5.17 and Fig. 5.18, the attack can be easily detected. However, it can only be detected the instant when the attack starts and this is because, as explained above, the model ends up learning the behavior of the time series when there is an attack. In addition, the rolling mean after the attack is lower, since the values of power consumed are 30% of the original and as a consequence, the reconstruction error is lower.

Therefore, using ARIMA models, we obtain an alert generator. In order to do this, it is necessary to calculate the rolling mean of the reconstruction error of a house without attack in order to compare it with the one obtained in an attacked house. At the moment the attack begins, the rolling mean will have a much higher value than the rest because the model has not yet learned the behavior of the attack.

5.3 Conclusions

As a conclusion of this section, the performance between the autoencoder and the ARIMA model is compared.

On the one hand, the autoencoder allows us to know when an attack starts, as well as its duration. Each point that forms the time series of the power consumed by the houses can be classified between normal and anomalous behavior.

On the other hand, the ARIMA model has been implemented as an alert generator. It allows us to know when there is a change in the trend of the data (an attack) but does not allow us to classify all the points of the time series.

Therefore the autoencoder provides better performance because it allows us to classify every power consumption value of the houses between normal or anomalous behaviour. In addition, the autoencoder only has to be trained once, while the ARIMA model is trained every time it is going to predict a new value. As a result, the ARIMA model is much slower and consumes more computational resources.

CHAPTER 6

Conclusions

In this chapter, the conclusions, the achieved goals, the problems encountered during the development of this project and the future work are described.

6.1 Conclusions

This master thesis presents the development of a system that allows detecting attacks in smart grids automatically. For this purpose, deep learning techniques have been implemented using Python libraries such as Keras and TensorFlow.

The developed system consists of four different modules: Multi-Agent System, Data Preprocessing Module, Autoencoder, and Anomaly Detection Module. The Multi-Agent System is responsible for generating the synthetic data of smart grids. In addition, it will generate different datasets: ones with a normal behaviour of the grid and others where an attack has taken place. Once the data is generated, they will be introduced in the Data Preprocessing Module to carry out the necessary transformations in order to be able to feed the autoencoder with them. In this module the data will be cleaned, features will be generated and finally, they will be normalized. The Autoencoder, with the features generated in the previous module, creates a model that is in charge of reducing the dimensions of the input data and then reconstruct them with a certain error. Lastly, this reconstruction error will be used in the Anomaly Detection Module to classify the data between normal or attacked.

The attack chosen for its study is a Topology Attack, specifically a False Data Injection attack. This attack consists of modifying, through the smart meters, the value of the power consumed by the user. Normally, this attack is carried out to economically defraud the electric company.

The performance provided by the developed system has been measured in different ways:

- On the one hand, by calculating the reconstruction error for the entire month of data available, it can be known if an attack has occurred, but not the time at which it has started. This detection is perfect regardless of whether the attack sets the power consumption values at 0, 10, 20 or 30% of actual consumption.
- On the other hand, in order to know if each value of power consumption is an attack or not, the reconstruction error of the autoencoder is calculated in another way. Instead of calculating it for the whole month of data, it is calculated for each dataframe entry generated in the Data Preprocessing Module. In addition, after calculating the reconstruction error, it is necessary to make certain transformations to be able to classify with greater precision between normal or attacked data. The results obtained have a higher F-score value for attacks of 0 and 10%, but in all cases, it is possible

to detect the attack without problems. In this way, it is possible to know at what moment the attack starts, as well as its duration.

In addition, an ARIMA model, which is a mathematical model, has also been implemented to detect attacks. This model would replace the autoencoder and the anomaly detection module. In this way, the following power consumption values of the time series are predicted and compared with the real ones. This comparison allows us to detect if an attack has taken place or not. With this model, an alert generator has been implemented, since it is possible to detect in which moment there is a change in the trend of the data.

If the autoencoder and the ARIMA model are compared, it can be seen that they have different functionalities: the autoencoder allows us to know when an attack begins, as well as its duration, while the ARIMA model is a generator of alerts, which allows us to know the moment in which there is a change in the trend of the data (an attack) but does not allow us to classify all the power consumption values between normal or anomalous behaviour.

In conclusion, in this project, a system that automatically detects smart grids attacks using deep learning techniques has been developed.

6.2 Achieved Goals

In this section, the goals achieved during the development of the project are described.

The main objective of this project was to develop a system that automatically detects different attacks that smart grids may suffer using deep learning techniques. In order to do so, the following goals have been achieved:

- **Study the current state of the art of different areas.** At the beginning of the project, it was necessary to carry out a state of the art study mainly of smart grids and deep learning techniques. Regarding smart grids, it has been studied how they work, the advantages they provide, the vulnerabilities they have, as well as the different attacks they may suffer. Regarding deep learning techniques, those that allow data to be analyzed in order to detect anomalies have been studied.

- **Simulation of a smart grid using Agent Based Modelling Simulation.** In order to obtain data related to a smart grid, ABMS was used, due to the impossibility of obtaining real data from them. This is because power consumption data is

confidential. In addition, the tool used to simulate smart grids is Mosaik, which is a smart grid simulator compositor.

- **Preprocessing of the data obtained from smart grids.** The data obtained from Mosaik must be preprocessed in order to give them the necessary format to apply deep learning algorithms on them.

- **Implementation of an Autoencoder.** In order to apply deep learning algorithms to the smart grids data, an autoencoder has been implemented. With the data obtained from the autoencoder, different attacks can be identified.

- **Processing of the data obtained from the autoencoder to detect attacks.** With the data obtained from the autoencoder the reconstruction error is calculated, and different transformations are applied to it in order to detect False Data Injection Attacks.

- **Implementation of an ARIMA model.** An ARIMA model, which is a mathematical model, has been implemented to develop an alert generator.

6.3 Problems Faced

During the development of this project, some problems have had to be faced. These problems are the following ones:

- *Simulation of a smart grid.* In order to simulate a smart grid, an attempt was made to use the Maverig tool. This tool provides to Mosaik a visual and interactive interface, which makes it easier to use. Several problems were encountered when generating a grid and due to this, the software developers were contacted. The developers themselves discourage its use because it is no longer maintained. Therefore, the solution to this problem was to use Mosaik, even though it is a more complex software.

- *Implementation of the attack simulator in Mosaik.* In order to simulate an attack, it was necessary to develop a new simulator for Mosaik. This simulator communicates with the HouseHoldSim simulator in order to modify the power consumption values of the houses. For this purpose, the *set_data* function provided by the Mosaik API was used, with the appropriate data format, but the power values did not suffer any change. After investigating this problem, we realised that HouseHoldSim was prepared to send data to another simulator but not to receive it, as this was not necessary. Therefore,

the solution finally applied was to modify this simulator so that it could receive data and simulate attacks.

- *Autoencoder reconstruction error.* In order to detect when an attack starts and its duration some problems arose. Mainly in the attack that sets the power consumption values to 30% of the originals. The problem was that when there was an attack, not all values were detected as an attack. This was due to the nature of the time series containing the power consumption values. Therefore, in order to solve this problem, several transformations had to be made to the reconstruction error in order to improve the precision of the detection of the attacks.
- *ARIMA model implementation.* During the implementation of the ARIMA model, an attempt was made to predict an entire day and then compare it with actual consumption in order to infer whether an attack had occurred. The problem was that when predicting an entire day, the values converged to the mean of the time series, so the prediction method had to be changed. Therefore, the next value in the time series was predicted and then the real value was introduced into the training set. Thus, the model was retrained and the next value was predicted.

6.4 Future Work

In this section, the implementation of possible new features or improvements of the project are presented.

- *Implementation of new attacks in Mosaik.* A possible improvement of the project is the implementation of new attacks in the Mosaik tool. This is because now only False Data Injection Attacks can be simulated. In addition, the simulation of new attacks would allow us to evaluate the performance of the system when detecting them.
- *Implementation of new algorithms to detect attacks.* By implementing another way of classifying the data, we will be able to compare the performance of the system. In addition, with the other algorithms, an ensemble could be made and, in this way, the results could be improved.
- *Integration of the developed system with a real system.* In this way, the performance of the system using real data could be evaluated. The performed evaluation of the system developed in this project was based on synthetic data obtained from a simulation carried out with the Mosaik tool.

APPENDIX A

Impact of the project

In this appendix, the possible impact that this project has on different areas is presented. First, Sect. A.1 explains the social impact of the project. Then Sect. A.2 describes the economic impact and Sect. A.3 the environmental impact. Finally, Sect. A.4 explains the possible ethical implications of the project.

A.1 Social Impact

The aim of this project is to develop a tool that will automatically detect possible attacks that smart grids may suffer.

The detection of these attacks in a power grid will improve the quality of the supply, as it will avoid various problems that can lead to power outages. Therefore, end users will benefit because the quality of the supply will be improved.

On the other hand, electric companies could reduce the price of electricity (as explained in Sect. A.2), which will also benefit consumers.

A.2 Economic Impact

Electricity fraud causes a great loss of money for electric companies. Only in Spain, it is estimated that approximately one hundred and fifty million euros per year are lost due to fraud.

The tool developed in this project allows the automatic detection of attacks suffered by electrical grids. Therefore, by knowing when and where an attack has been originated, the economic loss caused by fraud to electricity companies will be reduced.

In addition, fraud detection is not only a benefit for electricity companies, but also for the state. This is because all the energy consumed fraudulently does not pay taxes, which is also a great loss of money.

In addition, as the electric companies will have less economic losses related to fraud, they could lower the final price to the consumer. This would be a benefit for consumers as they would save money on the electricity bill.

A.3 Environmental Impact

The detection of attacks on electrical grids is the objective of the system developed in this project. In addition to the economic loss caused by the attacks, as explained above, these worsen the estimation of the energy that will be consumed.

For this reason, the detection of the attacks will improve the estimation of the energy

that is going to be consumed due to the fact that there is information about all the energy that is consumed. A better estimation of energy has several benefits for the environment.

The energy estimation is used to know which power generation plants need to be in operation. Therefore, a better estimation will ensure that only those plants that are really needed are put into operation, which is a benefit to the environment as it will reduce the emission of gases from them. In addition, a better estimation of energy encourages the use of renewable energy. Furthermore, the system will improve the efficiency of the electricity grid, reducing losses in the transport of energy.

Consequently, the system developed in this project will have a positive environmental impact.

A.4 Ethical Implications

The power consumption data used by the system developed in this project does not involve any ethical implications related to data collecting, because these data are already used by the electric company that would use this system. In addition, the purpose of the treatment of these data is to reduce electrical fraud.

The most important ethical implication of this project is to know who is defrauding the electricity companies. Electricity fraud is a global problem and this project aims to reduce its impact.

APPENDIX B

Economic Budget

In this appendix, the adequate economic budget for the implementation of this project is detailed. First, in Sect. B.1 the costs of the different material resources that are necessary are presented. Then, in Sect. B.2 is done an estimation of personnel costs for the development of the system and in Sect.B.3 the different licenses that are necessary are presented. Finally, Sect. B.4 details the costs related to the payment of taxes.

B.1 Material Resources

The material resources needed for the design and development of this project is mainly a computer. This computer must be powerful enough to run and train deep learning models. Therefore, the main features of the computer used are:

- **CPU**: Intel Core i7, 3,2 GHz.
- **Hard Drive**: 30 GB of free space.
- **RAM**: 8 to 16 GB.

Although it makes the price of the computer more expensive, the use of a GPU would accelerate the training of the deep learning model, as this is what consumes the most resources. Therefore, an estimation of the cost of a computer with these characteristics today is 1000 euros.

B.2 Human Resources

In this section, the cost that this project would have in human resources is estimated. So, to make this estimation, the time spent on its development is considered.

In order to estimate the number of hours worked in the development of this project, the number of ECTS credits assigned to it has been considered. In addition, 1 ECTS is approximately between 25 and 30 hours of work. Therefore, an estimation of the total number of hours spent in the realization of this project is 900 hours.

The salary of an engineer working as a scholarship holder at the university is 500 euros per month. Being the scholarship of 4 hours a day and 22 days of work per month, the estimated time for the completion of the project is 10 months.

Therefore, the cost in human resources of the project is 5000 euros.

B.3 Licenses

This section presents the costs associated with software licenses used for the development of the project.

The tools and software used are all open source. Therefore, the licenses do not represent an additional cost to this project.

B.4 Taxes

The system developed in this project could be sold to another company. In that case, taxes related to that sale would have to be paid.

These taxes correspond to 15% of the final value of the sale of the product, according to the Statue 4/2008 of the Spanish law. If the sale was to a foreign company, this tax would be different.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Fadi Aloul, AR Al-Ali, Rami Al-Dalky, Mamoun Al-Mardini, and Wassim El-Hajj. Smart grid security: Threats, vulnerabilities and solutions. *International Journal of Smart Grid and Clean Energy*, 1(1):1–6, 2012.

[3] Daniel B Araya, Katarina Grolinger, Hany F ElYamany, Miriam AM Capretz, and G Bitsuamlak. Collective contextual anomaly detection framework for smart buildings. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 511–518. IEEE, 2016.

[4] Daniel B Araya, Katarina Grolinger, Hany F ElYamany, Miriam AM Capretz, and Girma Bitsuamlak. An ensemble learning framework for anomaly detection in building energy consumption. *Energy and Buildings*, 144:191–206, 2017.

[5] Niyazi Ari and Makhamadsulton Ustazhanov. Matplotlib in python. In *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1–6. IEEE, 2014.

[6] PN Pereira Barbeiro, J Krstulovic, H Teixeira, J Pereira, Filipe Joel Soares, and José Pedro Iria. State estimation in distribution smart grids using autoencoders. In *Power Engineering and Optimization Conference (PEOCO), 2014 IEEE 8th International*, pages 358–363. IEEE, 2014.

[7] Imad A Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[8] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. Anomaly detection using autoencoders in high performance computing systems. *arXiv preprint arXiv:1811.05269*, 2018.

[9] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[10] John A Bullinaria. Recurrent neural networks. *Neural Computation: Lecture*, 12, 2013.

[11] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[12] Sam Clements and Harold Kirkham. Cyber-security considerations for the smart grid. In *Power and Energy Society General Meeting, 2010 IEEE*, pages 1–5. IEEE, 2010.

[13] Andrew Collette. *Python and HDF5: Unlocking Scientific Data.* " O'Reilly Media, Inc.", 2013.

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[15] Peter Eder-Neuhauser, Tanja Zseby, Joachim Fabini, and Gernot Vormayr. Cyber attack models for smart grid environments. *Sustainable Energy, Grids and Networks*, 12:10–29, 2017.

[16] Alexandros Eleftheriou. Smart grid simulation platforms with cyber-attack capabilities. 2018.

[17] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart grid—the new and improved power grid: A survey. *IEEE communications surveys & tutorials*, 14(4):944–980, 2012.

[18] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*, pages 14–36. Springer, 2012.

[19] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47. ACM, 2011.

[20] Peter Goldsborough. A tour of tensorflow. *arXiv preprint arXiv:1610.01178*, 2016.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[22] Antonio Gulli and Sujit Pal. *Deep Learning with Keras.* Packt Publishing Ltd, 2017.

[23] Vehbi C Gungor, Dilan Sahin, Taskin Kocak, Salih Ergut, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. Smart grid technologies: Communication technologies and standards. *IEEE transactions on Industrial informatics*, 7(4):529–539, 2011.

[24] Ping Han, Peng Xin Wang, Shu Yu Zhang, and De Hai Zhu. Drought forecasting based on the remote sensing data using arima models. *Mathematical and computer modelling*, 51(11-12):1398–1403, 2010.

[25] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[26] Ryan Hledik. How green is the smart grid? *The Electricity Journal*, 22(3):29–41, 2009.

[27] Keras. `https://keras.io`, 2018. Accessed: 2019-02-18.

[28] Mehdi Khashei and Mehdi Bijari. An artificial neural network (p, d, q) model for timeseries forecasting. *Expert Systems with applications*, 37(1):479–489, 2010.

[29] Jinsub Kim and Lang Tong. On topology attack of a smart grid: Undetectable attacks and countermeasures. *IEEE Journal on Selected Areas in Communications*, 31(7):1294–1305, 2013.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Oliver Kosut, Liyan Jia, Robert J Thomas, and Lang Tong. Malicious data attacks on the smart grid. *IEEE Transactions on Smart Grid*, 2(4):645–658, 2011.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[33] R Lincoln. Github repository for rwl/pypower. *GitHub, November*, 2012.

[34] Charles M Macal and Michael J North. Agent-based modeling and simulation. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 86–98. IEEE, 2009.

[35] David Masad and Jacqueline Kazil. Mesa: an agent-based modeling framework. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*, pages 53–60, 2015.

[36] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[37] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press USA, 2015.

[38] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[39] Ping-Feng Pai and Chih-Sheng Lin. A hybrid arima and support vector machines model in stock price forecasting. *Omega*, 33(6):497–505, 2005.

[40] Yao Pan, Fangzhou Sun, Jules White, Douglas C Schmidt, Jacob Staples, and Lee Krause. Detecting web attacks with end-to-end deep learning. 2018.

[41] Ivan LG Pearson. Smart grid cyber security for europe. *Energy Policy*, 39(9):5211–5218, 2011.

[42] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[43] Martin Renström and Timothy Holmsten. Fraud detection on unlabeled data with unsupervised machine learning, 2018.

[44] Philipp Ringler, Dogan Keles, and Wolf Fichtner. Agent-based modelling and simulation of smart electricity grids and markets–a literature review. *Renewable and Sustainable Energy Reviews*, 57:205–215, 2016.

[45] Thilo Sauter and Maksim Lobashov. End-to-end communication architecture for smart grids. *IEEE Transactions on Industrial Electronics*, 58(4):1218–1228, 2011.

[46] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[47] Steffen Schütte, Stefan Scherfke, and Martin Tröschel. Mosaik: A framework for modular simulation of active components in smart grids. In *Smart Grid Modeling and Simulation (SGMS), 2011 IEEE First International Workshop on*, pages 55–60. IEEE, 2011.

[48] Tobias Schwerdtfeger and Grid Simulationen. Erika root, gerrit klasen, hanno günther, jerome tammen, marina sartison, marius brinkmann, michael falk, rafael burschik, rouven pajewski sascha spengler, andrianarisoa a. johary ny aina und. 2015.

[49] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[50] R Shyam, Bharathi Ganesh HB, Sachin Kumar, Prabaharan Poornachandran, and KP Soman. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.

[51] Pierluigi Siano. Demand response and smart grids—a survey. *Renewable and sustainable energy reviews*, 30:461–478, 2014.

[52] Carina Silberer and Mirella Lapata. Learning grounded meaning representations with autoencoders. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 721–732, 2014.

[53] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[54] Song Tan, Wen-Zhan Song, Qifen Dong, and Lang Tong. Score: Smart-grid common open research emulator. In *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*, pages 282–287. IEEE, 2012.

[55] William G Temple, Binbin Chen, and Nils Ole Tippenhauer. Delay makes a difference: Smart grid resilience under remote meter disconnect attack. In *SmartGridComm*, pages 462–467, 2013.

[56] TensorFlow. `https://www.tensorflow.org`, 2018. Accessed: 2018-09-16.

[57] Alexandre Gustavo Wermann, Marcelo Cardoso Bortolozzo, Eduardo Germano da Silva, Alberto Schaeffer-Filho, Luciano Paschoal Gaspary, and Marinho Barcellos. Astoria: A framework for attack simulation and evaluation in smart grids. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 273–280. IEEE, 2016.

[58] Xinghuo Yu, Carlo Cecati, Tharam Dillon, and M Godoy Simoes. The new frontier of smart grids. *IEEE Industrial Electronics Magazine*, 5(3):49–63, 2011.

[59] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674. ACM, 2017.