FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Deep Learning Applied to PMU Data in Power Systems

Pedro Emanuel Almeida Cardoso

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Dr. Vladimiro Henrique Barrosa Pinto de Miranda

Second Supervisor: Dr. Ricardo Jorge Gomes de Sousa Bento Bessa

July 2017

Abstract

The analysis of power system disturbances is fundamental to ensure the reliability and security of the supply. In fact, capturing the sequence of system states over a disturbance is an increased value to understand its origin. Phasor Measurement Units (PMUs) have the ability to record these fast transients with high precision, by providing synchronized measurements at high sampling rates. Indeed, these events can occur in a few seconds, which hampers their detection by the traditional SCADA (Supervisory Control and Data Acquisition) systems and emphasizes the uniqueness of PMUs. With the advent of Wide Area Measurement Systems (WAMS) and the consequent deployment of such monitoring devices, control centers are being flooded with massive volumes of data. Therefore, transforming data into knowledge, preferably automatically, is an actual challenge for system operators.

Under abnormal operating conditions, the data collected from several PMUs scattered across the grid can shape a sort of a "movie" of the disturbance. The importance of WAMS is therefore sustained on their ability to capture the sequence of events resulting from a disturbance, helping the further analysis procedures. Driven by the amounts of data involved, this dissertation proposes the application of Deep Learning frameworks to perform automatic disturbance classification. In order to do so, a set of measurements from several PMUs - installed in the Low Voltage grid of an interconnected system - is used, from which representative patterns are extracted so as to endow a classifier of knowledge related to system disturbances.

In particular, the strategies herein adopted consist of the application of Multilayer Perceptrons, Deep Belief Networks and Convolutional Neural Networks, the latter having outperformed the others in terms of classification accuracy. Additionally, these architectures were implemented in both the CPU and the GPU to ascertain the resulting gains in speed.

Index terms: Artificial Neural Network, Convolutional Neural Network, Deep Belief Neural Network, Deep Learning, Power System Disturbance Classification, PMU, WAMS.

Resumo

A análise de perturbações severas num sistema elétrico de energia é fundamental para assegurar a fiabilidade e segurança do fornecimento de energia eléctrica. Efetivamente, capturar a sequência de impactos decorrentes de uma perturbação constitui um passo importante para a determinação da sua origem. As PMUs (Phasor Measurement Units) têm a capacidade de registar estas variações rápidas com elevada precisão, providenciando medições sincronizadas obtidas com elevada taxa de amostragem. Este tipo de eventos pode ocorrer em poucos segundos, o que dificulta a sua deteção através dos tradicionais sistemas SCADA (Supervisory Control and Data Acquisition), evidenciando as características únicas deste aparelho de medição. Com o advento dos sistemas de medição sincronizada - referidos na literatura anglo-saxónica como WAMS (Wide Area Measurement System) - e a consequente proliferação das PMUs, os centros de controlo veem-se inundados por uma quantidade infindável de dados. Por conseguinte, a transformação desses dados em conhecimento, preferencialmente de forma automática, representa um desafio atual para os operadores do sistema.

Numa situação de funcionamento anormal do sistema, os dados fornecidos por diversas PMUs espalhadas pela rede configuram uma espécie de filme que ilustra a perturbação ocorrida. A importância dos WAMS é, então, aferida na capacidade que estes têm para capturar a sequência de eventos resultantes da referida perturbação, sendo uma mais-valia para análises posteriores. Decorrente da quantidade de dados envolvida, a presente dissertação propõe a aplicação de arquiteturas de Deep Learning para classificação automática de perturbações. Para tal, são utilizadas medições registadas por PMUs instaladas na Baixa Tensão, das quais são extraídos padrões representativos de forma a dotar um classificador de conhecimento relativo a perturbações do sistema.

Mais especificamente, as estratégias adotadas consistem na aplicação the Multilayer Perceptrons, Deep Belief Networks e Convolutional Neural Networks, tendo estas últimas superado as restantes em termos de precisão na classificação. Adicionalmente, as arquiteturas mencionadas foram implementadas não só no CPU mas também com recurso a uma GPU, de forma a avaliar as acelerações daí decorrentes.

Acknowledgments

I would first like to thank my thesis supervisor, Prof. Dr. Vladimiro Miranda, for the opportunity, constant motivation and cutting-edge ideas to steer this research in the right direction. A word must also be addressed to Dr. Ricardo Bessa, as co-supervisor, for his valuable contribution to the work developed.

I am also very thankful to all my friends who took part in this journey. It would not have been this good without you all.

A special word goes to Mariana, for being by my side everyday, providing me with unfailing support and continuous encouragement.

Finally, I must express my very profound gratitude to my parents, for their loving support throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Pedro Emanuel Almeida Cardoso

"Without data, you're just another person with an opinion."

W. Edwards Deming

Contents

List of Figures

List of Tables

Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
CI	Communication Infrastructure
CNN	Convolutional Neural Network
CPU	Central Processing Unit
dAE	Denoising Autoencoder
DBN	Deep Belief Network
DeSTIN	Deep Spatio-Temporal Inference Network
DNN	Deep Neural Network
FFNN	Feed-Forward Neural Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
GT	Generation Tripping
LS	Load Shedding
LT	Line Tripping
MLP i	Multilayer Perceptron with i hidden layers
NLL	Negative Log-Likelihood
OS	Oscillation
PDC	Phasor Data Concentrator
PLC	Power-line Communication
PMU	Phasor Measurement Unit
RBM	Restricted Boltzmann Machine
RMS	Root Mean Square
ROCOF	Rate Of Change Of Frequency
SdA	Stacked Denoising Autoencoder
SGD	Stochastic Gradient Descent
SCADA	Supervisory Control and Data Acquisition
SPDC	Super Phasor Data Concentrator
UTC	Coordinated Universal Time
WAMS	Wide Area Measurement System

Chapter 1

Introduction

This chapter presents a brief overview on the main topics addressed to in this work. It introduces the benefits resulting from the integration of Phasor Measurement Units (PMUs) in power system operation and the corresponding usefulness of Deep Learning as a computational tool for harnessing the resulting surplus of data.

The purpose and main contribution of the developed work are then described.

1.1 Motivation

Electrical Power Systems are dynamic infrastructures facing significant changes nowadays. The growth of decentralized energy sources penetration in the system is boosting the modifications required for improvements in both monitoring and control of power systems. The operation paradigm has changed, evolving from a load-driven to a generation-driven mode. That is, generation is now leading the operation paradigm of power systems, since the increase of renewable energy sources leads to generation profiles that cannot be absolutely controlled. Therefore, a need arises for new solutions and technologies to improve the operation of the power system, with its growing complexity requiring more advanced and sophisticated monitoring capabilities.

Currently, transmission networks are endowed with more advanced solutions than distribution systems. In addition, the large presence of distributed energy resources connected at distribution levels is enhancing the role of this infrastructure in power system operation and emphasizing the lack of active distribution grids. The improvement of the monitoring resources for both transmission and distribution networks is conceived as the employment of new measurement technologies and the development of new algorithms for data management. Phasor Measurement Units (PMUs) represent a significant upgrade in the measurement devices being used to improve the monitoring of power systems. Its uniqueness relies on providing measurements of both Root Mean Square (RMS) and phase of an electrical signal, with high sampling rates and synchronized with very accurate time-stamps provided by the Global Positioning System (GPS).

More specifically, PMUs provide 10-60 samples per second (10-60 Hz), which is very high when compared to the traditional SCADA acquisition devices that sample every 2-4 seconds (0.25-0.5 Hz). This significant difference is illustrated in Figure 1.1, where the frequency variation resulting from an actual generation tripping, registered by both technologies, is shown.

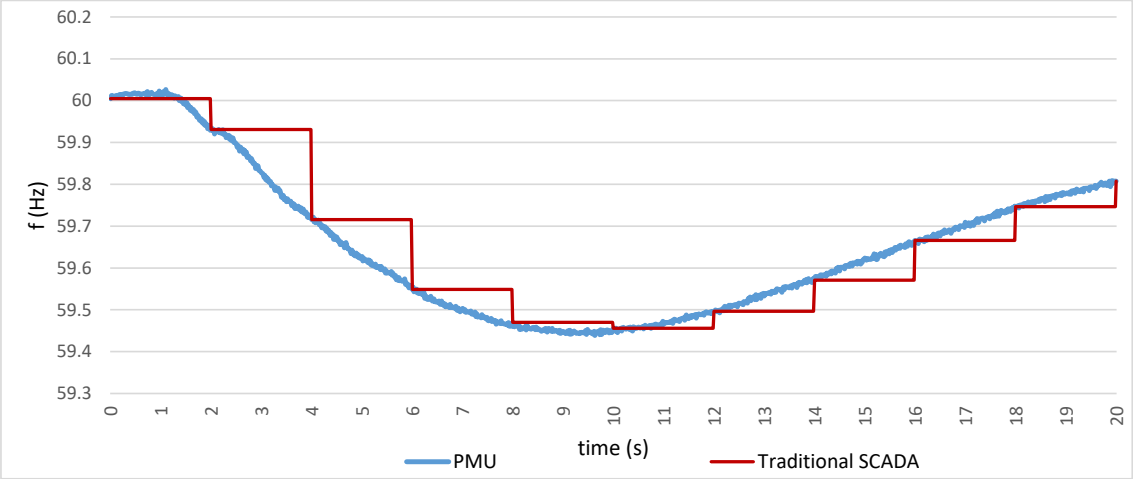

Figure 1.1: Frequency variation resulting from a generator tripping, registered with a PMU sampling at 60 Hz (blue) and a traditional SCADA acquisition device sampling at 0.5 Hz (red)

It is evident that the adoption of PMUs can have significant impacts in the operation of the system. To better understand the resulting changes, a comparison between both paradigms regarding their main attributes is presented in Table 1.1.

Table 1.1: Main attributes of traditional SCADA systems and PMU-based WAMS

Attribute	Traditional SCADA System	PMU-based WAMS
Resolution	1 sample every 2-4 seconds	10-60 samples per second
Observability	Steady State	Dynamic / Transient
Measured Quantities	Magnitude	Magnitude + Phase
Time Synchronization	No	Yes
Monitoring	Local	Global

By replicating several PMUs across the system and creating a Wide Area Measurement System (WAMS), the directly measured phasor data can help to infer the state of the power system, at a given instant, enhancing several control functionalities that the system operator has to deal with. So, PMUs are able to play a critical role in real-time operation, namely in stability surveillance throughout the electrical grid. The high precision knowledge PMUs can provide is required to minimize and control power outages and avoid problems such as cascading blackouts. This new paradigm results in a high resolution situational awareness, described perfectly by Terry Boston - CEO of PJM Interconnection, a USA region transmission organization - "It's like going from an X-ray to an MRI of the grid.".

Indeed, the time-synchronization of geographically dispersed measurements provides a better real-time operational awareness. This becomes particularly relevant in cases of abnormal operating conditions, in which the collection of measurements with high sampling rates captures the dynamic behaviour of the system, providing a deeper knowledge of the transients involved. Having the measurements duly synchronized, the sequence of grid states over an occurrence represents a kind of a "film" of the event, that can be reproduced and used for improving examination methods such as *post-mortem* analysis.

As PMUs are deployed in larger numbers, the transmission and distribution operators will need practical tools to efficiently deal with the arising amounts of data. Consequently, the importance of developing innovative methods for harnessing the data collected from PMUs becomes evident. Indeed, the workload imposed by the stream of data generated by those devices suggests the application of Artificial Intelligence (AI) methods. In particular, the recent interest in Deep Learning is partly due to the availability of large amounts of data and also the improvement of computational power. Driven by these assumptions, this dissertation suggests the application of Deep Learning frameworks to extract knowledge from patterns present in raw data generated by a real, PMU-based, measurement system.

1.2 Main Contribution

There is a willingness to bring closer the concepts of computer science and power systems, with the objective of using breakthrough strategies from the field of AI to leverage the way everyday problems of an electrical grid are solved. Since it is a very recent research subject, this dissertation represents a novelty with an added value for the state of the art of power systems.

The methodologies proposed intend to demonstrate how efficiently Deep Learning frameworks can perform on power system disturbance analysis, applied to a set of PMU measurements. The incursion herein conducted aims at developing disturbance classifiers based on the frequency change during a given occurrence - for instance, the one previously illustrated in Figure 1.1. The distinctiveness of such an approach is to model the power system dynamics as a movie from which snapshots are taken and duly evaluated with the help of new computational tools.

In general, the main contributions are:

- Application of Deep Learning frameworks for extracting patterns from PMU frequency measurements: development of disturbance classifiers for *post-mortem* analysis;
- Performance evaluation of three classifiers: Multilayer Perceptron, Deep Belief Network and Convolutional Neural Network;
- Demonstration of the usefulness of Convolutional Neural Networks as a classifier on non-image data;
- Assessment of gains in the processing time of the methodologies proposed by harnessing the properties of GPU computing.

Chapter 2

Phasor Measurement Units

This is the first of two chapters reviewing the literature on the concepts related to this thesis. Here, the main features regarding Phasor Measurement Units are addressed. The chapter starts by presenting the basic concepts and moves towards the explanation on how PMU-based measurement systems are designed. Then, the most important subjects harnessing the data collected by the PMUs are described. The chapter ends with the incentive for the application of Deep Learning to PMU data.

2.1 Brief Overview

A **Phasor Measurement Unit** (**PMU**), which can be a dedicated device or as a function integrated in other devices such as protective relays, measures electrical waveforms in a power system, specifically voltage and current signals. Moreover, PMUs provide frequency and rate of change of frequency (ROCOF) of local measurements. With the use of a common time source provided by a Global Positioning System (GPS) clock, voltage and current measurements are time-stamped with high precision, enabling synchronization. This synchronized phasor measurements, also called **synchrophasors**, are becoming an important element of Wide Area Measurement Systems (WAMS). Synchrophasors are enabling more advanced real-time monitoring, protection and control applications, therefore improving the electric power system operation.

2.2 Historical Background

The development of PMUs can be traced back to the field of computer relaying of transmission lines. The first works dedicated to transmission line microprocessor-based relaying (around the 1970s) were hindered by the insufficient computational capabilities to perform the calculations of all relaying functions. The research for reducing that computational effort came out with a solution based on symmetrical component analysis of line voltages and currents [9]. The calculation of positive-sequence voltages and currents using the algorithms presented in 1977 [9] represented a great contribution for modern phasor measurement systems. Effectively, the importance and

applications of such an advent were identified in 1983 by [10]. These developments matched the beginning of the GPS project (1978), which offered the possibility of synchronizing measurements. In the early 1980s, the first PMU prototype using GPS was designed at Virginia Tech. However, the commercial manufacture of PMUs only started in 1991, in a collaboration between Virginia Tech and Macrodyne. This initiative and the further growth of PMUs manufacturers led to a series of "IEEE Standards for Synchrophasor Measurements for Power Systems" governing all issues related to phasor measurements.

2.3 Fundamentals of Synchrophasors

Charles Proteus Steinmetz, back in 1893, revolutionized the AC circuit theory and analysis when he introduced a way to describe a sinusoidal signal, using a complex number that referred to its RMS value and phase-angle - a phasor. The classical mathematical definition of a phasor can be depicted considering a sinusoidal waveform with constant frequency and magnitude, x(t):

$$x(t) = X_m \, cos(wt + \phi) \tag{2.1}$$

where X_m is the signal peak value, $w = 2\pi f$ is the angular frequency and ϕ the phase-angle of the signal. The corresponding phasor representation can be determined in both polar and rectangular coordinates, respectively:

$$x(t) = \frac{X_m}{\sqrt{2}} e^{j\phi} = \frac{X_m}{\sqrt{2}} (cos\phi + jsin\phi) = X_r + jX_{im} \tag{2.2}$$

where the module of the phasor is the RMS value $\frac{X_m}{\sqrt{2}}$ of the sinusoid waveform and X_r and X_{im} are the real and imaginary rectangular components of the complex phasor. Positive phase angles are measured in a counterclockwise direction from the real axis. However, phase is actually dependent of the initial time instant used as a reference, so it has to be referred correctly to that reference. Both phasor and sinusoidal representations are illustrated in Figure 2.1.

Synchrophasors are as simple as synchronized phasors. More specifically, the phase-angle of a synchrophasor is calculated using the Coordinated Universal Time (UTC), provided by a GPS signal, as time reference. In doing so, a unique time reference is defined for all measured signals in a wide area, at a given instant, thus being synchronized.

2.4 Generic Configuration of a PMU

The purpose of a PMU is to provide phasor measurements of voltages and currents, clearly referenced to a time source, besides the additional frequency and ROCOF measurements. So, despite the existence of different manufacturers, a generic configuration of a PMU can be described and its

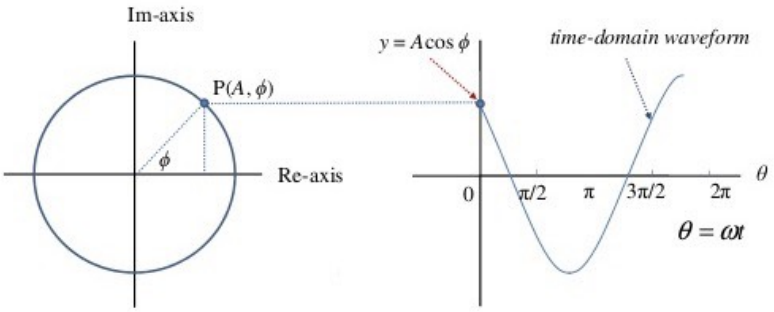

Figure 2.1: Phasor representation of a sinusoidal signal. (a) Phasor representation; (b) Sinusoidal Signal

key elements are presented in Figure 2.2. This structure is based on the first PMU built at Virginia Tech.

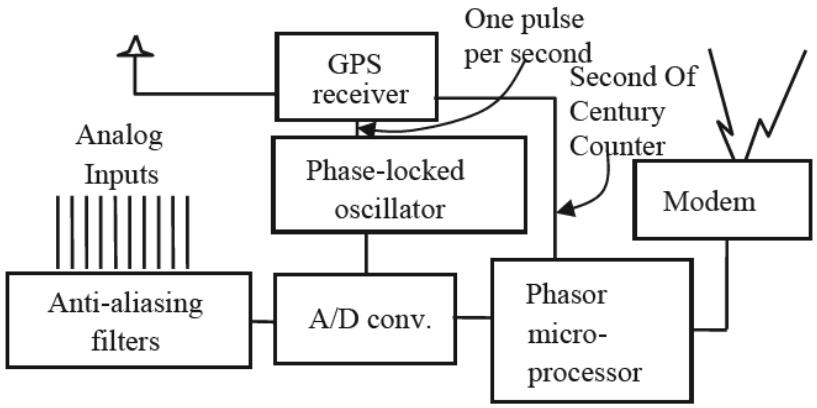

Figure 2.2: Elements of a PMU [1]

So as to compute positive-sequence measurements, all three-phase currents and voltages must be obtained from the respective current and voltage transformers. These can be acquired from several places in a substation and are identified as the analog inputs. To avoid aliasing errors, it is customary to use anti-aliasing filters, the frequency response of which is dictated by the sampling rate chosen for the sampling process. A common choice is to use analog-type filters with a cut-off frequency less than half the sampling frequency, meeting the Nyquist criterion [1]. Nevertheless, using a high sampling rate (thus oversampling) with a high cut-off frequency has some benefits. In fact, a digital decimation filter must be added so as to reduce the sampling rate, resulting in a digital anti-aliasing filter concatenated with the analog ones. In doing so, all analog signals have the same phase shift and attenuation, assuring that the phase-angle differences and relative magnitudes of different signals remain unchanged. Additionally, the oversampling technique can enable digital fault recording if raw data storage is available. It is important to note that current and voltage signals must be converted to appropriate voltages that match the requirements of the

analog-to-digital converters. The GPS receiver main task is to provide the UTC clock time used for time-tagging the sampled measurements. The phase locked oscillator converts the one pulse per second provided by the GPS into a sequence of high-speed timing pulses used in the waveform sampling. The microprocessor determines the positive-sequence of current and voltage signals. In phasor representations, the frequency of the sinusoidal signal is considered constant, implying stationary sinusoidal waveforms. Consequently, in practical applications where frequency does not remain constant, phasor representations must be handled carefully. This is usually surpassed considering the input signal over a finite data window, normally corresponding to one period of its fundamental frequency. Such a procedure can be conducted by several techniques depending on the instantaneous system frequency. Since that frequency is seldom equal to the nominal frequency, the PMU must use a frequency-tracking step so as to estimate the period of the fundamental frequency before the phasor is determined. Additionally, in order to do so, the PMU has to filter the input signal, separating the fundamental frequency from the harmonic or non harmonic components. The most commonly used technique for determining the phasor representation of an input signal relies on the application of the Discrete Fourier Transform (DFT) to a moving window over the sampled data. Considering N samples taken over one period and $x_k\{k = 1,2,...,N-1\}$ as the kth sample, the phasor representation is given by:

$$X = \frac{\sqrt{2}}{N}\sum x_k e^{-jk\frac{2\pi}{N}} \tag{2.3}$$

Finally, the output files of the PMU are transferred over the available communication facilities to a higher level in the measurement system hierarchy.

2.5 Measurement System Hierarchy

The architecture supporting modern measurement systems is commonly referred to as Wide Area Measurement System (WAMS). This term implies a set of new digital metering devices (e.g. PMUs), along with communication infrastructures, designed to acquire, transmit and process data over a wide geographic area.

Regarding PMUs as the quintessential example of metering devices, they are typically installed in power system substations. Despite local applications, the main use of PMUs information is carried out remotely at control centers. Therefore, a proper measurement system architecture is required, featuring, apart from the individual PMUs, dedicated communication channels and data concentrators. An illustrative scheme is shown in Figure 2.3.

The PMUs provide time-stamped measurements of positive-sequence voltages and currents of all monitored buses and feeders in the respective substation, in addition to frequency and ROCOF. These measurements can be stored for further exploration, like *post-mortem* analysis. Since the local data storage is limited, relevant events are to be flagged for permanent storage. Besides storage,

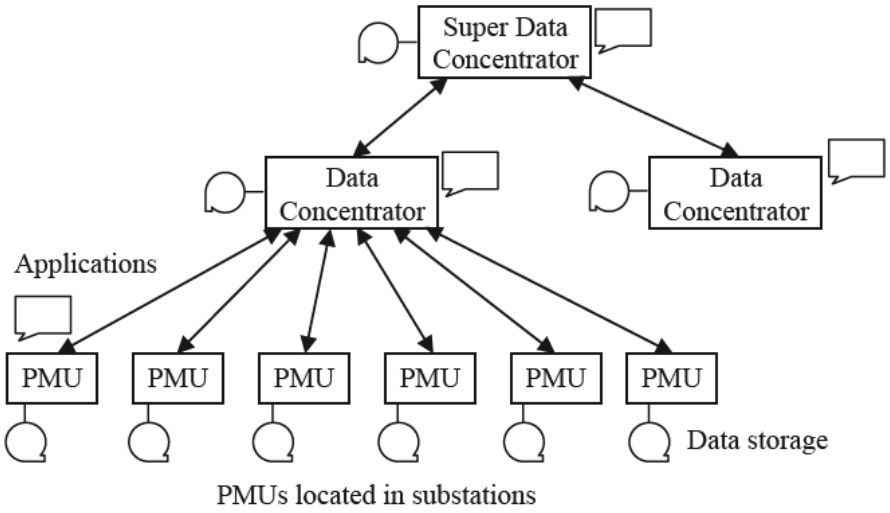

Figure 2.3: Measurement system hierarchy [1]

a stream of information flows upwards being available for other applications. At a higher hierarchical level, data from several remote PMUs is gathered in Phasor Data Concentrators (PDCs). The role of these devices depends on the application. For non-real-time scenarios, PDCs usually reject bad data and store all the information for future analysis. For real-time applications, such as network monitoring, PDCs align measurements according to each time-stamp. Subsequently, sorted measurements can be streamed to upper hierarchical levels such as control centers, where primary PDCs or Super Phasor Data Concentrators (SPDC) are placed. This hierarchical architecture enables information to flow from local to global entities.

The great advantage of such an architecture is that measurements coming from different substations can be correlated in respect to the common time reference. Consequently, the status of a wide network-area can be directly obtained.

2.6 Communication Infrastructures

The deployment of a Wide Area Measurement System regarding Phasor Measurement Units requires robust communication infrastructures. In fact, real-time applications rely on the effectiveness of information flow between all elements of the measurement system. Therefore, communication tasks should provide small latency - time lag between the instant of data creation and its availability for the desired application. In addition, an adequate channel capacity should be guaranteed, which is, in fact, rarely a limiting factor in PMUs applications.

Currently, optical-fiber is the physical medium of choice for communications, due to their unique channel capacity, high data transfer rates and immunity to electromagnetic interference [1], therefore meeting the requirements imposed. However, many existing applications are still using the conventional Power-line Communication (PLC) owing to its simplicity and low cost of implementation. Moreover, wireless infrastructures are emerging, with some applications using

the Internet via Virtual Private Network (VPN) systems. This is especially applied when large distances between the PMUs and the PDCs are involved, as seen in the Brazilian Synchronized Phasor Measurement System (SPMS) [11]. However, many existing applications are still using PLC.

With the advent of smart grids, significant investments are being made in the distribution grid, particularly in what regards the communication infrastructures (wired and/or wireless). This recent paradigm requires an almost absolute monitoring and control of the network, which is inherently dependent on the performance of the communication system. Therefore, it must be guaranteed a high level of robustness, with the infrastructure having some redundancy, high availability and speed, interoperability and security - especially cyber security, nowadays.

2.7 Output Data

The development of the synchrophasor technology and the corresponding need to transmit information over the network led to the establishment of several communication standards by the IEEE working group. Most recently, the IEEE C37.118-2 standard was developed, covering the communication framework and requirements for transmission of synchrophasors. Concretely, without imposing restriction on the communication media itself, the standard defines four file types for data transmission to and from PMUs:

- Header frame;
- Data frame;
- Configuration frame;
- Command frame.

The Header frame is the only human readable file, containing valuable information from the producer to the user of data. The Configuration frame is sent whenever the configuration of the system changes. Therefore, both Header and Configuration frame are transmitted by the PMU when the nature of data to be sent is defined for the first time. The Data frame, sent from the PMU to the PDC, contains the principal output of the PMU, such as the phasor measurements, frequency and ROCOF, voltages, currents, active and reactive powers. Finally, the Command frame is usually provided by higher levels of hierarchy for controlling the performance of the PMU (for instance, sent from a PDC to a PMU). Overall, under normal operation conditions, only the Data frame is communicated.

2.8 Harnessing PMU data

Synchronized phasor measurements are extremely important for monitoring and controlling the dynamics of a power system. Since PMUs were introduced into power systems in 1980s, their

value has been acknowledged by the extensive studied, proposed and implemented applications, which have shown significant benefits.

Nowadays, there are several PMUs installed around the world providing valuable information to improve power system operation. One of the major concerns regarding the deployment of PMU technology is its placement. Due to installation costs - devices, communication infrastructures and labouring - it is not cost-effective to install a PMU at every node of a system. That is why a great amount of research has been conducted to develop optimal placement strategies aiming at minimizing the number of devices. It is important to note that the intended applications of PMU installation are determinant to the strategies defined, with state estimation as one of the most common.

In [12], the authors present three different methods for optimal placement of PMUs: observability factor analysis, sequential orthogonalization algorithm and coherency identification technique combined with observability factor analysis. However, in addition to the cost of PMUs, the cost of the Communication Infrastructure (CI) has to be considered because it can even be dominant in relation to the cost of PMUs. Reference [13] shows that independent approaches for optimal placement of PMUs and optimal design of the CI might not lead to a global optimum solution in terms of cost; considering both simultaneously led to better solutions in terms of cost and state estimation observability. In addition, a genetic algorithm is used for optimizing the defined problems, however exposing the methodology to the vicissitudes inherent of genetic approaches, such as randomness and computation effort. An extension to that work is proposed by [14], where binary imperialistic competition algorithm and Dijkstra algorithms are combined to optimize the cost of WAMS. Here, optimal placement of PMUs and CI design are determined simultaneously, whilst N-1 contingency is considered and observability is ensured.

Finding the optimum design of the measurement infrastructures to be installed is of great importance for power system operators. In fact, the development of modern WAMS is inherently dependent on the strategies found for PMUs placement, which have to be considered in investment plans. As aforementioned, the optimum solutions are dependent on the application to be explored. In general, there are five major areas of interest in which the PMUs have significant impacts [15]:

- Oscillation Detection and Control;
- Load Modeling Validation;
- Voltage Stability Monitoring and Control;
- System Restoration and Event Analysis;
- Improvement on State Estimation;

2.8.1 Oscillation Detection and Control

The deregulation of electric power systems and consequent market-based scheme, along with continuously growing demand, led to increasingly stressed operation in terms of oscillatory stability.

Therefore, the dynamics involved in power transmission became more complex, specially for low frequency oscillations. These are of great importance because they not only limit the amount of power transfer, but also degrade the power system security. The traditional SCADA systems cannot detect these fast dynamics due to their low data sampling rate and lack of synchronization. These restraints are effectively surpassed by the high data reporting rate of synchronized PMUs and the availability of fast communication links. So, the advent of PMU-based WAMS offers great opportunities to monitor the dynamic behaviour of power systems and identify its oscillation modes. For instance, an online monitoring of power system dynamics by using the synchronized measurements is presented in [16]. Additionally, both [17] and [18] suggest other online applications, where Prony-based analysis - method used for the direct estimation of system modal information - is implemented for frequency and damping oscillations detection by harnessing PMU data.

2.8.2 Load Modeling Validation

The analysis of a power system consists of modelling its various components and it has been an object of extensive studies. Thereof, load modelling has always been a challenging area for power system engineers, due to its great uncertainty and impact on system voltage and dynamic stability. In this practice, two approaches are widely employed: the component-based approach and the measurement-based approach [19]. The latter provides direct monitoring of true dynamic load variations, hence easily updating the load model parameters in real time for transient stability studies [20]. The recent growth of synchronized PMUs employment enhanced the efficiency of the measurement-based approach. In fact, the inherent accuracy of PMU measurements is reflected in the precision of load models, whereas the high reporting rates enable real-time development and validation of the model [21]. Moreover, [15] discusses the use of PMUs for measurement-based load modelling in the light of real-time security assessment and dynamic characteristics of end-use load.

2.8.3 Voltage Stability Monitoring and Control

In electrical power systems, voltage stability is inherent to the loadability of transmission networks. Moreover, recent developments regarding distributed energy resources have brought increased uncertainty to power transfer as well as network expansion. Consequently, voltage stability has become a problem of major concern in both planning and operating of power systems. The advent of PMU technology brought the possibility of measurement-based real-time voltage stability monitoring and control to transmission systems, thus improving the management of power transfer and system security [15]. For instance, [22] proposes an algorithm for voltage stability improvement by optimally setting the output of control devices, assuming the availability of several dispersed PMUs. In a different perspective, [23] presents PMU-based Artificial Neural Network approach for faster than conventional real-time voltage stability monitoring. The synchronized

measurements are the inputs of the neural network, which is trained for providing the different voltage stability indices.

2.8.4 System Restoration and Event Analysis

Facing a major disturbance, system operators want to limit the resulting impact for consumers by restoring the system as quickly as possible. Afterwards, a complete event analysis must be driven in order to determine the root cause of the disturbance - commonly referred to as *post-mortem* analysis. However, the process of system restoration and event analysis is technically difficult to implement due to the absence of synchronized data and consequent computation time. Therefore, synchronized PMUs arise as a way to effectively deal with both issues.

As soon as a disturbance occurs, the protection system should detect it and operate appropriately so as to isolate and minimize the affected area. Then, the location and source of the disturbance should be identified and isolated in order to restore power service. Approaches to fault location regarding PMUs reveal advantages in computation burden and technical assumptions reduction [15]. In addition, [24] presents the effectiveness of high-speed synchrophasor data in a multi-terminal calculation of fault location.

As reported in [25], PMUs gave a significant contribution to information acquisition during a system islanding situation. The formation of the island was detected by monitoring the data from PMUs, where rapid diverging oscillations of frequencies were captured. Subsequently, [26] proposed a methodology for islands-synchronization with PMU measurements.

Overall, PMU-based systems can improve restoration time by providing high accuracy measurements that help pinpointing a fault. Also, the presence of synchronized data is greatly leveraged whenever *post-mortem* analysis of events is required. These unique properties of PMUs also prospect a successful application in preventive recognition schemes.

2.8.5 Improvement on State Estimation

State estimation plays an important role in real-time monitoring and control of a power system. The traditional procedure uses measured voltage and current, real and reactive powers to infer the operating condition of the grid, at a given instant. The synchronized measurements provided by PMUs can be easily incorporated into state estimators along with conventional measurements [27]. Since voltage angles (which are state variables to be estimated in conventional state estimators) are measured directly, the accuracy of the state estimation is increased. This accuracy improvement was also obtained for dynamic state estimators [28].

State estimators are intended to provide the most likely state of the network. Therefore, they must be able to adequately detect and correct measurement errors, in a procedure called bad data processing. The detection of bad data in a power system can be improved by the effective placement of PMUs [29].

2.9 WAMS Implementations

Nowadays, Phasor Measurement Units are being used for several large scale WAMS implementations around the world, from which some examples are mentioned.

In USA, The Bonneville Power Administration was the first electrical utility to implement a PMU-based architecture, in the beginnings of 2000. Thereafter, in response to the largest blackout in American history that affected about 50 million people in 2003, the New York Independent System Operator decided to install several PMUs in the system so as to prevent similar catastrophic events [30]. In addition, Mexico developed a WAMS architecture capable of reaching up to eight transmission regions, helping the real-time monitoring and operation and improving system reliability and security [31]. China has also made substantial investments in this technology. Besides producing their own devices and standards, its grids have thousands of PMUs currently installed, mainly in EHV levels [32]. In Brazil, a phasor measurement project called Medfasee was developed. The project started by implementing Medfasee BT, a low voltage synchrophasor system in which PMUs are installed in universities. In 2013, 22 low voltage PMUs were already installed in several universities scattered around the country [6]. This brazilian project will be paid detailed attention in chapter 4, since it provided a totally labelled dataset of disturbances recorded by the synchrophasor measurement system. Driven by the success of Medfasee BT, the Medfasee CTEEP was designed so as to implement a synchrophasor measurement system in a 440kV grid owned by CTEEP (Companhia de Transmissão de Energia Elétrica Paulista). In 2013, the project accounted 13 PMUs monitoring 13 transmission lines [33].

2.10 PMUs as a Big Data issue

With the advent of WAMS and the consequent the proliferation of digital measurement devices, control centers are being flooded with increasingly amounts of data. PMUs are capable of collecting samples at ratings from 10 to 60 (depending on system frequency) samples per second [34], which are much higher sampling rates than the employed in traditional SCADA systems, that typically collect measurements every 2-4 seconds [35]. This new paradigm represents a huge amount of raw data collected everyday. For instance, [36] refers that a single PMU sampling at 60Hz can create roughly 721MB of data per day. In addition, they use a dataset from Bonneville Power Administration's PMU installation containing 44 PMUs that generate approximately 1TB per month. Reference [37] prospects how large-scale PMU systems present challenges for such a volume of data processing. Therefore, power system operators are craving for efficient techniques to digest the incoming data, improving grid operations. The volumes of data involved in the operation of WAMS suggest that this is a Big Data issue, where techniques in the field of Artificial Intelligence, like Machine Learning, can be very helpful to extract features from raw data.

Chapter 3

Deep Learning

This second chapter of the literature review describes the state of the art concerning Deep Learning. After some initial considerations, the chapter addresses the concepts of learning from training and includes detailed description of the most important Deep Learning frameworks.

A section of final remarks regarding both chapters of the literature review can be found at the end.

3.1 Why Deep Learning?

Driven by the massive amounts of data involved in the operation of PMU-based WAMS, innovative methods in the field of Artificial Intelligence, such as Machine Learning, are emerging for harnessing the information provided and extract valuable knowledge for system operators. Instead of declaring complex analytical models, learning to recognize patterns and identifying features seems to be the answer to overcome the challenges imposed by processing the huge volumes of raw data involved in WAMS operation. Thus, as a class of the Machine Learning algorithms, **Deep Learning** arises as a computational learning technique in which high level abstractions are hierarchically modelled from raw data. In doing so, computers are taught to understand the world in terms of a hierarchy of concepts, where a complex concept can be defined in terms of simpler concepts. Why Deep Learning? The core of Deep Learning emergence lies on recent increase in computation power and access to enough data to train the algorithms. Also, in comparison to other learning algorithms, Deep Learning has shown better performance when larger datasets are involved. As illustrated in Figure 3.1, most learning algorithms reach a *plateau* in performance, whereas Deep Learning continues increasing.

The hierarchical learning representations of data in Deep Learning are usually inspired in neuroscience, underlying the interpretation of information processing in neural systems. Hence, the majority of Deep Learning frameworks are based on the working principle of Artificial Neural Networks (ANNs), leading to the appearance of Deep Neural Networks (DNNs). DNNs can be considered as a class of ANNs where multiple hidden layers are placed between the input and output layers. Each layer is composed of single units, referred to as neurons due to the similarities

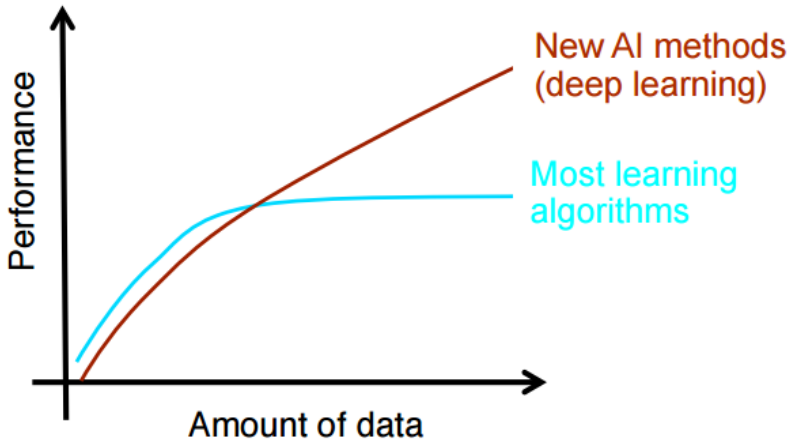

Figure 3.1: Performance of learning algorithms on available data [2]

they share with the behaviour of the neurons present in the human brain. Mathematically, a neuron can be considered as a non-linear function of the weighted sum of its inputs. This multi-layered processing enables the algorithm to learn representations from data with multiple levels of abstraction. By composing the simple non-linear modules, representations at one level (starting from the raw input) are transformed into a representation at a higher, slightly more abstract, level [38].

3.2 Learning from Training

Learning from training is perhaps the most interesting property of ANNs. The ability of learning is generally obtained by iteratively adjusting the connection weights of neural network layers, so as to reproduce features of the input data. The theory of Machine Learning presents a broad categorization of its algorithms as **supervised**, **unsupervised** or even **reinforcement learning**, by the kind of experience they are allowed to have during the learning procedure.

In **supervised learning**, the algorithm experiences a dataset, in which each example consists of input values and the respective label/target. The outputs of the network are compared to the targets and the corresponding difference is determined. The weights are then adjusted so as to minimize such a difference. The backpropagation algorithm, addressed to in the following section, is considered the quintessential example of supervised learning.

In **unsupervised learning**, algorithms are fed with sets of unlabelled examples. The objective is to learn the probability distribution that generated the dataset. The connection weights are therefore adjusted by a self-organizing process that learns useful properties of the input dataset structure.

The algorithm can also be in a dynamic environment, receiving a reward feedback from its current performance, in a context of learning called **reinforcement learning**, which should not be confused with the supervised learning paradigm.

3.2.1 Backpropagation

One of the most common and successful methods for supervised training of artificial neural networks is the backpropagation algorithm. Introduced in 1986 by [39], the process adjusts the weights of the hidden connections of the network in order to meet a given loss function. This adjustment showed that backpropagation training is able to generate useful internal representations of incoming data in the hidden layers of neural networks. A standard choice as the loss function is to minimize the difference between the actual and the desired output (for instance, the Mean Squared Error).

In general, the procedure comprises a two step sequence:

- **Propagation** — input data is presented to the network, propagating forward and producing an output. The output is compared to the desired/target values using a loss function that returns the resulting error;
- **Weight Adjustment** — the resulting error is then propagated backwards, updating the weights of the internal connections so as to minimize the loss function. This minimization is generally done through the application of a gradient descent method, in which the weight parameters are iteratively updated in the direction of the gradient of the loss function, until the minimum is reached.

The latter step is the trickiest. Indeed, the accuracy of the model depends directly on how good the connection weights are adjusted. Time efficiency is also a concern, since this step can take a significant part of the training duration.

3.2.2 Gradient-based Optimization

The inherent nonlinearity of ANNs implies that most loss functions become non-convex. Therefore, these networks are usually trained employing iterative, gradient-based optimization methods, in order to conduct the loss function to the lowest value possible. Regarding a learning paradigm, objective functions are typically additive, formed as a composition of parcels from the training set, all summed up. Consequently, a gradient optimizer is also an additive operation. For this reason, optimizers have the possibility of performing one step (update of model parameters) based on two distinct techniques:

- **Batch** or **Deterministic** Gradient Descent — a single step requires running over the entire training set, hence updating the weights at the end of each epoch (one epoch is considered as a pass over all examples of the training set);
- **Stochastic** or **Online** — a single example is used at a time. The term Online refers to a specific case of real-time Stochastic, employed whenever each example is drawn from continuously created examples. The weights are updated after every training example until the entire training set is treated. At that point, an epoch of training is considered as complete, having performed as much updates as training examples.

In general, machine learning algorithms require large datasets for good generalization. Thus, applying the Batch Gradient Descent to huge training sets causes the time efficiency of the optimization to drop tremendously. On the other hand, the Stochastic Gradient Descent will imply a stochastic approximation of the true gradient of the loss function. In order to overcome those disadvantages, most applications employ a hybrid solution and use more than one example at a time - a mini-batch - instead of all the training set. This is commonly referred to as Mini-batch Stochastic Gradient Descent (SGD). Indeed, the Mini-batch SGD provides a more accurate estimate of the gradient [7] and is computationally more efficient in memory usage of modern computers [3], specially for GPU (Graphics Processing Unit) computing [7].

3.2.3 Problems with Training

The training process acquires additional importance to avoid common issues such as overfitting and excessive computation time. Frameworks like DNNs are prone to overfitting, that is, to overadapt to the training set and lose capacity of generalization. In these cases, regularization methods such as dropout (where neurons are randomly removed during training, preventing the network from becoming overly dependent on any neuron) or early-stopping can be applied during training to help combat overfitting.

The success of backpropagation with stochastic gradient descent relies on its ease of implementation and tendency to converge to better local optima than other methods. However, especially for DNNs, these methods can be computationally expensive due to the many training parameters to be considered such as size of the network (number of layers, number of units per layer), learning rates and the initialization of weights. Covering all the parameter space to find the optimal parameters might not be feasible in terms of time or computation resources. This has led to several adaptations to speed up computation: mini-batching (determine the gradient for several training examples at once) and application of GPU computing.

In the development of the Deep Learning algorithms, both issues were considered and their solutions implemented. Therefore, these concepts are further detailed when appropriate - regularization in Section 5.6.1, mini-batching and its influence on GPU computing in Section 6.5.1.

3.3 Deep Learning Frameworks

Deep learning is a fast-growing field and new applications of its frameworks/architectures emerge often. The process of determining the best architecture is not straightforward, since it is not always possible or even advisable to compare their performance on the same datasets. That is, some architectures were developed for specific purposes and might not perform well on all tasks. Bearing that in mind, the following sections present a description of the most important Deep Learning Frameworks, especially for classification tasks (the aim of this dissertation). It is also important to note that Deep Learning has not come across the field of Power Systems very often, so the description given tends to be generic. Nevertheless, when appropriate, connections to Power Systems are traced.

3.3.1 Multilayer Perceptron

Multilayer Perceptron (MLP) is the name hereby given to a deep multilayered feedforward artificial neural network, which is perhaps the most basic example of a Deep Learning Framework. The concepts related to structure and working principle of DNNs were continuously introduced as Deep Learning ideas were being introduced. So, to avoid redundancy, only the main features will be highlighted. In addition, an illustration of a typical MLP is given in Figure 3.2.

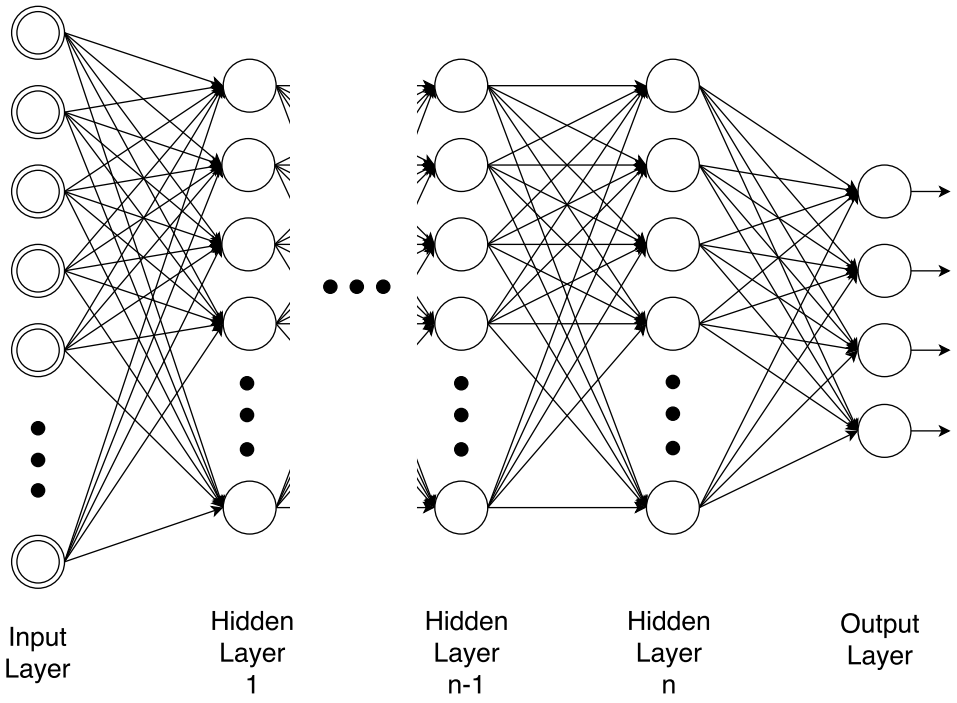

Figure 3.2: An example of the typical architecture of an MLP: input layer, a series of n hidden layers and an output layer

Defining the structure of an MLP for a specific application can assume a high level of complexity. The number of neurons in the input layer is completely and uniquely determined once the shape of the input data is acknowledged. The output layer is specified by the aim of the solution developed. For instance, classification and regression require different but well defined output configurations. However, defining the number of hidden layers and respective number of neurons is not straightforward. Regarding the number of hidden layers, literature in general suggests that ore or two layers is usually enough and going deeper not always implies accuracy improvements. On the other hand, some rules-of-thumb for the number of neurons have been developed but are too generic for a universal acceptance - for instance restricting the number of hidden units to be between the number of units in the input and output layers. Still, these guidelines provide a starting point for a final selection based on trial and error.

Having defined the configuration and purpose of an MLP, its training is conducted in a supervised fashion by employing the technique of backpropagation. If properly trained, these architectures are able to acquire sufficient knowledge to solve its given task very accurately. In fact, they have shown very good performances in computer science field, such as speech and image recognition systems.

In what regards power systems, the application of ANNs in general, and MLPs in particular, is wide and not a novelty. Therefore, a lot of subjects have at least experienced their use: for instance, the definition of protection schemes for transmission lines, determining the location of the fault with ANNs [40] [41]; transformer fault diagnosis [42] [43]; on-line voltage stability monitoring [44]; forecasting problems of load [45] [46] [47], wind power [48] [49] and photovoltaic production [50] [51] [52].

3.3.2 Autoencoders

Autoencoders (AEs) are a kind of artificial neural network trained to output a copy of its input. However, it is usual to apply some restrictions that allow them to copy approximate and only inputs that resemble the training data. This process forces to prioritize given aspects to be copied, providing the ability to learn useful properties of data. Since merely copying the inputs may seem useless, the objective of training the Autoencoder is that the hidden layers obtain useful properties from data. A way of doing that is to constrain the hidden layer to have smaller dimension than the input layer, as exemplified in Figure 3.3. This is a process commonly referred to as encoding, in which the Autoencoder is forced to capture the most salient features of the training data by representing it in an undercomplete representation, enabling dimensionality reduction [7]. Then a conversion of that representation is performed by a decoding function, retrieving the original format.

An interesting approach beyond the classic Autoencoders is found in the so-called Denoising Autoencoder (dAE). These can be interpreted as a stochastic version of the Autoencoder, receiving a corrupted input $\tilde{x}$ and trained to reconstruct and obtain the original, uncorrupted input x. That is, firstly, a stochastic corruption process randomly chooses some of the input points and sets them to zero, simulating the removal of these components from the input [53]. This procedure approximates an actual issue affecting power systems nowadays, that being the case of missing data. Control centers are constantly flooded with measurements arriving from the various measurement points, and sometimes that information can be distorted or even missing. This could be explained by the malfunction of the metering devices or the communication facilities. In fact, this is a common issue regarding the communication of the measurements by PMUs. Receiving that sort of data values, the dAE tries to predict the missing/corrupted values from the non-missing ones. More specifically, the dAE encodes the input by preserving as many features as possible, followed by the inversion of the stochastic corruption effect applied to the input. Therefore, it aims at capturing the statistical dependencies between the inputs [3].

Stacked Denoising Autoencoders (SdAs) consist of series of dAEs forming a deep network, in which the output of one dAE is the input of the subsequent dAE. Such a framework is usually

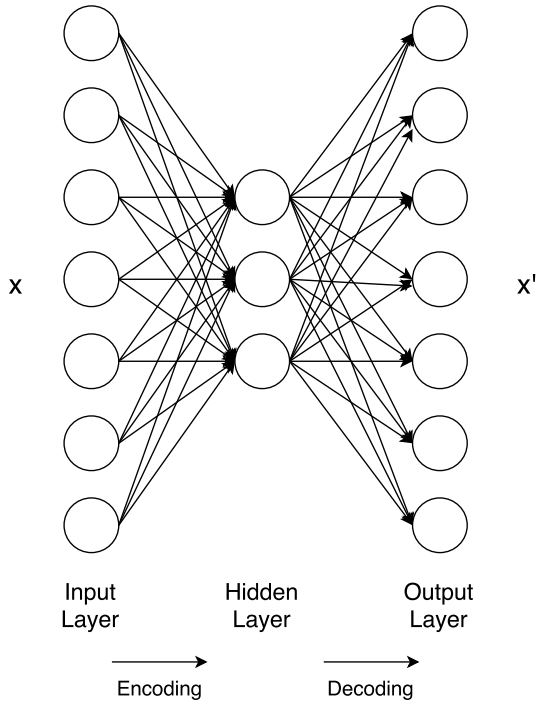

Figure 3.3: The structure of a classic Autoencoder. The network is trained to reconstruct the inputs x into x', capturing the most salient features of the inputs in the smaller dimension hidden layer

trained in an unsupervised, greedy (one layer at a time) fashion. Each layer is therefore trained as a single Denoising Autoencoder, focusing on reconstructing its input. Worth noticing in this case that greedy procedures are not always adopted. In fact, some alternatives regarding Information Theoretic and (top-down) Generative Model have been addressed to in both [53] and [3]. Additionally, as long as a classification task is desired, a second phase of supervised training is required. Slight adjustments are applied in this fine-tuning step, in which the network as a whole resembles an MLP.

3.3.3 Deep Belief Networks

The concept of Belief Networks is not new. In fact, their origin can be traced back to 1992 [54]. However, the depth of these and other architectures was not much explored due to the difficulty of training and consequent bad generalization performance. An explanation was found in the problems that randomly initializing parameters brought to gradient-based optimization [55]. The breakthrough for effectively train deep architectures was only introduced in 2006 by [56]: Deep Belief Networks (DBNs) are pre-trained in a greedy, layer-wise unsupervised manner, followed by a supervised fine-tuning process. More specifically, each layer is individually pre-trained with an unsupervised method, such as the Contrastive Divergence, for learning a non-linear transformation of its input. Afterwards, a gradient-based optimization performs the final supervised training

for fine-tuning the network. The unsupervised learning acts as an initialization of the network parameters by finding a good starting point in the parameter space for the supervised training, which represents better generalization [55]. Despite the initial eagerness, DBNs have been falling out of favour since they have been outperformed by other frameworks in several applications. Still, recent developments in power system forecasting research have reawakened the interest in DBNs, specially in the field of power systems, demonstrating its current usefulness and motivating the author to experience its performance.

Structurally, DBNs can be understood as a composition of single unsupervised networks - Restricted Boltzmann Machines (RBMs) - stacked to form a deep architecture. As the name implies, RBMs evolved from the simple Boltzmann Machine, which is a fully-connected network with two layers, one visible and one hidden. By restricting the connections between neurons within the same layer, an RBM is created. The individual RBMs are stacked in a way that the hidden units of one RBM correspond to the visible layers of the next RBM. In doing so, the Deep Belief Network structure is generated, as seen in Figure 3.4.

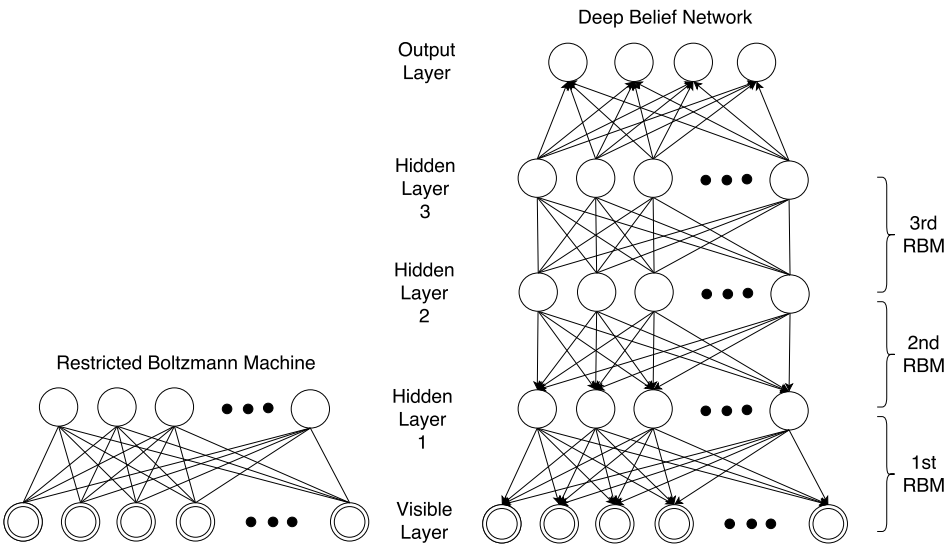

Figure 3.4: Individual RBMs (left) can be stacked to create the corresponding DBN (right). For classification tasks, an output layer is also added

In terms of structure, Deep Belief Networks are very similar to Multilayer Perceptrons. However, the employment of RBMs and the distinctive unsupervised pre-training method turns DBNs into probabilistic generative models due to their ability to provide the joint probability distribution of their inputs and outputs. The additional supervised fine-tuning learning can be handled as in the traditional MLPs, requiring the provision of labelled data.

Mainly, DBNs have been used for generating and recognizing images [56] [57], video sequences [58] and motion-capture data [59]. Moreover, Deep Belief Nets can also be structured so as to perform non-linear dimensionality reduction [60]. In fact, [55] showed that using Autoencoders as an alternative to RBMs produced very similar results.

As mentioned earlier in this section, Deep Belief Networks have recently been employed in forecasting problems. For instance, [61] applied deep feature learning using DBNs for day-ahead wind speed forecasting. The DBN model employed was trained by using a step-by-step greedy algorithm, enabling the extraction of complex features of wind speed due to its strong nonlinear mapping ability, demonstrating high forecast accuracy. Moreover, applications in wind power forecasting regard a hybrid combination of Wavelet Transform and DBN, as adopted in [62] and [63]. Besides wind forecasting, in [64] a DBN is used for electricity load forecasting and the model implemented demonstrated good results in 24h ahead forecasting; in [65], a DBN is used for solar power forecasting.

3.3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [4] are a special type of FFNN for processing data that has grid-like topology. Its structure is biologically-inspired, in which the connections between the neurons are based on the animal visual cortex. Each neuron responds to stimulus within their receptive field in a way mathematically approximate to a convolution operation. Receptive fields of different neurons partially overlap, tiling the visual field. Indeed, CNNs tend to explore spatially-local correlations, therefore enforcing local connectivity patterns between neurons of adjacent layers - a property sometimes called Sparse Connectivity. This special behaviour is illustrated in Figure 3.5: the inputs of a hidden unit in layer *m* come from a subset of units in layer *m-1* that are spatially contiguous (receptive field).

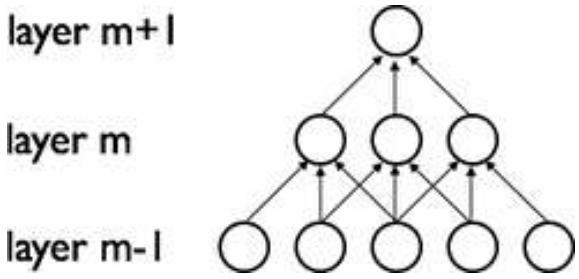

Figure 3.5: Connectivity pattern between neurons of adjacent layers - how CNNs emulate the neuron response to stimulus only within its receptive field: units in layer *m* are connected to 3 adjacent units in layer *m-1*, therefore having receptive fields of width 3; the unit in layer *m+1* is also connected to 3 adjacent units in layer *m*, therefore having a receptive field of width 3 with respect to that layer and a receptive field of width 5 with respect to layer *m-1* (input) [3]

Variations outside the receptive field of each unit do not interfere with its output. Thus, this architecture guarantees that the learnt "filters" produce an effective response to a spatially local input pattern. Stacking several layers, as shown in the previous figure, leads to "filters" that are more global, i.e., responsive to larger regions of the input space.

This connectivity pattern is replicated for all units within each layer to cover the entire visual field, which allows the detection of features regardless of their position in the visual field. In doing

so, a feature map is created. A parameter sharing technique is then applied to each feature map - all units share the same parameters (weights and biases), as demonstrated in Figure 3.6.

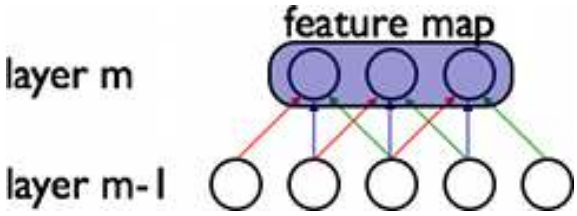

Figure 3.6: Parameter sharing technique: the three units of layer *m* form a feature map and the connections of the same color are constrained to be equal (shared) [3]

This characteristic of parameter sharing increases learning efficiency due to the reduction of the number of free parameters to estimate. Since CNNs can also be trained with the backpropagation algorithm, having fewer parameters to learn makes them typically easier to train than other frameworks. This is highly attractive because memory requirements for running the network are reduced, leading to the ability of training larger, more powerful networks. Additionally, CNNs have shown better performance in several applications, specially in both image and speech recognition. Consequently, deep CNNs are becoming one of the means to effectively extract the invariant structures and inherent hidden features in data.

In general, a feature map h is obtained by the *convolution* of the input x with the weights W (linear filters), to which a bias b is added and finally a non-linear transformation (for instance *tanh*) is applied:

$$h = \tanh((W * x) + b) \tag{3.1}$$

To better attain the inner properties of the input data, a convolution layer consists of multiple feature maps.

Typical deep CNN architectures consist of alternating convolution and pooling layers, followed by a fully-connected layer before the output layer. The common practice is to insert a pooling layer between one or more convolutional layers in a CNN architecture, as shown in Figure 3.7

The convolution layer - whose working principle was explained earlier - consists of multiple two-layer FFNNs that adopt the mathematical convolution operation in order to transform low-level maps with local features into several high-level maps with global features. The weight sharing technique between neurons in different layers helps the process of feed forward and back-propagation. This is the main attribute that makes this architecture different from others, due to the reduction in the number of parameters to be estimated, while still being able to effectively extract the hidden invariant features in data. The pooling layer implements a max-pooling effect, which

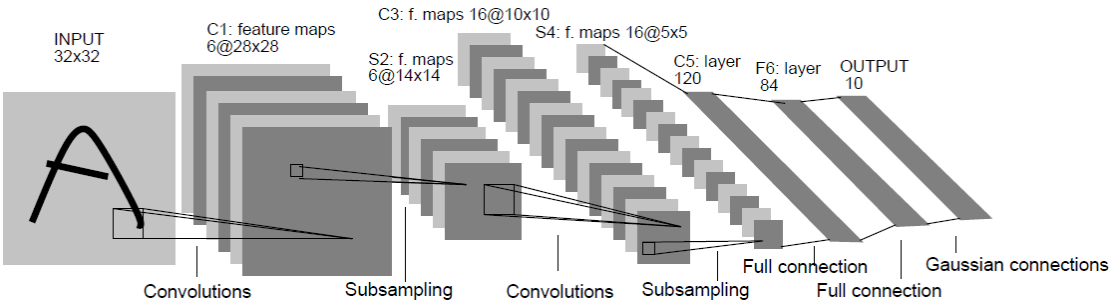

Figure 3.7: Architecture of the LeNet-5, a well-known convolutional neural network example [4]

is a form of non-linear down-sampling. The input is divided into a set of non-overlapping rectangles, outputting the maximum (max-pooling) of each sub-region. The pooling layer objective is to progressively reduce the spatial size of the representation, decrease the number of parameters and consequent amount of computational effort, and control overfitting issues.

Due to this visual influence, CNNs applications are primarily in the visual field, with several image recognition examples [66] [67] [68] [69]. For video classification, where complexity increases due to the temporal dimension, some extensions of CNNs have been explored [70] [71] [72]. Despite that, this architecture has also been tested outside the field, with both recommendation systems [73] and natural language processing [74]. It is worth mentioning that, in case of CNNs are to be used in non-visual data, a proper way of encoding that data must be found, so as to replicate the properties of visual information (grid-like topology). Therefore, the spread and success of this architecture for any other than image-based applications is inherently dependent on the ability to emulate the properties of an image.

Nevertheless, Convolutional Neural Networks have proven to be highly flexible and consequently implemented in several and distinct applications. Recently, CNNs have found their place within Electric Power Systems. For instance, in [75], a deep CNN architecture is employed for power line insulators classification from aerial images. The well-known performance in image classification tasks and the recent success of deep learning led to the first approach registered at the time to implement a deep CNN for insulator condition inspection. Another recent application of CNNs is presented in [76], where the uncertainties in wind power data can be effectively learnt for probabilistic wind power forecasting. In fact, the high variability and volatility exhibited by wind power data is soothed by the employment of a hybrid Wavelet Transform, deep CNN and ensemble technique. The deep CNN architecture demonstrates great efficiency in extracting the nonlinear and stochastic nature of each wind power frequency.

3.3.5 Spatio-Temporal Deep Learning

The majority of deep learning architectures focuses on extracting spatial features. However, a sequence of patterns can give an understanding that single and discrete patterns might not provide. So, capturing the additional temporal dependencies in observations is vital for an effective

and accurate inference of knowledge [5]. In spite of that, simultaneous integration of both spatial and temporal components has not been an object of extensive research. The very first work was presented by in [77], where DeSTIN - Deep Spatio-Temporal Inference Network - is introduced as a discriminative Deep Learning framework combining unsupervised learning for dynamic pattern representation together with Bayesian inference. DeSTIN is considered to model the spatio-temporal dependencies in the observations in an unguided manner, whilst successfully dealing with high dimensional signals.

The DeSTIN architecture consists of a hierarchy of layers, each having multiple nodes, such as the typical Deep Learning frameworks. Additionally, every node has a corresponding "child" from the layer below and a "parent" from the layer above, creating a hierarchical structure as presented in Figure 3.8.

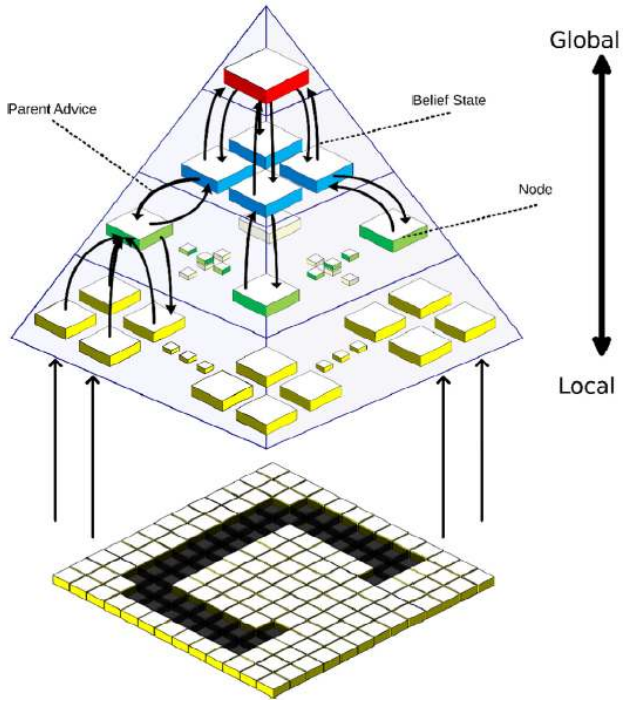

Figure 3.8: Hierarchical structure and signal flow of a DeSTIN architecture[5]

The nodes at the lowest layer receive the raw data as input - as exemplified in Figure 3.8 with the letter 'C' - producing an output, referred to as belief state, that is the input of its corresponding "parent" node. In doing so, each node attempts to capture the spatio-temporal features contained in its input and update a respective belief state. This belief state indicates the probability of each possible state given the known information about the system. This sort of receptive field is unique for each node and means to characterize the input and the sequences thereof. The beliefs created throughout the entire architecture represent rich features that can be harnessed by hybrid architectures that employ further supervised learning for classification purposes [5]. As it can be observed in Figure 3.8, the spatial and hierarchical disposition of each node implies that belief states extracted from the lower layers will describe local features whereas beliefs from higher layers will

characterize global features. Therefore, DeSTIN learns features from data in a total unsupervised fashion, which stands in contrast to the other architectures described that rely on previous knowledge of the problem at hand.

An enhancement to the original DeSTIN is proposed in [5], where an innovative recurrent clustering algorithm is employed for capturing the spatio-temporal dependencies, as an unsupervised learning procedure included in each node of the DeSTIN hierarchy. Furthermore, owing to parallel and independent operation of each node, remarkable scalability attributes, specially in GPU implementation, are offered. Note that modern GPU are highly parallel programmable processors that have a peak arithmetic and memory bandwidth much greater than any CPU [78]. Thus, the ability to map DeSTIN to a GPU implementation is of great importance in order to tackle large problems that use high resolution datasets, such as image or video.

Capturing the temporal dependencies on data is not straightforward. Also, as mentioned earlier, it has not been an object of intensive research since most architectures are focused on improving their performance on learning spatial features. Nevertheless, the ability to extract features based on the sequential behaviour of data, in an unsupervised manner, seems very promising. In the particular field of power systems, modelling the temporal dependencies inherent to consecutive information prospects great advances on monitoring and control the dynamics of the system.

3.4 Final Remarks

The advent of WAMS and consequent PMU mass-deployment is of great importance for the improvement of monitoring and control of power systems. However, as stated in section 2.10, power system operators are craving for techniques that efficiently deal with the amount of raw data flooding control centers nowadays. Artificial Intelligence, and more specifically Deep Learning approaches, seem to be a valid course of action to follow. The raw data provided by the PMUs prospects a need for unsupervised learning techniques that extract valuable information from patterns recognized in synchronized measurements. Otherwise, training supervised frameworks requires previous extensive labelling of their datasets. Due to its unsupervised pre-training phase, the Deep Belief Networks can emerge as a worthwhile framework to be employed. Alternatively, the flexibility exhibited by Convolution Neural Networks makes them highly attractive, despite needing a way to encode data as images. Moreover, Autoencoders are envisaged as a means of data dimensionality reduction. The issue regarding missing data on PMUs might aswell be a motivation for the employment of Autoencoders, in its Denoising version. In addition, the highly temporal component of the patterns present in PMU data opens doors for the application of spatio-temporal algorithms for harnessing temporal dependencies.

Recent deployments of Deep Learning approaches regarding electric power systems in general are depicting a bright future for the coupling of both activities. Indeed, several applications have proven to benefit from the employment of deep learning frameworks. Also, the current availability of GPU implementation is assuaging the computational effort inherent to those architectures. Thus, the previous and the present section comprised an overview of both PMU technology and

Deep Learning paradigm, aiming at a revision of concepts and respective applications. The intention of this dissertation is to propose an innovative application of deep learning frameworks that can effectively extract features from synchronized measurements provided by PMUs.

Chapter 4

Context of The Work

This chapter contextualizes the work developed. It describes the PMU-based measurement system that provided the data used, as well as the classification task carried out. The impacts of each disturbance in system frequency are detailed, hence providing an insight to the ongoing electrical phenomena.

4.1 The Medfasee BT Project

As described earlier in section 2.9, Brazil has implemented a Low Voltage, PMU-based, Wide Area Measurement System. By the end of 2013, 22 devices were already installed in several universities spread across the country. In addition, a 23rd PMU was envisaged to be installed but, at the time of this thesis, that still had not happened. The data collected from each PMU is sent to a central PDC installed in Universidade Federal de Santa Catarina (UFSC). So as to better understand the geographical disposal of the devices, the location of each university taking part in the project is identified in Figure 4.1. The acronym of each measurement point is described in Appendix D.

Besides R&D, the data produced is being harnessed by the brazilian system operator for *post-mortem* disturbance analysis. Also, the geographical distribution of the installed devices allows the monitoring of electromechanical oscillations [6].

It is worth noticing that the Medfasee BT Project contributed greatly for the development of this thesis by providing a complete dataset with several events properly labelled. Indeed, a manual database was created including real events registered by the system operator between 2010 and 2015 [79]. Since the PMUs are installed in the low voltage grid, a careful selection of cases was performed, which highlighted four types of events:

- Generation Tripping - cases of generation loss that caused frequency changes greater than 0.08 Hz (approximately 530 MW);
- Load Shedding - cases of load loss that resulted in frequency changes greater than 0.07 Hz (approximately 440 MW);

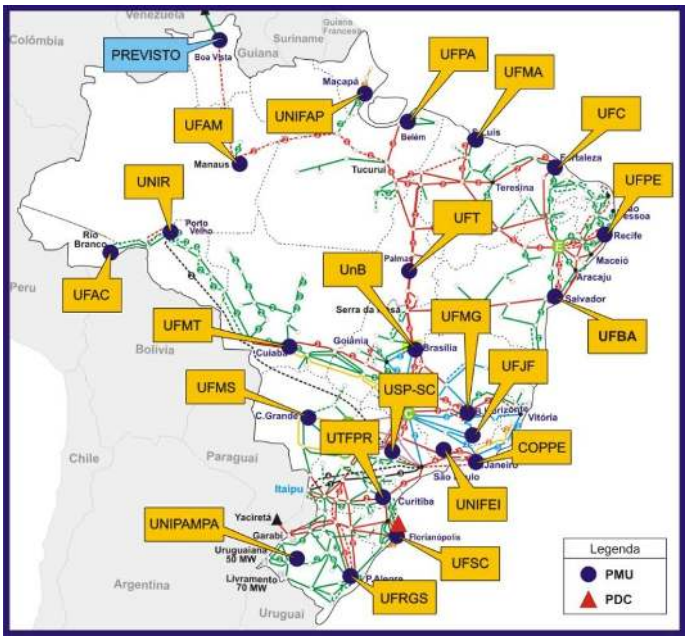

Figure 4.1: Geographical disposal of PMUs implemented in Medfasee Project [6] - details in Appendix D

- Line Tripping - disconnection of $\geqslant$ 500kV transmission lines with significant power flow;
- Oscillation - cases of inter-area oscillations caused by the disconnection of a specific 600kV DC link.

The events detected are distinguishable by the extent of their impacts. That is, both Generation Tripping and Load Shedding are considered systemic events, therefore reaching a larger number of PMUs. In contrast, Line Trippings and Oscillations are observed locally, hence affecting less PMUs. As a result, the number of PMUs worth accessing is highly event-dependent. Having gathered the exact time and location of each disturbance, the data measured by the affected PMUs (stored in the main PDC) was then collected using a software for offline synchrophasor analysis, which was specially developed for the project (MedPlot [33]). Table 4.1 lists the number of events considered for each disturbance and the respective total number of cases (which corresponds to the number of events multiplied by the number of PMUs affected in each event).

Table 4.1: List of extracted events

Events	no. of events	no. of examples
Generation Tripping	55	876
Load Shedding	29	421
Line Tripping	17	46
Oscillation	7	14
Total	108	1357

The ratio between the number of examples and the number of events illustrates the different range of impacts: Generation Tripping and Load Shedding are detected, in average, by 15 PMUs whereas Line Tripping and Oscillation are caught by an average of 2 PMUs. Thus, the distribution of cases is uneven, in which 65% of cases correspond to Generation Trippings, 31% to Load Shedding, 3% to Line Trippings and only 1% of Oscillation examples. So, the total 1357 examples compose a possibly too small dataset in what concerns typical deep learning applications.

Each example from the dataset provided corresponds to the frequency variation registered in a PMU during a time window of 20 seconds: 1 second pre-event (system frequency normal behaviour) and the following 19 seconds of post-event. Since the sampling rate is 60 samples/second, each example is fully characterized by 1200 frequency values. Therefore, each disturbance is characterized by the changes it provoked in the frequency of the low voltage distribution grid. An outlook of the impacts on frequency is given in the following sections.

4.2 The Importance of Frequency in Disturbance Detection

Frequency is one of the linchpin electrical variables in a power system. It can be understood as a parameter of the voltage signal or assumed as a measurement of the system synchronous speed, therefore proportional to the rotational speed of the generators. In what concerns the system operator, frequency is regarded as an image of the generation-consumption balance. Mathematically, that balance is depicted by Newton's 2nd Law of Motion for rotating masses, commonly referred to as the swing equation for power systems:

$$\frac{2H}{w_o}\frac{dw(t)}{dt} = P_{mec} - P_e \tag{4.1}$$

Here, H represents the normalized inertia constant in seconds or MJ/MVA, w_o is the nominal system frequency in rad/s, P_{mec} is the per-unit mechanical power applied to the generator by its prime mover and P_e is the per-unit electrical power load being supplied.

As expressed by the swing equation, mismatches between generation and load result in changes in system frequency: if generation exceeds actual demand (increase in the mechanical power applied to the generator or reduction of the electrical power being supplied) the generators will accelerate and hence the frequency will increase; conversely, a surplus of load forces the generators to slow down since the excess power is extracted from the rotating masses, and frequency decreases. In both cases, the respective control systems should be activated so as to counterbalance the effects and restore the production-consumption equilibrium. Under normal operating conditions, the instantaneous frequency is not always constant due to small generation or load variations. However, large and fast frequency deviations from its nominal value point towards the existence of a disturbance or event in the system. That is, abnormal situations such as short circuits, can lead to the activation of the respective protection systems, removing from the grid the component or a set of components where the fault was occurring. The corresponding impacts are component-dependent,

so a classification can be performed in order to identify their origin. As explained earlier, the dataset used in this thesis comprises four types of events properly labelled - Generation Tripping, Load Shedding, Line Tripping and Oscillations - each of them described in the following sections.

4.2.1 Generation Tripping

Nowadays, electrical facilities with large exploitation of renewable resources are exposed to an inherent and inevitable variability that affects the electricity production. Therefore, system operators have to handle this new paradigm with special care so as to minimize the resulting impacts and prevent excessive frequency variations. For instance, such fluctuations are particularly felt in systems with high wind power penetration, in which sudden climate changes can cause the need to stop the turbines. In addition to dispatch orders, losses of generation can also result from the activation of its protective system due to external factors. Overall, the loss of generation and consequent imbalance will result in the drop of frequency. A real occurrence of a generation tripping is shown below, as it was caught by several PMUs:

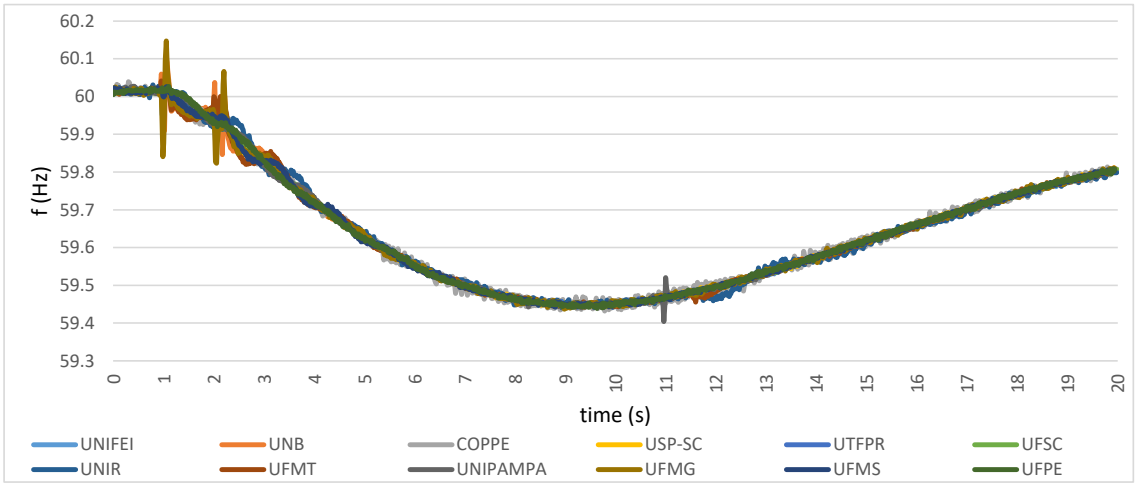

Figure 4.2: Typical frequency change in the presence of a Generation Tripping. Details about the PMUs that caught the event in Appendix D

The frequency progress demonstrates the initial normal behaviour and the following frequency drop caused by the disturbance felt. Moreover, the action of both primary and secondary frequency controls is also well observed a few seconds after the occurrence - respectively, stabilizing and returning the frequency value to its nominal (the latter not shown completely). This is a standard and automatic procedure in which the generators outputs are rapidly redefined so as to avoid a continuous decline in frequency that can lead to under-frequency problems and consequent blackouts.

4.2.2 Load Shedding

Load shedding regards the process of dropping off important loads to restore the balance between demand and generation. A particular and common case is found in the disconnection of pumped

storage power plants after dispatch orders.

However, under the paradigm of system security assessment, load shedding is held as an ultimate resource of control/protection to solve excessive frequency decline and avoid damaging system components. In doing so, the system transits from an Emergency State to a *In Extremis* State (DyLiacco, 1967; Fink and Carlsen, 1978), in which partial or total collapse of the system is a possibility.

Moreover, a load drop off might occur unpredictably, i.e., a severe disturbance or even a malfunction of a component can result in the disconnection of a substantially large load. Normally, such cases are more easily circumvented in highly meshed grids, by adjusting its topology and retrieving the standard operating conditions, as seen below.

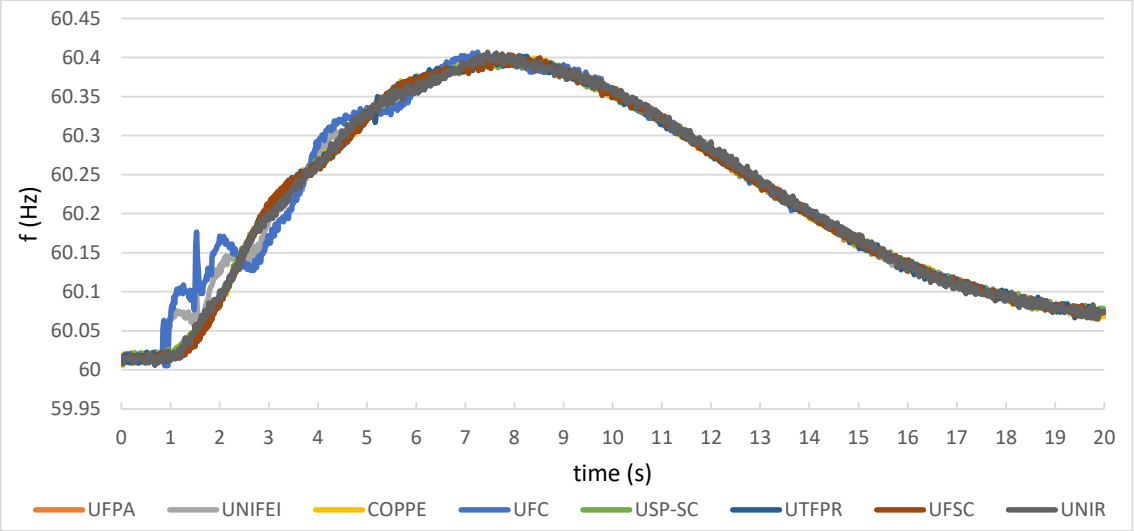

Figure 4.3: Typical frequency change in case of a load shedding - details in Appendix D

Overall, the electrical power supplied decreases significantly and the excess of mechanical power applied to the generator will imply a frequency increase. Again, the actions of counterbalancing are also observed.

4.2.3 Transmission Line Tripping

Whenever a transmission line is switched off, the power flow in its surroundings is changed. In addition, the switching operation creates frequency transients that are transmitted over the form of oscillations around the nominal frequency. As seen in Fig 4.4, in which the event was zoomed in for a better understanding, a rapid, high amplitude transient is caught. Indeed, this kind of phenomena is typically not widely observed, therefore being caught within a small range of nearby PMUs.

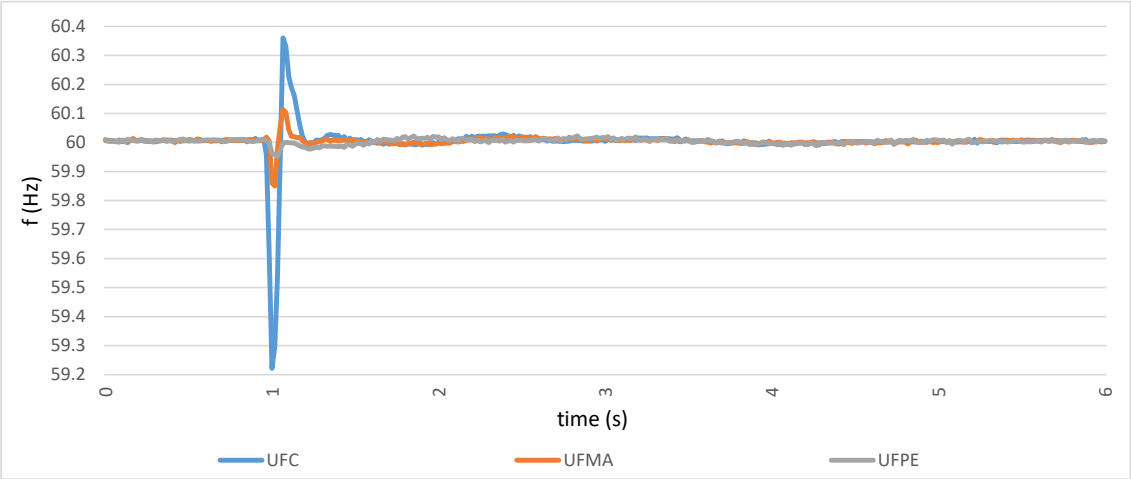

Figure 4.4: Typical frequency change in case of a transmission line tripping off - details in Appendix D

4.2.4 Oscillations

Frequency oscillations are quite often in electrical power grids due to their dynamic behaviour under normal or abnormal working circumstances. In fact, the occurrence of the previously described disturbances can produce additional impacts in the form of frequency oscillations. Therefore, sometimes, a specific origin of an oscillation might not be easily acknowledged. Nevertheless, it is crucial that the system has the ability of maintaining stability by properly damping those electromechanical oscillations. If not, negative consequences affect system components which are disconnected to avoid damage, leading to potential islandings or blackouts. Consequently, transient stability and small signal stability studies assume a major relevance for system operators preventive analysis.

In what regards the dataset supplied, the registered frequency oscillations are originated in a particular HVDC link (600 kV). Thus, they are captured by the PMUs installed in UFAC (Universidade Federal do Acre) and UNIR (Universidade Federal de Rondônia), as seen in Figure 4.5.

4.3 Existing Disturbance Identification Methods

The procedure of identifying a disturbance comprises a series of sequential steps: preprocessing, detection, classification and location. The preprocessing phase performs the signal filtering in order to attenuate noise and leaps. Then, a commonly used method for detecting power system disturbances is to monitor the rate of change of frequency (df/dt) [80]. Whenever the frequency variations exceed a given threshold, the trigger is activated and a disturbance is flagged. Here, the choice for the threshold value (sensitivity of the detector) is critical so as to avoid false triggering. Afterwards, the process of event classification is employed. The traditional classifiers are mainly deterministic, in which a specific algorithm is defined so that for a given input the output is always

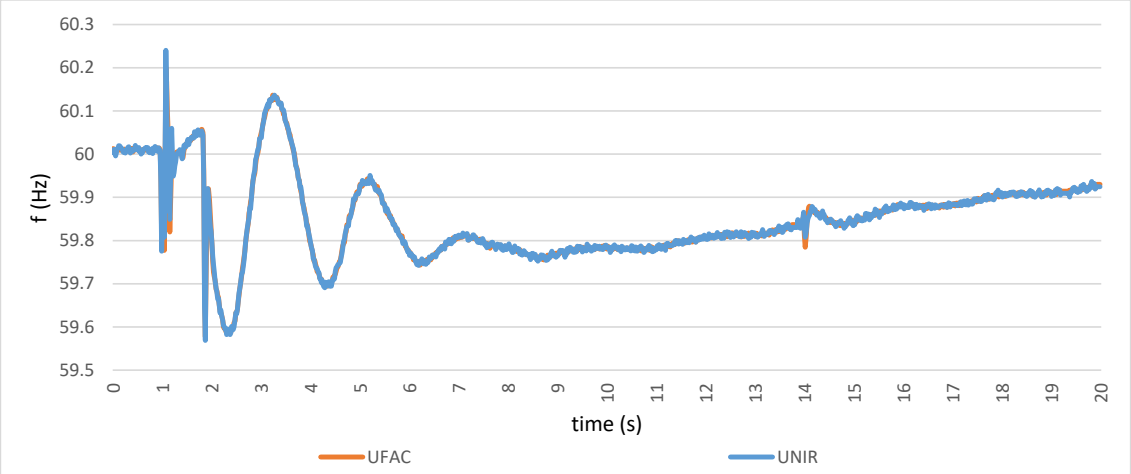

Figure 4.5: Typical frequency change in case of an oscillation - details in Appendix D

the same. Recently, statistical methods have gained special attention. They have been proven to be better suited for dealing with power system randomness, especially with the increasing penetration of renewable sources. Indeed, several recent methods for statistical classification have been proposed, such as artificial neural networks [79], support vector machines [81] [82], naive Bayes classifiers [82], decision trees [82], etc. After classification, the location of the event can also be estimated employing the concepts of electromechanical wave propagation [83]. In general, a lot of research has been conducted in the area of disturbance identification and system operators are usually endowed with such functionalities. However, the fresh developments in the area of artificial intelligence, namely in deep machine learning, foresee the application of new methodologies for disturbance classification.

4.4 Proposed Classifier

Inspired by the advances in Deep Learning, this thesis brings forward new procedures for performing automatic classification of power system disturbances. More specifically, the aim is to learn features from the temporal frequency variation involved in a disturbance. The classifiers herein developed will be applied to the dataset provided and trained to distinguish the four types of events described earlier in this section. In order to do so, three different frameworks were explored:

- Multilayer Perceptron - this type of artificial neural network was adopted here to perform a direct comparison to the work developed in [79], in which a one hidden layer architecture is proposed. Thus, this work designed a one hidden layer architecture and also deeper architectures with four and eight hidden layers;

- Deep Belief Network - the depth of the network is obtained by stacking individual RBMs on top of each other (in this case, 3 RBMs);

- Convolutional Neural Network - its visual field inspiration was used for dealing with the whole event frequency change as an image/frame snapped from the continuous movie that is the operation of the system.

In the following chapter, the specifications of each framework engineered are detailed. The respective achievements are presented in chapter 6.

Chapter 5

Methodology

In this chapter, the frameworks designed for the classification procedure are explained, detailing the chosen structure and respective parameters. For a deeper understanding of the concepts involved in each implementation, the reader is advised to visit Section 3.3, where the respective theoretical explanations are given. What's more, an observation is made on how the data was preprocessed before being fed to the algorithms adopted. It is important to note that this is a different kind of data treatment, typically applied in Machine Learning, and should not be confused with the filtering phase of the disturbance identification methods. The end of the chapter presents the motivations for the use of GPU computing as a means to accelerate the computational procedures.

5.1 Data Preprocessing

In Deep Learning, data preprocessing is almost essential, since many frameworks achieve better performance when the dataset is rescaled. However, there is not a defined recipe that could be followed. Sometimes, the distinctiveness of data requires a trial and error procedure and some experience for finding the best way to work out the dataset. Still, two generic and widely applied techniques are normalization and standardization. Normalization involves the rescaling of data in order to set all values within the range of 0 and 1. Standardization sets the observations to fit a Gaussian distribution, hence examples are rescaled to have zero-mean and unit-variance.

As mentioned in Section 4.1, the data entries correspond to the frequency variation registered in the 20 seconds of disturbance. Since samples were obtained with a rate of 60 samples per second, one disturbance is characterized by 20 x 60 = 1200 frequency values. Moreover, the preprocessing method in use is standardization. Sometimes, standardization is more robust when dealing with observations that might step outside the range of expected values, which can be very helpful since the disturbance can lead to unpredictable frequency values.

5.2 Logistic Regression

The logistic regression is considered one of the most basic classifiers in machine learning theory. It is a probabilistic classifier as it performs the classification by projecting the input values onto a set of hyperplanes, each corresponding to a given class. The distance from the input to each hyperplane reflects the probability that the input is a member of the matching class [3].

The logistic activation function is usually determined by the number of classes to predict. In binary classification, the *sigmoid* function is preferably employed. In cases of multiclass classification, the *softmax* function is mostly used. In fact, *softmax* is no more than an extention of the *sigmoid* function, hence both being equivalent for binary classification. Regarding the case of multiclass classification, the probability of an input x belonging to a class i is expressed as:

$$P(Y = i|x,W,b) = softmax_i(Wx+b) = \frac{e^{W_i x+b_i}}{\sum_j e^{W_j x+b_j}} \tag{5.1}$$

This formulation can be replicated to calculate the probability distribution of the input example over the different possible classes. The output probabilities will range from 0 to 1, all summing up to 1. Thus, the actual prediction $f(x)$ is determined by the class i, whose probability is maximal (therefore the *argmax* function):

$$f(x) = argmax_i [P(Y = i|x,W,b)] \tag{5.2}$$

Regarding ANN-based classification tasks, logistic regression is regularly employed as the output layer. The event classifiers implemented in this thesis establish a multiclass logistic regression layer as the output. The class predictions are used for determining the accuracy of the model under a function called Zero-One Loss (detailed below in Section 5.3.1). However, due to computation efficiency issues, the optimal model parameters are obtained with the minimization of the Negative Log-Likelihood as loss function (explained in Section 5.3.2), by conducting a mini-batch SGD optimization.

5.3 Loss Functions

Loss functions differ greatly with respect to the paradigm and objective of training. In classification problems, it is very useful to measure the obtained miss-classifications, since they provide the accuracy of the model. This measurement is commonly referred to as Zero-One Loss. Nevertheless, it is not always mathematically possible to learn the classifier on that measurement, and a probabilistic meaning is assigned - for instance, with the Negative Log-Likelihood. These concepts are adopted in this thesis, so a fully description is given in the following sections.

The classification tasks herein conducted are supervised, hence requiring the dataset $\mathscr{D}$ in evaluation to comprise a set of pairs $(x^{(i)}, y^{(i)}) \rightarrow$ (i-th input, i-th target). Accordingly, the i-th input

of dimensionality D, $x^{(i)} \in \mathscr{R}^D$, is assigned the i-th target from a set of L targets $y^{(i)} \in \{0,...,L\}$. The loss functions implemented regard this organization in their description.

5.3.1 Zero-One Loss

The main objective in classification tasks is to minimize the number of errors in unseen examples. The so called zero-one loss function performs exactly that as it measures the accuracy of the classification: returns 0 if the predicted class is equal to the true class or 1 if missclassified.

Mathematically, having a prediction function for L classes as $f : \mathscr{R}^D \rightarrow \{0,...,L\}$, the corresponding loss ℓ on the dataset $\mathscr{D}$ is given as:

$$\ell_{0,1} = \sum_{i=0}^{|\mathscr{D}|} I_{f(x^{(i)}) \neq y^{(i)}} \tag{5.3}$$

in which $f(x^{(i)}) = argmax_i \left[P(Y = k|x^{(i)}, W, b)\right]$ is the class predicted for the input $x^{[i]}$ (equation 5.2) and $y^{(i)}$ is the correct label.

The indicator function I_z determines the correctness of the classification through the direct comparison between the predicted classes and the real targets:

$$I_z = \begin{cases} 1 & \text{if } z \text{ is True (missclassification)} \\ 0 & \text{if } z \text{ is False (properly classified)} \end{cases}, \quad z \rightarrow f(x^{(i)}) \neq y^{(i)} \tag{5.4}$$

By doing so for every example i on the dataset $\mathscr{D}$, the accuracy of the model is determined.

5.3.2 Negative Log-Likelihood

The non differentiability of the zero-one loss function requires excessive computational resources when optimizing large models - huge number of parameters θ - during the training procedure. Thus, an alternative is found in maximizing the log-likelihood of the classifier:

$$\mathscr{L}(\theta, \mathscr{D}) = \sum_{i=0}^{|\mathscr{D}|} \log P(Y = y^{(i)}|x^{(i)}, \theta) \tag{5.5}$$

Since typically the loss functions are to be minimized, the learning can be adapted so as to perform the training data on a negative log-likelihood (NLL):

$$NLL(\theta, \mathscr{D}) = -\sum_{i=0}^{|\mathscr{D}|} \log P(Y = y^{(i)}|x^{(i)}, \theta) \tag{5.6}$$

Therefore, the *NLL* is assumed to be the differentiable surrogate for the zero-one loss function. In doing so, the gradient can then be efficiently applied to the training set of examples. However, it should be pointed out that the zero-one loss function and the *NLL* represent different objectives.

5.4 Multilayer Perceptron

The Multilayer Perceptron implemented here is a typical feedforward, fully-connected, DNN with an input layer, hidden layers and an output layer. The input layer contains 1200 neurons so as to be fed with the 1200 frequency values (20 seconds x 60 samples/second) as inputs. Several experiments were conducted in what regards the number of hidden layers and the respective number of neurons. Particularly, in order to ascertain the effect of the model's depth, 3 architectures were tested: 1, 4, and 8 hidden layers - MLP 1, MLP 4 and MLP 8, respectively. The number of neurons chosen for each layer is detailed in Table 5.1. Both the input and hidden layer neurons were implemented with the hiperbolic tangent (*tanh*) as activation function.

Moreover, the weights of the hidden layers should be initialized to be small enough so that the activation function operates in its linear region, where gradients are larger. Therefore, the initialization is dependent on the activation function in use. For *tanh*, this is held by uniformly sampling from the following symmetric interval, as proposed in [84]: $[-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$, in which fan_{in} and fan_{out} are the number of units in the $(i-1)-th$ and $i-th$ layers, respectively.

The classification task is performed by employing a *softmax* logistic regression layer as the output layer. The number of output neurons corresponds to the number of events in classification. The training procedure is carried out by minimizing the Negative Log-Likelihood and updating the parameters with a mini-batch SGD. The minibatch size is variable. Several sizes were experimented and the ones adopted are shown in Appendix B. Finally, the accuracy of the model designed is observed by applying the Zero-One loss function.

Table 5.1: Specifications of the several Multilayer Perceptrons developed

Settings	MLP 1	MLP 4	MLP 8
input neurons	1200	1200	1200
hidden neurons	500	500 | 200 | 100 | 50	1200 | 1000 | 800 | 500 300 | 200 | 100 | 20
output neurons	no. events	no. events	no. events
hidden layer activ. func.	*tanh*	*tanh*	*tanh*
ouput layer activ. func.	*softmax*	*softmax*	*softmax*
training loss func.	neg. log-likelihood	neg. log-likelihood	neg. log-likelihood
training optimizer	mini-batch SGD	mini-batch SGD	mini-batch SGD
accuracy evaluation	zero-one loss	zero-one loss	zero-one loss

5.5 Deep Belief Network

The Deep Belief Network used in this thesis is implemented by stacking 3 Restricted Boltzmann Machines on top of each other. Each individual unit was assigned the *sigmoid* as activation function. The first layer RBM is the input of the network, hence having 1200 neurons in its visible layer. Since the DBN is used for classification purposes, the hidden layer of the last RBM is not the output of the network. Instead, the DBN is treated as a special MLP and a logistic regression layer is added on top for performing the classification task. Again, the output units are determined by the number of events to distinguish. These settings are organized in Table 5.2.

Table 5.2: Specifications of the Deep Belief Network designed

Settings	DBN Architecture
input neurons	1200
units in 1st RBM	1200 800
units in 2nd RBM	800 800
units in 3rd RBM	800 800
units activ. func.	*sigmoid*
output neurons	no. of events
ouput layer activation	*softmax*
pre-training method	CD-1
fine-tune cost	neg. log-likelihood
accuracy evaluation	zero-one loss

As described in section 3.3.3, DBNs can be efficiently pre-trained in a greedy, layer-wise, unsupervised manner. Thereafter, a supervised fine-tuning for a final weight adjustment is employed. More specifically, the whole training procedure followed the subsequent guidelines:

- Unsupervised pre-training:
  1. The first RBM is trained to obtain a representation of the input data;
  2. The second RBM is trained using the outputs from the first RBM. That is, the transformed inputs in the hidden layer of the first RBM are the visible layer inputs of the second RBM;
  3. Points 1. and 2. are repeated until every layer in the network is trained - in this case, 3 RBMs were implemented;
- Supervised fine-tuning of the whole model.

The objective in training an RBM is to maximize its log-likelihood. However, applying the stochastic gradient descent to an RBMs is mathematically intractable. For that reason, Hinton proposed a new method - Contrastive Divergence [85] - which approximates the log-likelihood gradient calculus.

In RBM training, maximizing the log-likelihood of the input data is equivalent to minimizing a Kullback-Leibler divergence $(Q^0||Q^\infty)$ between the data distribution Q^0 and the equilibrium distribution of the model Q^∞. The latter implies running several steps of Gibbs sampling (each variable draws a sample given the current state of the other variables) over the model, therefore resulting in high computational effort. The reasoning behind CD-1 is to minimize the difference between $(Q^0||Q^\infty)$ and $(Q^1||Q^\infty)$, where Q^1 is the resulting distribution after one-step of Gibbs sampling. Surprisingly, performing just one-step approximates very well the theoretical result and is computationally efficient, so it is widely applied in the pre-training phase of DBNs. In the context of RBMs, one-step of Gibbs sampling comprises the following steps:

1. Start with a training vector on the visible units;
2. Update all the hidden units in parallel;
3. Update all the visible units in parallel, getting a reconstruction of the input data;
4. Update all hidden units again.

The key factors to achieve both speed and correctness is to start the network with small wights and use the CD-1. Therefore, the DBN implemented consists of the one-step Contrastive Divergence (CD-1) as the unsupervised method, running for 100 epochs for each RBM layer. Since the activation function in use is the *sigmoid*, the weights were initialized as proposed in [84]: $[-4 \times \sqrt{\frac{6}{fan_{in}+fan_{out}}}, 4 \times \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$, in which fan_{in} and fan_{out} are the number of units in the $(i-1)-th$ and $i-th$ layers, respectively.

The end of the pre-training stage results in that each RBM has learned to detect inherent patterns in data. However, the network cannot still identify each of the patterns learned. To finish training, labels of the given patterns need to be provided so as to fine-tune the net with supervised learning. An important feature of this architecture is that it only needs a small labelled dataset for performing the final fine-tuning, which is revealed as of great importance in real-world applications. This supervised fine-tuning phase is similar to the training procedure implemented for learning the MLP, although now only small changes are applied to its parameters. With the help of the output logistic regression layer, fine-tuning is performed via minibatch stochastic gradient descent in the negative log-likelihood cost function. Moreover, the accuracy of this framework is evaluated by quantifying the wrong classifications, that is, by using the Zero-One Loss function.

5.6 Convolutional Neural Network

The trick to apply Convolutional Neural Networks is to find the right settings for the filters implemented. The filter shapes found in the literature vary greatly according to the dataset in use. In addition, CNNs are an image-based methodology, so its implementation is easier when working with images as inputs, specially squared images that involve symmetrical structures. For non-image applications, a way to replicate a visual shape is required.

In this thesis, the 1200 frequency values of each event can be understood as a 1200x1 vector of data. For enabling the CNN training, these 1200 "pixels" have to be re-shaped so as to produce a matrix structure that resembles the frame of an image - height x width. Two image shapes were developed: 30x40 and 20x60. Unfortunately, fitting the 1200 values in a symmetrical structure (height = width) is not possible. Thus, this asymmetry carries an extra complexity to the definition of the filter shapes to be applied. An illustration of the resulting 30x40 images, for a specific event of each class, is shown in Figures 5.1-5.4. For a better understanding of the pattern involved in a given event, the 30x40 pixels were assigned a color as a function of its frequency value: green for nominal frequency, red and blue for under and over the nominal frequency, respectively.

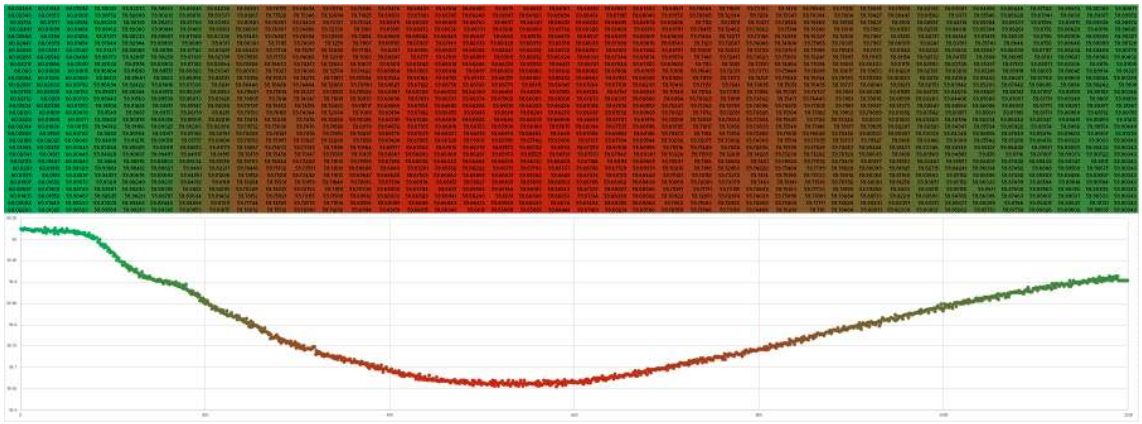

Figure 5.1: Generation Tripping: the input pattern of the CNN and the corresponding original frequency variation

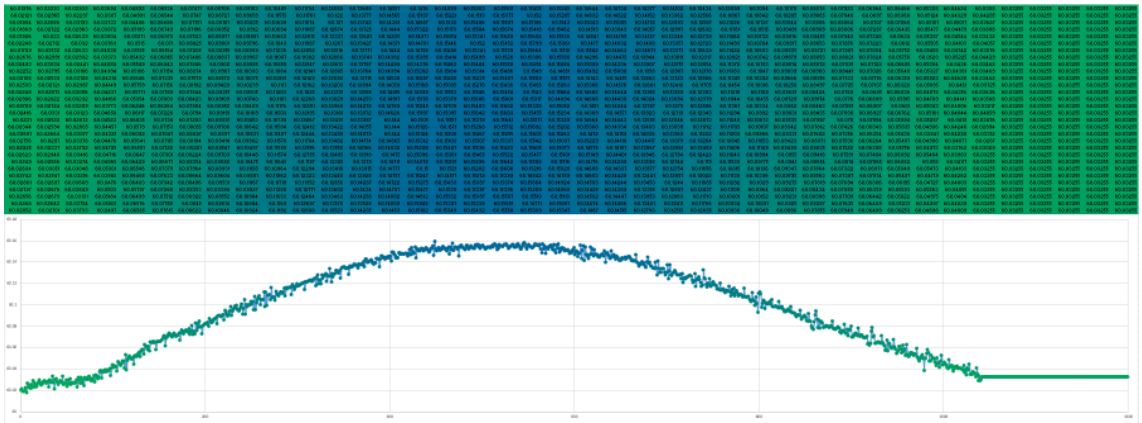

Figure 5.2: Load Shedding example: the input pattern of the CNN and the original frequency variation

The architecture designed will also be explained for the 30x40 case. For an improved understanding of the subsequent description, the reader is advised to simultaneously visualize its concepts in Figure 5.5.

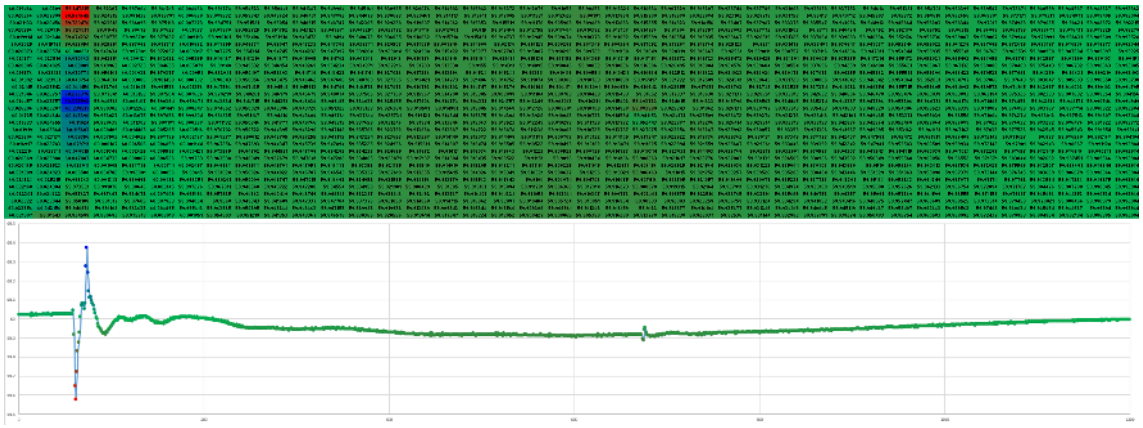

Figure 5.3: Line Tripping example: the input pattern of the CNN and the original frequency variation

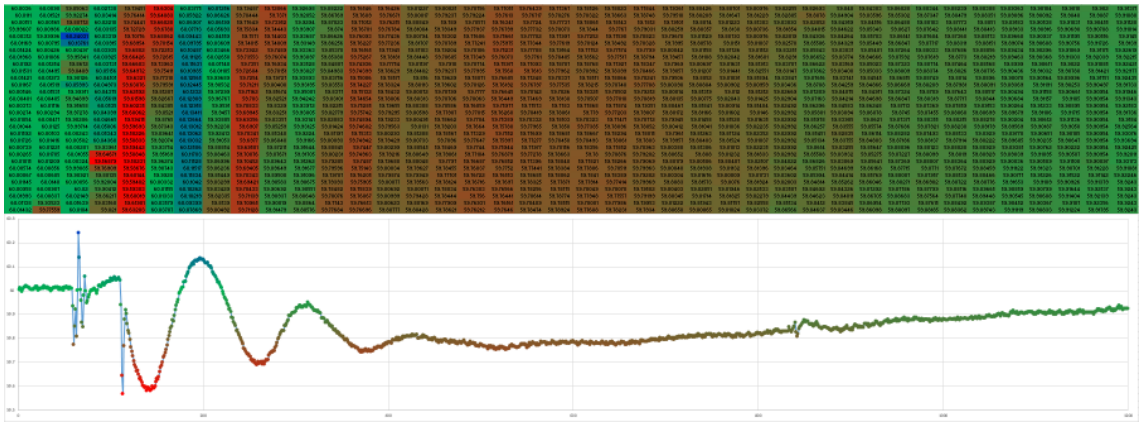

Figure 5.4: Oscillation example: the input pattern of the CNN and the original frequency variation

The CNN consists of 2 convolutional and 2 max-pooling layers alternately stacked, followed by a fully connected layer and a final logistic regression layer to carry out the classification task. The input image contains 1200 pixels (30 x 40 image). The first convolution layer performs the convolution of twenty (7x9) filters with the input image, thus creating twenty feature maps of size (30-7+1 x 40-9+1) = (24x32). Then, the pooling layer does a (2x2) max-pooling over each feature map re-shaping them to a (12x16) size. The first convolutional-pooling filtering is completed. This 2-step process is again repeated: the twenty (12x16) feature maps are now convoluted with fifty (7x9) filters resulting in frames of size (6x8); the subsequent sub-sampling/max-pooling layer leads to fifty feature maps of size (3x4). The output of this second procedure is a 4D tensor of shape (mini-batch size, number of feature maps, filter height, filter width) = (mini-batch size, 50, 3, 4). Since the fully-connected layer operates only on 2D matrices of shape (mini-batch size, number of pixels), the 4D output tensor is flattened to (mini-batch size, 50 * 3 * 4) = (mini-batch size, 600).

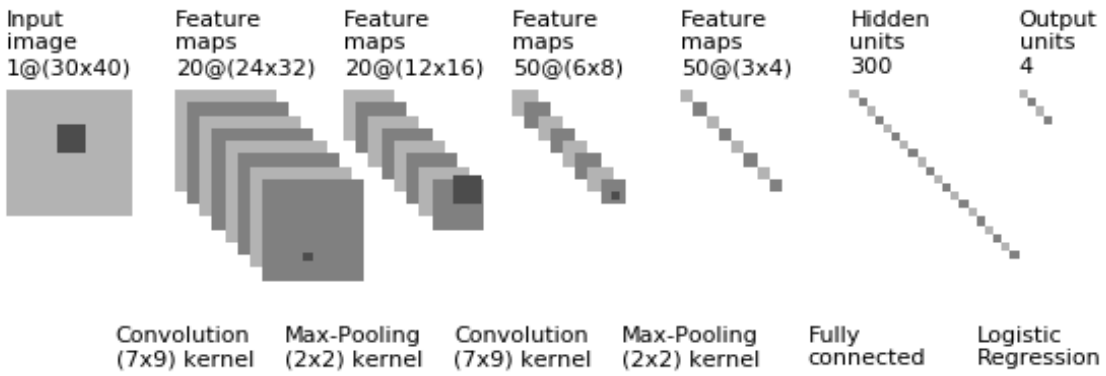

Figure 5.5: CNN designed for performing classification in 30x40 images (adapted from [4])

The hidden layer has 300 neurons and the logistic regression layer contains the number of units corresponding to the number of events to distinguish. In summary, the procedure is as follows:

1. Convolutional layer with 20 (7x9) filters → creating 20 (24x32) feature maps;
2. Max-Pooling layer of (2x2) → re-shaping for obtaining 20 (12x16) feature maps;
3. Convolutional layer with 50 (7x9) filters → creating 50 (6x8) feature maps;
4. Max-Pooling layer of (2x2) → re-shaping for 50 (3x4) feature maps;
5. Fully-connected layer of 300 hidden neurons.
6. Output layer of one neuron per class.

The same reasoning can be done for handling images of shape 20x60. Table 5.3 organizes the settings defined for both cases.

The weights of each convolutional and sub-sampling layer are initialized similarly to as with the MLP. Slight differences are worth mentioning: here, fan_{in} corresponds to the number of inputs to a hidden unit (number of input feature maps * filter height * filter width); fan_{out} is determined by the number of output feature maps and filter shape. The weights of the fully-connected layer are initialized exactly as described for the Multilayer Perceptron.

Since a logistic regression layer is used for performing the classification task, the training procedure does not differ much from the previous architectures. Indeed, the negative log-likelihood is used as loss function and the weights and biases of each layer are updated by the mini-batch stochastic gradient descent. Finally, the accuracy of both architectures is evaluated under the zero-one loss criterion.

5.6.1 Regularization Methods

The objective in training a machine learning algorithm is to obtain good generalization, i.e., to perform well on unseen data. However, a commonly observed behaviour is the model overfitting to

Table 5.3: CNN settings defined for processing the input 30x40 and 20x60 images

CNN Architecture	Settings	30x40 image	20x60 image
1st convolutional layer	no. of kernels filter shapes	20 (7, 9)	20 (5, 13)
1st sub-sampling layer	no. of kernels pool size	20 (2,2)	20 (2,2)
2nd convolutional layer	no. of kernels filter shapes	50 (7, 9)	50 (5, 13)
2nd sub-sampling layer	no. of kernels pool size	50 (2,2)	50 (2,2)
Fully-connected layer	input units hidden units activation	600 300 *tanh*	600 300 *tanh*
Logistic Regression layer	input units output units activation	300 no. of events *softmax*	300 no. of events *softmax*

the training set throughout the training procedure, losing its generalization capacity. An illustration of these phenomena is found in Figure 5.6, showing the generalization error increasing with the increase of model capacity - capacity is hereby relative to the complexity of the model, such that a very complex model is overly attached to the training data; also, the capacity increase is correspondingly connected to the course of training.

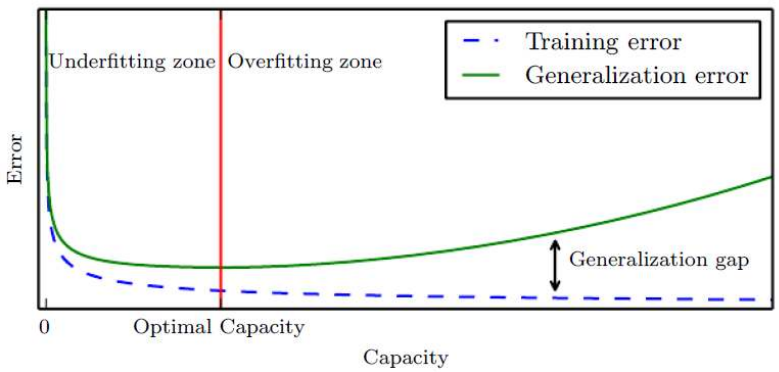

Figure 5.6: Evolution of the training and generalization errors along with training epochs [7]

These phenomena can be combated by applying the so-called regularization methods: the objective is to reduce the generalization error even if at the expense of increased training error. There are many regularization techniques, such as the addition of extra terms to the loss function (L1 and L2 Regularization), hence penalizing specific parameter configurations. A particular case of regularization is found in Early Stopping (identified as the optimal capacity in Figure 5.6),

which requires splitting the dataset into a separate Validation set, besides the Train and Test sets. Overfitting is then tackled by monitoring the performance of the model on the validation set. With the training procedure running, the parameter configuration is saved every time the validation error decreases. Conversely, if the validation error ceases to improve significantly, the heuristic implemented gives up on much further optimization and the procedure is stopped. This is based on the concept of patience, i.e. the number of iterations to wait (examples to experience) before stopping the procedure.

5.7 Introducing GPU Computing

GPU-accelerated computing, also referred to as General-Purpose Computing on Graphics Processing Units (GPGPU), regards the use of Graphics Processing Units (GPUs) along with the Central Processing Unit (CPU) to accelerate computational applications. Whilst traditional methods operate only on the CPU, the new paradigm offloads the computationally heavy portions of code to the GPU, processing the remainder on the CPU, as shown in Figure 5.7.

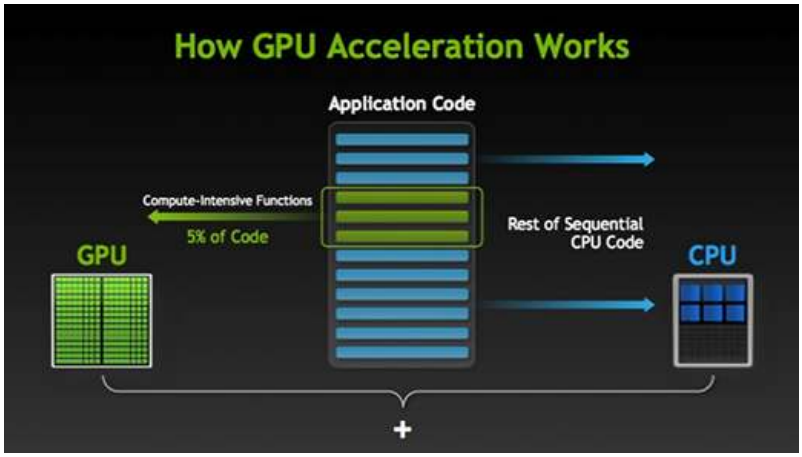

Figure 5.7: How the division of code sections is made between the GPU and the CPU [8]

In doing so, applications benefit from huge time reductions. The reason why is found on how both process their tasks: the CPU consists of few cores optimized for sequential serial processing, whereas the GPU owns a massively parallel architecture composed by thousands of smaller and more efficient cores conceived for processing multiple tasks concurrently - Figure 5.8. Thus, GPU computing is all about leveraging the parallelism in its architecture to perform mathematically compute intensive operations that the CPU is not designed to handle with ease.

So as to harness the performance of the GPU architecure, NVIDIA created a parallel programming model called CUDA that is supported by all NVIDIA GPU models. Moreover, several programming libraries have emerged or been updated to provide the ability of easy GPU computing. For instance, cuDNN is an NVIDIA library specially developed for deep neural networks

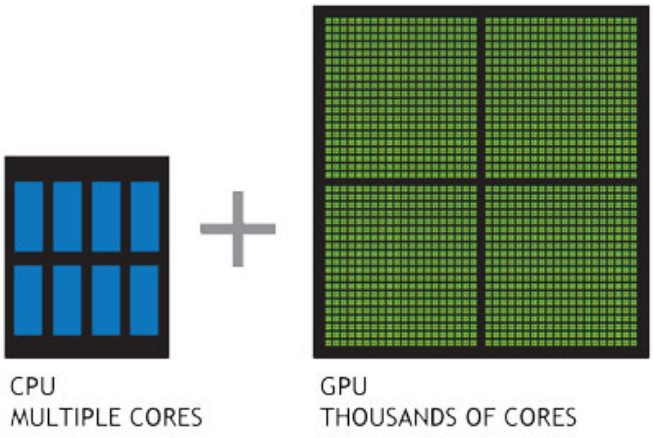

Figure 5.8: CPU versus GPU regarding the number of cores [8]

GPU-accelerated computing. These libraries can be further integrated with more generally numerical computation libraries, such as Theano that is built on top of Python and is compiled to run efficiently on either CPU or GPU architectures. A more detailed description of these concepts and the effort required for their understanding is given in Appendix A.

Overall, the combination of GPU computing and Deep Learning looks very promising. Deep Learning requires huge amounts of mathematical operations involving matrices and vectors. These are essentially multiplications that can be massively parallelized. Therefore, harnessing the properties of the GPU architecture can result in speeding up the algorithms. Additionally, as models increase in depth, more of these operations are required. Depending on the structure, the benefits in time can be proportional to the dimension of the model employed. Another interesting feature is that the resulting accuracy of training a model is not relevantly affected by the use of GPU computing libraries. Thus, in what concerns Deep Learning and specially its applications to the field of power systems, it is foreseen that the increased computational effort inherent to deeper networks can be massively reduced, without losing accuracy.

Chapter 6

Results

The following sections show the results obtained with the application of the three methodologies proposed. Firstly, some general specifications are explained considering their transverse nature. Then, the outcome of each framework is detailed.

6.1 Dataset Splitting

In Machine Learning applications, a mandatory step is to perform a proper data splitting to the dataset that is going to be used. There are several rules-of-thumb for choosing the ratio of each subset (for instance techniques such as Holdout or k-fold Cross Validation) supported by series of arguments that are out of the scope of this work. In fact, the partition hereby conducted is very common when working with three distinct sets. Nevertheless, the main reason of its choice was to match the one held in the classifier proposed in [79], since doing so allows tracing some direct comparisons that are going to be carried out in this work. Accordingly, the data was split in the sets of training, validation and test, as follows:

- Training Set - 70%: consists of the examples used in training, whose objective is to fit the parameters of the model (connection weights);
- Validation Set - 15%: its examples are used for selecting hyper-parameters of the architecture and controlling of optimization;
- Test Set - 15%: a set of examples that are used to assess the generalization capacity of the model, i.e., its performance in "unseen" examples.

Note that the objective of building these computational models is that they gain the ability to generalize well the extracted knowledge. Therefore, the accuracy of a given methodology is measured in the performance it exhibits when evaluating examples from the test set, simulating a real application in unseen data. So, it is very important to guarantee that the test examples are not used in any way that influences choices about the model. For this reason, the validation set

is considered as part of the training data, since both are used for leading the model to its best performance on the test data.

What's more, the classifier proposed in [79] demonstrated results for the distinction of only two events - Generation Tripping and Load Shedding. This dissertation can be regarded as an extension of those results, since all the possible combinations between the four events were experimented. Thus, the database presented in Table 4.1 is further divided into 11 partial datasets. The number of examples and their division is listed in the subsequent table according to each possible combination of 2, 3 or 4 events. Obviously, the distinction between 4 events corresponds to the use of the total database provided.

The nomenclature followed was: GT for Generation Tripping; LS for Load Shedding; LT for Line Tripping; OS for Oscillation.

Table 6.1: List of cases regarding each possible combination of 2, 3 and 4 events

Events to distinguish	no. examples	Training set - 70%	Validation set - 15%	Test set - 15%
GT vs LS	1297	908	195	194
GT vs LT	922	645	138	139
GT vs OS	890	623	134	133
LS vs LT	467	327	70	70
LS vs OS	435	305	65	65
LT vs OS	60	42	9	9
GT vs LS vs LT	1343	940	201	202
GT vs LS vs OS	1311	918	197	196
GT vs LT vs OS	936	655	140	141
LS vs LT vs OS	481	337	72	72
GT vs LS vs LT vs OS	1357	950	204	203

In general, all frameworks developed will be evaluated in the eleven different combinations.

6.2 Hardware Specifications

The time efficiency of the files developed is directly influenced by the specifications of the machine used for extracting the results. Therefore, it is a good practice to present the characteristics of both CPU and GPU hardware so that the time associated to each file has a base of comparison between different hardware:

- CPU: The CPU in use is a 4-core Intel (R) Xeon (R) E5-1620 V3, 10 MB Cache, 3.50 GHz with 3.60 GHz Turbo - specifications can be found in [86];
- GPU: The GPU is a 12 GB NVIDIA GeForce GTX TITAN X - specifications are available in [87].

Due to driver memory issues, the use of the GPU is limited to 95% of its maximum capacity.

6.3 Selection of Hyperparameters

The so-called hyperparameters are several settings regarding each framework that can be used for controlling the behaviour of the learning procedure. Generally speaking, these include learning rates, size of mini-batch, number of hidden units, etc., that are not appropriate to be adjusted (and sometimes cannot be) with the course of training because this would almost certainly result in overfitting to the training data. In addition, the effects that each has on the results are not independent, therefore anticipating that finding their optimum values is not a feasible problem. So, a procedure of trial-and-error was conducted to choose the hyper-parameters based on the assessment of the validation set accuracy. This is a very time-consuming step, which was based on several rules-of-thumb found on the literature and fully documented in [88].

It is worth noticing that the mini-batch size was handled separately from the other hyper-parameters due to its impacts on the efficiency of GPU computing. Therefore, the reasons supporting the choices of size made for each architecture and dataset are explained in Section 6.5.1.

Since this is merely informative, the settings are compiled and presented in Appendix B.

6.4 Classification Results

As mentioned earlier, the original dataset was replicated into the different possible combinations between the events to distinguish. Then, the architectures proposed were evaluated in all those datasets. The classification results are presented in Table 6.2. The accuracy is measured in the test set with respect to the zero-one loss function, i.e., it identifies the number of miss-classifications for each case. The corresponding percentage results are indicated in Table C.1 of Appendix C.

Table 6.2: Accuracy of each architecture developed in each event distinction

Dataset \ File	MLP 1	MLP 4	MLP 8	DBN	CNN 20x60	CNN 30x40
GT vs LS	0	0	0	0	0	0
GT vs LT	0	0	0	0	0	0
GT vs OS	1	1	1	2	0	0
LS vs LT	0	0	0	0	0	0
LS vs OS	1	1	0	1	0	0
LT vs OS	1	0	0	1	0	0
GT vs LS vs LT	2	0	0	0	0	0
GT vs LS vs OS	2	2	2	2	0	0
GT vs LT vs OS	0	0	0	0	0	0
LS vs LT vs OS	1	0	0	0	0	0
GT vs LS vs LT vs OS	3	3	3	3	1	0

Since this work shares the dataset with the classifier described in [79], an immediate comparison is inevitable. The corresponding circumstances herein are the application of the MLP with 1 hidden layer to the dataset that distinguishes Generation Tripping from Load Shedding (GT vs

LS). In those conditions, the classifier developed showed to perfectly distinguish between the two events, therefore achieving a performance equal to the one presented in Brazil [79].

Regarding the application of the three MLPs, the first thing to notice is that there are no substantial improvements with the increase of depth. This could be a symptom of the universal approximation theorem, which states that one hidden layer is enough to make the network represent a wide variety of functions, when given the appropriate parameters. Nonetheless, these improvements cannot be neglected due to the number of examples in evaluation. For instance, whilst in the case of GT vs LS vs LT classifying correctly 2 more events represented an accuracy enhancement of 0.99% (2 out of 202 examples), the improvement on classification obtained in LT vs OS led to a reduction of error in 11.11% (1 out of 9 examples) - these are results that can be consulted in Table C.1, which shows the error accuracy in terms of percentage of the number of examples in each set.

In addition, the comparison between the different MLPs needs to consider not only the gains in accuracy, but also the amount of computational burden each architecture carries. Moving from 4 hidden layers to 8 hidden layers only represented a gain of 1.54% and only in the evaluation of one combination (LS vs OS). In fact, for the distinction of all 4 events, it is clear that there are no such improvements with the increase of depth, prompting the current need for exploring different Deep Learning frameworks.

The performance of the Deep Belief Network is roughly identical to the one obtained with the 1 hidden layer MLP. With the exception of the case LS vs LT vs OS, the previous unsupervised pre-training applied to the DBN did not in any way leverage the network with respect to the basic MLP, for this specific classification task.

The Convolutional Neural Networks absolutely outperformed the remaining architectures. Both 20x60 and 30x40 frameworks show great capacity of distinguishing the several events by making use of image properties. This is inherently dependent on how the "picture" was organized: re-shaping the original vector to a matrix created new relations between "pixels" that were previously located far from each other. Since CNNs exploit local spatially correlations found in connections between nearby pixels, the new arrangement produced patterns that were easily learned by the architecture. This also supports that different shapes are predictable to result in different accuracy. However, the lack of discrepancies in the results obtained is due to the flexibility of this architecture, which led to an almost absolute correctness of classification. The exception is found in the distinction of the 4 types of events, in which only the 30x40-sized images managed to obtain 100% of correct classification. Overall, the successful application of both CNNs illustrates that the patterns hidden in data are easily learned when working with image-shaped configurations. By emulating the behaviour of the visual cortex - the most powerful visual processing system in existence - one is able to achieve better performance, at least in this recreated "image" classification.

In short, considering only the best architectures from each framework, it becomes clear that the DBN held the worst results. It was outperformed by the MLP of 8 hidden layers, which was then greatly surpassed by the CNN 30x40.

6.4.1 Classification Details

A closer look to the miss-classified cases is worth having. To perform that sort of analysis, the confusion matrices for each dataset and architecture were obtained and can be found in Section C.2 of Appendix C. The confusion matrices allow to determine how many examples were duly classified. That information can be organized into a table which reflects the predicted classes *versus* the original target classes.

Table 6.3: Predicted vs real events

Dataset \ File	MLP 1		MLP 4		MLP 8	
	predicted	real	predicted	real	predicted	real
GT vs LS	-	-	-	-	-	-
GT vs LT	-	-	-	-	-	-
GT vs OS	GT	OS	GT	OS	GT	OS
LS vs LT	-	-	-	-	-	-
LS vs OS	LS	OS	LS	OS	-	-
LT vs OS	LT	OS	-	-	-	-
GT vs LS vs LT	LT LS	GT LT	-	-	-	-
GT vs LS vs OS	GT GT	OS OS	GT GT	OS OS	GT GT	OS OS
GT vs LT vs OS	-	-	-	-	-	-
LS vs LT vs OS	LT	OS	-	-	-	-
GT vs LS vs LT vs OS	GT GT LS	OS OS OS	GT GT GT	OS OS OS	GT GT LT	OS OS OS

Dataset \ File	DBN		CNN 20x60		CNN 30x40	
	predicted	real	predicted	real	predicted	real
GT vs LS	-	-	-	-	-	-
GT vs LT	-	-	-	-	-	-
GT vs OS	GT GT	OS OS	-	-	-	-
LS vs LT	-	-	-	-	-	-
LS vs OS	LS	OS	-	-	-	-
LT vs OS	LT	OS	-	-	-	-
GT vs LS vs LT	-	-	-	-	-	-
GT vs LS vs OS	GT GT	OS OS	-	-	-	-
GT vs LT vs OS	-	-	-	-	-	-
LS vs LT vs OS	-	-	-	-	-	-
GT vs LS vs LT vs OS	GT GT GT	OS OS OS	GT	LT	-	-

The table shows an interesting feature: almost all errors result from miss-classifying an Oscillation. In fact, whenever evaluating a dataset containing OS, the errors in prediction are made in OS examples. That is, the real target was OS but the model predicted something else. A possible reason is found in the lack of examples from this particular class, which implies insufficient exposure to that kind of event in training. Therefore, the model is unable to catch representative features and fails to generalize. The exception to this subject lies in the Convolutional Neural Networks. Re-shaping the inputs led to the creation of new patterns that better identified the unique features of oscillations. Thus, both CNNs managed to classify correctly all OS examples, demonstrating

their exceptional ability to generalize well even in datasets with few examples from a given class.

6.5 The Outcome of GPU Implementation

The main outcome in employing GPU computing to Deep learning frameworks is the gain in time efficiency. The typical neural network based architectures have a myriad of parameters to adjust during the course of training. Also, their structures are composed of large numbers of identical units, thus being highly parallel by nature and foreseeing significant speed-ups over CPU-only training.

It should be mentioned that the transmission of data between the CPU and the GPU is time costly. Specially when working with mini-batch SGD, the training procedure must be optimized so as to avoid constant flux of information between both devices that can degrade the GPU-computing profits. To achieve effective time improvements, this situation was considered and handled by harnessing the Theano's library ability to work with shared variables - for a deeper understanding of the GPU-accelerated computing experience, Appendix A should be consulted. The theoretical explanation is given as follows.

6.5.1 Mini-batch Size and its Influence on GPU Computing

When applying the mini-batch SGD, a size of the mini-batch has to be selected. In addition, the time acceleration obtained with the GPU is dependent on the mini-batch size, so this choice has to be made with caution. In practice, using larger sizes in mini-batch SGD reduces the variance of the stochastic gradient updates, since it takes the average of the gradients in the mini-batch. Thus, bigger step-sizes are allowed and the optimization algorithm converges much faster in terms of total computation. It should be pointed out, however, that it requires experiencing more examples in order to reach the same accuracy because there are less updates per epoch - no. updates = no. training examples / no. examples in the mini-batch. Therefore, the number of gradient computations is actually the same: a mini-batch of size N takes step-sizes N times larger, so a single step will result in roughly the same accuracy as N steps of SGD with mini-batch size of 1.

The interesting feature is that large-minibatch parallelization with GPUs is easier to perform than using a single example per iteration. In this thesis, each dataset used was divided in mini-batches of a fixed size. So as to fully harness the capacities of GPU-computing, the dataset is stored in Theano's shared variables and entirely copied to the GPU in a single call. The GPU further accesses any minibatch by their respective index. The reasoning behind this procedure is the computational expense inherent to transmitting data from the CPU into the GPU memory. In case minibatches were copied when needed, the resulting speed-up could be almost insignificant or even negative (delaying the process) [3].

6.5.2 CPU *versus* GPU Time Results

As of today, the benefits from the employment of GPU computing into Deep Learning frameworks are related with time. Indeed, the implementation of the several architectures in both CPU and GPU produced significant time improvements. The results illustrated in Table 6.4 are referred to the speed-up obtained with the use of the GPU. Here, the speed-up is measured as the ratio between the CPU and GPU elapsed time.

Table 6.4: Speed-ups obtained with using the GPU: ratio time(CPU) / time(GPU)

Dataset \ File	MLP 1	MLP 4	MLP 8	DBN (pre-train)	DBN (fine-tune)	CNN 20x60	CNN 30x40
GT vs LS	3.4	4.3	6.9	8.4	6.2	21.1	24.2
GT vs LT	3.1	4.0	6.8	9.1	6.2	20.2	23.3
GT vs OS	2.6	3.0	8.6	9.0	6.4	19.7	22.4
LS vs LT	3.0	3.6	7.4	9.0	5.5	26.2	27.3
LS vs OS	3.4	3.5	6.4	8.2	5.3	23.5	21.2
LT vs OS	1.5	2.0	5.0	4.0	6.8	9.8	9.6
GT vs LS vs LT	3.2	3.6	6.8	9.5	6.3	23.9	27.1
GT vs LS vs OS	3.5	4.1	7.2	8.7	6.6	23.6	22.5
GT vs LT vs OS	3.1	4.0	6.4	9.0	6.2	16.8	20.3
LS vs LT vs OS	3.2	3.7	6.7	7.8	5.9	21.0	21.9
GT vs LS vs LT vs OS	3.6	4.1	8.1	8.8	6.0	31.6	27.2

Comparisons between the several results obtained must be handled with care. Different architectures should be evaluated in the same dataset. Sometimes, this comparison gets biased due to the choice of different hyper-parameters for each model. Even in the same dataset, the regularization method of early stopping implies that the total number of epochs is different between the several architectures. Yet, it remains the same between the CPU and the GPU. Consequently, presenting the time elapsed for each architecture in each dataset does not make sense.

An alternative was found in determining the ratio between the time elapsed when running on the CPU and the GPU - time(CPU) / time(GPU). In doing so, a relative measure is obtained, vanishing the influence that the number of epochs would have. Such a procedure could also be understood as a measure of relative speeds: if the speed in epochs/time for both GPU and CPU is determined, then its ratio is exactly the same as the one obtained from the elapsed times. So, both inform how much faster the algorithm ran on the GPU. Comparisons between different frameworks in the same dataset are now possible, enabling to depict some general assumptions.

As introduced earlier in Section 5.7, the inherent computations of ANNs in general fit well with the GPU architecture. Indeed, ANNs are typically structured in such a manner that at each layer, hundreds of units perform the same computation. Also, the underlying updates for training an ANN are basically large matrix manipulations, which are highly parallelizable. Therefore, the resulting gains in speed for the MLPs are somewhat expected: for every dataset in evaluation, the bigger acceleration is found in the MLP with 8 hidden layers, followed by the MLP with 4 hidden

layers and then the MLP with 1 hidden layer. Whilst increasing the depth of the model implies more computational time (due to the increased amount of mathematical operations conducted), the space for parallelization grows. That is, the MLP 8 takes longer to converge in both CPU and GPU but the ratio between those times is roughly two times bigger than the one of MLP 4. Comparing the MLP 4 and the MLP 1, the difference is a lot tightened. Thus, the increase of parallelization with the increase of depth is more noticeable when dealing with more complex models.

In what concerns the Deep Belief Networks, it is seen that the process of Contrastive Divergence (pre-training) was more subject to parallelization than the mini-batch Stochastic Gradient Descent (fine-tuning). When conditioned to one layer, the elements in the other layer (considering an individual RBM) are independent. Therefore, the Gibbs sampling can be held for all units in parallel, which represents a significant time reduction for the Contrastive Divergence when the process is conducted on the GPU.

The greatest speed-ups are obtained in the Convolutional Neural Networks. The explanation relies on their working principle, since they process the inputs as images (grid-like topology), which is the perfect environment for GPUs. The convolutional layer comprises a set of independent filters - for instance, 20, in the first convolutional layer of the architecture employed - which are independently convolved with the input image. Then, the sub-sampling layer operates in each feature map also independently. So, this procedure can be massively parallelized by harnessing the GPU architecture and running these operations at the same time. This is a remarkable feature that produced speed-ups ranging from 20 to 30 times. Indeed, for a long time, CNNs were hindered by the amount of computation time they required, being considered almost unfeasible. The advances in CPU computing aroused the interest in this framework, but it was the throughput obtained with the GPUs that established their worldwide success.

Chapter 7

Conclusions and Future Work

This chapter states the major conclusions resulting from the research conducted and presents some recommendations for upgrading the work developed.

7.1 Conclusions

The need to improve the monitoring and control of electric power systems led to the proliferation of PMU devices. As a result, massive amounts of data are generated and sent to control centers, which are eager to find new approaches for harnessing this type and volume of information. A new paradigm emerges, envisaging the application of new and unconventional methods, such as Deep Learning frameworks. These are capable of dealing more efficiently with such quantities of raw data than the traditional and analytical methods.

Several applications regarding PMU data were outlined. PMU-based WAMS benefit from the high sampling rate of these devices, being able to monitor dynamic behaviours that are undetectable by traditional SCADA acquisition devices. Apart from real-time employments, the availability of synchronized measurements is very important for offline purposes, such as *post-mortem* analysis. Indeed, system security depends on the accuracy of this procedure, which can be highly improved by using PMU data.

In general, this thesis focused on harnessing PMU data for *post-mortem* analysis of a system that suffered from several disturbances. Four types of events were to be classified: Generation Tripping, Load Shedding, Line Tripping and Oscillation. Due to the limited access to PMU data, the classifiers developed consider only the frequency variation as a consequence from each disturbance. To address this classification task, Deep Learning frameworks were employed. In particular, three Multilayer Perceptrons, a Deep Belief Network and two Convolutional Neural Networks. The three MLPs differ in the depth of each model. It was observed that increasing the depth of an MLP did not result in substantial accuracy improvements. In fact, the MLP with a single hidden layer showed a similar performance to the DBN. The breakthrough of this work is the application

of CNNs to PMU data, which outperformed significantly the other architectures in terms of accuracy. By simulating the properties of an image, this framework recreates patterns of the input data, thus achieving 100% of correct classifications when distinguishing between the four events.

Evaluating the miss-classifications obtained, it was verified that they all corresponded to oscillation cases, with only one exception. This resulted from the fact that the dataset provided contained very few examples of this event. The exception was found in CNNs due to the recreation of input patterns that led to the correct classification of the OS cases. This is another remarkable achievement, in the sense that this architecture is able to perform well even in non-representative classes.

Moreover, each framework was implemented on a GPU so as to leverage its massive parallel architecture. In doing so, significant time improvements were achieved. As a matter of fact, training time has been a real concern for the employment of these complex structures. However, the recent advances in computational efficiency and the usage of GPUs have propelled their wide application.

The organization of ANNs makes them highly parallelizable. Also, the results regarding the MLPs showed that the acceleration is proportional to the complexity (depth and individual units) of the model.

In terms of structure, the DBNs are quite similar to the MLPs, therefore prone to run efficiently on the GPU. However, training the DBN is a more complex task owing to its pre-training and fine-tuning phases. In fact, it was the former (Contrastive Divergence method) that held the most significant speed-up.

In CNNs, the convolution operation performed in a feature map requires significant computational resources. Since each feature map is independent, this procedure can be massively parallelized on the GPU. The same applies for the pooling operation. Such a structure implied that this was the architecture in which the acceleration was larger.

Overall, the deep learning frameworks have proved to be valuable tools for harnessing data from PMUs. Their scalability adapts very well to such volumes of data, fulfilling a current demand from power system operators. The speed-ups obtained with the use of GPU computing (up to 30 times in the case of CNNs) transports the training of deep networks into the realm of on-line use, which is remarkable. This allows the algorithms to be trained and re-trained in real time, instead of the traditional off-line training, prospecting their adoption in system control centers. Indeed, not only they will be able to perform real-time guidance for operators and trigger system responses, but also continuously learn new features from the real-time observation of events.

7.2 Future Work

Since this is an on-going research subject, several ideas can be mentioned as an improvement to the work developed. One can also speculate over other ideas related to this work but not necessarily as complementary developments. In general, the author proposes the following advances:

- Extension of the dataset: if continuing the labelling process manually seems unsustainable, some sort of data augmentation could be performed, specially for reproducing more examples of Line Tripping and Oscillation events. In addition, with the use of techniques of that nature, an entire training set could be created, the validation and test accuracy being evaluated in real events;

- Consider additional types of disturbances or detail the existing ones. For instance, identify the root cause of a Load Shedding event;

- The malfunctioning of the device itself or the respective communication system may imply corrupted or missing data. In order to ascertain its impact in this type of algorithms, an application regarding (Stacked) Denoising Autoencoders could be employed;

- Explore the temporal dependencies on data with the employment of spatio-temporal methodologies, such as the DeSTIN architecture described in Section 3.3.5;

- Move from an offline to an online environment: apply Deep Learning techniques to perform preventive recognition of upcoming events;

- The classification task conducted in this work is mainly local. That is, the classifier is fed with one PMU at a time, outputting the expected class of that specific event. A classifier can be implemented in every local with a PMU installed. By collecting the outputs of each classifier, an estimation of event location could be made;

- As an alternative to the previous suggestion, a global classifier could be developed. In this case it would receive the input data from all the PMUs in the system, at the same time. This was, in fact, conceived but not possible to implement due to the limitations of the dataset available: each event is only represented by the affected PMUs. For the possibility of such an implementation, each event must be characterized by all the PMUs present in the system, so that a global picture can be taken. It is worth mentioning that an architecture of that size will lead to significant increases in computational time, so this development would need to be combined with GPU computing.

Appendix A

Python as a Deep Learning Tool

Python is one of the most employed languages for developing Deep Learning applications. It is a high-level programming language, whose design emphasizes code readability. Python regards extensive math libraries that can be used in conjunction with third-party libraries such as Numpy, providing the basic tools for its applicability to scientific purposes that involve numerical data processing and manipulation. Then, specialized libraries were developed for machine learning purposes. For instance, both Theano and TensorFlow provide automatic differentiation and easy GPU implementation, proving to be extremely valuable Deep Learning tools. Additional libraries are built on top of these for increased user friendliness. The most common are Keras and Lasagne, with the first wrapping both Theano and TensorFlow and the latter running exclusively on top of Theano.

Part of the time spent exploring the concepts inherent to this work was dedicated to learn how to program in Python and fully explore its libraries. In particular, the specific deep learning features were developed with the use of Theano. It was favoured over Keras/Lasagne for reasons of availability of documentation and flexibility of implementation. At its very core, Theano allows the definition, optimization and evaluation of mathematical expressions, particularly the ones having multi-dimensional arrays. However, one of the main reasons for using Theano lies in its ease of GPU computing. Note that the main workload in machine learning models is the training procedure. With the use of Theano shared variables, the mini-batch SGD can be effectively offloaded to the GPU as described in section 6.5.1. Also, the automatic differentiation ability is of great value for this type of gradient-based optimizers.

Apart from installing all GPU related drivers, one has to write the code in a way to be interchangeably run either on the CPU or the GPU. Then, a file named .theanorc is created, in which several flags define the properties to be used whenever running the program. That is, the device can be chosen (device=cpu or device=gpu) and, in case of using the GPU, the amount of memory allocated can also be defined (95% - cnem=0.95)

Overall, the incursion in Python and all the idiosyncrasies of Deep Learning represented a challenge that was successfully overtaken, in part due to the employment of Theano, that provided an enhanced understanding of the GPU computing methods.

Appendix B

Architecture Settings

This chapter appends the several hyper-parameters chosen for each architecture.

B.1 Multilayer Perceptron

B.1.1 MLP with 1 hidden layer

Table B.1: Hyper-parameters defined for the application of the 1 hidden layer MLP

Events to distinguish	Learning Rate	Mini-batch Size	No of hidden units
GT vs LS	0.1	25	500
GT vs LT	0.1	25	500
GT vs OS	0.1	10	500
LS vs LT	0.1	25	500
LS vs OS	0.1	25	500
LT vs OS	0.1	3	500
GT vs LS vs LT	0.1	25	500
GT vs LS vs OS	0.1	25	500
GT vs LT vs OS	0.1	25	500
LS vs LT vs OS	0.1	25	500
GT vs LS vs LT vs OS	0.1	10	1000

B.1.2 MLP with 4 hidden layer

Table B.2: Hyper-parameters defined for the application of the 4 hidden layer MLP

Events to distinguish	Learning Rate	Mini-batch Size	No of hidden units
GT vs LS	0.1	25	500 I 200 I 100 I 50
GT vs LT	0.1	25	500 I 200 I 100 I 50
GT vs OS	0.1	10	500 I 200 I 100 I 50
LS vs LT	0.1	25	500 I 200 I 100 I 50
LS vs OS	0.1	25	500 I 200 I 100 I 50
LT vs OS	0.1	3	500 I 200 I 100 I 50
GT vs LS vs LT	0.1	25	500 I 200 I 100 I 50
GT vs LS vs OS	0.1	25	500 I 200 I 100 I 50
GT vs LT vs OS	0.1	25	500 I 200 I 100 I 50
LS vs LT vs OS	0.1	25	500 I 200 I 100 I 50
GT vs LS vs LT vs OS	0.1	25	500 I 200 I 100 I 50

B.1.3 MLP with 8 hidden layer

Table B.3: Hyper-parameters defined for the application of the 8 hidden layer MLP

Events to distinguish	Learning Rate	Mini-batch Size	No of hidden units
GT vs LS	0.1	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
GT vs LT	0.1	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
GT vs OS	0.1	10	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
LS vs LT	0.1	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
LS vs OS	0.1	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
LT vs OS	0.1	3	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
GT vs LS vs LT	0.2	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
GT vs LS vs OS	0.01	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
GT vs LT vs OS	0.1	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
LS vs LT vs OS	0.01	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20
GT vs LS vs LT vs OS	0.01	25	1200 I 1000 I 800 I 500 I 300 I 200 I 100 I 20

B.2 Deep Belief Network

Table B.4: Hyper-parameters defined for the application of the Deep Belief Network

Events to distinguish	Pre-training Learning Rate	Fine-tuning Learning Rate	Mini-batch Size	No. of hidden units
GT vs LS	0.000001	0.1	25	800 ǀ 800 ǀ 800
GT vs LT	0.000001	0.1	25	800 ǀ 800 ǀ 800
GT vs OS	0.000001	0.1	10	800 ǀ 800 ǀ 800
LS vs LT	0.000001	0.1	25	800 ǀ 800 ǀ 800
LS vs OS	0.000001	0.1	25	800 ǀ 800 ǀ 800
LT vs OS	0.000001	0.1	3	800 ǀ 800 ǀ 800
GT vs LS vs LT	0.000001	0.1	25	800 ǀ 800 ǀ 800
GT vs LS vs OS	0.000001	0.1	25	800 ǀ 800 ǀ 800
GT vs LT vs OS	0.000001	0.1	25	800 ǀ 800 ǀ 800
LS vs LT vs OS	0.000001	0.1	25	800 ǀ 800 ǀ 800
GT vs LS vs LT vs OS	0.000001	0.1	25	800 ǀ 800 ǀ 800

B.3 Convolutional Neural Network

B.3.1 20x60 case

Table B.5: Hyper-parameters defined for the application of the 20x60 Convolutional Neural Network

Events to distinguish	Learning Rate	Mini-batch Size	No. of hidden units
GT vs LS	0.1	25	200
GT vs LT	0.1	25	200
GT vs OS	0.1	25	200
LS vs LT	0.1	35	500
LS vs OS	0.1	25	200
LT vs OS	0.1	6	200
GT vs LS vs LT	0.1	35	500
GT vs LS vs OS	0.1	25	200
GT vs LT vs OS	0.1	15	500
LS vs LT vs OS	0.1	25	300
GT vs LS vs LT vs OS	0.1	45	300

B.3.2 30x40 case

Table B.6: Hyper-parameters defined for the application of the 30x40 Convolutional Neural Network

Events to distinguish	Learning Rate	Mini-batch Size	No. of hidden units
GT vs LS	0.1	25	200
GT vs LT	0.1	25	200
GT vs OS	0.1	25	200
LS vs LT	0.1	35	500
LS vs OS	0.1	25	200
LT vs OS	0.1	6	200
GT vs LS vs LT	0.1	35	500
GT vs LS vs OS	0.1	25	200
GT vs LT vs OS	0.1	15	500
LS vs LT vs OS	0.1	25	300
GT vs LS vs LT vs OS	0.1	30	300

Appendix C

Ancillary Results

This chapter appends several information to complement the understanding of the results obtained with the application of the different methodologies.

C.1 Accuracy as a Percentage of the Test Set

Table C.1: Accuracy obtained as a percentage of the total examples in each dataset

Dataset \ File	MLP 1	MLP 4	MLP 8	DBN	CNN 20x60	CNN 30x40
GT vs LS	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)
GT vs LT	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)
GT vs OS	0.75 % (1)	0.75 % (1)	0.75 % (1)	1.50 % (2)	0.00 % (0)	0.00 % (0)
LS vs LT	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)
LS vs OS	1.54 % (1)	1.54 % (1)	0.00 % (0)	1.54 % (1)	0.00 % (0)	0.00 % (0)
LT vs OS	11.11% (1)	0.00 % (0)	0.00 % (0)	11.11% (1)	0.00 % (0)	0.00 % (0)
GT vs LS vs LT	0.99 % (2)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)
GT vs LS vs OS	1.02 % (2)	1.02 % (2)	1.02 % (2)	1.02 % (2)	0.00 % (0)	0.00 % (0)
GT vs LT vs OS	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)
LS vs LT vs OS	1.39 % (1)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)	0.00 % (0)
GT vs LS vs LT vs OS	1.48 % (3)	1.48 % (3)	1.48 % (3)	1.48 % (3)	0.49 % (1)	0.00 % (0)

C.2 Confusion Matrices

The following sections present the confusion matrices relative to the test set obtained with the application of each architecture to every combination possible. The first section presents the number of examples in each class for each dataset evaluated, that is, the expected confusion matrix in case the architecture classifies every example correctly. Then, for each architecture, it is shown the confusion matrix for each dataset with classification errors.

C.2.1 Expected Confusion Matrices

Table C.2: Number of examples of each class in the dataset GT vs LS

GT vs LS	GT	LS
GT	135	0
LS	0	59

Table C.3: Number of examples of each class in the dataset GT vs LT

GT vs LT	GT	LT
GT	132	0
LT	0	7

Table C.4: Number of examples of each class in the dataset GT vs OS

GT vs OS	GT	OS
GT	129	0
OS	0	4

Table C.5: Number of examples of each class in the dataset LS vs LT

LS vs LT	LS	LT
LS	62	0
LT	0	8

Table C.6: Number of examples of each class in the dataset LS vs OS

LS vs OS	LS	OS
LS	62	0
OS	0	3

Table C.7: Number of examples of each class in the dataset LT vs OS

LT vs OS	LT	OS
LT	5	0
OS	0	4

Table C.8: Number of examples of each class in the dataset GT vs LS vs LT

GT vs LS vs LT	GT	LS	LT
GT	135	0	0
LS	0	60	0
LT	0	0	7

Table C.9: Number of examples of each class in the dataset GT vs LS vs OS

GT vs LS vs OS	GT	LS	OS
GT	136	0	0
LS	0	58	0
OS	0	0	2

Table C.10: Number of examples of each class in the dataset GT vs LT vs OS

GT vs LT vs OS	GT	LT	OS
GT	130	0	0
LT	0	8	0
OS	0	0	3

Table C.11: Number of examples of each class in the dataset LS vs LT vs OS

LS vs LT vs OS	LS	LT	OS
LS	64	0	0
LT	0	6	0
OS	0	0	2

Table C.12: Number of examples of each class in the dataset GT vs LS vs LT vs OS

GT vs LS vs LT vs OS	GT	LS	LT	OS
GT	140	0	0	0
LS	0	53	0	0
LT	0	0	6	0
OS	0	0	0	4

C.2.2 MLP with 1 hidden layer

Table C.13: Confusion matrix for the MLP of 1 hidden layer applied to the dataset GT vs OS

GT vs OS	GT	OS
GT	129	0
OS	1	3

Table C.14: Confusion matrix for the MLP of 1 hidden layer applied to the dataset LS vs OS

LS vs OS	LS	OS
LS	62	0
OS	1	2

Table C.15: Confusion matrix for the MLP of 1 hidden layer applied to the dataset LT vs OS

LT vs OS	LT	OS
LT	5	0
OS	1	3

Table C.16: Confusion matrix for the MLP of 1 hidden layer applied to the dataset GT vs LS vs LT

GT vs LS vs LT	GT	LS	LT
GT	134	0	1
LS	0	60	0
LT	0	1	7

Table C.17: Confusion matrix for the MLP of 1 hidden layer applied to the dataset GT vs LS vs OS

GT vs LS vs OS	GT	LS	OS
GT	136	0	0
LS	0	58	0
OS	2	0	0

Table C.18: Confusion matrix for the MLP of 1 hidden layer applied to the dataset LS vs LT vs OS

LS vs LT vs OS	LS	LT	OS
LS	64	0	0
LT	0	6	0
OS	0	1	1

Table C.19: Confusion matrix for the MLP of 1 hidden layer applied to the dataset GT vs LS vs LT vs OS

GT vs LS vs LT vs OS	GT	LS	LT	OS
GT	140	0	0	0
LS	0	53	0	0
LT	0	0	6	0
OS	2	1	0	1

C.2.3 MLP with 4 hidden layers

Table C.20: Confusion matrix for the MLP of 4 hidden layer applied to the dataset GT vs OS

GT vs OS	GT	OS
GT	129	0
OS	1	3

Table C.21: Confusion matrix for the MLP of 4 hidden layer applied to the dataset LS vs OS

LS vs OS	LS	OS
LS	62	0
OS	1	2

Table C.22: Confusion matrix for the MLP of 4 hidden layer applied to the dataset GT vs LS vs OS

GT vs LS vs OS	GT	LS	OS
GT	136	0	0
LS	0	58	0
OS	2	0	0

Table C.23: Confusion matrix for the MLP of 4 hidden layer applied to the dataset GT vs LS vs LT vs OS

GT vs LS vs LT vs OS	GT	LS	LT	OS
GT	140	0	0	0
LS	0	53	0	0
LT	0	0	6	0
OS	3	0	0	1

C.2.4 MLP with 8 hidden layers

Table C.24: Confusion matrix for the MLP of 8 hidden layer applied to the dataset GT vs OS

GT vs OS	GT	OS
GT	129	0
OS	1	3

Table C.25: Confusion matrix for the MLP of 8 hidden layer applied to the dataset GT vs LS vs OS

GT vs LS vs OS	GT	LS	OS
GT	136	0	0
LS	0	58	0
OS	2	0	0

Table C.26: Confusion matrix for the MLP of 8 hidden layer applied to the dataset GT vs LS vs LT vs OS

GT vs LS vs LT vs OS	GT	LS	LT	OS
GT	140	0	0	0
LS	0	53	0	0
LT	0	0	6	0
OS	2	0	1	1

C.2.5 DBN

Table C.27: Confusion matrix for the DBN applied to the dataset GT vs OS

GT vs OS	GT	OS
GT	129	0
OS	2	2

Table C.28: Confusion matrix for the DBN applied to the dataset LS vs OS

LS vs OS	LS	OS
LS	62	0
OS	1	2

Table C.29: Confusion matrix for the DBN applied to the dataset LT vs OS

LT vs OS	LT	OS
LT	5	0
OS	1	3

Table C.30: Confusion matrix for the DBN applied to the dataset GT vs LS vs OS

GT vs LS vs OS	GT	LS	OS
GT	136	0	0
LS	0	58	0
OS	2	0	0

Table C.31: Confusion matrix for the DBN applied to the dataset GT vs LS vs LT vs OS

GT vs LS vs LT vs OS	GT	LS	LT	OS
GT	140	0	0	0
LS	0	53	0	0
LT	0	0	6	0
OS	3	0	0	1

C.2.6 CNN 20x60

Table C.32: Confusion matrix for the CNN 20x60 applied to the dataset GT vs LS vs LT vs OS

GT vs LS vs LT vs OS	GT	LS	LT	OS
GT	140	0	0	0
LS	0	53	0	0
LT	1	0	5	0
OS	3	0	0	1

Appendix D

Brazillian Medfasee BT Project - Complementary Information

In order to fully understand the charts presented in chapter 4 and provide complementary information regarding the Medfasee BT Project, the meaning of each location with a PMU installed is given:

- UFAC - Universidade Federal do Acre;
- UFAM - Universidade Federal do Amazonas
- UFBA - Universidade Federal da Bahia;
- UFC - Universidade Federal do Cerará;
- UFJF - Universidade Federal de Juíz de Fora;
- UFMA - Universidade Federal do Maranhão;
- UFMG - Universidade Federal de Minas Gerais;
- UFMS - Universidade Federal de Mato Grosso do Sul;
- UFMT - Universidade Federal de Mato Grosso;
- UFPA - Universidade Federal do Pará;
- UFPE - Universidade Federal de Pernambuco;
- UFRGS - Universidade Federal do Rio Grande do Sul;
- UFRJ/COPPE - Universidade Federal do Rio de Janeiro;
- UFSC - Universidade Federal Santa Catarina;
- UFT - Universidade Federal do Tocantins;

- UNB - Universidade de Brasília;
- UNIFAP - Universidade Federal do Amapá;
- UNIFEI - Universidade Federal de Itajubá;
- UNIPAMPA - Universidade Federal do Pampa;
- UNIR - Universidade Federal de Rondônia;
- USP-SC - Universidade de São Carlos, Campus de São Carlos;
- UTFPR - Universidade Tecnológica Federal do Paraná;

References

[1] A.G. Phadke and J.S. Thorp. *Synchronized Phasor Measurements and their Applications*. 2008. doi:10.1007/978-0-387-76537-2.

[2] A. Ng. Deep Learning. URL: http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf.

[3] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, and Sameep Tandon. *Deep Learning Tutorial - Release 0.1*. Sep. 2015.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition". *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998. doi:10.1109/5.726791.

[5] S. Young, A. Davis, A. Mishtal, and I. Arel. "Hierarchical spatiotemporal feature extraction using recurrent online clustering". *Pattern Recognition Letters*, 37(1):115–123, Feb. 2014. doi:10.1016/j.patrec.2013.07.013.

[6] R. Leandro, T. Jeremias, I. Decker, A. Silva, and M. Agostini. "Monitoramento on-line de oscilações eletromecânicas no SIN utilizando sincrofasores". *XIII SEPOPE - Simpósio de Especialistas em Planejamento da Operação e Expansão Elétrica*, May 2014.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[8] "GPU vs CPU? What is GPU Computing? | NVIDIA". URL: http://www.nvidia.com/object/what-is-gpu-computing.html.

[9] A.G. Phadke, M. Ibrahim, and T. Hlibka. "Fundamental basis for distance relaying with symmetrical components". *IEEE Transactions on Power Apparatus and Systems*, 96(2):635–646, March 1977. doi:10.1109/T-PAS.1977.32375.

[10] A. G. Phadke, J. Thorp, and M. Adamiak. "A New Measurement Technique for Tracking Voltage Phasors, Local System Frequency, and Rate of Change of Frequency". *IEEE Transactions on Power Apparatus and Systems*, PAS-102(5):1025–1038, May 1983. doi:10.1109/TPAS.1983.318043.

[11] V Zimmer, T. Jeremias, I. Decker, and M. Agostini. "Detecção de perturbações sistémicas usando dados de medição fasorial sincronizada". *XIX Congresso Brasileiro de Automática*, pages 4180–4187, 2012.

[12] A. M. Almutairi and J. V. Milanovic. "Comparison of different methods for optimal placement of PMUs". *2009 IEEE Bucharest PowerTech*, pages 1–6, June 2009. doi:10.1109/PTC.2009.5281976.

[13] M. Shahraeini, M. S. Ghazizadeh, and M. H. Javidi. "Co-optimal placement of measurement devices and their related communication infrastructure in wide area measurement systems". *IEEE Transactions on Smart Grid*, 3(2):684–691, 2012. doi:10.1109/TSG.2011.2178080.

[14] M. B. Mohammadi and H. F. Hooshmand, R.and Fesharaki. "A New Approach for Optimal Placement of PMUs and Their Required Communication Infrastructure in Order to Minimize the Cost of the WAMS". *IEEE Transactions on Smart Grid*, 7(1):84–93, Jan. 2016. doi:10.1109/TSG.2015.2404855.

[15] P. Zhang. "Phasor Measurement Unit (PMU) Implementation and Applications". Technical Report 3, EPRI - Electric Power Research Institute, California, 2007.

[16] T. Hashiguchi, M. Yoshimoto, Y. Mitani, O. Saeki, and K. Tsuji. "Analysis of Oscillation Characteristics Followed by Power System Disturbance Based on Multiple Synchronized Phasor Measurements". Jan. 2015.

[17] Camilo A. Ordóñez and Mario A. Ríos. "Electromechanical Modes Identification Based on Sliding-window Data from a Wide-area Monitoring System". *Electric Power Components and Systems*, 41(13):1264–1279, Oct. 2013. doi:10.1080/15325008.2013.816982.

[18] C. A. Ordonez, , H. R. Chamorro, J. Quintero, R. Leelaruji, J. Peng, and L. Nordström. "Prony-based on-line oscillation detection with real PMU information". In *2016 IEEE Colombian Conference on Robotics and Automation*, pages 1–5. IEEE, Sep. 2016. doi:10.1109/CCRA.2016.7811401.

[19] W. Price, K. Wirgau, A. Murdoch, V. Mitsche, E. Vaahedi, and M. El-Kady. "Load modeling for power flow and transient stability computer studies". *IEEE Transactions on Power Systems*, 3(1):180–187, 1988. doi:10.1109/59.43196.

[20] J. Shi and H. Renmu. "Measurement-based load modeling - model structure". In *2003 IEEE Bologna Power Tech Conference Proceedings*, volume 2, pages 631–635. IEEE, June 2003. doi:10.1109/PTC.2003.1304621.

[21] V Vignesh, S. Chakrabarti, and S. C. Srivastava. "An experimental study on the load modelling using PMU measurements". In *2014 IEEE PES T&D Conference and Exposition*, pages 1–5. IEEE, Apr. 2014. doi:10.1109/TDC.2014.6863487.

[22] C. Reddy, S. Chakrabarti, and S. Srivastava. "Reduced network based voltage stability monitoring by using PMU measurements". In *2016 IEEE Region 10 Conference (TENCON)*, pages 762–765. IEEE, Nov. 2016. doi:10.1109/TENCON.2016.7848106.

[23] H. Shah and K. Verma. "PMU-ANN based approach for real time voltage stability monitoring". In *2016 IEEE 6th International Conference on Power Systems (ICPS)*, pages 1–5. IEEE, mar 2016. doi:10.1109/ICPES.2016.7584137.

[24] S. Picard, M. Adamiak, and V. Madani. "Fault location using PMU measurements and wide-area infrastructure". In *2015 68th Annual Conference for Protective Relay Engineers*, pages 272–277. IEEE, Mar. 2015. doi:10.1109/CPRE.2015.7102170.

[25] S. Kolluri, S. Mandal, F. Galvan, and M. Thomas. "Island formation in entergy power grid during Hurricane Gustav". In *2009 IEEE Power & Energy Society General Meeting*, pages 1–5. IEEE, July 2009. doi:10.1109/PES.2009.5275340.

[26] Z. Qin, S. Liu, and Y. Hou. "Virtual Synchroscope: a novel application of PMU for system restoration". In *2011 International Conference on Advanced Power System Automation and Protection*, pages 193–197. IEEE, Oct. 2011. doi:10.1109/APAP.2011.6180407.

[27] R. Zivanovic and C. Cairns. "Implementation of PMU technology in state estimation: an overview". In *IEEE AFRICON 1996*, volume 2, pages 1006–1011. IEEE, Sep. 1996. doi:10.1109/AFRCON.1996.563034.

[28] A. Jain and N. R. Shivakumar. "Impact of PMU in dynamic state estimation of power systems". In *2008 40th North American Power Symposium*, pages 1–8. IEEE, Sep. 2008. doi:10.1109/NAPS.2008.5307352.

[29] J. Chen and A. Abur. "Placement of PMUs to enable bad data detection in state estimation". *IEEE Transactions on Power Systems*, 21(4):1608–1615, Nov. 2006. doi:10.1109/TPWRS.2006.881149.

[30] M. Wald. "New Tools for Keeping the Lights On - The New York Times". URL: http://www.nytimes.com/2013/08/01/business/new-tools-for-keeping-the-lights-on.html?ref=science.

[31] E. Martinez. "Wide area measurement & control system in Mexico. *2008 Third International Conference on Electric Utility Deregulation and Restructuring and Power Technologies*, (Apr.):156–161, 2008. doi:10.1109/DRPT.2008.4523394.

[32] C. Lu, B. Shi, X. Wu, and H. Sun. "Advancing China's Smart Grid: Phasor Measurement Units in a Wide-Area Management System". *IEEE Power and Energy Magazine*, 13(5):60–71, Sep. 2015. doi:10.1109/MPE.2015.2432372.

[33] R. Leandro and I. Decker. "Ambiente Computacional de Análise do Desempenho Dinâmico de Sistemas Elétricos Usando Sincrofasores". *XXII SNPTEE - Seminário Nacional de Produção e Transmissão de Energia Elétrica, At Brasília*, Oct. 2013.

[34] IEEE Power & Energy Society. *C37.118.1-2011 - IEEE Standard for Synchrophasor Measurements for Power Systems*. Dec. 2011. doi:10.1109/IEEESTD.2011.6111219.

[35] B. Yang, J. Yamazaki, N. Saito, Y. Kokai, and D. Xie. "Big data analytic empowered grid applications - Is PMU a big data issue?". *International Conference on the European Energy Market, EEM*, Aug. 2015. doi:10.1109/EEM.2015.7216718.

[36] D. Nguyen, R. Barella, S. A. Wallace, X. Zhao, and X. Liang. "Smart grid line event classification using supervised learning over PMU data streams". *6th International Green and Sustainable Computing Conference*, Dec. 2015. doi:10.1109/IGCC.2015.7393695.

[37] Y. Hu and D. Novosel. "Challenges in Implementing a Large-Scale PMU System". In *2006 International Conference on Power System Technology, PowerCon2006*, pages 1–7. IEEE, Oct. 2006. doi:10.1109/ICPST.2006.321829.

[38] Y. LeCun, Y. Bengio, and G. E. Hinton. "Deep learning". *Nature*, 521(7553):436–444, May 2015. doi:10.1038/nature14539.

[39] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". *Nature*, 323(6088):533–536, Oct. 1986. doi:10.1038/323533a0.

[40] M. Hashim, M. Osman, M. Ibrahim, A. Abidin, and M. Mahmud. Single-ended fault location for transmission lines using traveling wave and multilayer perceptron network. In *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 522–527. IEEE, 2016. doi:10.1109/ICCSCE.2016.7893632.

[41] E. Koley, S. Shukla, S. Ghosh, and D. Mohanta. Protection scheme for power transmission lines based on SVM and ANN considering the presence of non-linear loads. *IET Generation, Transmission & Distribution*, 11(9):2333–2341, June 2017. doi:10.1049/iet-gtd.2016.1802.

[42] A. Venkatasami. Application of neural networks for transformer fault diagnosis. *2012 IEEE 10th International Conference on the Properties and Applications of Dielectric Materials*, pages 3–6, July 2012. doi:10.1109/ICPADM.2012.6318975.

[43] M. Ali, A. Siddique, and S. Mehfuz. Artificial Neural Networks Based incipient fault diagnosis for Power Transformers. *2015 Annual IEEE India Conference (INDICON)*, pages 1–6, Dec. 2015. doi:10.1109/INDICON.2015.7443174.

[44] C. Bulac, I. Tristiu, A. Mandis, and L. Toma. On-line power systems voltage stability monitoring using artificial neural networks. In *2015 9th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pages 622–625. IEEE, May 2015. doi:10.1109/ATEE.2015.7133884.

[45] K. Kandananond. Forecasting Electricity Demand in Thailand with an Artificial Neural Network Approach. *Energies*, 4(12):1246–1257, Aug. 2011. doi:10.3390/en4081246.

[46] D. Ortiz-Arroyo, M. Skov, and Q. Huynh. Accurate Electricity Load Forecasting with Artificial Neural Networks. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 1, pages 94–99. IEEE. doi:10.1109/CIMCA.2005.1631248.

[47] K.P.M. Madhugeeth and H.L. Premaratna. Forecasting Power Demand Using Artificial Neural Networks For Sri Lankan Electricity Power System. *2008 IEEE Region 10 and the Third international Conference on Industrial and Information Systems*, pages 1–6, Dec. 2008. doi:10.1109/ICIINFS.2008.4798394.

[48] D. Lee and R. Baldick. Short-term Wind Power ensemble Prediction based on Gaussian processes and Neural Networks. *IEEE Transactions on Smart Grid*, 5(1):501–510, Jan. 2014. doi:10.1109/TSG.2013.2280649.

[49] S. Li, P. Wang, and L. Goel. Wind Power Forecasting Using Neural Network Ensembles with Feature Selection. *IEEE Transactions on Sustainable Energy*, 6(4):1447–1456, Oct. 2015. doi:10.1109/TSTE.2015.2441747.

[50] V. Singh, B. Ravindra, and V. Vijay. Forecasting of 5MW Solar Photovoltaic Power Plant Generation Using Generalized Neural Network. *2015 39th National Systems Conference (NSC)*, (3):1–2, Dec. 2015. doi:10.1109/NATSYS.2015.7489107.

[51] S. Watetakarn and S. Premrudeepreechacharn. Forecasting of solar irradiance for solar power plants by Artificial Neural Network. *2015 IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA)*, pages 1–5, Nov. 2016. doi:10.1109/ISGT-Asia.2015.7387180.

[52] A. Rashkovska, J. Novljan, M. Smolnikar, M. Mohorcic, and C. Fortuna. "Online short-term forecasting of Photovoltaic Energy Production". *2015 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, (Nov.):1–5, feb 2015. doi:10.1109/ISGT.2015.7131880.

[53] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders". Feb. 2008.

[54] Radford M Neal. "Connectionist learning of belief networks". *Artificial Intelligence*, 56(1):71–113, 1992. doi:10.1016/0004-3702(92)90065-6.

[55] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. "Greedy Layer-Wise Training of Deep Networks". *Advances in Neural Information Processing Systems*, 19(1):153, 2007. doi:citeulike-article-id:4640046.

[56] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". *Neural Computation*, 18(7):1527–1554, July 2006. doi:10.1162/neco.2006.18.7.1527.

[57] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun. "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition". In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, June 2007. doi:10.1109/CVPR.2007.383157.

[58] I. Sutskever and G. Hinton. "Learning Multilevel Distributed Representations for High-Dimensional Sequences". In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 548–555. PMLR, Mar. 2007.

[59] G. Taylor, G. Hinton, and S. Roweis. Modeling Human Motion Using Binary Latent Variables. In *Advances in Neural Information Processing Systems 19*, pages 1345–1352. MIT Press, 2006.

[60] G. Hinton and R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". *Science*, 313(5786):504–507, July 2006. doi:10.1126/science.1127647.

[61] J. Wan, J. Liu, G. Ren, Y. Guo, D. Yu, and Q. Hu. "Day-Ahead Prediction of Wind Speed with Deep Feature Learning". *International Journal of Pattern Recognition and Artificial Intelligence*, 30(05):1650011, June 2016. doi:10.1142/S0218001416500117.

[62] H. Wang, G. Wang, G. Li, J. Peng, and Y. Liu. "Deep belief network based deterministic and probabilistic wind speed forecasting approach". *Applied Energy*, 182:80–93, 2016. doi:10.1016/j.apenergy.2016.08.108.

[63] X. Zhao, J. Wan, G. Ren, J. Liu, J. Chang, and D. Yu. "Multi-scale DBNs regression model and its application in wind speed forecasting". In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 1355–1359. IEEE, Oct. 2016. doi:10.1109/IMCEC.2016.7867434.

[64] Aleksandra Dedinec, Sonja Filiposka, Aleksandar Dedinec, and Ljupco Kocarev. "Deep belief network based electricity load forecasting: An analysis of Macedonian case". *Energy*, 115:1688–1700, 2016. doi:10.1016/j.energy.2016.07.090.

[65] A. Gensler, J Henze, B. Sick, and N. Raabe. Deep Learning for Solar Power Forecasting — An Approach Using Autoencoder and LSTM Neural Networks. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002858–002865. IEEE, Oct. 2016. doi:10.1109/SMC.2016.7844673.

[66] S. Lawrence, C. Giles, A. Tsoi, and A. Back. "Face Recognition: A Convolutional Neural Network Approach". *IEEE Transactions on Neural Networks*, 8(1):98—-113, 1997.

[67] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda. "Subject independent facial expression recognition with robust face detection using a convolutional neural network". *Neural Networks*, 16(5-6):555–559, June 2003. doi:10.1016/S0893-6080(03)00115-1.

[68] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". Sep. 2014. arXiv:1409.4842.

[69] S. Farfade, M. Saberian, and L. Li. "Multi-view Face Detection Using Deep Convolutional Neural Networks". page 19, Feb. 2015. doi:10.1145/2671188.2749408.

[70] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li. "Large-scale video classification with convolutional neural networks". In *Conference on Computer Vision and Pattern Recognition*, pages 1725–1732. IEEE, June 2014. doi:10.1109/CVPR.2014.223.

[71] K. Simonyan and A. Zisserman. "Two-Stream Convolutional Networks for Action Recognition in Videos". June 2014. arXiv:1406.2199.

[72] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. "Sequential Deep Learning for Human Action Recognition". pages 29–39. Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-25446-8_4.

[73] A. Van Den Oord, S. Dieleman, and B. Schrauwen. "Deep content-based music recommendation". In *Advances in Neural Information Processing Systems 26*, pages 2643–2651. 2013.

[74] R. Collobert and J. Weston. "A unified architecture for natural language processing". *Proceedings of the 25th International Conference on Machine learning - ICML '08*, 20(1):160–167, July 2008. doi:10.1145/1390156.1390177.

[75] Z. Zhao, G. Xu, Y. Qi, N. Liu, and T. Zhang. "Multi-patch deep features for power line insulator status classification from aerial images". In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3187–3194. IEEE, July 2016. doi:10.1109/IJCNN.2016.7727606.

[76] H. Wang, G. Li, G. Wang, J. Peng, H. Jiang, and Y. Liu. "Deep learning based ensemble approach for probabilistic wind power forecasting". *Applied Energy*, 188:56–70, 2017. doi:10.1016/j.apenergy.2016.11.111.

[77] I. Arel, D. Rose, and R. Coop. "DeSTIN: A Scalable Deep Learning Architecture with Application to High-Dimensional Robust Pattern Recognition". *AAAI Fall Symposium Series 2009*, pages 11–15, 2009.

[78] J. Owens and M. Houston. "GPU Computing". *Proceedings of the IEEE*, 96(5):879 – 899, Apr. 2008. doi:10.1109/JPROC.2008.917757.

[79] I. Decker, M. Zarzosa, and V. Zimmer. "Classificador de Eventos no SIN baseado em Redes Neurais Artificiais e Sincrofasores". *VI Simpósio Brasileiro de Sistemas Elétricos*, Jan. 2016. doi:10.20906/CPS/SBSE2016-0250.

[80] T. Xia, H. Zhang, R. Gardner, J. Bank, J. Dong, J. Zuo, Y. Liu, L. Beard, P. Hirsch, G. Zhang, and R. Dong. "Wide-area Frequency Based Event Location Estimation". In *2007 IEEE Power Engineering Society General Meeting*, pages 1–7. IEEE, June 2007. doi:10.1109/PES.2007.386018.

[81] G. Zheng. "Classification of power system disturbances based on wide-area frequency measurements". In *IEEE PES Innovative Smart Grid Technologies Conference Europe, ISGT Europe*, pages 1–4. IEEE, Jan. 2011. doi:10.1109/ISGT.2011.5759179.

[82] P. Sangitab and S. Deshmukh. "Use of Support Vector Machine, decision tree and Naive Bayesian techniques for wind speed classification". In *2011 International Conference on Power and Energy Systems, ICPS 2011*, pages 1–8. IEEE, Dec. 2011. doi:10.1109/ICPES.2011.6156687.

[83] T. Xia, H. Zhang, R. Gardner, J. Bank, J. Dong, J. Zuo, Y. Liu, L. Beard, P. Hirsch, G. Zhang, and R. Dong. "Wide-area Frequency Based Event Location Estimation". *2007 IEEE Power Engineering Society General Meeting*, pages 1–7, June 2007. doi:10.1109/PES.2007.386018.

[84] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256, 2010. doi:10.1.1.207.2059.

[85] G. Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". *Neural Computation*, 14(8):1771–1800, 2002. doi:10.1162/089976602760128018.

[86] "Intel® Xeon® Processor E5-1620 v3 (10M Cache, 3.50 GHz) Product Specifications". URL: https://ark.intel.com/products/82763/Intel-Xeon-Processor-E5-1620-v3-10M-Cache-3{_}50-GHz.

[87] "GeForce GTX TITAN X | Specifications | GeForce". URL: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications.

[88] Y. LeCun, L. Bottou, G. Orr, and K. Müller. "Efficient backprop". *Computer Science*, 7700 LECTU:9–48, 2012. doi:10.1007/978-3-642-35289-8-3.