UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Intrusion Detection in Web Systems Using Deep Learning Techniques

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:
Hidalgo Espinoza Sergio Hernán

Tutor:
PhD. Chang Tortolero Oscar Guillermo

Cotutor:
PhD. Ortega-Zamorano Francisco

Urcuquí, Agosto 2019

UNIVERSIDAD YACHAY TECH

Urcuquí, 26 de agosto de 2019

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2019-00005-AD

l la ciudad de San Miguel de Urcuquí, Provincia de Imbabura, a los 26 días del mes de agosto de 2019, a las 09:30 horas, el Aula AI-102 de la Universidad de Investigación de Tecnología Experimental Yachay y ante el Tribunal Calificador, egrado por los docentes:

Presidente Tribunal de Defensa	Dr. IZA PAREDES, CRISTHIAN , Ph.D.
Miembro No Tutor	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.

presenta el(la) señor(ita) estudiante **HIDALGO ESPINOZA, SERGIO HERNAN**, con cédula de identidad No. **0706006590**, la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA FORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, n el objeto de rendir la sustentación de su trabajo de titulación denominado: **Intrusion Detection in web systems using eep Learning techniques**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN.**

citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.

ecibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) tudiante.

eviamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y aminado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación, que integró la posición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se ifica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.	7,5
Presidente Tribunal De Defensa	Dr. IZA PAREDES, CRISTHIAN , Ph.D.	10,0
Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.	10,0

que da un promedio de: **9.1 (Nueve punto Uno)**, sobre 10 (diez), equivalente a: **APROBADO**

ra constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

DALGO ESPINOZA, SERGIO HERNAN
tudiante

IZA PAREDES, CRISTHIAN , Ph.D.
esidente Tribunal de Defensa

CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.
tor

Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Miembro No Tutor

TORRES MONTALVAN, TATIANA BEATRIZ
Secretario Ad-hoc

AUTORÍA

Yo, **SERGIO HERNÁN HIDALGO ESPINOZA**, con cédula de identidad 0706006590, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autora(a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Agosto 2019.

Sergio Hernán Hidalgo Espinoza
CI: 0706006590

AUTORIZACIÓN DE PUBLICACIÓN

Yo, SERGIO HERNÁN HIDALGO ESPINOZA, con cédula de identidad 0706006590, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior

Urcuquí, Agosto 2019.

Sergio Hernán Hidalgo Espinoza
CI: 0706006590

Acknowledgment

I would like to express the deepest gratitude to my principal advisor, Professor Oscar Chang, who has been the main mentor to me along these months, giving me the main concepts to develop and finish successfully this work. His vast experience and leadership have been the fundamental support to clarify the concepts and experiments reflected here.

I express my appreciation and thanks to Professor Francisco Ortega, who was my teacher on repeated occasions at Yachay Tech and one of the first people to make me feel familiar with one of the main topics of this work: Artificial Neural Networks, giving me a clear notion to devise and start my grade thesis. Also, he has taken the work of being my co-advisor, helping me mainly in the practical development of the thesis and correcting important written sections of this work.

Thanks to all my teachers, classmates and any people who have helped me, from the primary school until university, to progress in my academic life to acquire my current level of knowledge and capacity of researching, specially to Yachay Tech's professors.

Finally, I would like to thanks all my family because they are always supporting me in different aspects of my life, I LOVE THEM.

Sergio Hernán Hidalgo Espinoza

Abstract

Intrusion detection into computer networks is a topic tightly related to hacking attacks, becoming one of the most important issues to take account in cybersecurity since attackers remain always researching to discover new vulnerabilities to break the information security systems, thus these systems must be updated daily using the most powerful tools and techniques to perform the work of avoiding damages from hackers in an optimal way. Starting with this premise, this research focuses on the design and implementation of an intrusion detection system based on Deep Learning architectures. As a starting point, a shallow (three layers) network is trained with labelled log-in [into a computer network] data taken from the Dataset CICIDS2017. The internal behavior of this network is carefully tracked and tuned by using plotting and exploring routines until it reaches a functional peak in intrusion prediction accuracy. As a second step, an autoencoder, trained with big unlabeled data, is used as a middle processor which feeds compressed information and abstract representation to the original shallow net. It is proven that this resultant deep architecture has a better performance than any version of the shallow net alone. The resultant functional code routines, written in MATLAB, represents a re-trainable system which has been proved in real time producing good precision and fast response.

Key Words:

Artificial Neural Networks, intrusion detection, hacking attacks, Deep Learning, cybersecurity.

Resumen

La detección de intrusiones en redes de computadoras es un tema estrechamente relacionado con ataques informáticos, convirtiéndose en uno de los asuntos más importantes a tomar en cuenta en ciberseguridad, ya que los atacantes se mantienen siempre investigando para descubrir nuevas vulnerabilidades para romper los sistemas de seguridad informática, por lo tanto dichos sistemas tienen que ser actualizados día a día utilizando las herramientas y técnicas más poderosas para realizar el trabajo de evitar daños por parte de los hackers de la manera más óptima. Partiendo de esta premisa, este trabajo de investigación se enfoca en el diseño e implementación de un sistema de detección de intrusiones basado en arquitecturas Deep Learning. Como punto de partida, una red no profunda (de tres capas) es entrenada con datos de entradas [a una red de computadoras] etiquetados, tomados de la base de datos CICIDS2017. El comportamiento interno de esta red es cuidadosamente observado y calibrado usando gráficos y explorando rutinas hasta alcanzar un pico funcional en la precisión de detección de intrusiones. Como segundo paso, un autoencoder, entrenado con una gran cantidad de datos no etiquetados, es usado como un procesador intermedio el cual suple de información comprimida y representaciones abstractas a la red no profunda original. Se prueba que esta arquitectura profunda tiene un mejor rendimiento que cualquier versión de la red no profunda en solitario. Las rutinas funcionales de código resultantes, escritas en MATLAB, representan un sistema re-entrenable que puede ser probado en tiempo real produciendo una gran precisión y respuestas inmediatas.

Palabras Clave:

Redes Neuronales Artificiales, detección de intrusiones, ataques de hacking, Deep Learning, ciberseguridad.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Talking about information security leads us to talk about *hacking* attacks, security failures/warnings, threats, sensible information thief and a lot of terms that may cause scare to systems and business administrators when they are not properly trained to counteract this kind of problems.

Internet-connected networks and devices increase at incredible rates and at the same time increase the threats that imply being connected to the great world network, one of the main threats are **hacking activities** which are performed daily and hourly. The fight against the malicious hackers, hacktivists or even *script-kiddies*[1] is not easy and never stops by the fact of they (mainly the first two) are always developing and implementing new attack techniques that go from being simple warnings in the system until serious sensible information engages, leaving, in most of cases, significant losses of money.

To counteract and prevent any kind of hacking attacks, a large number of software and techniques have been developed through the years such as *firewalls*, *demilitarized zones* (DMZs) and *Intrusion Detection Systems/Intrusion Prevention Systems* (IDS/IPS), being these last two tools the main concern of this work. IDS/IPS use to be stand-alone systems that observe the traffic into a network connected to internet in real-time, comparing each single event with a labeled database and, depending on the similarity of a real-time event with a known attack [1], the system raises an alarm and warns to the respective personal for attacking situations. In some cases, the system responds against the attack by itself depending on its sophistication and degree of intelligence. The final product of these systems is typically a program which analyzes the parameters of a single event trying to label it as an attack (within a range of possible attacks) or a normal (friendly) operation. Modern anti-hacking systems incorporate advanced techniques such as ***Artificial Neural Networks***.

Artificial Neural Networks (ANNs) approaches have come gaining strength in the last years because of their incredible power to predict results in different science/researching fields such as Engineering, Medicine, Biology, Geology, Physics and Economics [2]. These useful tools are processing data structures inspired on the human brain functionality [3]: human senses receive external information, nervous system carries this information to the brain where neurons share information between them by *synapses* [4] to interpret this information and turn it into impulses immediately. ANNs perform a similar work: processing units, called *artificial neurons*, receive *input data* obtained from the outside to be **shared** between these neurons and **processed** by means of **mathematical operations** obtaining a final result [3]. Another point that human brains and ANNs share is the fact that both learn by experimenting: in case of humans, the brain catches all kind of information in the childhood stage and replicates similar

[1] This term is applied by many authors referring to people whom uses programs already developed by skilled community to perform basic hacking activities such as *sniffing* and *port scan/mapping* (see *Subsection 2.2.2* and *Subsection 2.2.6* in *Chapter 2: Theoretical Framework*).

synapses almost pragmatically performing a human activity; ANNs effectuate training stages to 'learn' about input data and then return an acceptable result when new patterns are inserted [5]. The great difference is the ability to reason in humans [6] (from a determined age, obviously), an aptitude impossible to constitute in machines at this moment; while a human is able to contemplate a reasonable answer to face a threatening and/or unknown situation, the ANN just will execute its mathematical operations systematically outputting an answer resulting from said operations, it does not matter how trained the ANN is. Although it is a mystery for the science nowadays, it could be said that the thinking ability in humans are just more advanced and adapted synapses acquired during a complex evolution process across the time.

There are a lot of ANN architectures going from simply one-single neurons structures to record few patterns, passing by various neurons linearly connected by levels to cluster inputs by its similarity, until a lot of neurons organized in fully-connected layers performing giant matrix operations between thousands of data and values to classify very similar but different patterns. Some of these implementations perform their *training* operations over and over again using the same training data with the purpose of getting an optimal training decreasing its error, each round where a complete dataset is processed is called iteration or *epoch* [7]. When more than one single process of data is repeated and/or an ANN have at more than 10 layers[2], some authors have named this approach as **"Deep Learning"** [8], [9] and [10].

Deep Learning is a relatively recent term that was born with the intention of covering a definition for any learning method in ANNs with neurons/layers-of-neurons containing information hide to our visualization applying a high level of abstraction [8]. Deep Learning is related to multi-epochs and multi-layer architectures; if it is taken the output from a random hidden layer in any epoch, the result looks like it has nothing to do with the input parameters (or with the final outputs) then a Deep Learning approach is present. Sometimes, Deep Learning architectures (and ANNs in general) are considered **'black boxes'** where a number is inserted and another one is waited for. Although the developer says how the ANN will work, operations inside it (especially in hidden layers) are so many and so large that a person would not have any clue what is really going on and the idea of tracking the results turns into an impossible task after a few operations.

The main objective of this work is to develop a particular design and implementation of an intrusion detector using a Deep Learning Architecture based on Artificial Neural Networks and train it by using a large dataset previously created by a Cybersecurity Laboratory in which they have been registered and labelled normal operations and different kinds of hacking attacks by hostile intruders. Finally, after the ANN is successfully trained, hacking intrusions will be automatically detected by taking randomly, from the dataset, events never before evaluated by the ANN, which will process each event individually and returns a statically correct answer about the transaction nature (friend or foe). The ANN's accuracy will be obtained by observing its ability to classify every single event as benign or attack, from which it is performed an analysis of false positives and false negatives. It is expected a percentage greater than 95% in successfully classified events. In this text are documented all theoretical definitions and mathematical issues necessary to put in context the implemented programs and performed experiments as well as the respective results, analysis and conclusions.

[2]This number is not strictly defined, it depends on the author, like any other feature of an ANN implementation.

Chapter 2

Theoretical Framework

In consideration of the non-existence of a complete consensus about some terminology used for this work, this chapter goes about the explanation, in a technical and an easy way at the same time, of the most of important definitions to understand this work properly. These definitions go from hacking/hackers, kind of hacking attacks, passing by IDS/IPS and finally clarifying terms used for the practice part of this work such as Artificial Neural Network structures, the Backpropagation Algorithm and autoencoders.

2.1 Hacking and Information Security Terminology

Hacking jargon and related terminology have been used from immemorial time by people, media and popular culture, but, in general, these terms are applied wrong and distorted. This section is intended to explain, in the most understandable way, some terms that may be a confusion to some readers. Below is presented the definition of some of these terms:

2.1.1 Hacker

This term does not have a strict definition (being this is one of the main reasons for confusing) and it will depend on the author and applications. Also, this concept has been changing along to the time.

According to Pandey [11] and his colleagues the original definition of a hacker is:
"A person who enjoys learning the details of computer systems and how to stretch their capabilities-as opposed to most users of computers, who prefer to learn only the minimum amount necessary. One who programs enthusiastically or who enjoys programming rather than just theorizing about programming."

From this point of view, the hacker definition is kind of passive and even it is referred to helpful people, who enjoyed use the intelligence, common sense and natural skills to "look around" to please their curiosity and fun-oriented challenges. In fact, the first hacking attacks were benign and used just to look for limitations/bugs into the system. Some years later, when computers and internet spread to home-people leaving the laboratories and business exclusiveness, the definition of this community was changing and increasing. In some cases, files were broken causing damages onto the systems given by hackers with lack of preparation or umbrage when they were discovered and system administrators thrown them out. The second case is the reason why both, the hackers' purposes and thus its definition start to take a different course [11].

Driven by wicked objectives, some people begging to break security systems illegally to perform intentional damages or information theft. Some of them were vengeful people who were fired out from

his work, people interested on money stealing from banks and business, credit cards or any kind of lucrative items and classified information, leaving catastrophic amounts of monetary losses to business and natural people. To differentiate them from any kind of benign hackers it is necessary to use the term *malicious hackers* [12]. Some authors call them *crackers* [13]. On the other hand, there exist professional and academically well-skilled people who perform *penetration tests*[3], fight against malicious hackers and defend computer systems using their professional preparation and well-developed techniques/tools. They are called *ethical hackers*, people with permission to intrude into the systems in a legal way to locate any kind of vulnerability for the correspondent patching [11].

Then, to avoid any distortion and confusion this concept will be divided into two main groups: **malicious hackers** (black hat) and **ethical hackers** (white hat). Any hacker could use all levels of knowledge, reasoning, tools and skills to operate, what sets them apart is the pursued objectives and, the legality and conferred permissions on them to perform the hacking activities. The first group is in which this research will focuses on to fulfill its purposes and they will be referred as 'hackers' from this point.

Another class of hackers are the **gray hat hackers**, whom are over the line between black and white, acting illegally like a black hat hacker but with benign and useful intentions in the middle as a white hat hacker. *Hacktivist* is the noun assigned to hackers whom have political motivations to exploit vulnerabilities of governments and important enterprises. In general, their objectives are focused on getting sensitive information for showing to the world the obtained information without any profit [14], they are considered black/gray hat hackers, depending on the situation. A notable current case is the *WikiLeaks*[4] organization with *Julian Assange* as one of the most important activists whom have hacked and filter classified information from important world-power countries and business.

2.1.2 Hacking

Given and cleared the hacker definition above, it is necessary to remove the idea that hacking are only illicit activities but is any performing of computer skills to achieve a determined objective. The Pandey [11] description that says

"*the term hacking was used to describe the rapid crafting of a new program or the making of changes to existing, usually complicated software.*"

is compatible with his own description of 'hacker' then compatible with the description redacted here (at the start of the paragraph). Nevertheless, for the interests of this work, it is used the term 'hacking' to refer to any activity that compromise the integral security of a computer system.

2.1.3 Penetration Testing

Penetration Test, also referred by its trade name '*pentesting*' for short, is the term used professionally by ethical hackers to describe, briefly, the process of researching, attacking[5] and patching potential security vulnerabilities into a determined system. To perform optimal pentestings, professionals use to apply the same malicious attack techniques and dedicated software developed by security business [11].

[3]See *Subsection 2.1.3: Penetration Testing.*

[4]Wikileaks official site: `https://wikileaks.org/`.

[5]The term 'attack' is used explicitly for malicious actions, being the reason for which professionals have included the term 'pentesting' in their lexicon. In this case, the verb 'attack' means perform an activity to test a vulnerability into a computer system.

2.1.4 The CIA Triad

It is a definition usually used in Information Security to describe three concepts as the most important to keep data safe to the fullest. These concepts are **Confidentiality**, **Integrity** and **Availability**. **Confidentiality** is assuring that information is known just by sender and receiver that have special credentials to gain access to this information; to get this objective it is necessary the use of techniques to encrypt and decrypt data (*cryptography*) using combinations of alphanumerical characters, called keys, to keep data hidden to third parties independently of the used channel to transmit the information. When the sender and the receiver uses two different private keys mixed with a different public key to encrypt and decrypt the information respectively, the system is called *asymmetric* or *public key* cryptography. On the other hand, when the two parts use the same key, it will always be private and the cryptographic implementation is called *symmetrical* or *private key*. Confidentiality also involves the information saved on servers/computers which not necessarily will be sent to a destination. **Integrity** refers that data is received as it was sent, without any change nor being corrupted in the way; there exist techniques such as *hashing* to make sure data has not been affected before arriving at the destination. Keeping stored data integral is important too. The last concept is **Availability**, which means that legitimate users are able to access data whenever they need it, it is in general achieved by using different methods that include physical assurance and constant maintenance of the equipment. [15] These three terms must be applied together to constitute appropriately the CIA triad; as you may have noticed the first letter of each term articulate this abbreviation.

2.2 Kind of Hacking Attacks

This section covers some of the most harmful, known and launched attacks: *DoS/DDoS*, *sniffing*, *Man-in-the-Middle*, *phishing*, *SQL injection*, even *viruses* may be considered hacking attacks[6] depending on the use, but this is not the target of this study then it is not needed to focus on them.

2.2.1 DoS (Denial of Service)

These attacks are those which to be executed affect directly the availability of a network connection, disabling specific required services by the users and dropping those temporary. Depending on the scope [of the attack], some DoS attacks affect the system files integrity, destroying them or causing irreversible damage. These attacks are performed making thousands or millions of requests to a specific service on a computers network. When the attack is performed **concurrently** by several devices is called **DDoS** that means **Distributed Denial of Service**. Whether or not the service is actually affected by the attack depends on its capacity of requests at the same time and the number of requests sent by the attacker. Then, in both cases if the service is enough potent or the attack is not well-prepared, the system will not have problem with this kind of attacks, in the opposite case, the service will drop. Nowadays, some servers are even provided with tools to detect these attacks and avoid them. If a system does not count with these tools and it is already affected by a DoS/DDoS attack, the solution is to detect the attackers to eject them from the system and restart the service. When the problem persists it is necessary to add the affecting addresses to a black-list to prohibit them from entering at the moment and in the future. In general, the targets are websites or web-servers. Affecting the site availability could cause serious money losses since a lot of people could require the services at the moment, services which people are paying

[6] Viruses may perform any kind of hacking activities and their power just depends on the script of code programmed to be executed once the virus explodes.

for using [16]. For example, if the web-site of a store is disrupted by this attack, until the system is recovered, people will not be able to buy and will search for another store leaving monetary losses to the attacked store.

2.2.2 Sniffing

Or **eavesdropping attack** is the action of **intercepting** the traffic for **catching** continuously the packets on *TCP ports*[7] into a network [18]. For this attacks, it is used specialized software, such as *Wireshark*[8] and *Ettercap*[9] which require waiting for until the desired information is caught or looking for it among all captured information once the attack is stopped. To execute the attack the sniffer must be connected to the network. Because the TCP flow is accessed and able by all connected devices [17], there is no way to avoid a sniffing attack by a connected part. Then, the consequences for the affected network are given by the level of packets encryption in the traffic: when the packets are strong-encrypted, the sniffer is able to see the traffic but it may be an impossible task to decipher the packets; in cases that the encryption be weak, the use of basic decryption software/techniques will be enough to get the authentic information; and, finally, if directly does not exist an encryption mechanism, the sniffing techniques will get the data easily and even files can be caught and downloaded by the sniffer [19], since files are interactive-transferred via *FTP* (Files Transfer Protocol) which is connected via TCP ports [17].

2.2.3 Man in the Middle (MITM)

MITM attack is the scenario where a third part gets control over data flow between two users (end-points) who are interchanging information through a network channel, without the two users noticing the attack. In general, this kind of attacks are performed in the next way: the attacker intercepts the victims' messages and keys, ciphering and deciphering the messages whatever the attacker wants. MITM attackers generally do not interrupt the data flow, convincing the victims that they are into a secure environment free of hackers. The MITM attacks damages are not limited only to the theft of plane information (confidentiality) but involves loss of them and a direct threat to data integrity because data can be manipulated by the attacker before to send it to the receptor and even the availability of data may be compromised if the attacker so requires [20], thus breaking totally the CIA triad. Since the attacker first must get access to the communication channel, the way to prevent this kind of attacks is avoiding the entrance of these third parts or expelling them in the case they already gained access breaking down the security tools. Another solution[10] for this problem is the use of advanced cryptographic techniques for sending and receiving the messages [20]. The term "*Man in the Middle*" comes from the basketball case when two players are trying to pass the ball among them and there exist a rival player between them trying to possess the ball [21].

2.2.4 SQL Injection

They are attacks explicitly oriented explode database security gaps being webpages the main affected targets. The objective of these attacks is retrieving the database content injecting (hence its name)

[7] TCP (Transmission Control Protocol) ports are the via to create communications paths to transfer integral and complete information between devices into a network [17].

[8] Wireshark official web: https://www.wireshark.org/

[9] Ettercap official web: https://www.ettercap-project.org/

[10] It would be more like a **contingency plan** against these attacks because the implementation of this solution assumes that the MITM attacker have already gained access into the communication channel, but at the same time, the implementation of a strong encryption system must be mandatory into any communication system to avoid any issue related to any kind of attack.

or introducing **"malicious"**[11] **commands** into a database program to change the behavior of a *SQL query* and then the database behavior. In general, the SQL injection techniques exploit the incorrect implementation of input/output data validation, for example one of the first points that an attacker tests is the response of an incorrect[12] user/password validation. The consequences of these attacks go from a simple illicit access to sniff into a system until the theft of high valuable information from the affected databases. To avoid this kind of attacks, it may be used *Graphical User Interfaces* (GUI's) with text-boxes that limit the range of characters to be used into them, for example removing the option to use the equal symbol (=) which is one of the main symbols at the moment to execute a SQL query and so is the case with other special symbols. Another way (that would be stronger putting it together the first one) to do not become a victim of these attacks is showing correct messages in the GUI's and perform clear responses by the database to face any kind of event [22].

2.2.5 Brute-Force Attacks

Brute-force attacks occur when an attacker uses systematical and automatic tools/programs to get access into a system which needs user/password authentication, basically stealing the authentication credentials of an existent user. These techniques consist of trying every random combination of characters to gain access to a system. When pre-combined strings of characters are used, such as proper names or well-known typical combinations, the attack is called **Dictionary attack** [23], existing all kind of dictionaries for all kind of systems, even there exist software that helps with this work, being *The Harvester*[13] one of the best examples. Once a valid user/password is caught, the attacker will have access to the system using that credentials until these are modified or removed from the system. The dangerous of this attack lies in the fact that licit credentials are used to access into the system then there is no way to know when credentials have been stolen and an attacker is using it to gain access. Several techniques are used to avoid this kind of attacks, for example, the limitation of tries for each device every so often, since Brute-Force attacks generally need a lot of tries to discover valid credentials and, having a black list for users with anomaly behavior. Social networks such as Facebook and Twitter and, services providers such as Google have implemented advising via e-mail or cellphone messages each time a new device is trying to get access with your account, giving the option to terminate the sessions and change the password at the moment. *Secure Shield Protocol* (SSH), FTP and *Telnet* (remote access) are the services mainly attacked by using Brute-Force techniques [24].

2.2.6 Port Scan/Mapping

This activity consists on ***footprinting*** and **scanning** a network, device or service looking for vulnerabilities into the attacked system deduced from information gathered during the scan by sending packets to retrieve another packets. In general, these attacks look for open ports to get a *back door* where to enter into the system and initiate a determined attack. If the scan is performed in a coarse way, without targeting a determined port and launching packets continuously it is called **Brute Force Scan**. In the opposite case, when a full-connection is not established, single packet with a particular *flag*[14] is sent and, the scan is dedicated to a determined port, the scan is known as **Stealth Scan** [25]. Since Port Scans

[11] There not exist malicious commands per se, what turn a command into a malicious one is the way in which they are injected and executed and, the objectives for what they are used.

[12] Incorrect not only in the sense that what happens when a non-existent user is trying to get access but what happens if it is inserted, into the text-box, a random string using any symbol or what if it is inserted a SQL query command.

[13] The Harvester GitHub repository: `https://github.com/laramies/theHarvester`.

[14] The flag is a portion of the packet that will retrieve specific information to be analyzed, depending on that response it is concluded if the port is open or not [25].

are one of the first phases of a Penetration Testing [26] it can be understood that their dangerousness lies in the fact that once a vulnerable port is discovered by attacker, it can be exploded with other kind of attacks [25] that could bring more catastrophic consequences. Nevertheless, a scan is not always considered as an attack as such, because it may be performed easily by script-kiddies using basic tools like *Metasploit Framework*[15] without representing a real threat.

2.2.7 Phishing

They are the kind of attacks where some of the *social engineering* is applied, it means the **human factor** is involved too [27] in the sense of it is necessary that **the victim "collaborates" with the attacker**. Phishing attacks are in general performed online via webpages or e-mails; the attacker cheats the victim using false webpages, e-mails via re-direction links that look like confidential and trustful sources. The aim of these attacks is stealing personal information [27] given meekly by the victim, causing significant damages and losses depending on the case. For example, the classical case in which an advertisements banner into a website advises that the user has won a free gift and then it link to a website asking for credit card data to reclaim the suppose gift; it is obvious the only gift the victim receives is an indebtedness with the bank caused by the theft of the credit card information. The way to avoid becoming a phishing victim is installing programs/browser-plugins to detect phishing-webpages patterns to advise to the user about it [28]. Major of banks and financial business recommend checking the veracity of the online services accessed by the users just following easily distinguishable features.

2.2.8 Bots

Bots are programs, computers or computers networks working automatically to perform a determined activity. Some bots are used to propagate itself into a computer network for its later execution [24] to steal and/or damage information. Some bots are made to perform multiple entrances into a system with the aim of cause overflow, send spam [29] or to make a post/topic a trend in social networks although these tools are used for benign and useful tasks too, for example sending automatic reminders, news and updates [30]. To protect a system against bots, they are used sophisticated solutions to detect them based on the behavior of a connected device [30]. Another practice solution is the use of advanced *CAPTCHA's* (Completely Automated Public Turing Tests to Tell Computers and Humans Apart) to detect bots and forbid them for the entrance.

2.3 IDS and IPS

IDS and IPS are the abbreviations that correspond to **Intrusion Detection System** and **Intrusion Prevention System** respectively, being them a set of computer techniques and software/programs used by computer systems for defense purposes. The objectives of these tools are to detect anomalies, log events and report to security managers about malicious activities [1]. Depending on the implemented technology these systems could proceed in an automatic way, deploying activities such as *session termination* that means expelling the suspicious user or, in critical cases, forcing to terminate sessions for all users and reboot the system. It can be done by using *dynamic reconfiguration* (re-configuring networks parts/devices such as firewalls or blocking the potential source/destination attack) or *attack content manipulation* (removing any infected file that allows the malicious connection) [31].

[15] Metasploit official web: https://www.metasploit.com/.

Even though IDS and IPS implementations can be merged and used as a single tool called **Intrusion Detection/Prevention Systems** (IDPS) [31], each one of them has particularities and differences and can be used separately. The main difference between them is that an IDS just detects anomalies or inappropriate entrances once the attackers are inside the system and warn about it, but an IPS can prevent these entrances and, in case these entrances occur, it is able to take actions against the attackers, for example removing the connection attacker-network in real-time. In general, IDS/IPS tools are like collector that have several parameters to recognize malicious from benign connections in a brute way, using information about previous attacks to detect a new one of them (*rule-based analysis*), without having a true learning and without increasing its level of recognizing [32]. By its attack identification techniques, IDS/IPS have been classified into two groups: **misuse** and **anomaly**. The first one technique is based on the idea of comparing the involved event with a previously assembled database with different kind of attacks classified depending on the parameters of each event. On the other hand, the anomaly technique takes as attack any unknown and abnormal activity, compares it with benign and normal activities saved on a database and gives us a classification to each event [33].

Some IDS/IPS implementations are already using Machine Learning and Artificial Neural Networks approaches [34], [35], [36], [37] and [10]. These articles will be analyzed to contrast their results with those presented in this work.

2.4 Artificial Neural Networks

An Artificial Neural Network (ANN) is a processing data architecture conformed by *artificial neurons* which take a dataset to learn about it by the action of capturing and isolating data features during a prolonged training period. After training, the network accepts inputs which are analyzed by the neurons to produce a final predictive output, quite useful for the underlying data processing system. One critical step is to study the performance of the ANN when it learns from patterns [2], previously sampled in clusters of data, to adjust their structure operating mathematically this data and, later be ready to process new patterns never seen before. ANNs are used for classification, clustering and prediction [2] and their neural connections are based on the human brain model that consists on *neural synapses* in which neurons send, receive and process electrical impulses (data) between them [3].

Each kind of neural network has its own singularities and each of these features could be borrowed by any implementation depending on the application, but in general, they have some components in common:

2.4.1 Artificial Neurons

An artificial neuron or simply a 'neuron' or 'node' is the **basic processing unit** into an ANN [38]. A neuron receives as input a value obtained from the previous *level of processing*[16], performing operations between it, the *weights*, the *activation function* and the *learning rate* to get another value[17], being this value the output of the neuron.

[16] It will be called 'level of processing' to any structure returning outputs or providing values that could be used as inputs by another processing structure (a single neuron or a layer of neurons). A level of processing could be both: a previous neuron/layer or the input data.

[17] This output number is not necessarily different than the input, depending on the applications neurons can return infinity sets of numbers as well as just two different values. For example, certain NN's return a value into the continuous range between [0 - 1], but other NN's return 0 or 1.

2.4.2 Layers of neurons

It is important to clear this concept since a lot of useful ANN's are conformed by this architecture. A layer is a set of neurons at the same level of processing in which, in general, each neuron receives each output returned from the last level of processing [7]. If the neurons of a layer receive the input data, the current layer is called **Input layer**, if the neurons receive outputs from other layer as inputs but they do not return the final output, these layer is called **Hidden layer** and, the last case, if the neurons return the final output of the ANN, the layer is called **Output layer**. There not exists a criterion to prohibit a layer being the input/output layer at the same time, in this case, it would have a one-layer ANN without hidden layers and, and there is no problem with a layer containing one single neuron, it just will depend on the implementation requirements. In fact, there not exist strict rules for the layers, each one of them may has any number of neurons, learning rate, transfer function and any other independent value.

2.4.3 Weights

Sets of numbers, generally organized in vectors/matrices. Each one of them represents the **strength** of a neuron signal [39] which 'saves' the learning between neurons or layers. These matrices are modified each time an input is processed to accommodate the network learning. Weights are generally initialized randomly in the range determined for the ANN functionality, being [0 - 1] one of the most used.

2.4.4 Learning rate (LR)

It is a constant value[18], generally between [0 - 1], that determines the **learning speed** of the ANN giving a **change-ratio** to the weights in each epoch [2]. For example, in an ANN to classify animals, if the learning rate is higher than the necessary, when recognizing a cat (for instance) from a set of different animals, the ANN would take any feline animal as if it were a cat or even worse, the ANN will consider any four-footed animal as a cat. In the opposite case, when the learning rate is very low, the ANN will separate all cats with different colors and sizes as different animals. In the ideal case, when LR is optimal, all cats will be classified as cats and the other animal species will be classified properly. For this reason, the learning rate should be correctly calibrated during the ANN *validation phase*[19].

2.4.5 Transfer/activation function

A non-linear mathematical function that processes the *synaptic potential*[20] to transforms it into the neuron's output [40]. More used functions as transfer functions are the **sigmoid**, the **arc tangent**, the **hyperbolic tangent** and **ReLU**[21] [36], functions which returns values between [-1 - 1] (excepting the last one), although these functions may be manipulated at convenience to return results in a different range[22]. In some cases, this function is two combined functions, the first one process normally the synaptic potential and the second one process the result of the first to return the neuron's output. Sometimes it is used a beta (β) value as a gradient regulator for the transfer function. For the first layer $y^{(1)}$ this function g looks like:

[18] It might be variant in some *Unsupervised ANNs*.

[19] See *Section 2.6: Artificial Neural Networks Learning Phases*.

[20] Also called 'activation level of the neuron', it is the product resultant from the operations between the neuron inputs, its weights and, the learning rate.

[21] The Rectified Linear Unit (ReLU) function defined as $f(x) = max(0, x)$ or softly approximated by $f(x) = log(1 + e^x)$ [41] returning values in the range [0 - 1].

[22] For this work, it is used a function returning results between [0 - 1], see *Chapter 4: Methodologies*.

$$y^{(1)} = g\left(W^{(1)}x + b^{(1)}\right) \tag{2.1}$$

, where
$W^{(1)}$ = the weights of the first layer,
x = the inputs,
b = the bias of the layer, which is a value that adds a *freedom degree* to the ANN.

2.4.6 Data normalization

Because most of data is parameterized in arbitrary ways and measures, it is necessary to give them a tractable and standardized shape to avoid severe fluctuations and unexpected behaviors by our ANN. Normalizing is a linear transformation of data [42], re-arranging it into a correspondent shorter range. The range of normalization applied in the experiments of this work is [0 - 1] and each parameter of the set of patterns is normalized individually.

2.4.7 Input data

It is the final data (previously refined) to be processed by the ANN obtained from a dataset. It consists on normalized data divided into *patterns* that are each one of the examples or samples taken for the ANN. Every pattern has a constant number of parameters[23] (which may be a disadvantage when using ANN implementations since not always all these data could be retrieved because not existence or fails in the used tools) which must always be passed in the same order; these parameters are inserted into the neurons for their later processing. It is worth to say that it is practically sure that variations in patterns change the ANN behavior and performance. In general, ANN's match every input neuron to every input parameter.

2.4.8 Output data

These data are the final output obtained from the NN using the aforementioned parameters, which later is treated to deduce a final result depending on the kind of ANN used. Sometimes it is applied the term 'output' just to refer the result returned by a single processing structure.

2.4.9 Epochs

To adjust the weights optimally, some ANNs perform the calculations repeated times, each one of these times is called *epoch* [7]. An epoch is a complete round of performing all calculations for all inputs in the *training* and validation phases, but it is not applied in the *testing* phase. To improve the results, the inputs used to be randomized for each different epoch. The number of epochs must be minutely selected since some ANN approaches might be *over-trained* having a critical number of epochs where learning begins to get worse giving as result an increase in the error and therefore a decrease in the performance. There not exists a fixed number of epochs to have a better accuracy of an ANN, in fact, all parameters are fixed by the developer performing a lot of executions with different values, according to what the developer thinks it is the best choice.

[23] If not the case, the pattern could be suppressed from the input data or should be defined a default value, although it is not secure since it could affect the ANN's performance, then the first option use to be the most applied.

2.5 Classification of Neural Networks

By learning methods, ANNs are divided into two large groups: **Supervised Neural Networks** and **Unsupervised Neural Networks** [7], having each one different and unique applications. Furthermore, several ANNs can be mixed to form a new one. The first one method is what this work is focused on.

2.5.1 Supervised Neural Networks

These ANNs have a **target** output which is compared with the ANN final output. Depending on the similarity between the target and the output it will be concluded the goodness of the NN. Some examples of Supervised ANNs are detailed next:

Perceptron. This is the particular case where the network is formed by a single neuron. All parameters are connected to the perceptron to be processed. This implementation carries to the **Multi-layer perceptron**: single perceptrons connected between them in different and separated sections of processing [34]. These networks are in general *fully-connected*, it means all inputs parameters are connected to all neurons.

Convolutional Neural Networks. Its name comes from the mathematical function called **convolution**, because one of its elements is the fact of reducing the number of parameters that the ANN process. Another important point of these ANNs is the extraction of hidden details in processed data [9] that could be relevant in the performance at the time of obtaining the results.

Deep Learning. Being maybe one of the most applied ANN implementations, it does not have a precise definition which nowadays depends on the author and its application. The definition applied in this work to differentiate any ANN implementation from Deep Learning will be a mix of the next concepts: the use of different ANN approaches in which there exist totally separated epochs/iterations and data representations. The fact of using different layers of neurons and data representations makes the ANN's learning deeper and more powerful in terms of processing capacities [8]. Each one of the approaches does know nothing about the others, they just share data already processed by them. In this work, two specific different kind of ANNs are used to be trained and handled separately: **1.- an *autoencoder*** to process and compress the original data for giving it as input to **2.- a classifier** consisting of on a tree-layers ANN, both using the *Backpropagation Algorithm*[24] as the training tool. Merging these two concepts, it is accomplished the final Deep Learning Architecture seen in detail in *Chapter 4: Methodology*.

2.5.2 Unsupervised Neural Networks

These NNs are generally used for clustering and they do not need a target to be compared with the output, they just return the result and the author decides how to judge the useful of the results which in general recognize patterns difficult to detect with a naked eye [7]. Another difference is that the learning rate is not fixed, it varies with an equation that changes in each iteration depending on the last calculated LR and the total number of iterations which makes the LR smaller after each iteration. A practical way to test the reliability of an Unsupervised ANN is to compare the result with another one obtained for applying another class of techniques with the same dataset. Two specific Unsupervised ANNs are:

Competitive Neural Networks. As the name implies, in these NN's, all neurons 'fight' one against the other between all of them to appropriate the input pattern, being the **winner** the physically closest neuron to the pattern and it will be influenced by the **neighbor** neurons too [43]. There exist different

[24]The definitions of 'Backpropagation Algorithm' and 'Autoencoder' are extended in *Section 2.7: Backpropagation* and *Section 2.8: Autoencoder* respectively.

kind of architectures for these NN's: not-connected where are the patterns which are changing its position and the neurons are fixed and, connected networks, where neurons are moving towards the pattern.

Self-organized maps or *Kohonen Maps*, named so by its creator Teuvo Kohonen, in 1990. In this ANN, patterns are auto-clustered not necessarily into divided groups, but they are mapped depending on the hidden and abstract similarities of their attributes [44]. Clustering countries depending on their economics, social status, levels of poverty, education, illiteracy percentage and other parameters is a nice example to apply this kind of ANN.

There exist some examples of ANNs with a kind of learning independent from these two seen above: **Hopfield Networks** are those which are trained to storage patterns and then recognize a new pattern and match it with one of the stored before, this is one of the first compact ideas and was developed by J.J. Hopfield in 1982 [45].

2.6 Artificial Neural Networks Learning Phases

To get a final Supervised ANN ready to face totally new patterns and work with them properly consists on a process given by the next stages:

2.6.1 Training

This is the initial phase. It consists on fitting the weights, performing all ANN calculations, to optimize the ANN learning. This phase is executed by epochs in which the learning is expected to improve, i.e. the error must decrease. If the improvement does not occur, then the input data used for training should be changed or being reduced the number of epochs to avoid over-training.

2.6.2 Validation

It is executed into the same epochs number than the training phase. In each training epoch, the weights are fit by validation immediately using a different dataset, generally smaller than the training dataset. The error of this phase should be lower than the training to show a real improvement, since this is the objective of the validation phase.

2.6.3 Testing

This final phase is the fireproof for the ANN. It is not executed by epochs because the objective of this step is to prove the performance of the ANN with unique weights and a new portion of the dataset, in general, larger than validation dataset and smaller than training dataset. Weights selected for this phase are those which showed the better performance in the validation phase and, they are fixed for all samples in this phase.

2.7 Backpropagation Algorithm

Backpropagation is an algorithm as old as the ANN idea. If one goes back to the eighties decade could find works explaining the theoretical framework of this algorithm. Le Cun [46] in 1988 and Hecht-Nielsen [47] in 1989 already mentioned the Backpropagation Algorithm as the most used indisputably for ANN implementations. Nowadays, this algorithm is quite used and its essence has not changed for more than 30 years of application.

The Backpropagation Algorithm is divided into two main stages: **forward** and **backward** which are executed sequentially in each epoch. In the forward process it is calculated an output for each neuron in all layers: the inputs together the weights from the first layer are processed to calculate a value called *synaptic potential*. For the rest of layers, the correspondent weights (for each layer) and the outputs of the previous layer are taken to get the synaptic potential. Each synaptic potential is processed individually by the transfer function to get the output of each layer. On the other hand, in the backward process, the targets, the ANN's outputs and their derivatives are processed in the output layer to get a value called *delta* (δ); for calculating δ in the rest of layers, the derivatives of the outputs in the next layer, the deltas in the next layer and the respective weights are handled. Finally, weights in the first layer are updated using the learning rate, the deltas for the first layer and the inputs patterns. For updating the weights in the rest of layer, the learning rate, the rest of deltas and the outputs are taken.

The ANN general error (E) is calculated using the ***Mean Square Error*** **(MSE)** equation in each epoch:

$$E = \frac{1}{2}\sum_{k=1}^{P}\sum_{i=1}^{M}\left(z_i(k) - y_i(k)\right)^2 \tag{2.2}$$

, where

P = total of epochs,

M = total of patterns,

z = the target of an i pattern,

y = the output of the ANN of an i pattern.

The accuracy is equal to

$$accuracy = \frac{well\ classified\ patterns}{total\ patterns} \tag{2.3}$$

2.8 Autoencoder

Autoencoders are data predictors divided into two fundamental parts: the **encoder** and the **decoder** [48]. The encoder contains an input layer, which takes the parameters of each pattern (one neuron per parameter) to process them along the hidden layer; since this hidden layer is (in our case) smaller than the input layer, it contains compressed parameters based on the original data. Because the information generated in the hidden layer can be used to reconstruct the initial dataset it may be considered abstract representations of the input data [49], like a **DNA** of the input patterns: **dimensionality reduction** and **data featuring extraction** of each individual pattern are being executed. The second part, the decoder, converts the DNA data of the hidden layer back into the original input data. Autoencoders are a form of Unsupervised Learning and due to its data compression and feature detection capacities are a common way of assembling Deep Architectures [48]. Learning is unsupervised because its main objective is to replicate the input data as it is, and no labelling or targeting is required [48], in fact the expected value would be the same inputs, but several iterations to encode and compress the data into the hidden layer has to be carried out.

One of the most used autoencoder is the **Sparse Autoencoder** which handle the sparsity of the prediction by using a *regularizer* ($\hat{\rho}$) in the training, which averages the output of a neuron [50]. For an individual i-neuron in the input layer, the regularizer is defined by:

$$\hat{\rho}_i = \frac{1}{n}\sum_{j=1}^{n} z_{(i)}^{1}\left(x_j\right) = \frac{1}{n}\sum_{j=1}^{n} g\left(w_i^{(1)T} x_j + b_i^{(1)}\right) \tag{2.4}$$

, where
n = the number of patterns,
x_j = the j-th pattern,
g = the transfer function,
$w_i^{(1)T}$ = the i-th row of the weights matrix,
$b_i^{(1)}$ = the i-th element of the bias vector.

Another important factor in this kind of autoencoders is the **Sparsity Regularization** which restricts the output of a hidden layer (this process is applied in hidden layers only) by adding a regularization which takes a large value when the regularizer ($\hat{\rho}$) and its expected value (ρ) are very different [50]. The *Kullback-Leibler divergence*[25] (KL) could be applied to get this regularization:

$$\Omega_{sparsity} = \sum_{i=1}^{D^{(1)}} KL\left(\rho||\hat{\rho}_i\right) = \sum_{i=1}^{D^{(1)}} \rho\ log\left(\frac{\rho}{\hat{\rho}_i}\right) + (1-\rho)\, log\left(\frac{1-\rho}{1-\hat{\rho}_i}\right) \tag{2.5}$$

, where
D = a hidden layer.

When training this kind of autoencoder, the Sparsity Regularizer may decrease, increasing the value of the weights and decreasing the value of the output. It is added the **L_2 Regularization** in the weights to avoid it [50], defined by:

$$\Omega_{weights} = \frac{1}{2}\sum_{l}^{L}\sum_{j}^{n}\sum_{i}^{k}\left(W_{ji}^{(l)}\right)^2 \tag{2.6}$$

, where
L = the number of hidden layers,
n = the number of input patterns,
k = the number of neurons in the l layer.

The last important function for these kind of autoencoders is the **Cost/Loss Function** (E) which is an adaptation of the MSE:

$$E = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(x_{kn} - \hat{x}_{kn}\right)^2}_{MSE} + \ \lambda\ * \underbrace{\Omega_{weights}}_{L_2\ regularization} + \ \beta\ * \underbrace{\Omega_{sparsity}}_{sparsity\ regularization} \tag{2.7}$$

, where
λ = the coefficient of L_2 Regularization,
β = the coefficient of Sparsity Regularization[26].

[25] This funciton returns 0 when ρ and $\hat{\rho}$ are equal or, a value which diverges depending on the difference between them.
[26] λ and β are manually handled by the developer.

Chapter 3

Related Works

This chapter is intended to analyze five different works, developed in different years, implementing Intrusion Detection Systems together Artificial Neural Network approaches. It is important to announce that in this chapter may be appear some terms that have not been explained in the last chapter, because they are never used for the purposes of this work, then, the explanation of these works tries to be the most summarized and understandable. The recommendation to the reader is to analyze in detail the articles presented in this chapter to acquire a solid interpretation of them.

3.1 Cannady (1998)

This kind of works have been studied from the last century being **Cannady** [34], in 1998, who presented his work emphasizing the importance of using systems to protect network computers to avoid loss of clients by lack of trust after being a victim of an attack. He gives some important definitions about Intrusion Detection Systems which were based on rule-based analysis or *expert systems* which takes away the idea of flexibility at the time of evaluating an individual event. In the same way, he discloses the Artificial Neural Networks implementations which would allow characterize flexibly each incoming event by learning about its features. The disadvantage that the author punctuates is the fact of the ANNs need always correct characterized data and the strictness of the network to learn properly.

His implementation consists of a Multi-Layer Perceptron with layers of neurons adapted by trial and error, using the sigmoid function described as **1 / (1 + exp(-x))**, with an output layer with two neurons that returns [0.0, 1.0] for benign events and [1.0, 0.0] when an attack is detected. He used labelled data analyzed by Real Secure™ network monitor from Internet Security Systems, Inc that includes more than 360 kinds of attacks to classify, adding the Internet Scanner™ from ISS, Inc, and the Satan Scanner. A dataset of 10000 events was collected by using these tools in which approximately 30% of them was hacking attacks. The preparation of data was divided into three sequential processes: 1.- taking the most relevant parameters of each event (data compression), 2.- converting three of these nine parameters into a single normalized numeric value and 3.- the last phase is querying all parameters of the event to create a unique ASCII comma-delimited value to introduce into the ANN, including the value 1 (for attacks) or 0 (for benign events).

Once executed his implementation he obtained a performance at **98.2%** and **97.6%** of correlation for training and testing respectively after apply the Mean Squared Error.

3.2 Sammany, Sharawi, El-Beltagy and Saro (2007)

In that year, Sammany, et al. [35] analyze seriously the consequences of lack of protection in information systems, noting the importance of a powerful Intrusion Detection System being, according to them, the rule-based approaches the most implemented. They take the two established detection techniques: misuse (different kind of known hacking attacks) and anomaly (irregular events).

For their approach, they have used the 1999 MIT Lincoln Laboratory – DARPA (Defense Advanced Research Projects Agency) Intrusion Detection Data which consists of more than 450000 individual connection events from which were taken 13000 properly selected and normalized events with 35 parameters by each one which in turn was divided into three subsets: training (10400 inputs), validation (20% of training inputs) and testing (2600 inputs). It was used a Multi-Layer Perceptron using Backpropagation algorithm with three layers: the input layer taken each pattern parameter, two hidden layers with 50 and 30 neurons and, the output layer with three neurons, since they just had three kinds of events: [1 0 0] for normal connections, [0 1 0] for Neptune attacks (DoS) and [0 0 1] for Satan attacks (Port Scan). They have chosen a high value for the learning rate between the input and hidden layers and, a small value for the second hidden and output layer.

Using this setup, they performed different tests from 100 epochs, reaching a break-point at 1000 epochs given by the increasing of *Cross Validation Error* (CVE) checked by the Mean Square Error, obtaining a maximum correct classification at **93.4%** for the testing set.

3.3 Y. Liu, S. Liu, Zhao (2017)

These authors [36] explain us that nowadays already exist intelligent implementations to get a high capacity of detection, using tools such as *Genetic Algorithms* (GA), *decision-making trees* and *fuzzy logic* which are part of flexible and robust *Soft Computing* (SC) techniques.

In this work, authors apply an ANN technique have not seen in the works before: Convolutional Neural Networks. For training and testing, they use the 10% of the KDD Cup 1999 dataset assembled by DARPA which has more four million of events with 41 parameters by each one, divided into 22 different attack denominations.

After a complex process of pre-processing data and training the network, they reached a **97.7%** of accuracy to detect attacks and a 10% of benign events classified as an attack category (false positive).

3.4 Biswas (2018)

Biswas [37] performs a comparative study of different *Machine Learning* approaches, for which they are chosen different features selectors (these algorithms will compress the input parameters by taking the most relevant characteristics of the inputs): Correlation based Feature Selection method (CFS), Principal Component Analysis (PCA), Information Gain Ratio (IGR) based feature selection and Minimum Redundancy Maximum Relevance. Likewise, they are used several classifier models, which catalogue each individual pattern with a class (normal or suspect traffic) depending on its features: Nave Bayes, Support Vector Machine, Decision Tree, Neural Network, k- nearest neighbor algorithm (k-NN). Each one of these algorithms has different characteristics and way of work. Every feature selector is tested with every classifier.

NSL-KDD (a variant of KDD99) dataset has been used that consists of approximately 10000 patterns with more than 40 parameters, for the final work they have been selected 10000 patterns. The higher

accuracy was obtained from the combination of k-NN with IGR: >**99%**.

3.5 Vinayakumar, Alazab, Soman, Poornachandran, Al-Nemrat and Venkatraman (2019)

In this work [10] the authors highlight the relevance to have a well-prepared Intrusion Detector showing some values of monetary losses caused by hacking attacks. Their application is treated by them as a Deep Learning approach (by the use of several hidden layers) that use a Multi-Layer Perceptron with a *Feed Forward* algorithm between layers and using the ReLU function as activation function, although sigmoid and hyperbolic tangent are used too in some tests. Different portions of the dataset have been selected and pre-processed to perform different tests. Also, the implementation is tested using a different number of hidden layers (from 1 to 5) in each test. After a huge batch of tests, they obtain results at values greater than **99%** of accuracy, depending on the dataset. They explain that the dataset configuration influence the results very much because some of them are not characterized in the correct way.

Well, different points of view about advances in the studied field have been acquired, it is obvious that hacking attacks evolve then requirements to counterattack them do it too, then the older works might be obsolete nowadays although they were novel and worked at their time and, in that sense, newer implementations must be trained constantly to avoid become useless along the time. It is sure that there exist a lot of researches in this topic using different approaches, but it is enough to us with the analyzed studies to have a clear idea of how to handle our own development. Now, it is time to check how effective and different our implementation is with respect to these works.

Chapter 4

Methodologies

The aim of this chapter is to provide a clear idea to the reader about the procedures followed to obtain the results and therefore achieving the planned objectives. It is going to explain the software tools selected to code our ANN structure, then the database extracting and pre-processing to be entered into the ANN. Finally, it is examined the theoretical architecture of the ANN explaining the selected parameters and mathematical equations used for it.

4.1 MATLAB

MATLAB is a mathematical processor developed by MathWorks®[27] which uses its own programming standard named **M language** for developing the scripts to be run into the software. In this case, it is used the MATLAB2015b version. ANN automatic approaches can be meet in MATLAB but it is used just for the ease that this software provides to work with large matrices and their respective operations. All commands and functions mentioned in this chapter come from MATLAB software. It is worth to say that all scripts developed by us on MATLAB[28] have been checked meticulously to avoid fake/wrong results. All results showed in this work have been documented (in this handwritten) and saved to be replicated in case revisers and/or readers require it.

4.2 Dataset

First of all, it is necessary to get a large amount of data (*Big Data*) to be processed by any deep ANN architecture. This data could be created using dedicated tools to catch the traffic of a network or download created data from Information-Security-Research Laboratories, being this second the chosen option for this research.

The used dataset was developed in 2017 by the **Canadian Institute for Cybersecurity (CIC)** from the **University of New Brunswick** and was named by them as 'Intrusion Detection Evaluation Dataset (CICIDS2017)'[29]. This dataset consists of eight different *Comma Separated Values* (CSV) files which collect network traffic events with 79 parameters by each event, being the first 78 the values with the information in real-time about the event and last one the label of the event: benign or attack. The 78 informational parameters include true real-world data (PCAPs) such as network traffic analysis with

[27] MathWorks official web: https://www.mathworks.com/.
[28] MATLAB scripts of this project are stored online in https://github.com/cheo2322/Neural-Network-for-hacking-attacks-detection.
[29] Official site for dataset download: https://www.unb.ca/cic/datasets/ids-2017.html.

CICFlowMeter, flows based on time stamp, source/destination IP's, ports and protocols. All parameters of this dataset are specified in *Appendix A*.

4.2.1 Dataset Reading and Normalization

Once the CSV files are obtained, they are able to be read and manipulated by using MATLAB. First, these files are loaded and it is created a script to read the data and then generate a *.mat* file[30] with a half of benign inputs and the other half of attack inputs. It is necessary to save this .mat file and then it is just called using the function `load()`. Random inputs have been selected to create three different input sets: the first one with 150 inputs for small tests, another one with 6000 inputs to verify a trustworthy functionality of the ANN and finally a 2000 inputs dataset to perform the final tests.

For practical issues, the normalization has been performed into the same MATLAB script where it is executed the ANN, normalizing each x column (parameter) and adopting the next equation to normalize them in the range [0 - 1]:

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

, where:
$i = 1, 2, 3, \ldots,$ number of patterns in the dataset,
$n =$ the resultant column after normalization.

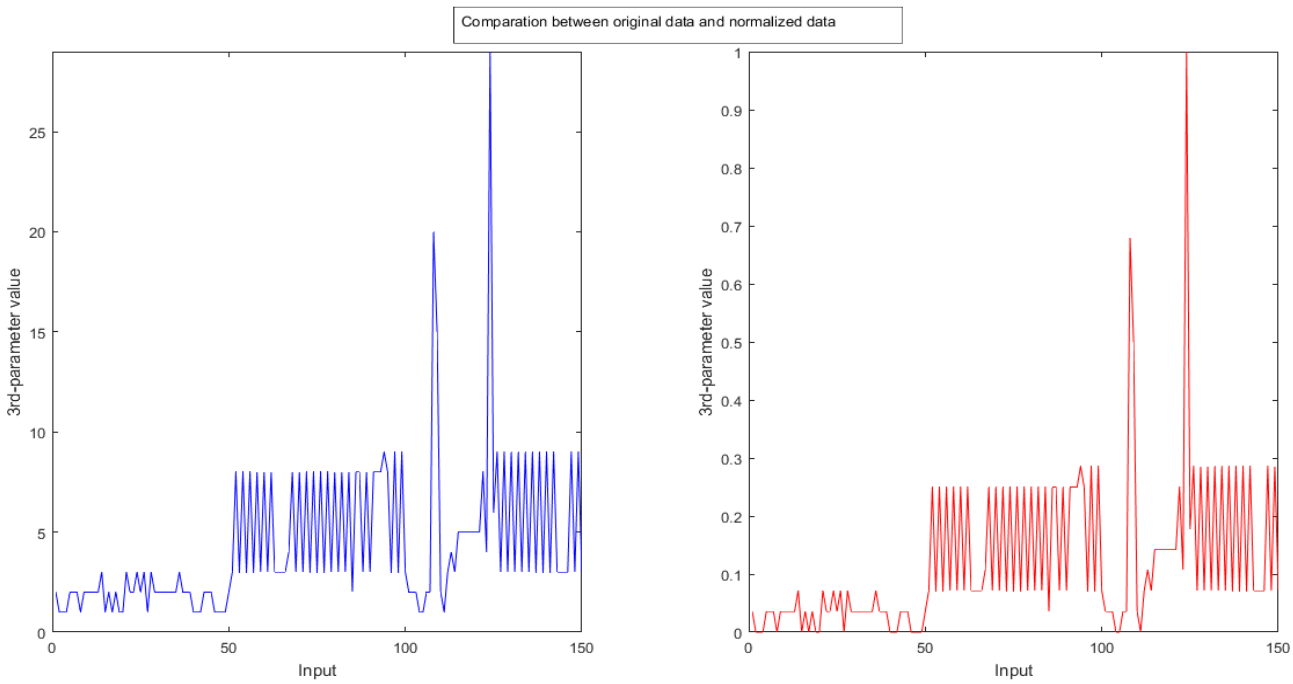

Figure 4.1: Parameter 3, left: original distribution, right: normalized distribution.

Applying the equation 4.1 in each parameter separately gives us as a result an identical distribution of data but in a new range: [0 - 1]. Figure 4.1 shows us an example of normalization in the 150 inputs dataset,

[30]The CSV files could be read directly in the ANN implementation, but the pre-processing performed in the MATLAB script to create the .mat files makes easy the data reading and guarantees freedom of errors related with data at runtime, for example *Not a Number error*, the most common in this kind of datasets.

the only difference in the parameter 3 ("Total Forwarded Packets") before and after the normalization is the range of its distribution, all inputs have a correspondent normalized value. It will be the same case for any parameter.

> ***Note 1***
> *From this point, it will be called 'original data' or 'input data' to normalized data. It does not matter the not-normalized data anymore.*

The last value (the target of the event) is the only value that is not normalized, a different process is performed for this value: since this value is a string of characters, a 0 is used to replace the benign target and a 1 otherwise. In this way, it is obtained a practical way to perform operations with this value, such as the subtraction of the output with the target.

It is important to mention that not all patterns in the dataset had all parameters correctly characterized, for this reason, a very small portion of the dataset was omitted at the time of reading the patterns and normalize them.

4.3 Deep Learning Architecture

For the experiments of this work, two neural architectures have been designed to test the learning algorithms, being the ultimate goal to learn from the data and get predictive results with solid statistical advantages. The first learning architecture is a three layers Autoencoder, which through industrious training with unlabeled data, learns to reproduce as output the data presented as input. Under controlled conditions, the hidden layer of the autoencoder capture important features of the unlabeled data and these features prove to be useful for other downstream processors. The second network is a three-layer network which never sees the real data but only neural representations provided by the hidden layer of the trained autoencoder. It is proved that this deep learning architecture can be trained to become a reliable online intrusion detector. The specifications of the mentioned architectures are shown below.

4.3.1 Backpropagation in a shallow architecture

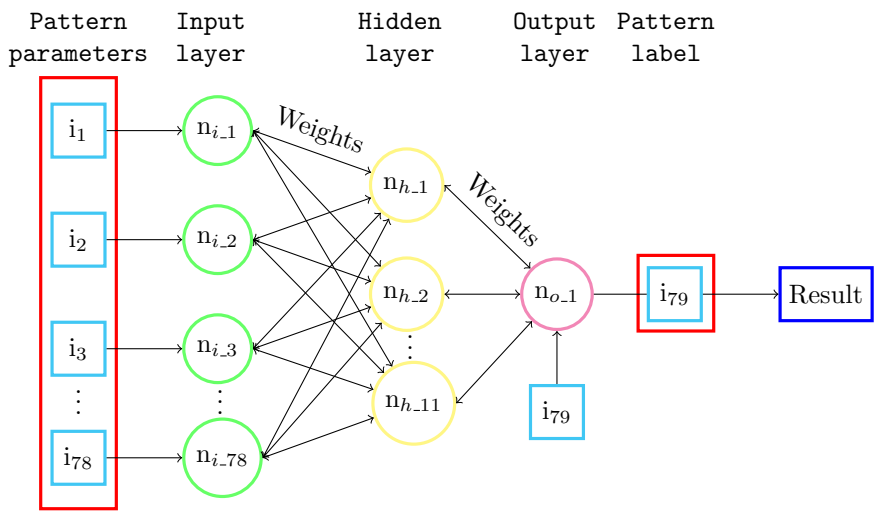

Figure 4.2: First architecture: Shallow backpropagation topology.

As a first step, a shallow architecture has been trained (shown in Figure 4.2), where the inputs are the normalized 78 parameters of the dataset and the target is the labelled information of 'benign' and 'attack' events (these parameters are bounded by a red rectangle, labelled with `i`'s), taken in a balanced way which is approximately the same amount of each label. The input layer is configured to take all input patterns, each neuron takes a different parameter; a hidden layer composed by 11 neurons and finally an output layer with a single neuron that returns 1 when an attack is detected and 0 when the ANN consider the event as benign. Then, it starts to be executed the Backpropagation algorithm performed in the input layer, the hidden layer and the output layer. Finally, the result of the output layer is compared with the label corresponding to each pattern to give us the final result shaped like error and accuracy values.

Below are described other features of this architecture:

- Learning rate: 0.1.
- $\beta = 1$.
- Initial weights for an a layer:

$$W^a_{i,j} = \texttt{rand([-1:1])}$$

  , where
  a = number of the layer,
  i = number of values returned by the last level of processing,
  j = number of neurons in an a layer.

- Transfer function g: *logistic sigmoid function*

$$g(h) = \frac{1}{1 + e^{-\beta h}} \tag{4.2}$$

  , where
  h = the synaptic potential.

  The synaptic potential is calculated using different equations depending on the needs, which are specified in equations 4.3, 4.4. Figure 4.3 shows the function plotting with $\beta = 1$.

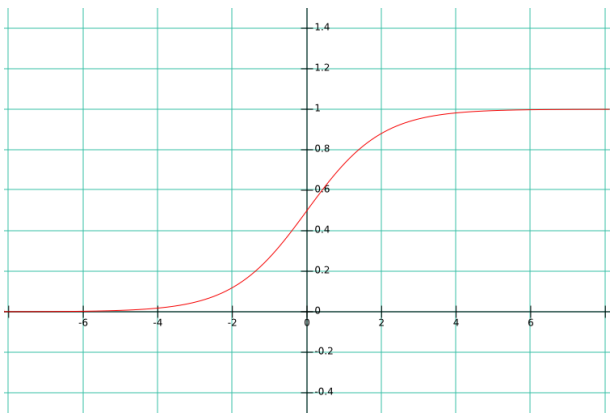

Figure 4.3: Transfer function with $\beta = 1$.

- Output neurons calculation in forward phase (S): For neurons from the first layer:

$$S^1 = g(h) = g\left(\sum_{j=1}^{L} W_{i,j}^1 \cdot X_i\right) \tag{4.3}$$

For neurons from an a layer:

$$S^a = g(h) = g\left(\sum_{j=1}^{L} W_{j,i}^a \cdot S_i^{a-1}\right) \tag{4.4}$$

, where
X = input data,
a = number of layer, $\forall a = 2, \ldots, n$,
L = number of total neurons in a,
g = equation 4.2,
h = the synaptic potential.

- Output neurons calculation in backward phase (δ): For neurons from the last layer n:

$$\delta_l^n = (S_l^n)' \cdot [Z_l - S_l^n] \tag{4.5}$$

For neurons of a b layer:

$$\delta^b = (S^b)' \cdot \left(\sum_{i=1}^{M} W_{i,j}^b \cdot \delta^{b+1}\right) \tag{4.6}$$

, where
l = the number of outputs in the final layer / target of each parameter,
Z = the targets for an l input pattern,
b = number of layer, $\forall b = 1, 2, 3, \ldots, n-1$,
S' = the derivative of S, calculated by using

$$S' = g'(h) = \frac{\beta e^{-\beta h}}{(1 + e^{-\beta h})^2} \tag{4.7}$$

- Calculation of weights update:

$$\Delta W_{i,j}^1 = \eta \cdot \delta_j^1 \cdot X_i \tag{4.8}$$

$$\Delta W_{i,j}^a = \eta \cdot \delta_j^a \cdot S_i^{a-1} \tag{4.9}$$

, where
η = learning rate.

- Calculation of testing error (E_t): It is taken the best weights (those in whose epoch it was obtained the lowest error) for all the layers and then it is applied the MSE equation in the next way:

$$E_t = \frac{1}{N}\sum_{i=1}^{N}(Y_i - Z_i)^2 \tag{4.10}$$

, where
N = the total number of input patterns in the testing set,
Y = the output of the network for a i pattern,
Z = the target of the network for a i pattern.

4.3.2 Autoencoder architecture

For this work, the implemented autoencoder trainer from MATLAB has been used with the next parameters:

- 78 neurons for the input layer (one neuron per parameter).
- 1200 iterations.
- 19 neurons for the hidden layer.
- L_2 weights regularization: $1x10^{-5}$.
- Encoder/decoder function: logistic sigmoid function (equation 4.3).

The rest of the values not specified are the default values given by MATLAB. Figure 4.4 shows the autoencoder topology:

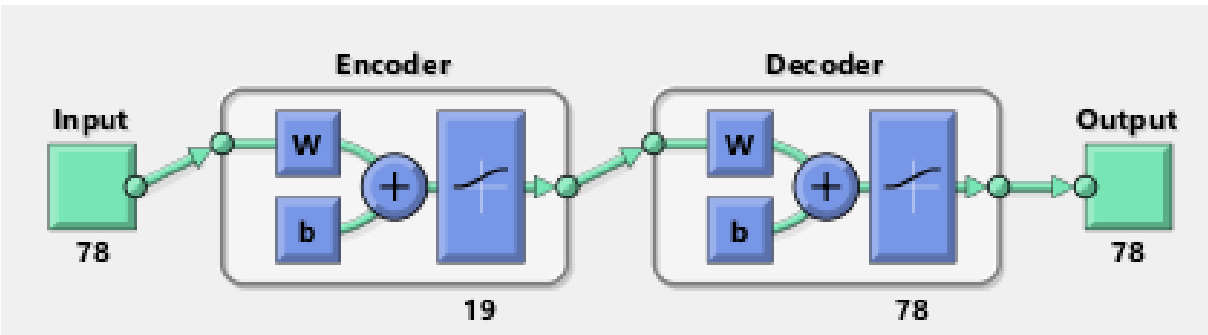

Figure 4.4: Autoencoder architecture. W = weights, b = bias. In the autoencoder 78 inputs are taken, input 79 (the label of the pattern) is not processed here, it does not matter because it is not performed a prediction of the nature of the patterns but just extracting information from their features.

4.3.3 Ultimate Deep Learning Architecture

Our final step is to train a deep network where a second shallow net learns from labeled abstract neural representations provided by the hidden layer of the autoencoder. Figure 4.5 describes the topology of this architecture: the red square contains the normalized parameters that are the inputs: the features and the label of each pattern. The orange square is the autoencoder, in which the first layer takes all parameters of each pattern (one parameter per neuron) to encode it in the hidden layer (of the encoder) which is taken as new inputs of the intrusion detector, this hidden layer extract the DNA of inputs, it is assumed that this DNA data is rich in features; the decoder is forgotten since it returns the same initial

values. The green square is the detector which takes these new inputs in a smaller layer than in the fist architecture (remember data is compressed from 78 to 19 parameters), the process is normally executed here by the backpropagation algorithm and then the outputs are compared with the target to get the errors (black square). Notice that here, the shallow network never sees the real data coming from the outside world, just the features extracted by the encoder. Finally, after performing this process for all patterns in each epoch, it is obtained the accuracy which means the final result (blue square).

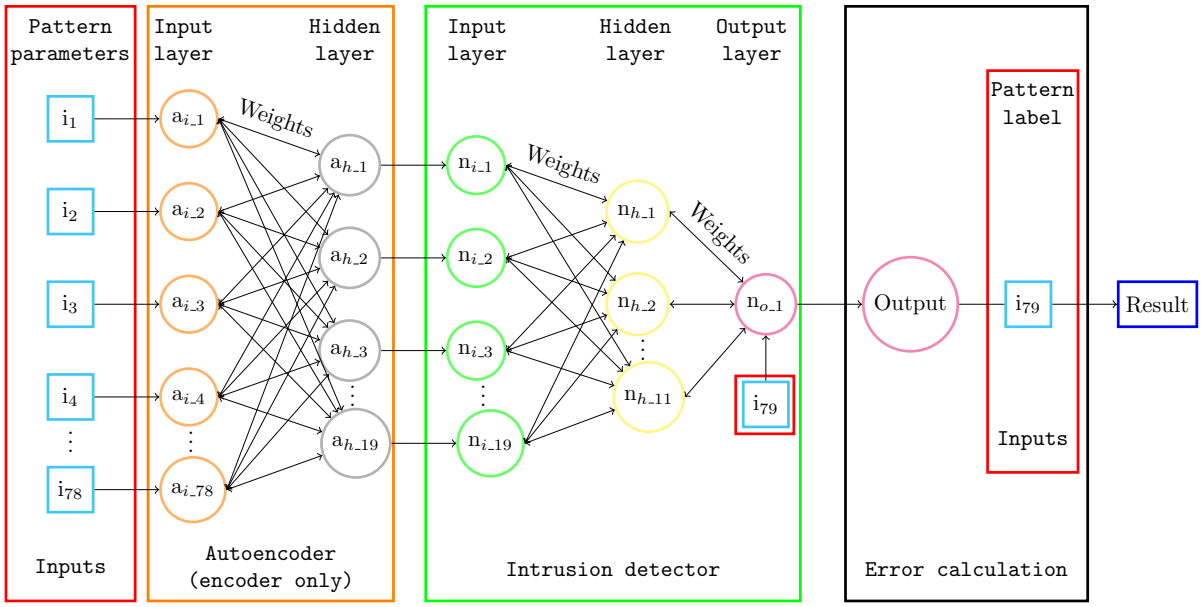

Figure 4.5: Ultimate architecture: Final deep network topology.

Chapter 5

Results

In this chapter, a set of tests performed by different architecture deepness are presented. It starts trying a simple one-hidden-layer Backpropagation classifier, making possible to test the overall performance of the system and polish the graphic-plotting routines used in the developed implementation. Then, it is checked the performance of the final Deep Learning Architecture. Finally, the results from a fully-trained stand-alone system are showed to verify the quantitative efficiency of this work.

During this stage, a set of graphics, tables and statistics are generated to improve the interpretation of the results, together with explicit paragraphs and values which reflects the real results of all tests. The comparisons between all independent results are performed to prove the advance of each approach to get a final optimized ANN architecture.

5.1 One hidden layer shallow network

These tests were performed just to try the functionality of the main structure: the shallow backpropagation classifier, the autoencoder is not used yet. It was divided into two parts: in the first one, it has been used the 150 inputs dataset and the second one using the 6000 inputs dataset.

5.1.1 150 inputs

Using 1000 epochs, an error of **0.0434** is reached. It looks like good and goes according to our objectives achieving an accuracy at **95.66%**. But, these results are misleading and in Section 6.3 will prove it. Also, Figure 5.1 is not trustworthy, it looks weird, enough errors graphics have been checked in the works studied before to know this behavior is not normal. Another point that does not convince is the fact of the error is very fluctuating using the same values, going from 0.1 to values in scales of 10^{-4}, these fluctuations are impossible to justify just by the random initialization of the weights.

5.1.2 6000 inputs

Now, it is time to test our ANN using more data. First, a test with 300 epochs has been performed. Figure 5.2 shows some points of over-training, with an error at **0.0961**, accuracy equal to **90.39%**. None of these are good enough results to achieve our objectives.

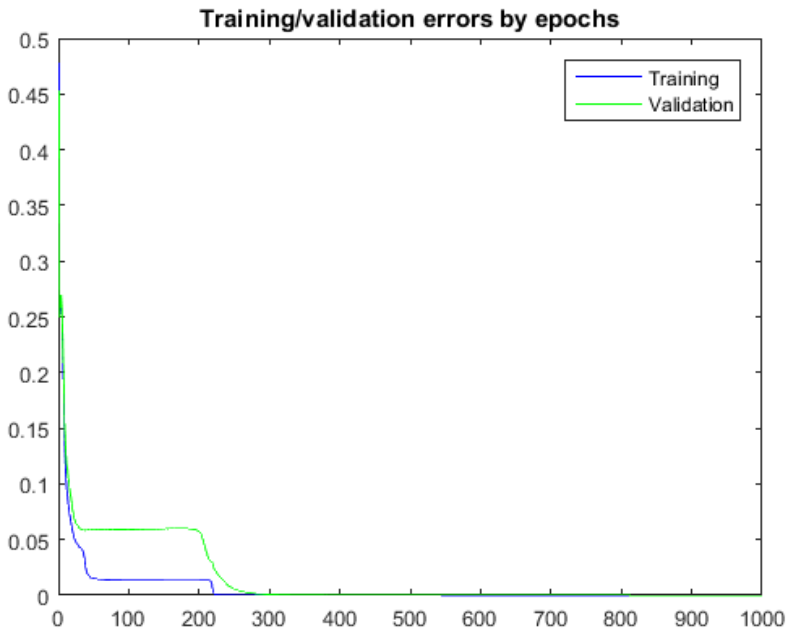

Figure 5.1: Epochs vs errors for training and validation phases: 1000 epochs, 150 inputs.

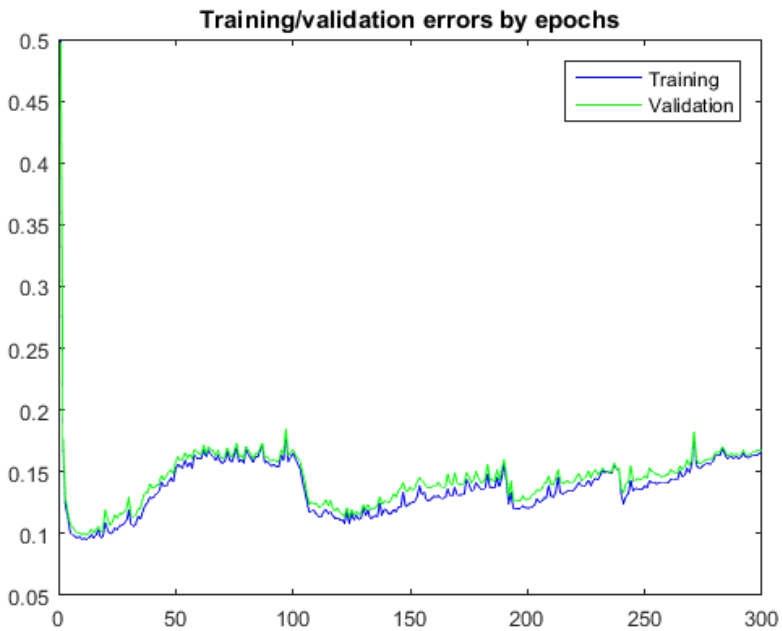

Figure 5.2: Epochs vs errors for training and validation phases: 300 epochs, 6000 inputs.

Using 1000 epochs, an error of **0.0873** is obtained, corresponding an accuracy of **91.27%**, that is our trained network can detect real intrusion activity in 91.3 out of 100 cases. Figure 5.3 presents over-training fluctuations in many points of the training epochs. It is not a common behavior in ANNs of this type, there are two options to explain this particular conduct: the ANN cannot process properly the data with its current parameters arrangement or, the ANN is not designed correctly. It is expected that

the first one option is happening.

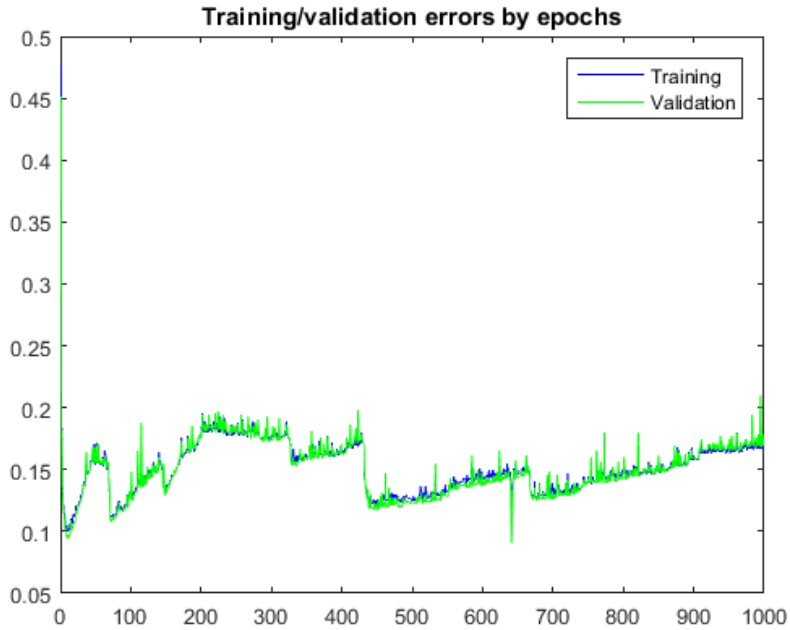

Figure 5.3: Epochs vs errors for training and validation phases: 1000 epochs, 6000 inputs.

It has been tried out the one hidden layer classifier for our intrusion detector and all its operations are working well. The 91% accuracy seems to be the limit of the learning capacity of the shallow network when it processes the 6000 inputs dataset. Let's see now what will happen if our detector network is fed with abstract representations of data coming from the hidden layer of the autoencoder. Will this deep arrangement improve our prediction accuracy?

5.2 Autoencoder and Shallow Backpropagation Architecture

In this case, the autoencoder is used as a way to use features of the data and not the data itself as an information source. Thus, after being trained the autoencoder outputs are neglected and its hidden outputs are given as inputs to the new three-layers learning network. This arrangement has three advantages: **1.-** the number of parameters to the ANN inputs is reduced. **2.-** the data has been compressed **3.-** the autoencoder performance is a good data reconstruction, then the autoencoder hidden layer must be good in data feature processing.

In Figure 5.4 the next is showed: in blue, 4 inputs taken randomly to plot the value of their features and, in red, the respective data reconstructed by the autoencoder is plotted. The reconstruction almost overlaps the original input, then it can be concluded that the hidden layer contains important compressed information and features from which the autoencoder output layer decodes the resultant reconstructed data. The great performance of the autoencoder is not only graphical but also it is proved numerically with a **Mean Square Error** equal to **4.7670x10**$^{-4}$ [31] between the original data and the reconstruction.

[31] It was the MSE obtained for this single test, this value could change for each different test, but the results from our autoencoder are always in the order of 10^{-4}.

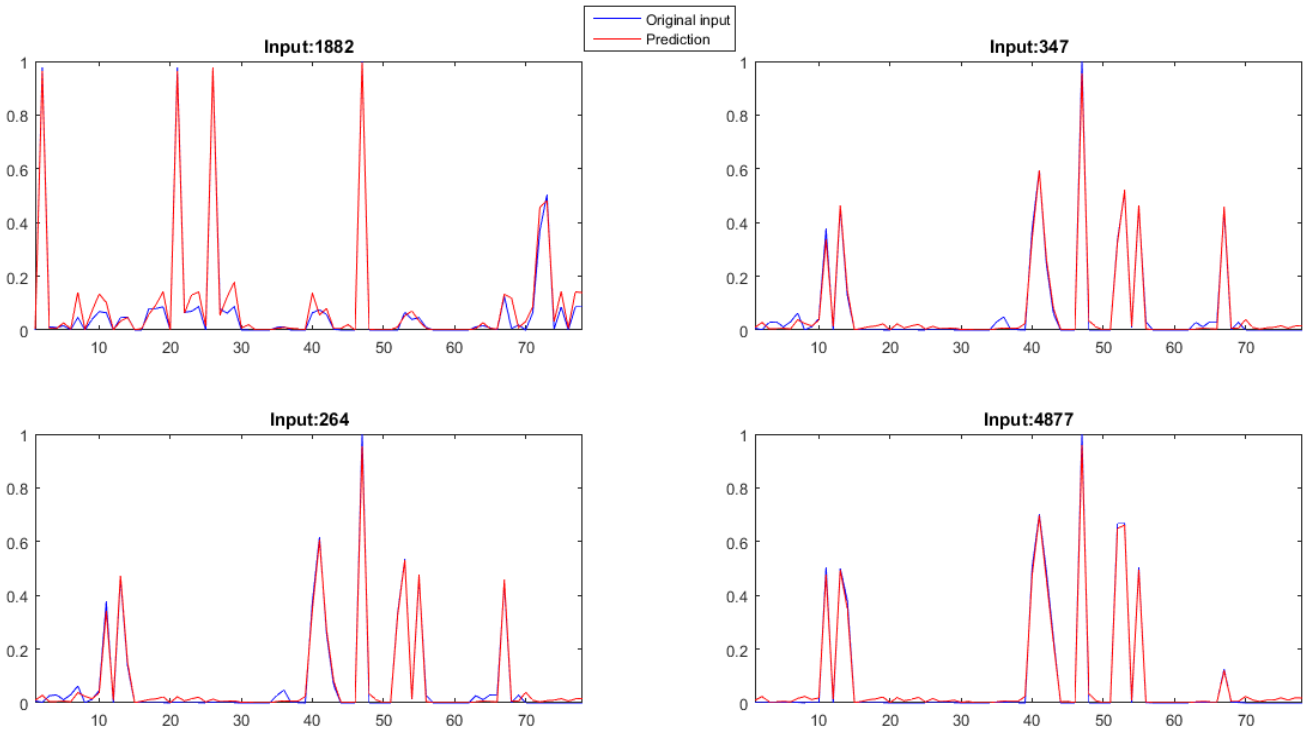

Figure 5.4: 4 random inputs (blue) and its respective autoencoder reconstruction (red). Inputs parameters in x-axis, normalized values in y-axis.

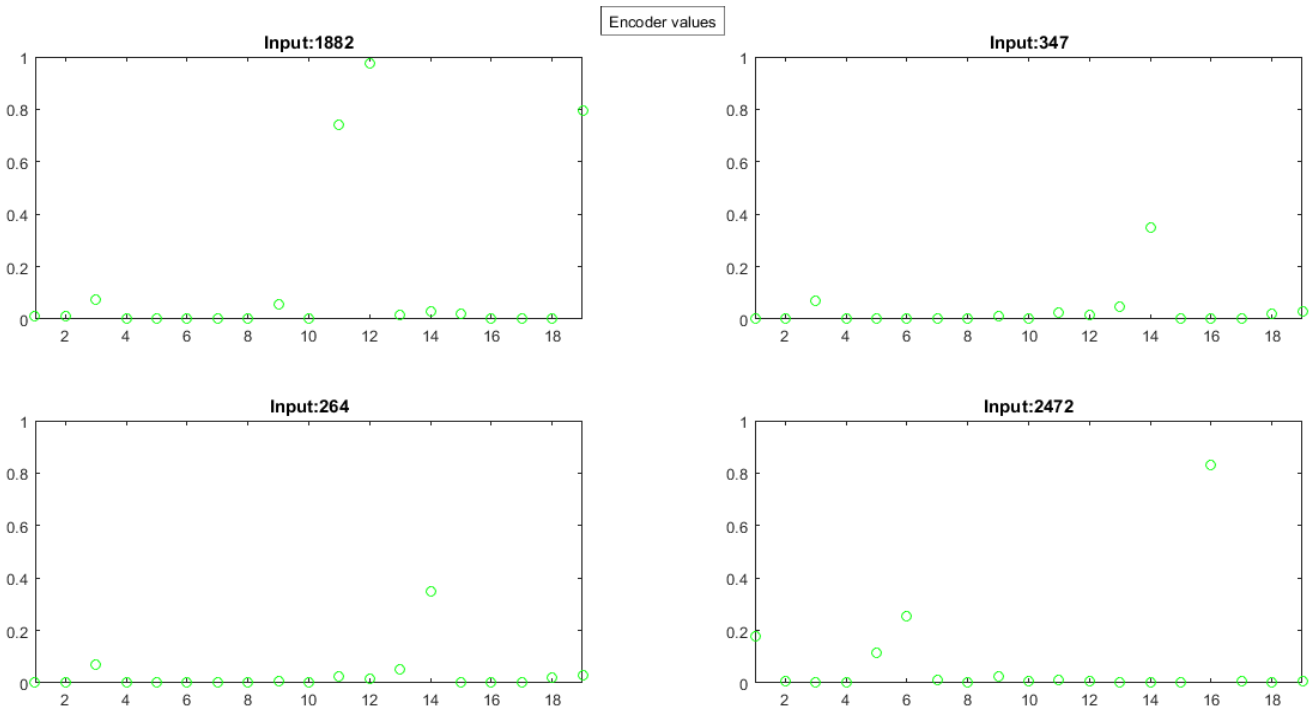

Figure 5.5: Encoder values: new data taken as inputs. Neurons in x-axis vs their respective output in y-value.

From these results, it looks promising to use the compressed data of the autoencoder hidden layer as input for the next trainable ANN and convert 78 to 19 parameters (without the targets), reducing by almost 25% the number of parameters. Red graphics from Figure 5.4 is data which would have been taken if predicted data by autoencoder is required as inputs, but it is not the case, because it is almost the same that taking the original data. Encoded data for the inputs in Figure 5.4 is represented in Figure 5.5, being this last the values taken as the new inputs for the ANN.

It has been already checked that the two architectures work very well, then the mix of these two approaches will be tested directly with the 6000 inputs dataset. Once the data have been compressed by the autoencoder, this data is inserted into the intrusion detector, obtaining an error of **0.0084** which results in an accuracy of **99.16%** in the test phase. Figure 5.6 shows the progress of the ANN accuracy for training and validation phases with only 300 epochs.

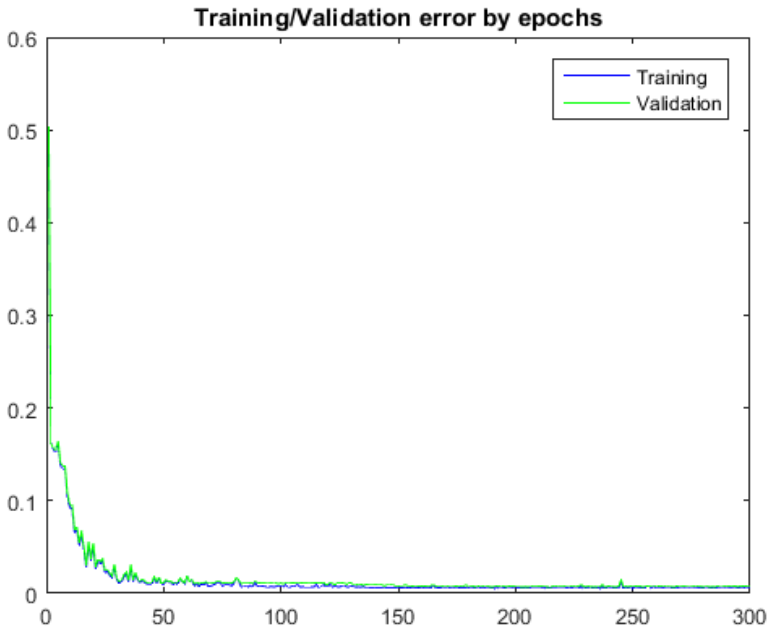

Figure 5.6: Epochs vs errors for training and validation phases: 300 epochs.

Let's see what happens when the number of epochs is increased to 500. Figure 5.7 shows us a similar behavior of the network which was shown in the two first phases with 300 epochs. The real difference here is performed by the testing phase where is achieved **0.0068** of error, corresponding to **99.32%** of accuracy. Nice!

Now, the ANN is tested directly to 1000 epochs. It is showed an error equal to **0.0068** that corresponds to **99.32%** of accuracy, a totally equal result to the last one, with a similar behavior in the two first phases as shown in Figure 5.8.

Our last try using this approach is using 5000 epochs, just to check if our ANN behaves weird or in an unexpected way. Figure 5.9 shows us the same behavior by the ANN. Even the testing error does not decrease so much with respect to five times less than this test, returning an error at **0.0060** equivalent to **99.40%** of accuracy.

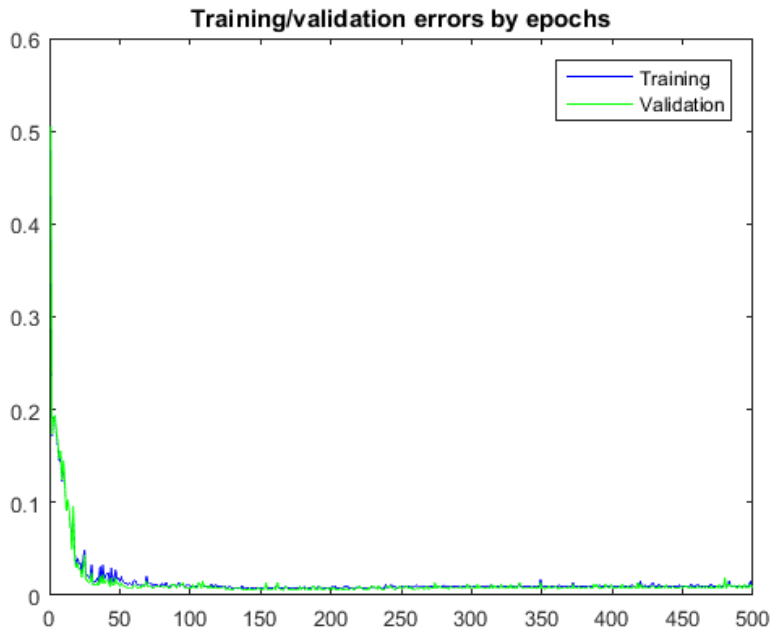

Figure 5.7: Epochs vs errors for training and validation phases: 500 epochs.

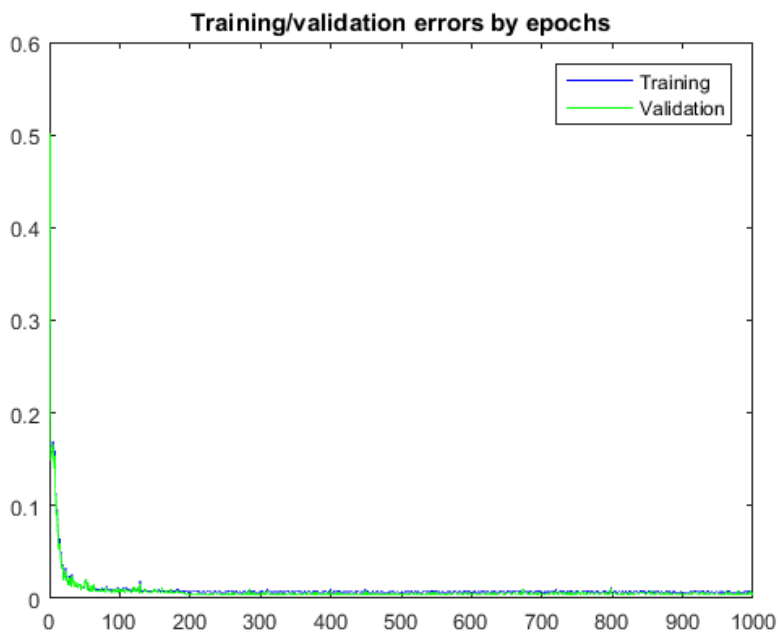

Figure 5.8: Epochs vs errors for training and validation phases: 1000 epochs.

Until now, only the number of epochs has been manipulated. An interesting issue is what would happen when one of the most important factors in the ANN, the learning rate, is manipulated? Well, let's see. If it is decreased 10 times: $\mathbf{LR = 0.01}$, the ANN will require more epochs to get good results as seen in Figure 5.10 and it is proved by the error equal to **0.0165** that corresponds to an accuracy of **98.35%**. It is not necessary to perform more tests to insist on this LR.

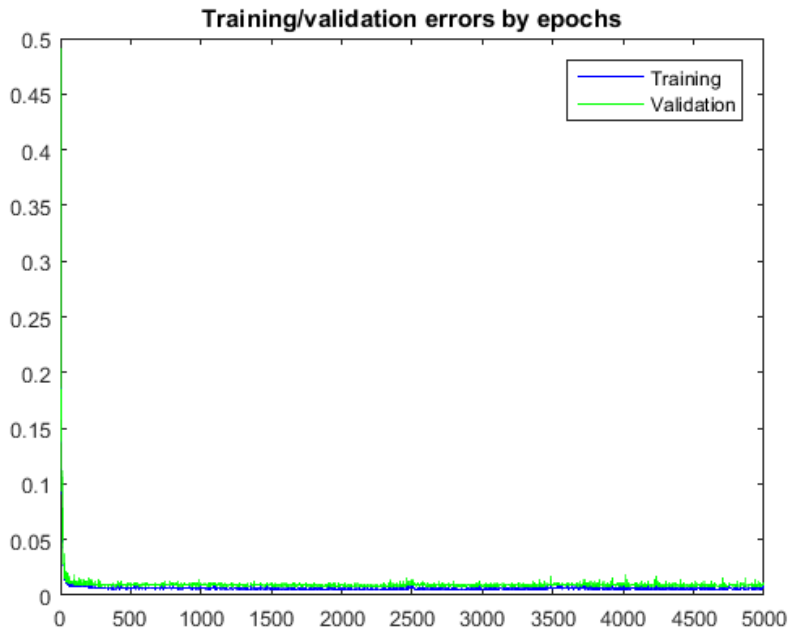

Figure 5.9: Epochs vs errors for training and validation phases: 5000 epochs.

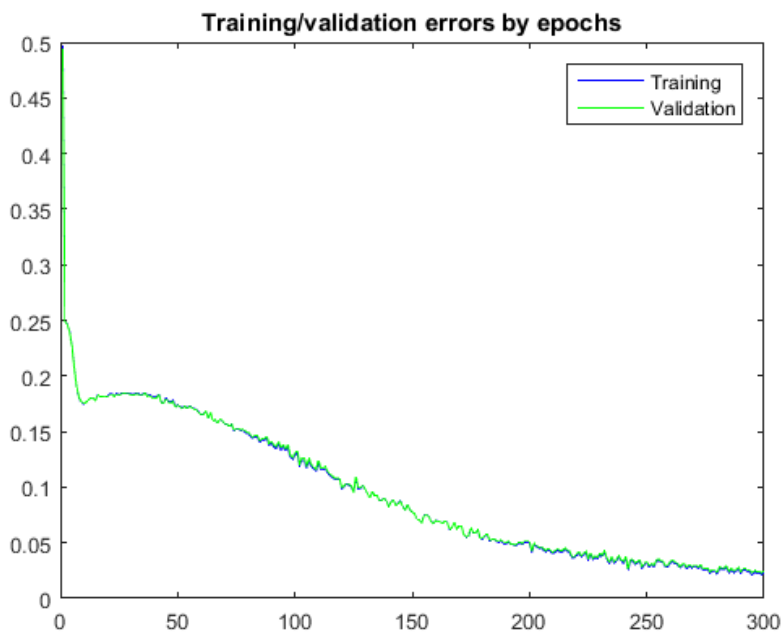

Figure 5.10: Epochs vs errors for training and validation phases: 300 epochs, LR=0.01.

What about increasing the learning rate? Using a **LR = 0.5** (increased five times) it does not look bad with an error at **0.0090** and an accuracy at **99.10%**, but it is not good the fluctuations in the phases showed in Figure 5.11. It looks like the optimal learning rate has been chosen: **0.1**.

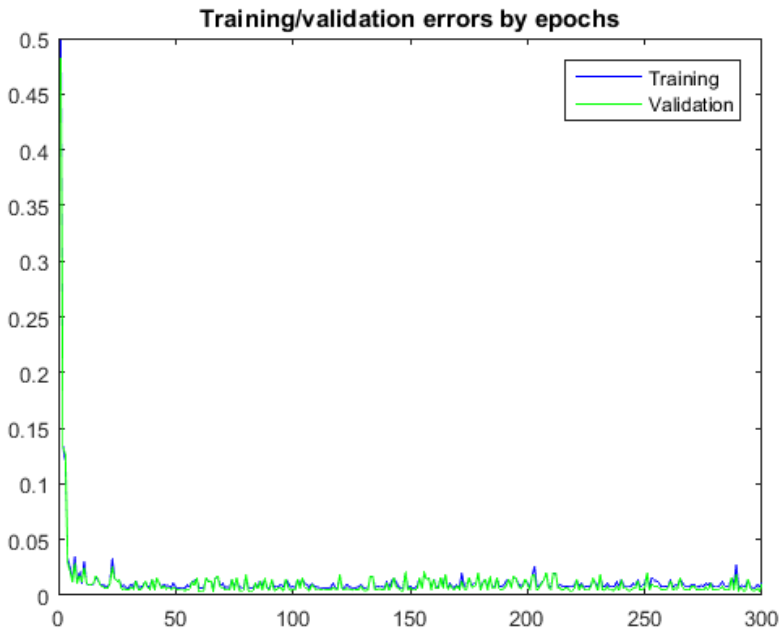

Figure 5.11: Epochs vs errors for training and validation phases: 300 epochs, LR=0.5.

All these results are summarized in Table 5.1. The best performance is highlighted in green color, results which could be selected with an acceptable performance are highlighted in yellow color and the worst results are in red, being parameters **Performance** and **Stability** the main considerations for this classification. The last parameter '**Stability**' refers to the capacity of the ANN to keep a performance into a range where **Performance** does not vary by more than **1%** and/or, **the fact of having a regular training without severe fluctuations**.

N.	# patterns	Epochs	LR	Autoencoder	Error	Performance(%)	Stability
1	150	1000	0.1	No	0.0434	**95.66**	No
2	6000	300	0.1	No	0.0961	**90.39**	No
3	6000	1000	0.1	No	0.0873	**91.27**	No
4	6000	300	0.1	Yes	0.0084	**99.16**	Yes
5	6000	500	0.1	Yes	0.0068	**99.32**	Yes
6	6000	1000	0.1	Yes	0.0068	**99.32**	Yes
7	6000	5000	0.1	Yes	0.0060	**99.40**	Yes
8	6000	300	0.01	Yes	0.0165	**98.35**	Yes
9	6000	300	0.5	Yes	0.0090	**98.10**	No

Table 5.1: ANN's results summary.

5.3 False positives and false negatives (stand-out system)

For this experiment, it will be used the best performance test features of the ANN given by the **Test 7** in the Table 5.1 and the 2000 inputs dataset. The test began taking the best performance weights, which are the same for the testing phase, these weights will be used in the transfer function. Then the input dataset is taken to pass the patterns one each one with the next procedure: Take the 78 first parameters of the pattern (not-labelled data) and compress it to 19 parameters using the autoencoder;

for this work, it is necessary to take the 6000 inputs dataset to get the features of the new pattern, if this phase is not performed in this way, it will be obtained a poor accuracy. Once the input is compressed, the transfer functions and the selected weights are used to compare the output with the target. In this case, the output is rounded to the nearest integer value using the function `round()`[32]. This process is repeated for all the input patterns and finally it is checked the accuracy of the ANN by using the equation 2.3. Also, two other measures were obtained, for which two terms used in Biology and Medicine have been borrowed: **sensibility** and **specificity** [51], adapted to our applications. Sensibility indicates the capacity of detect the attacks as such (true positive) and it is measured by

$$Sensibility = \frac{TP}{TP + FN} \tag{5.1}$$

, where
TP = true positive events,
FN = false negatives events.
The specificity is the fact of classify correctly the benign cases as such (true negative), measured by

$$Specificity = \frac{TN}{TN + FP} \tag{5.2}$$

, where
TN = true negative events,
FP = false positive events.
The results are reflected in the Table 5.2:

Total patterns	**Well-classified**	**False positives**
2000	1870	130
True Positive	**True Negative**	**False Negatives**
1000	870	0
Sensitivity	**Specificity**	**Accuracy**
1	0.87	93.5%

Table 5.2: False positives and false negatives: first results.

Well, these results look good, to finish this set of tests the last one is going to be performed: re-training the autoencoder using the 6000 inputs and adding more epochs (3500)[33], then the new inputs (the same 2000 inputs used before) will pass by this new trained autoencoder to get its respective result. Table 5.3 summarize this test:

Total patterns	**Well-classified**	**False positives**
2000	1915	85
True Positive	**True Negative**	**False Negatives**
1000	915	0
Sensitivity	**Specificity**	**Accuracy**
1	0.915	95.75%

Table 5.3: False positives and false negatives: second results.

[32] As any other function of this type, this MATLAB function turns into 1 the values greater or equal to 0.5 and to 0 in another case.
[33] This is the approximately maximum value that MATLAB allows us to train its autoencoder, with an own implementation it could be observed what would happen if the autoencoder is re-trained with any number of epochs.

An additional interesting result is the next: the **accuracy does not change** whether the 2000 input patterns are processed by the autoencoder as a single dataset or each pattern is processed one by one. The most important variation here is the taken **execution time** for these two different approaches. Table 5.4 shows this result:

Time execution 1st test (sec.)

All inputs as a single dataset		Inputs taken one by one	
Total time	**Average time by input**	**Total time**	**Average time by input**
0.127289	0.0000636445	119.316180	0.05965809

Time execution 2nd test (sec.)

All inputs as a single dataset		Inputs taken one by one	
Total time	**Average time by input**	**Total time**	**Average time by input**
0.156625	0.0000783125	127.861567	0.0639

Table 5.4: Execution time (in seconds) for false positives/negatives tests.

Chapter 6

Discussion

Once showed and explained images and tables in *Chapter 5: Results*, resulting from the execution of the implemented scripts of code with varied attributes, this chapter is designed to analyze and compare those results in the most detailed way to get an interpretative mean of them and achieve a complete understanding of the present work.

For the 150 patterns dataset, it was performed only one single experiment with 1000 epochs, it was just to warm up our ANN. The result of this test looks like there is a good approximation to our objectives, nothing is further from reality. The problem with this experiment is as follows: if it is reproduced, very fluctuating performances with variations from 85% to 99% will be found, which means that the ANN is not learning in a stable way. Nonetheless, thanks to this experiment it is already known that there are no failures in our ANN, but still not sure whether the architecture will fulfill the expectations changing the parameters. It is worth to remember that this experiment was performed without using the autoencoder.

Well, it continues with a pair of examples without using the encoded patterns generated by the autoencoder just to study the behavior of our ANN. In this case, it will be changed the length of the input set using now the 600-patterns dataset. First, the ANN is tried out using 300 and then 1000 epochs, the behavior of the network does not present changes (rather it does not present improvements). The same instability with signals of over-training in some points of training and validation epochs. Will it be a failure in the dataset?

Then, it is time to pass from testing the complete deep architecture by using the autoencoder to capture data features and reduce the dimensionality of the patterns from 78 to 19 parameters. In these experiments the scenario starts to change and looks better, returning results more acceptable and easy to analyze. From this point, all results present great performances and stability in the behavior of the ANN, without variations bigger than 1% in each test performed under the same conditions. By the way, for this architecture it was used the 6000-patterns dataset, being the changes executed in the epochs and learning rate.

With a learning rate fixed to 0.1, the first test was performed using 300 epochs with a performance at 99.16% and given its stability this configuration of the ANN could be already considered to be used to execute the final experiments. Next, the number of epochs was incremented to 500 with an improvement of 0.12%, it results in 99.32% of performance, the same value presented with 1000 epochs. Applying an increasing 5 times bigger than the same test, the improvement starts to be very small, obtaining 0.08% which means 99.40% of performance, becoming the final test taken to value our ANN with respect to the accuracy with false positives and false negatives. From this point the small improvements may not compensate the time of execution when running the training and validation phases, it will be interesting

observing the behavior of the ANN by performing a lot of epochs in a more powerful computing equipment. An interesting point in these experiments is that the error decrease abruptly in the first 10 epochs of training, it could be said that the rest of epochs are just for optimizing the weights and keep the stability of the ANN.

Now, decreasing the LR value to 0.01 stability is maintained with an admissible performance at 98.35%, smaller than in the other tests but it is still good. This is a nice try but this experiment is not used to perform the final test because better performances have been obtained.

The last case is performing a test increasing the learning rate to 0.5. It is appreciated a similar ANN's behavior as in the other tests, but there exist small peaks (but bigger than in the other tests) in the error corresponding to the training and validations phases; these peaks reach variations near to 0.04% and although it looks very small, since the ANN will face a serious problem, it is not convenient a wrong classification in a real-world implementation. Then, this is not the desired result. It is enough reason to consider not increasing the learning rate, it is possible the existence of a LR value between [0.1 - 0.5] but testing more values in this range is pending for future works.

All results discussed until now are qualitative, the quantitative accuracy of our ANN are those presented in the *Section 5.3: False positive and false negative* in the *Results* chapter. An interesting issue to analyze here before to observe the results, is the compression of every pattern parameters using the 6000 inputs dataset added to our new input to encode it; although without using this dataset, the autoencoder will predict correctly the input, as seen before, it does not matter, the importance of the autoencoder lies in the extraction of characteristics in its hidden layer, therefore it is performed the encoding process in this way to take advantage of the features of the data already processed to train the autoencoder, to perform a better featuring to our new input compressed patterns which will be introduced into the shallow network. On the other hand, the results showed in Tables 5.2 and 5.3 are clear: the ANN classify totally the attacks as such, the problem is caused by some benign events that look like attacks for our ANN. This behavior could be corrected by training constantly the ANN with new events since they could present totally unknown features as the used in this last test. The invariable accuracy given in the tests using the dataset to be compressed, by the autoencoder, on the whole and compressing pattern by pattern, means that the autoencoder is not learning from the new patterns but it is just compressing them according to the given inputs because it is already trained (with the 6000 inputs).

Chapter 7

Conclusions

Artificial Neural Networks have been applied in a lot of fields of researching and science; this work has demonstrated the useful and effectiveness of these interesting and advanced tools in a very important field in Computer Science: **cybersecurity** by carrying out, efficiently, an intrusion detection system, based on deep learning architectures.

The used tools do not have fixed parameters and characteristics (such as the learning rate, training/-validation epochs, transfer function, weights fitting). Many trial and error executions has been performed to achieve an optimal architecture according to the required problems. It can be observed that although weights are initialized randomly, it does not affect significantly the training of the ANN since the fast descent of error just later a few epochs of training.

The introduction of an autoencoder as a middle processor has worked well giving stability to the ANN and raising the overall performance and accuracy of the intrusion detector to a respected 99.4% and 95.75%, respectively, from an unstable initial performance of 95% without using the autoencoder. This confirms the assumption that a well-trained autoencoder develops good feature detectors in its hidden layer and is ready to pass this valuable information to other downstream networks. Also proves that deep architectures, with separate training with unlabeled and labelled data, yield improved neural processors.

Finally, it has been obtained a robust, trainable, fast program capable of work as an online intrusion detector which will improve its capacity of correct characterization by training using each incoming event, winning more flexibility to perform this activity. It is evident the need for more processing power equipment, for example, if the autoencoder is required to be re-trained each time a new event is processed (or even re-training the complete ANN) in the stand-out system, in the opposite case the process would be slow, removing robustness to the system. Since the accuracy does not change whether the inputs are compressed in sets or individually and the execution time of the first approach is approximately 1000 times better, the detector could be implemented not to process each event individually but take a determined amount of patterns every so often into a dataset and then process them by the system.

Chapter 8

Future Works

Given the time and the available resources, not all desired tests could be performed for this topic that has a scope of implementations bigger than one could think. More powerful computers, equipment with parallel and multi-GPU/CPU[34] interfaces, hardware/software for catching and analysis traffic network are some of the tools needed to focus on more advanced goals in the future. Below are shown some interesting activities pending for our next works:

8.1 Use of different datasets

There are many cybersecurity laboratories and institutes creating data from their networks and events and, they upload these valuable datasets to be accessible by everybody. The main problem when downloading these datasets is the lack of a target, that is knowing when an event is or not an attack. This data is used to analyze and obtain own conclusions (not related with the objectives of this work) and this kind of data is not needed in this occasion. Since the number of parameters and the parameters are variables, it will be interesting study and analyze how the ANN's behavior changes too. Also, the changes would be seen in the ANN's parameters such as the training epochs and the hidden layer of encoding into the autoencoder.

8.2 Creation of a dataset of its own

It could be an easy and fast solution to the problem of lack of targeting in the patterns, it could be created a simple network connected to the internet with different computers and devices using different operative systems. For this implementation, it is necessary the use of specialized tools to catch and track each individual event in each machine, installed in a common point of the network such as the router or switch used for internet connection or in a computer with manager permission over all the network. Even a virtual network could be created using virtual machines which will require a powerfully-featured single computer to install the virtual machines. The attacks could be performed by ourselves using pentesting tools going from simple and basics attacks until advanced and distributed attacks from different points, it technique gives the advantage of having always the control of the attacks and any event in the network traffic and it is not necessary to be connected to the internet but just using an internal network. The attacks could also be performed by third parties from other points in the world publishing our network

[34] Graphics Processing Unit and Central Process Unit respectively. A multi-GPU/CPU approach allows parallel computing performances.

to be attacked on internet which could give us more and more sophisticated different attacks but the loss of control over them.

8.3 Coupling the system into a real-time IDPS

It would be a remarkable project in which at the same time data is collected to other researches, real-time attacks could be detected to take the correct decisions. It would be installed a trained ANN ready to process any event with the determined parameters, process it and then warn to the network managers. Also, it will be a complement of an IDS/IPS already installed like an additional layer of protection for which entrance events must be analyzed.

8.4 Training network using several GPUs

Parallel computing is a technique in which researches don not have just high expectations but also have good results. It will be interesting the use of *High-Performance Computing* (HPC) techniques in this ANN to process millions of patterns to train the ANN. Using these techniques, performance could better or not, it is nothing sure about how it will change the performance and accuracy of the ANN but it is guaranteed that good results for new researches could be reached. It could be analyzed what happens when using millions of data performing millions of epochs with an ANN formed by hundreds of layers and computing more complex operations and functions. Could it be possible to obtain a 100% of accuracy training the ANN applying the characteristics mentioned before? Will it affect the performance in an unknown way the fact of calculating the ANN's parameters in parallel? This kind of questions can just be answered putting in practice HPC techniques to train the ANN.

8.5 Researching about new Artificial Neural Network features

The selection of each single feature for an ANN implementation just depends on the developers, thus the setup is configured whatever they want. Improving and optimizing transfer functions and process algorithms (logical sigmoid and backpropagation, respectively, for our case) is the work of some researches in Computing Science and Mathematics and it has been made a lot of investigation in this topic such that there exist a lot of these functions and algorithms, some are news and others are variants from an existent one and, this is the reason why there exist a lot of ANN architectures. Using an optimal transfer function and process algorithm according to our needs would be like our perfect ANN. In fact, making subtle changes in the current parameters could be a topic for another work, analyzing the general changes in the ANN with all permutations of hundreds of infinitesimally variations in each individual parameter.

8.6 Researching about Deeper Artificial Neural Network

In theory, the process of data autoencoding can be repeated many times over, that is the output of the first autoencoder hidden layer being used to train a second autoencoder whose hidden layer will create new data representation. It is expected that this new data is richer in data features and will eventually produce better predictions, and so on.

Bibliography

[1] M. Korcak, J. Lamer, and F. Jakab, "Intrusion prevention/intrusion detection system (ips/ids) for wifi networks," *International journal of Computer Networks & Communications*, vol. 6, pp. 77–89, 07 2014.

[2] G. Jha, "Artificial neural networks and its applications," 01 2019.

[3] M. Mijwel, "Artificial neural networks advantages and disadvantages," 01 2018.

[4] H. Lodish, A. Berk, C. A. Kaiser, M. Krieger, A. Bretscher, H. Ploegh, A. Amon, and M. P. Scott, *Molecular Cell Biology*, 7th ed. W.H. Freeman and Company, 2013.

[5] D. Lacrăma, L. Vîşcu, and C. V. Anghel Drugarin, "Artificial vs. natural neural networks," *13th Symposium on Neural Networks and Applications (NEUREL)*, 11 2016.

[6] D. Shanth, G. Narsimha, and R. Mohanthy, "Human intelligence vs. artificial intelligence: Survey," *3rd National Conference on Research Trends in Computer Science & Technology (NCRTCST-2015)*, vol. 6, pp. 30 – 34, 09 2015.

[7] R. Sathya and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification," *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 2, 2013. [Online]. Available: http://dx.doi.org/10.14569/IJARAI.2013.020206

[8] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[9] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, Aug 2017, pp. 1–6.

[10] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[11] B. Pandey, A. Singh, and L. Balani, "Ethical hacking (tools, techniques and approaches)," 01 2015.

[12] A. Trowbridge, F. Sharevski, and J. Westbrook, "Malicious user experience design research for cybersecurity," *CoRR*, vol. abs/1806.11060, 2018. [Online]. Available: http://arxiv.org/abs/1806.11060

[13] B. Guo, "Why hackers become crackers – an analysis of conflicts faced by hackers," *Public Administration Research*, vol. 5, no. 1, p. 29, 2016.

[14] L. A. Long and E. H, "Profiling hackers," *SANS Institute*, p. 22, January 2016. [Online]. Available: https://www.sans.org/reading-room/whitepapers/hackers/profiling-hackers-33864

[15] J. Andress and S. Winterfeld, *Basics of information security - understanding the fundamentals of infosec*. Syngress Media,u.s., 2014.

[16] H. Nemati, "Information security and ethics: Concepts, methodologies, tools, and applications," 01 2007.

[17] C. Hunt, *TCP/IP network administration*. OReilly, 1994.

[18] S. Kumar and D. Agarwal, "Hacking attacks, methods, techniques and their protection measures," *International Journal of Advance Research in Computer Science and Management*, vol. 4, pp. 2353–2358, 05 2018.

[19] M. Karpinski, A. Korchenko, P. Vikulov, R. Kochan, A. Balyk, and R. Kozak, "The etalon models of linguistic variables for sniffing-attack detection," 09 2017, pp. 258–264.

[20] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2027–2051, thirdquarter 2016.

[21] G. Nath Nayak and S. Ghosh Samaddar, "Different flavours of man-in-the-middle attack, consequences and feasible solutions," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 5, July 2010, pp. 491–495.

[22] S. M. S. Sajjadi and B. T. Pour, "Study of sql injection attacks and countermeasures," *International Journal of Computer and Communication Engineering*, vol. 2, no. 5, 9 2013.

[23] T. Gautam and A. Jain, "Analysis of brute force attack using tg — dataset," in *2015 SAI Intelligent Systems Conference (IntelliSys)*, Nov 2015, pp. 984–988.

[24] J. Owens and J. Matthews, "A study of passwords and methods used in brute-force ssh attacks," 06 2019.

[25] J. Gadge and A. A. Patil, "Port scan detection," in *2008 16th IEEE International Conference on Networks*, Dec 2008, pp. 1–6.

[26] P. Kesharwani, S. S. Pandey, V. Dixit, and L. K. Tiwari, "A study on penetration testing using metasploit framework," *International Research Journal of Engineering and Technology(IRJET)*, vol. 5, 12 2018.

[27] M. Khonji, Y. Iraqi, and A. Jones, "Phishing detection: A literature survey," *IEEE Communications Surveys & Tutorials*, vol. PP, pp. 1–31, 04 2013.

[28] A. A. Ahmed and N. A. Abdullah, "Real time detection of phishing websites," in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Oct 2016, pp. 1–6.

[29] D. Geer, "Malicious bots threaten network security," *Computer*, vol. 38, no. 1, pp. 18–20, Jan 2005.

[30] A. Karataş and S. Şahin, "A review on social bot detection techniques and research directions," *ISCTurkey 10th International Information Security and Cryptology Conference*, 10 2017.

[31] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *National Institute of Standards and Technology, Special Publication 80094*, 2 2007.

[32] N. Chakraborty, "Intrusion detection system and intrusion prevention system: A comparative study," *International Journal of Computing and Business Research (IJCBR)*, vol. 4, 5 2013.

[33] Wenke Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*, May 1999, pp. 120–132.

[34] J. Cannady, "Artificial neural networks for misuse detection," 11 1998.

[35] M. Sammany, M. Sharawi, M. El-Beltagy, and I. Saroit, "Artificial neural networks architecture for intrusion detection systems and classification of attacks," 01 2007.

[36] Y. Liu, S. Liu, and X. Zhao, "Intrusion detection algorithm based on convolutional neural network," *2017 4th International Conference on Engineering Technology and Application (ICETA 2017)*, pp. 9–13, 2017.

[37] S. Biswas, "Intrusion detection using machine learning: A comparison study," *International Journal of Pure and Applied Mathematics*, vol. 118, pp. 101–114, 07 2018.

[38] F. G. Beckenkamp, "A component architecture for artificial neural network systems," 01 2002.

[39] C. Gershenson, "Artificial neural networks for beginners," *CoRR*, vol. cs.NE/0308031, 2003. [Online]. Available: http://arxiv.org/abs/cs.NE/0308031

[40] A. Vehbi Olgac and B. Karlik, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 02 2011.

[41] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[42] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *Nuclear Science, IEEE Transactions on*, vol. 44, pp. 1464 – 1468, 07 1997.

[43] A. R. Abas, "Adaptive competitive learning neural networks," *Egyptian Informatics Journal*, vol. 14, p. 183–194, 11 2013.

[44] T. Kohonen, "Essentials of the self-organizing map," *Neural networks : the official journal of the International Neural Network Society*, vol. 37, 10 2012.

[45] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, pp. 2554 – 2558, 01 1982.

[46] Y. L. Cun, "Theoretical framework for back-propagation," *Proceedings of the 1988 Connectionist Models Summer School*, pp. 21 – 28, 1988.

[47] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 593–605 vol.1.

[48] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2014, pp. 496–503.

[49] O. Chang, P. Constante, A. Gordon, and M. Singania, "A novel deep neural network that uses space-time features for tracking and recognizing a moving object," *Journal Artificial Intelligence and Soft Computing*, vol. 7, no. 2, pp. 125–136, 2017.

[50] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision Research*, vol. 37, no. 23, pp. 3311 – 3325, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0042698997001697

[51] F. De Smet, Y. Moreau, K. Engelen, D. Timmerman, I. Vergote, and B. De Moor, "Balancing false positives and false negatives for the detection of differential expression in malignancies," *British Journal of Cancer*, vol. 91, no. 6, pp. 1160–1165, 2004. [Online]. Available: https://doi.org/10.1038/sj.bjc.6602140

Appendices

Appendix A

CICIDS2017 dataset

This is a summary from the official website of Canadian Institute for Cybersecurity from the University of New Brunswick from where the used dataset for this work has been downloaded. All this information has been retrieved from https://www.unb.ca/cic/datasets/ids-2017.html.

CICIDS2017 have most typical hacking attacks and benign activities into a computers network in a controlled environment consisting on 25 users in 12 different machines operating on HTTP, HTTPS, FTP, SSH, and email protocols. Attacks detected to build this dataset were: Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS. The traffic of the network has been monitored in specific hours during 5 days, giving as result CSV files.

These files shows thousands of individual patterns divided into 79 parameters, being the first 78 the informational parameters and the last one the label of each pattern. The informational parameters are:

1. **Destination port**: Port in which connection was established.
2. **Flow Duration**: Connection time in seconds.
3. **Total Fwd Packets**: Number of total packets in the forward direction.
4. **Total Backward Packets**: Number total of packets in the backward direction.
5. **Total Length of Fwd Packets**: Length in bytes of total packets in the forward direction.
6. **Total Length of Bwd Packets**: Length in bytes of total packets in the backward direction.
7. **Fwd Packet Length Max**: The largest value in bytes of the packet(s) length in the forward direction.
8. **Fwd Packet Length Min**: The smallest value in bytes of the packet(s) length in the forward direction.
9. **Fwd Packet Length Mean**: Average of the total packets length in the forward direction.
10. **Fwd Packet Length Std**: Standard Deviation of the total packets length in the forward direction.
11. **Bwd Packet Length Max**: The largest value in bytes of the packet(s) length in the backward direction.
12. **Bwd Packet Length Min**: The smallest value in bytes of the packet(s) length in the backward direction.
13. **Bwd Packet Length Mean**: Average of the total packets length in the backward direction.
14. **Backward Packet Length Std**: Standard Deviation of the total packets length in the backward direction.
15. **Flow Bytes/s**: The number of bytes per second in the flow.
16. **Flow Packets/s** The number of packets per second in the flow.
17. **Flow IAT Mean**: Average of the Inter-Arrival Time in the flow.

18. **Flow IAT Std**: Standard Deviation of the Inter-Arrival Time in the flow.
19. **Flow IAT Max**. The largest value of the Inter-Arrival Time in seconds in the flow.
20. **Flow IAT Min**: The smallest value of the Inter-Arrival Time in seconds in the flow.
21. **Fwd IAT Total**: The total value of the Inter-Arrival Time in seconds in the forward direction.
22. **Fwd IAT Mean**: Average of the Inter-Arrival Time in the forward direction.
23. **Forward IAT Std**: Standard Deviation of the Inter-Arrival Time in the forward direction.
24. **Fwd IAT Max**: The largest value of the Inter-Arrival Time in the forward direction.
25. **Fwd IAT Min**: The smallest value of the Inter-Arrival Time in the forward direction.
26. **Bwd IAT Total**: The total value of the Inter-Arrival Time in seconds in the backward direction.
27. **Bwd IAT Mean**: Average of the Inter-Arrival Time in the backward direction.
28. **Bwd IAT Std**: Standard Deviation of the Inter-Arrival Time in the backward direction.
29. **Bwd IAT Max**: The largest value of the Inter-Arrival Time in the backward direction.
30. **Bwd IAT Min**: The smallest value of the Inter-Arrival Time in the backward direction.
31. **Fwd PSH Flags**: The number of times the packets in the flow had the Pushing Flag bit set as 1 in the forward direction.
32. **Bwd PSH Flags**: The number of times the packets in the flow had the Pushing Flag bit set as 1 in the backward direction.
33. **Fwd URG Flags**: The number of times the packets in the flow had the Urgent Flag bit set as 1 in the forward direction.
34. **Bwd URG Flags**: The number of times the packets in the flow had the Urgent Flag bit set as 1 in the backward direction.
35. **Fwd Header Length**: The header length of the packets flow in the forward direction.
36. **Bwd Header Length**: The header length of the packets flow in the backward direction.
37. **Fwd Packets/s**: The number of packets per second in the forward direction.
38. **Bwd Packets/s**: The number of packets per second in the backward direction.
39. **Min Packet Length**: The smallest value of the length of the packets.
40. **Max Packet Length**: The largest value of the length of the packets.
41. **Packet Length Mean**: Average of the packets length.
42. **Packet Length Std**: Standard Deviation of the packet length.
43. **Packet Length Variance**: Variance of the packets length.
44. **FIN Flag Count**: The number of times the packets in the flow had the Finish Flag bit set as 1.
45. **SYN Flag Count**: The number of times the packets in the flow had the Synchronize Flag bit set as 1.
46. **RST Flag Count**: The number of times the packets in the flow had the Reset Flag bit set as 1.
47. **PSH Flag Count**: The number of times the packets in the flow had the Pushing Flag bit set as 1.
48. **ACK Flag Count**: The number of times the packets in the flow had the Acknowledged Flag bit set as 1.
49. **URG Flag Count**: The number of times the packets in the flow had the Urgent Flag bit set as 1.
50. **CWE Flag Count**: The number of times the packets in the flow had the Congestion Window Reduced Flag bit set as 1.
51. **ECE Flag Count**: The number of times the packets in the flow had the Explicit Congestion-Notification Echo Flag bit set as 1.
52. **Down/Up Ratio**: Download and upload ratio.
53. **Average Packet Size**: Average of packets header size in the flow.
54. **Avg Fwd Segment Size**: Average of the segment size in the forward direction.

55. **Avg Bwd Segment Size**: Average of the segment size in the backward direction.
56. **Fwd Header Length**: The header length of the packets flow in the forward direction.
57. **Fwd Average Bytes/Bulk**: Average of the bytes/bulk ratio in the forward direction.
58. **Fwd Avg Packets/Bulk**: Average of the packets/bulk ratio in the forward direction.
59. **Fwd Avg Bulk Rate**: Average of the bulk rate in the forward direction.
60. **Bwd Avg Bytes/Bulk**: Average of the bytes/bulk ratio in the backward direction.
61. **Bwd Avg Packets/Bulk**: Average of the packets/bulk ratio in the backward direction.
62. **Bwd Avg Bulk Rate**: Average of the bulk rate in the backward direction.
63. **Subflow Fwd Packets**: Average of the number of packets in the sub-flows of the flows in the forward direction.
64. **Subflow Fwd Bytes**: Average of the number of bytes in the sub-flow in the forward direction.
65. **Subflow Bwd Packets**: Average of the number of packets in the sub-flows of the flows in the backward direction.
66. **Subflow Bwd Bytes**: Average of the number of bytes in the sub-flow in the backward direction.
67. **Init_Win_bytes_forward**: Number of bytes in the initial window in the forward direction.
68. **Init_Win_bytes_backward**: Number of bytes in the initial window in the backward
69. **act_data_pkt_fwd**: Number of packets with at least one byte of TCP payload in the forward direction.
70. **min_seg_size_forward**: The smallest segment size in the forward direction.
71. **Active Mean**: Average of active flow time (before becoming IDLE).
72. **Active Std**: Standard Deviation of active flow time.
73. **Active Max**: The largest time a flow was active.
74. **Active Min**: The smallest time a flow was active.
75. **Idle Mean**: Average of idle flow time.
76. **Idle Std**: Standard deviation of idle flow time.
77. **Idle Max**: The largest time a flow was idle.
78. **Idle Min**: The smallest time a flow was idle.

And finally, the 79th parameter is the label of the pattern: ‘benign’ or ‘attack’ (0 or 1).