Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master in Data Science

Master Thesis

Explainable Cascading System for Network Intrusion Detection in Industry

Author: Nicolás Amigo Sañudo

Supervisors: Concha Bielza Lozoya and Pedro Larrañaga Múgica

Madrid, July 2023

Master Thesis
Master of Science in Data Science

Title: Explainable Cascading System for Network Intrusion Detection in Industry
July 2023

Author: Nicolás Amigo Sañudo

Supervisor:
Concha Bielza Lozoya
ETSI Informáticos
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid

Supervisor:
Pedro Larrañaga Múgica
ETSI Informáticos
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid

Acknowledgments

This work was partially supported by the Spanish Ministry of Science and Innovation through the RTC2019-006871-7, PID2019-109247GB-I00 and TED2021-131310B-I00 projects.

I would like to express my thanks to my supervisors, Concha and Pedro, for their invaluable guidance and expert insight throughout the entire research process. Their mentorship has been fundamental in shaping this master thesis.

I extend my thanks to Titanium Industrial Security Madrid for providing me with a stimulating work area. I would also like to thank Jorge and Carlos, who took the time to discuss interesting aspects that have contributed to the development of this work.

Finally, I would like to acknowledge the support received from my family and my girlfriend Paola.

Resumen

La detección de intrusiones desempeña un papel fundamental en la protección de la integridad y la seguridad de las redes informáticas. A medida que evolucionan la complejidad y la sofisticación de los ataques, aumenta la necesidad de sistemas eficaces. Sin embargo, los procedimientos actuales se enfrentan a menudo a diferentes problemas a la hora de reconocer con precisión las actividades maliciosas y, al mismo tiempo, ofrecer explicaciones transparentes de sus decisiones.

En este trabajo se abordan estas cuestiones proponiendo un sistema explicable en cascada que combina las ventajas de las redes bayesianas con algunas técnicas de aprendizaje profundo para mejorar la interpretabilidad y la precisión del proceso de detección. Además, se incorporan algunos métodos *post-hoc* de explicabilidad del aprendizaje profundo para proporcionar una visión amplia del proceso de toma de decisiones, facilitando así una mejor comprensión de las intrusiones detectadas.

La metodología propuesta es evaluada mediante un conjunto de datos actualizado con diversos tipos de ataques. La solución supera a los métodos individuales tradicionales y proporciona aclaraciones transparentes de las intrusiones detectadas, mejorando la fiabilidad de los resultados. Asimismo, este trabajo supone un comienzo para futuros desarrollos en el ámbito de la detección explicable lo que permitirá una defensa más segura contra las comunicaciones anómalas.

Abstract

Intrusion detection performs a fundamental roll in safeguarding the integrity and security of computer networks. As the complexity and sophistication of attacks continue to evolve, the need for effective systems increases. However, existing approaches often face challenges in accurately recognizing malicious activity while providing transparent explanations for their decisions.

These issues are addressed in this master thesis by proposing an explainable cascading system that combines the strengths of bayesian networks and some deep learning techniques to improve the interpretability and accuracy of the detection process. Additionally, some *post-hoc* deep learning explainability methods are incorporated to provide meaningful insights of the decision-making process, facilitating a better understanding of the detected intrusions.

The proposed methodology is evaluated using an updated dataset with various types of attacks. The solution outperforms traditional single methods and yields transparent explanations of detected intrusions, improving the reliability of the decisions. This work opens avenues for future development in the domain of explainable network intrusion detection, facilitating a more trustworthy defense against anomalous communications.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, the industrial landscape has undergone major transformations with the advent of new technologies. From the rise of the Internet of Things (IoT) to the widespread integration of automation and cloud computing, these advances have brought numerous benefits and opportunities to businesses. However, along with these advances, the risks, and challenges of maintaining robust cybersecurity have increased.

After the COVID-19 pandemic, the digitization process accelerated significantly. According to a report by Accenture Research (Accenture, 2021), the overall rate of adoption of the main new technologies increased from 75% to almost 95%. Industrial systems are now highly intelligent and interconnected with unlimited potential to improve efficiency, productivity, and competitiveness in different sectors. It is possible to monitor equipment through IoT sensors that provide real-time information. Another vital enabler for industrial operations is cloud computing. Industrial companies can securely store, and process large amounts of data generated by IoT devices and other sources in geographically dispersed locations. In addition, advances in wireless communication technologies such as 5G are set to revolutionize industrial networks, providing ultra-low latency and high bandwidth connectivity. As a result, industrial systems can optimize their operations, accelerate innovation cycles, and gain a competitive advantage in the rapidly evolving marketplace. These advances have therefore changed traditional industrial systems and have set the pace for smart factories around the world.

With the arrival of Industry 4.0, the attack surface for potential cyber threats has expanded significantly. The integration of digital technologies introduces new vulnerabilities and risks that malicious actors can exploit. These breaches can have severe consequences. Thus, the importance of cybersecurity has become paramount. Ensuring the integrity, confidentiality, and availability of industrial systems is very important to protect sensitive data and maintain business continuity.

Security breaches have significant repercussions that go beyond the immediate consequences for the organizations affected. For example, they erode public trust in companies. This can hinder the growth of digital economies and impede technological advances. They can also disrupt critical services and infrastructures. This affects not only the organizations, but also the people who rely on these services. Another of the most important consequences is economic. According to a McAfee report titled “The Hidden Costs of Cybercrime” (Tom Gann, 2020), these attacks cost the global economy more than $1 trillion. The study states that 92% of companies felt effects beyond monetary losses. Moreover, these security issues often carry legal and regulatory consequences. Affected organizations can face significant legal action, lawsuits, and financial penalties, further impacting their operations and reputation.

As for the sophistication of cyberattacks, this has been on the rise, posing significant challenges. An annual report from Microsoft (Microsoft, 2022) states that the sophistication of actors has increased rapidly in recent years, using techniques that make them harder to detect and threaten even the most astute targets. Cybercriminals employ different types of tactics to exploit vulnerabilities and gain unauthorized access. These techniques are often highly targeted and customized, making it difficult for traditional security measures to detect.

To address these specific cybersecurity challenges, industrial companies adopt certain strategies. Some of these are antivirus software to detect and remove malware, firewalls to control network traffic or intrusion detection and prevention systems to monitor suspicious activity. With the evolution of threats, it is increasingly important to look for new methods to approach these new challenges.

The advent of artificial intelligence (AI) technologies in cybersecurity presents a significant opportunity to improve attack identification capabilities. These advanced technologies can analyze large amounts of data, detect patterns, and identify anomalies that may indicate cyber-threats. By leveraging these technologies, organizations can strengthen their cybersecurity posture to stay one step ahead of attackers.

Stanford University’s “Artificial Intelligence Index 2023 Report” (Stanford University, 2023) states that private investment in AI applied to cybersecurity is currently one of largest and the one that grew the most from 2021 to 2022. However, despite its great benefits, adoption has stagnated at around 60% in recent years. Thus, this represents a great opportunity for cybersecurity companies to differentiate themselves from their competitors.

1.2 Problem Statement

Titanium Industrial Security[1] is a leading company at national and international level in advising and supporting companies in the field of cybersecurity in the connected industry. It offers industrial cybersecurity services that allow to improve and adapt to the most demanding security requirements, to face the growing threats and to comply with the imposed regulations.

This company proposes to implement a local intrusion detection system capable of effectively identifying anomalies in network traffic communications between industrial devices. The system aims to monitor the network in real time and flag any deviations from normal behavior. To enhance understanding, it also intends

[1] https://www.titaniumindustrialsecurity.com/

to provide detailed explanations of its detection decisions. This additional functionality enables security analysts and other operators to understand the reasoning behind alerts and make informed decisions. By combining accurate detection with explanatory functions, the overall security posture and responsiveness of the industrial network is improved.

Network anomaly detection poses several challenges to effectively identify and mitigate potential threats. Current networks provide increasingly high transmission rates associated with the advent of 5G communications (Fernandez Maimo et al., 2018). This means that more data is transmitted and processed in a shorter amount of time, which can make it more difficult for anomaly detection systems to keep up in real-time.

The nature of the input data is another key aspect of any anomaly detection technique (Chandola, 2009). Most of the existing anomaly detection techniques deal with point data, in which no relationship is assumed among the data instances. However, there exist situations where such relationships among data instances become relevant for anomaly detection.

Anomaly detection methods frequently result in a high rate of false alarms (Aldini et al., 2005). This is because deviations from the expected behavior model can occur for many reasons. This may lead to a large volume of alerts to process making the system unusable by flooding it with some irrelevant notifications.

It should also be noted that anomaly detection is mostly carried out using deep learning techniques at present. This does not allow interpretation of the reasons why a certain decision has been arrived at. Therefore, it is very important to present the results in a way that is understandable to humans.

1.3 Objectives

The objective of this project is to develop an intrusion detection system for the detection of attacks on industry network traffic using machine learning methods. The system aims to analyze the communication patterns and detect any suspicious activity or deviation from normal behavior. It should also provide explanations enabling administrators and analysts to understand the nature of the threats.

Given the characteristics of the system, it is claimed that semi-supervised learning techniques can be used, and they are combined with other supervised methods for classification tasks. Bayesian networks and advanced recurrent neural networks will form the basis of a cascade system where both algorithms are trained by the normal behavior of the network. The first learn the probability distribution of the variables and estimates the likelihood that a connection is within the usual range. On the other hand, the latter captures sequential dependencies and temporal patterns to certify the previously indicated anomalies.

Directly interpretable models such as Bayesian networks are used and *post-hoc* explainability methods are added to those that function as a black box. Feature relevance techniques and transparent design methods work together with the recurrent neural networks to accomplish this goal. Consequently, the whole system provides explanations to enhance the understanding of detected anomalies.

Another key objective is to address performance requirements within the developed system. The use of lightweight models such as Bayesian networks as the first component of the system helps to filter the traffic received by the neural network. This involves ensuring efficient resource utilization to minimize processing overhead and response time.

1.4 Document Structure

The master thesis is structured in five chapters, each addressing specific aspects of the research.

- Chapter 1, already presented, is an introduction that provides a concise overview of the research topic, highlighting the problem statement and objectives of the study.

- Chapter 2, “Theoretical Background”, provides a comprehensive review of existing literature and research related to network intrusion detection. It explores the techniques, and algorithms commonly employed in network intrusion detection systems and explains in detail the main models employed in the system. In addition, it examines explainable artificial intelligence and the challenges associated with the machine learning methods discussed above.

- Chapter 3, “Contribution Methodology”, provides the details of the architecture and components of the proposed cascading system for network intrusion detection. It gives a brief description of the data used and the preprocessing techniques, ensuring reliability and its quality. Further, the built-in explainability methods incorporated are presented, along with their rationale.

- Chapter 4, “Results”, presents the results of experiments performed with the proposed framework. It offers a complete analysis and interpretation of the performance metrics. The chapter also evaluates the effectiveness of the explanatory techniques employed.

- Chapter 5, “Conclusions and Future Work”, summarizes the main research findings and their implications for the industry. It reflects on the contributions of the master thesis while acknowledging its limitations. In addition, the chapter offers recommendations for further improvement and outlines possible areas for future research.

Chapter 2

Theoretical Background

2.1 Intrusion Detection Systems

The increasing reliance on digital systems in the industry has led to an increase in malicious activities. Intrusion detections are crucial in protecting these organizations by identifying and responding to unauthorized access attempts and malicious activities.

An intrusion detection system (IDS) is a software or hardware that automates the process of monitoring the events occurring in a computer system or network, analyzing them for signs of security problems (Bace & Mell, 2001). On a more informal basis, an IDS is defined as the “burglar alarm” of the computer security field (Axelsson, 2000) whose goal is to defend a system by a combination of an alarm that sounds whenever the site’s security has been compromised. An entity then responds to the alarm and takes appropriate action. These “burglars” are normally attackers accessing from the Internet, authorized users attempting to obtain privileges for which they are not authorized, and other users misusing the privileges granted to them.

The main set of rules that limits access to information can be compromised when an intrusion occurs. Therefore, IDSs play a very important role in maintaining security. There are some reasons to use these mechanisms beyond attack detection. They can serve as a deterrent by increasing the perceived risk of discovery and punishment for potential attackers. In addition, they document the existing threat landscape providing valuable information to improve diagnosis and recovery.

IDSs can be categorized according to different monitoring and analysis approaches. Understanding the characteristics helps select and customize the solutions that best suit their specific security requirements.

In terms of detection methodology, there are two main approaches. When incoming events are compared against known attack signatures, the technique is known as *signature-based*. This type of detection is effective in identifying recognized attacks, but can have problems with new or modified threats that do not match existing signatures. On the other hand, *anomaly-based* detection focuses on identifying deviations from normal patterns. Anything that does not correspond to a previously learned behavior is considered intrusive. This is advantageous for detecting unforeseen attacks, but can also result in a high false alarm rate due to legitimate variations in behavior (Debar et al., 2000).

Additionally, a *hybrid* approach is formed by combining the above two to improve accuracy and coverage.

Another common way to classify IDSs is to group them by information source. *Network-based* systems monitor network traffic in real time to identify suspicious patterns. They do this in specific network segments using sensors and analyzing the activities of applications and protocols (Debar et al., 2000). In contrast, *host-based* systems focus on the security of individual systems and their local resources. This point of advantage allows to determine exactly which processes are involved in a particular attack on the operating system. A special subset of this category are the *application-based* systems that are designed to detect attacks targeting application-layer vulnerabilities and exploits (Bace & Mell, 2001).

2.1.1 Network Intrusion Detection Systems

Traditional network intrusion detection systems (NIDS) inspect the contents of every packet to find known attacks or unusual behavior. This is what is known as *payload-based* systems. This approach is effective in detecting specific attacks, as it looks for exact matches or variations of known attacks.

The problem with packet inspection is to perform it at the speed of several gigabits per second (Gbps) (Sperotto et al., 2010). Therefore, for high-speed lines it is important to investigate alternatives to packet inspection. One of the most widely used options today is *flow-based* intrusion detection. It analyzes the flow of network traffic, which represents a sequence of related packets exchanged between network hosts during a given time interval.

These systems examine network traffic metadata such as source and destination IP addresses, ports, timestamps, packet sizes and communication patterns. This type of detection provides a broader perspective of network activity and can detect anomalies even if the payload content is encrypted or not directly visible.

However, flow-based detection may have limitations in detecting attacks distributed across multiple flows or attacks that rely heavily on payload content for detection. Therefore, it can be considered a complement to packet inspection and not a substitute.

2.2 Signature-based Detection

Signature-based systems (Khraisat et al., 2019) rely on pattern matching techniques to find a known attack. They are also called knowledge-based detection or misuse detection. The main idea is to build a database of intrusion signatures and to compare the current set of activities against the existing signatures and raise an alarm if a match is found. However, these techniques are unable to identify attacks that span several packets. As modern malware is more sophisticated it may be necessary to extract signature information over multiple packets.

The effectiveness of *signature-based* detection lies in its ability to quickly detect, and block known attack patterns, providing a proactive defense mechanism. This is sometimes a costly trade-off if the event stream does not contain all the data being sought, or if multiple encodings must be taken into account. For example, if an attack uses advanced techniques to evade its known patterns or if signatures are designed to detect specific types of attacks, the system may have difficulty identifying the threat accurately.

This detection approach offers excellent detection accuracy for previously known intrusions. It should be able to generate very few false alarms. False positives (Aldini et al., 2005) come primarily from mischaracterization of the vulnerability. This often occurs when the IDS attempts to detect the execution of an application without differentiating between normal usage and actual malicious intent. In addition, it is often difficult to differentiate interactions between an attacker and a vulnerable information system from interactions between normal users and the same system.

On the other hand, false negatives occur on new attacks, when there is no signature associated to the vulnerability. Collecting vulnerability information of sufficient quality to write adequate signatures is a time-consuming task, and validation of this information is often limited, due to the large number of attack combinations possible.

The rapid pace at which new attack techniques and vulnerabilities emerge is another challenge associated with this approach. This involves frequent updates to the signature database. It requires a robust and efficient system to quickly incorporate the latest signatures, ensuring detection of the latest threats. Delays in updating the signature database can leave networks vulnerable to newly discovered attacks.

Furthermore, these systems have difficulty detecting zero-day attacks which exploit unanticipated security breaches that are not known to the party responsible for fixing the failure. They can go undetected until new signatures are deployed because these attacks lack pre-existing signatures. Therefore, the goal is to mitigate zero-day attacks before they cause significant damage to the network.

2.3 Anomaly-based Detection

Although effective against known threats, *signature-based* approaches often fail to detect new sophisticated intrusions that deviate from established patterns. This limitation requires the adoption of alternative detection methods, which has resulted in the emergence of *anomaly-based* detection as a vital component of modern IDSs.

Anomalies are patterns in data that do not conform to a well-defined notion of normal behavior (Chandola, 2009). In the context of IDSs, they can indicate security breaches or malicious activities. It is important to differentiate them from other related concepts, such as novelties and outliers. The first ones are elements that have not been observed before in the system under analysis. After their detection, they are usually incorporated into the normal model unlike anomalies. The outliers are observations that differs significantly from most instances. Although all anomalies can be considered outliers, not all outliers are necessarily anomalies. Outliers can exist within normal behavior and do not always indicate cyberattacks.

An important aspect is the nature of the desired anomaly. Anomalies can take various forms and occur in different domains. According to Chandola (2009), they can be classified into three categories. If an individual data instance is considered isolated and distinct with respect to the rest of data, then the instance is termed as a *point anomaly.* Although single observations within the cluster may not be anomalous, the collective behavior may deviate significantly from the expected pattern. This is what is known as a *collective anomaly*. Another type of anomalies is when some instances are considered anomalous in a very specific context. These are then called *contextual anomalies*.

Another way to categorize anomalies is to distinguish between global and local anomalies (Goldstein & Uchida, 2016). A *global anomaly* refers to an observation that is significantly different from the overall normal behavior of the dataset. A *local* anomaly, on the other hand, may not be an extreme value in the general dataset, but anomalous within a particular cluster of data points.

Obtaining accurately labeled data covering all types of behaviors can be a costly task as manual labeling by human experts is often necessary. In particular, acquiring a labeled set of anomalous data instances that covers the full spectrum of possible anomalies is more difficult than labeling normal behaviors. Anomaly detection techniques can operate in three different ways, depending on the availability of labels.

Supervised anomaly detection methods are based on labeled data, where anomalies are explicitly tagged. These methods learn from the instances to build a model that can accurately classify new data as normal or anomalous. They usually provide high accuracy, but require a significant amount of labeled data for training, which is not always available. In addition, some problems may arise due to the smaller number of attacks compared to normal observations.

Semi-supervised anomaly detection methods operate under the assumption that the training data contains only labeled instances of the normal class. This feature makes them more applicable compared to supervised techniques, as obtaining labeled anomalies can be particularly difficult. They can learn the underlying patterns of normal instances to identify deviations indicative of anomalies.

Unsupervised anomaly detection methods work in the absence of labeled data, relying solely on the characteristics of the data itself to identify anomalies. These methods aim to discover patterns or structures that are significantly different from the majority of data points, considering them as potential anomalies. This is an advantage when labeled data are costly to obtain.

False alarms are a significant concern in anomaly detection systems as they can lead to unnecessary disruptions and resource wastage. This phenomenon arises from the fact that deviations from the model are often observed for any incident occurring on the monitored information system.

On the other hand, false negatives in anomaly detection have two main causes in accordance with Aldini et al. (2005): corruption of the behavioral model and absence of measurement. Behavioral model corruption occurs when the model learns an intrusive behavior and incorporates it into its coverage. The IDS becomes unable to detect occurrences of the attack that have been accepted as normal. In addition, attacks sometimes do not affect the measures used by the normal behavior model. For example, in a scenario where a NIDS is designed to monitor network traffic patterns, but does not account for anomalies in network bandwidth usage. In this case, an attack that consumes a lot of network bandwidth might not be detected by the system.

2.3.1 Techniques

A wide range of techniques have been developed to address the challenges posed by anomalies in different domains. The categories into which they are organized are based on the nature of the algorithms used and there may be overlap between methods. According to Bhuyan et al. (2014), the most commonly used techniques used in anomaly detection are *statistical, classification-based*, *soft computing* and *hybrid learners*.

Statistical techniques exploit mathematical models to identify deviations from expected behavior. They are particularly effective when data follow well-defined statistical properties that can be captured mathematically. Bayesian networks are considered part of this group. They represent probabilistic relationships among variables and are used to make inferences about the likelihood of anomalous events.

Classification-based methods rely on the establishment of a model that allows categorizing network traffic patterns into various classes. These techniques require labeled data to train the behavioral model. One of the most widespread methods are *one-class classifiers* that can detect instances that do not belong to the learned class. For example, in a one-class support vector machine (OC-SVM) classifier the learning objective during training is to determine a function that is positive when applied to points on the boundary circumscribed around the training points and negative outside it.

Soft computing techniques are mainly associated with neural networks. They are well suited for anomaly detection because they can provide flexible and adaptive capabilities. They acquire knowledge of the environment through a learning process, which systematically modifies the interconnection strengths to achieve a desired goal. These networks usually perform well using massive neural connections.

Hybrid approaches are increasingly used to overcome the limitations of anomaly detection. This integration makes it possible to detect both known attacks through signature comparison and anomalies using anomaly detection algorithms. As a result, the overall performance of the detection system is improved.

Creating an accurate model when working with unbalanced data presents a significant challenge in anomaly detection. Traditional methods tend to classify all data in the majority class, which is not ideal. To address this problem, several methods have been proposed, which can be classified into data-level strategies and cost-sensitive strategies (Goldstein & Uchida, 2016).

Data-level strategies, also known as resampling strategies, aim to balance the class distribution of the training data. This can be achieved by undersampling the majority class or oversampling the minority class. However, random resampling has its limitations. Undersampling can discard valuable data, while oversampling can introduce a risk of overfitting if exact copies of existing instances are generated. To overcome these challenges, a synthetic minority over-sampling technique (SMOTE) was introduced (Chawla et al., 2002). It works by selecting a minority class instance and finding its k nearest neighbors in the feature space. It then creates new synthetic instances by interpolating between the selected instance and its neighbors.

Alternatively, cost-sensitive strategies focus on the minority class by incorporating misclassification costs. When assigning costs to correct and incorrect predictions, an instance is predicted to have the label that results in the lowest expected cost.

2.4 Bayesian Networks

In the search for effective reasoning about uncertain situations, the use of probability calculus has been a prominent approach. Probability theory has found wide application in a variety of fields. One such approach is Bayesian networks. These allow rational decisions to be made even in the presence of limited or ambiguous data. Their transparent representation of the probabilistic relationships between variables facilitates the understanding and explanation of the reasoning behind the decisions of AI systems. Thus, the advent of Bayesian network technology provides a promising solution to pave the way for broader and more sophisticated applications of probabilistic reasoning.

However, despite the advantages they bring, there is not much literature applied to intrusion detection in industry. An interesting paper by Kruegel et al. (2003) proposes an event classification scheme based on a Bayesian network. This improves the aggregation of different model outputs and allows one to seamlessly incorporate additional information.

Bayesian networks (Koller & Friedman, 2009)(Pearl, 1988) are probabilistic graphical models that represent the joint probability distribution (JDP) over a set of random variables. In other words, they capture the probabilistic relationships and dependencies between variables in a compact manner.

A Bayesian network can be formally defined as a tuple (G, P), where:

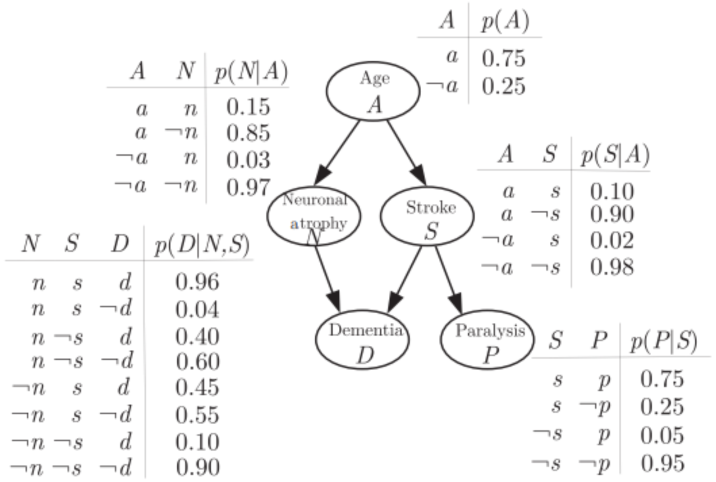
- G is a directed acyclic graph (DAG) representing the conditional (in)dependencies among a set of random variables $\{X_1, \ldots, X_n\}$.
- P is a set of conditional probability distributions (CPDs) associated with each variable and its parents in the graph G.

G consists of nodes representing the random variables and directed edges representing the probabilistic dependencies between variables. For each variable X_i, there is a corresponding node in G, and the arcs represent the influence between variables as illustrated in Figure 2.1.

P specifies the conditional probabilities for each variable given its parents in the graph G. Each node X_i in G has an associated CPD that defines the conditional probability distribution $p(X_i|\mathbf{Pa}(X_i))$, where $\mathbf{Pa}(X_i)$ denotes the parents of X_i in G.

Formally, the JDP defined by the Bayesian network is given by the product of the individual CPDs:

$$p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i|\mathbf{Pa}(X_i)) \tag{2.1}$$



A	$p(A)$
a	0.75
$\neg a$	0.25

A	N	$p(N\|A)$
a	n	0.15
a	$\neg n$	0.85
$\neg a$	n	0.03
$\neg a$	$\neg n$	0.97

A	S	$p(S\|A)$
a	s	0.10
a	$\neg s$	0.90
$\neg a$	s	0.02
$\neg a$	$\neg s$	0.98

N	S	D	$p(D\|N,S)$
n	s	d	0.96
n	s	$\neg d$	0.04
n	$\neg s$	d	0.40
n	$\neg s$	$\neg d$	0.60
$\neg n$	s	d	0.45
$\neg n$	s	$\neg d$	0.55
$\neg n$	$\neg s$	d	0.10
$\neg n$	$\neg s$	$\neg d$	0.90

S	P	$p(P\|S)$
s	p	0.75
s	$\neg p$	0.25
$\neg s$	p	0.05
$\neg s$	$\neg p$	0.95

Figure 2.1: Example of Bayesian network (Bielza & Larrañaga, 2021).

Using different properties (Bielza & Larrañaga, 2021), we can derive conditional independencies encoded by a BN. These play a critical role in understanding the behavior of a distribution. They are essential not only for gaining insights into the relationships between variables, but also for efficient query answering and inference.

In a BN, each variable is conditionally independent of its non-descendants, given its parents. The descendants of a node X_i are all the nodes reachable from X_i by repeatedly following the arcs. Then it is said that G satisfies the local Markov property. This property implies that a variable is only dependent on its immediate parents in the graph, and it is independent of all other variables in the network once its parents are known.

Additionally, the global Markov property states that a variable is conditionally independent of all other variables in the network, given its Markov blanket. The Markov blanket of a variable consists of its parents, children, and the other parents of its children.

2.4.1 Semiparametric Bayesian Networks

In a parametric BN for continuous domains, the CPDs are specified parametrically, assuming a specific (usually Gaussian) form and a fixed number of parameters. This approach works well when the underlying distribution conforms to the assumed form. However, in real-world scenarios, the true underlying distribution may not strictly adhere to a specific form, leading to potential inaccuracies or limited flexibility in modeling.

Semiparametric Bayesian networks (SPBN) address this limitation by incorporating nonparametric components into the modeling process. SPBNs (Atienza et al., 2022b) are a variant of BNs that combines parametric and nonparametric modeling approaches. These networks provide a framework allowing for a balance between the flexibility of nonparametric models and the bounded complexity and efficiency of parametric models.

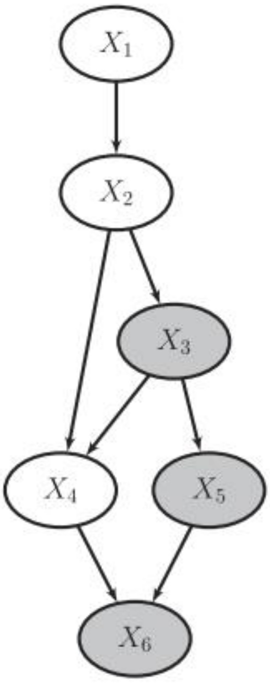


Figure 2.2. Graph structure of an SPBN (Atienza et al., 2022b).

It is important to note that semiparametric BNs may require more data to estimate the nonparametric components accurately, as they rely on the empirical distribution. Furthermore, the computational complexity of inference in SPBNs can be higher compared to standard BNs due to the increased flexibility and complexity of the models.

SPBNs are composed of parametric and nonparametric CPDs. For the parametric CPDs, linear Gaussian (LG) CPDs are used, as they are easy to train and usually provide good performance when there is a linear relationship between the variables. Let X_i be a random variable following an LG conditional distribution, then, the conditional distribution of X_i given $\mathbf{X}_{\text{Pa}(i)}$ can be formulated as:

$$f\left(x_i \middle| \mathbf{x}_{\text{Pa}(i)}\right) = \mathcal{N}\left(\beta_{i0} + \sum\nolimits_{k \in \text{Pa}(i)} \beta_{ik} \cdot \mathrm{x}_k, \sigma_i^2\right) \tag{2.2}$$

where $\mathcal{N}$ is the normal probability density function with variance σ_i^2; β_{ik} is the regression coefficient for variable X_k, $k \in \text{Pa}(i)$, in the linear regression of variable X_k, and β_{i0} is its intercept.

The nonparametric CPDs are represented as the ratio of two joint kernel density estimation (KDE) models. This type of CPDs is denoted as conditional kernel density estimation (CKDE) distributions. Then, let X_i be a random variable following a CKDE conditional distribution, then, the conditional distribution of X_i given $\boldsymbol{X}_{\text{Pa}(i)}$ is defined as:

$$\hat{f}_{CKDE}\left(x_i \middle| \mathbf{x}_{\text{Pa}(i)}\right) = \frac{\hat{f}_{KDE}\left(x_i, \mathbf{x}_{\text{Pa}(i)}\right)}{\hat{f}_{KDE}\left(\mathbf{x}_{\text{Pa}(i)}\right)} = \frac{\sum_{j=1}^{N} K_{\mathbf{H}}\left(\begin{bmatrix} x_i \\ \mathbf{x}_{\text{Pa}(i)} \end{bmatrix} - \begin{bmatrix} x_i^j \\ \mathbf{x}_{\text{Pa}(i)}^j \end{bmatrix}\right)}{\sum_{j=1}^{N} K_{\mathbf{H}-i}\left(\mathbf{x}_{\text{Pa}(i)} - \mathbf{x}_{\text{Pa}(i)}^j\right)} \tag{2.3}$$

where $x_{(i)}^{j}$ and $\mathbf{x}_{\mathrm{Pa}(i)}^{j}$ are the values of the j-th training instance among N instances for the variables X_i and $\mathbf{X}_{\mathrm{Pa}(i)}$, respectively. $\mathbf{H}$ is a symmetric positive definite $n \times n$ matrix called bandwidth. A bandwidth matrix can be used to define the smoothness of density estimation. Higher values in a bandwidth produce smoother densities, while smaller values generate wiggly density estimations. $K(x)$ is an n-variate kernel function that integrates to 1, and $K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-1/2}K\left(\mathbf{H}^{-1/2}\mathbf{x}\right)$. A Gaussian kernel, $K(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}}\exp\left(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{x}\right)$, is typically used since it is a well-known distribution with remarkable theoretical properties. Specifically, when a Gaussian kernel is used, the KDE model is equivalent to a Gaussian mixture model with an equiprobable component for each training instance.

In the SPBN model, the graph contains the type of each node, which determines the type of the corresponding CPD. There are no restrictions on the arcs, so the parent sets of each variable can be of different types: only LG parents, only CKDE parents or a mix of both options. LG and CKDE nodes are represented by white shaded and gray shaded nodes, respectively, in Figure 2.2.

2.4.1.1 Parameter Learning

The maximum likelihood estimation (MLE) is used to estimate the parameters of the CPDs based on the observed data, under the assumption that the network structure is known. The MLE involves finding the parameter values that maximize the likelihood of the observations.

Let $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$, with $\mathbf{x}^j = (x_1^j, \ldots, x_n^j)$, be a set of N independent and identically distributed training instances, and $\boldsymbol{\theta}$ denote a particular set of parameters. Then, the likelihood function is defined as the density assigned to the training data $\mathcal{D}$ by the Bayesian network:

$$\mathcal{L}(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{G}) = \prod_{j=1}^{N} f(\mathbf{x}^j \mid \boldsymbol{\theta}, \mathcal{G}) = \prod_{j=1}^{N} \prod_{i=1}^{n} f\left(x_i^j \mid \boldsymbol{\theta}_i, \mathbf{x}_{\mathrm{Pa}(i)}^{j}\right) \tag{2.4}$$

where $\boldsymbol{\theta}_i$ is the set of parameters for a CPD of node i. This set, for a random variable X_i, is defined depending on the distribution it follows (Atienza et al., 2022b).

2.4.1.2 Structure Learning

The structure learning process is used to automatically infer the graphical structure of the network from data. It determines the accuracy and interpretability of the model. Several algorithms have been developed to automate this task. One of these is the greedy hill-climbing (HC) that provides a simple and efficient approach to learn the structure.

HC is a search algorithm whose goal is to find an optimal network structure by iteratively performing local improvements based on a scoring metric through a set of operators. It explores the search space of possible Bayesian network structures locally. At each step, the operator that produces the highest score improvement is applied to generate a new candidate structure. The algorithm runs until no further improvement is possible or until a stopping criterion is met.

In general, three operators are utilized in HC: arc addition, arc removal, and arc reversal. In SPBNs, the structure consists of arcs in a graph and the types of nodes, namely, LG or CKDE conditional distributions. Then, a new operator is added into the HC algorithm to learn SPBNs: node type change.

The definition of a score function is an important part of the score and search algorithms. Commonly used scoring metrics include the log-likelihood function and Bayesian information criterion (BIC). However, for an SPBN, any score including the log-likelihood of the training data, such as the maximum log-likelihood score or BIC, is inappropriate because the training data constitute part of the KDE model. Instead, Atienza et al. (2022b) propose to use the k-fold cross-validated log-likelihood:

$$\mathcal{S}_{CV}^{k}(\mathcal{D},\mathcal{G}) = \sum_{m=1}^{k} \mathcal{L}\left(\mathcal{G},\boldsymbol{\theta}_{\mathcal{J}_{train}^{m}} : \mathcal{D}_{\mathcal{J}_{test}^{m}}\right) \tag{2.5}$$

where $\mathcal{L}\left(\mathcal{G},\boldsymbol{\theta}_{\mathcal{J}_{train}^{m}} : \mathcal{D}_{\mathcal{J}_{test}^{m}}\right)$ is the log-likelihood of the m-th test fold data element in a model composed of graph $\mathcal{G}$ and parameters $\theta_{\mathcal{J}_{train}^{m}}$. The log-likelihood, can be also expressed as a sum of terms:

$$\mathcal{L}(\mathcal{G},\boldsymbol{\theta} : \mathcal{D}) = \sum_{j=1}^{N} \sum_{i=1}^{n} \log f\left(x_i^j \mid \boldsymbol{\theta}_i, \mathbf{x}_{\mathrm{Pa}(i)}^j\right) \tag{2.6}$$

The $\boldsymbol{\theta}_{\mathcal{J}_{train}^{m}}$ parameters are estimated based on the data $\mathcal{D}_{I_{test}^{m}}$ using the parameter learning procedure described in Section 2.4.1.1.

It is common in k-fold cross-validation that $\mathcal{J} = \left\{\mathcal{J}_{test}^{i}\right\}_{i=1}^{k}$ corresponds to k disjoint sets of instance indices, and $\mathcal{J}_{train}^{i} = \cup_{j\neq i} \mathcal{J}_{test}^{j}$, for all $i = 1, \ldots, k$. The contribution of each node depends on its type.

Further, the score used in the learning stage of Bayesian networks possesses the decomposability property, which allows it to be expressed as the sum of local score terms associated with each node and its parents, given a specific selection of indices representing disjoint folds of data. To take advantage of this decomposability property during learning, it is necessary to fix a particular set of indices, denoted as $\mathcal{J}$. However, this can potentially lead to over-fitting of that set. To mitigate this problem, Atienza et al. (2022b) divide the available data into two disjoint sets: the training set $\mathcal{D}^{train}$ and the validation set $\mathcal{D}^{val}$. The learning process is guided by $\mathcal{D}^{train}$, whereas $\mathcal{D}^{val}$ fixes the overfitting by measuring the goodness of the new structure at each iteration as:

$$\mathcal{S}_{\text{validation}}\left(\mathcal{D}^{train},\mathcal{D}^{val},\mathcal{G}\right) = \mathcal{L}\left(\mathcal{G},\boldsymbol{\theta}_{D^{train}}:\mathcal{D}^{val}\right) \tag{2.7}$$

where $\boldsymbol{\theta}_{D^{train}}$ are the parameters estimated using the full training set $\mathcal{D}^{train}$.

2.4.2 Anomaly Detection with Bayesian Networks

BNs can be used to model the normal behavior in a network communication domain by capturing the relationships between the variables. The graphical structure of the network and the associated probabilities provide an intuitive representation of the relationships between the features. This allows security analysts to understand the reasoning behind detection decisions.

Anomaly detection typically consists of two main phases: training and inference. During the training phase, the BN is constructed using a known normal network traffic dataset. The network parameters are estimated from this observed normal data. Once the BN is trained, it can be used for inference during the anomaly detection phase. When a new communication case is found in the network, the BN is used to calculate the likelihood of observing that case given the learned network structure and parameters. If this value falls below a given threshold, it indicates that the observed case deviates significantly from the normal learned behavior, suggesting an anomaly.

An advantage of using BNs for anomaly detection is their ability to handle uncertainty and incomplete information. IDSs often face uncertainty due to noise in the data. BNs can handle uncertainty and provide a principled way of reasoning about it. They can quantify uncertainty using probabilities, allowing for more nuanced analysis of events and better decision making in the presence of uncertainty.

In addition, BNs can incorporate prior knowledge and expert opinion into the model. In this way, analysts can contribute their knowledge and encode it in the form of CPDs or arcs.

2.5 Autoencoders

Deep learning (DL) has emerged as a powerful transformative technology in the industry revolutionizing different domains. It has achieved unprecedent levels of performance thanks to the improvement of computing capabilities which has reduced training time on much more complex models. In the IDS context, they are able to capture complex relationships and detect anomalies that traditional rule-based or statistical methods may miss. One of the most well-known DL techniques within anomaly detection are *autoencoders*.

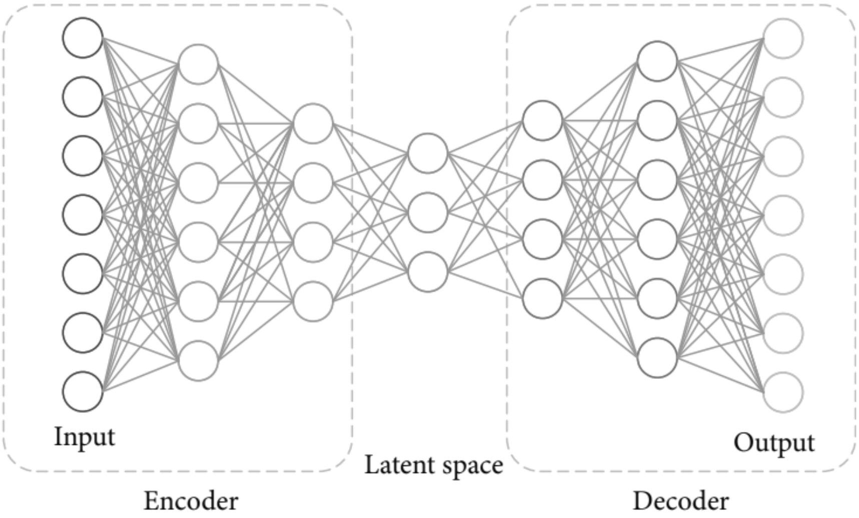


Figure 2.3. Deep autoencoder (Wang et al., 2020).

An autoencoder (Kramer, 1992) is a feedforward neural network trained to produce an approximation of the identity mapping between inputs and outputs using backpropagation or similar learning procedures. This is with the least possible amount of distortion. It is an unsupervised and semi-supervised learning framework composed of two modules (Zhou & Paffenroth, 2017), an encoder, and a decoder as illustrated in Figure 2.3. It maps the input to an output through these two phases

$$\bar{X} = D\big(E(X)\big) \tag{2.8}$$

where X is the input data and $\bar{X}$ is the recovered version of it. E is an encoding map from the input data to the hidden layer as in Equation (2.9) and D is a decoding map from the hidden layer to the output layer as in Equation (2.10),

$$h = \sigma(w_{xh}x + b_{xh}) \tag{2.9}$$

$$z = \sigma(w_{hx}h + b_{hx}) \tag{2.10}$$

where w and b are the weight and bias of the neural network, respectively, and σ is the nonlinear transformation function.

An autoencoder can be viewed as a solution to the following optimization problem as shown in Algorithm 1, where the idea is to train E and D to minimize the difference between X and $\bar{X}$:

$$\min_{D,E} \quad \left|\left|X - D\big(E(X)\big)\right|\right| \tag{2.11}$$

An autoencoder with more than one hidden layer is called a deep autoencoder and each additional hidden layer requires an additional pair of encoders $E(\cdot)$ and decoders $D(\cdot)$. By allowing many layers of encoders and decoders, a deep autoencoder can effectively represent complicated distributions over the input X.

Algorithm 1 Autoencoder training algorithm

```
INPUT: Dataset $\mathcal{D} = \{x_1, \dots, x_N\}$
OUTPUT: encoder $E$, decoder $D$
  $\boldsymbol{\theta}$ ← Initialize parameters
  repeat
    $error = \sum_{i=1}^{N} \|x_i - D\big(E(x_i)\big)\|$
    $\theta$ ← Update parameters using gradients of $error$
  until convergence of parameters $\boldsymbol{\theta}$
```

Autoencoders have proven to be versatile and powerful models with applications in various fields. Different types of autoencoders can be modified or combined to form new models for various applications. Since their introduction, they have been used as a dimensionality reduction technique. They compress the data by obtaining the most important features. On the other hand, variations of autoencoders can act as a generative model. They learn to generate data that resemble training examples, offering creative and exploratory possibilities. They are also useful in recommender systems. Autoencoders can capture user preferences to facilitate personalized experiences. In addition, autoencoders have attracted attention in anomaly detection due to their ability to identify deviations from learned patterns.

2.5.1 Anomaly Detection with Autoencoders

Autoencoders are becoming increasingly popular in addressing anomaly detection due to their proven effectiveness. The autoencoder process begins with training on normal data, which helps the algorithm to capture the underlying structure and the features associated with it. Once trained, the autoencoder tries to reconstruct the new observations using the knowledge stored during training and calculating the differences between the inputs and outputs or the reconstruction error, which serves as the anomaly score. Instances with a higher reconstruction error are considered potential anomalies.

Petsche et al. (1996) propose one of the first papers in DL focused on anomaly detection for rare events. They develop a model that called *autoassociator* (autoencoder) for imminent engine failures. In the network intrusion field, Zavrak & İskefiyeli (2020) focus on the detection of anomalous network traffic from flow-based data using two types of autoencoders. Finding a threshold that serves as a boundary between normal and anomalous behavior is crucial. Aygun & Yavuz (2017) propose two autoencoder models whose threshold value is determined using a stochastic approach instead of those available in most of the literature.

It should be noted that autoencoders may have difficulty detecting local anomalous behavior. This limitation can be addressed by incorporating additional techniques, such as recurrent neural networks, which improve identification capabilities by providing more comprehensive analysis of contextual dependencies over time.

2.6 Recurrent Neural Networks

Continuous advances in recurrent neural network (RNN) technologies have transformed the way sequential data is analyzed and modeled today. Resulting from these advances, these frameworks hold an important place in modern machine learning methodologies. Unlike traditional networks, these ones present loops, allowing information to persist. They can efficiently retain data from past inputs to follow temporal patterns with greater accuracy. RNNs are especially suitable for time series prediction and linguistic analysis where sequence understanding is paramount.

However, most RNNs became obsolete because of their inability to learn long-term memories. This was due to the vanishing gradient problem (Hochreiter, 1998), in which gradients decrease exponentially with time. This limitation hindered the ability of RNNs to capture and retain information from previous time steps, resulting in a loss of contextual understanding in long sequences. As a result, RNNs had difficulty effectively modeling complex temporal dependencies in the data.

With the introduction of advanced variants, which incorporated gating mechanisms to select information, significant progress was made. These innovations revolutionized the field, enabling them to capture long-term dependencies, leading to their widespread adoption.

2.6.1 Long Short-Term Memory

Long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) is a type of RNN architecture that addresses the vanishing gradient problem and enables the modeling of long-term dependencies in sequential data. An LSTM layer consists of a set of recurrently connected nodes, known as memory cells. Each cell can maintain its state over time, and non-linear gating units which regulate the information flow into and out of the cell. For easier comprehension, an LSTM is deconstructed into its elements from a high-level representation to a low-level one, following the fundamentals in Ranjan (2020).

2.6.1.1 Layer and Network Structure

A single input that receives the network is a time window, defined as $\boldsymbol{x}_{(T-\tau):T}$, of τ observations and p features. As it is illustrated in Figure 2.4, this is represented as a two-dimensional array with features and time-steps along the rows and columns, respectively.

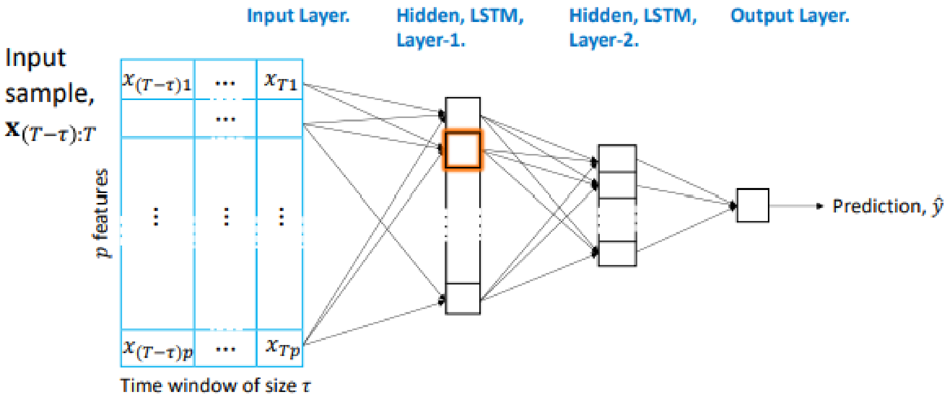


Figure 2.4. High-level representation of an LSTM network (Ranjan, 2020).

A batch of these windows is, thus, a three-dimensional array as shown in Figure 2.5. Each layer cell, represented with a blue box, takes all the time-step inputs and they are processed sequentially. The cells transmit their states within themselves to perform their internal operations and the connected arcs between the layers show the transmission of time-indexed information. The sequences are processed in the same order by the second LSTM layer and so on.

A LSTM can be either stateful or stateless, depending on the transmission mode. If the model is stateful, the cell state of the previous batch processing is retained and can be accessed by the next batch. In contrast, if the model is stateless, it processes each time-window independently. That is, there is no interaction or learning between two time-windows. The latter is the default when the sequence dataset is stationary.

In addition, the cells in an LSTM layer may return sequences or not. If they do, the cell outputs a sequence of the same length as the input. This approach is used when it is necessary to preserve the temporal structure. Otherwise, a cell emits only the last time-step output $\boldsymbol{h}_t$ that is an amalgam of the information present in all the cell states and the outputs of the previous cells.

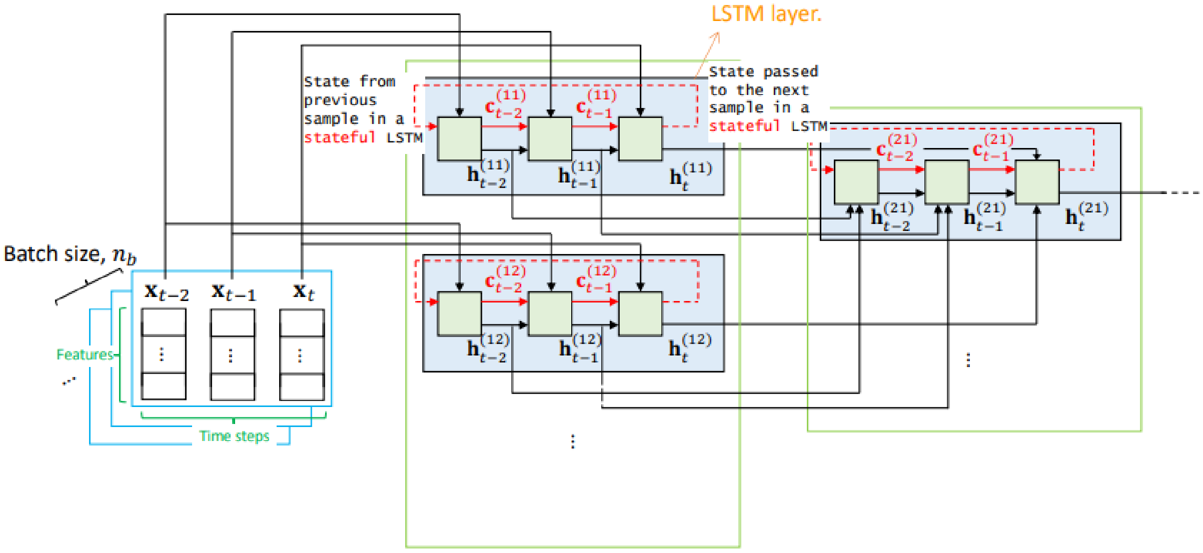

LSTM layer.
State from previous sample in a stateful LSTM
State passed to the next sample in a stateful LSTM
Batch size, n_b
Features
Time steps
$\mathbf{x}_{t-2}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_{t}$
$\mathbf{c}_{t-2}^{(11)}$
$\mathbf{c}_{t-1}^{(11)}$
$\mathbf{h}_{t-2}^{(11)}$
$\mathbf{h}_{t-1}^{(11)}$
$\mathbf{h}_{t}^{(11)}$
$\mathbf{c}_{t-2}^{(12)}$
$\mathbf{c}_{t-1}^{(12)}$
$\mathbf{h}_{t-2}^{(12)}$
$\mathbf{h}_{t-1}^{(12)}$
$\mathbf{h}_{t}^{(12)}$
$\mathbf{c}_{t-2}^{(21)}$
$\mathbf{c}_{t-1}^{(21)}$
$\mathbf{h}_{t-2}^{(21)}$
$\mathbf{h}_{t-1}^{(21)}$
$\mathbf{h}_{t}^{(21)}$

Figure 2.5. LSTM network input and hidden layers (Ranjan, 2020).

2.6.1.2 LSTM Cell

As previously discussed, the nodes of a layer are called LSTM cells. They are formed by one time-step iteration for each observation of the time window. These iterations, represented in green boxes in Figure 2.5 and Figure 2.6, consist in turn of two activation functions and three main gates: input gate i_t, forget gate f_t and output gate o_t. In Figure 2.5, the cell to which each iteration belongs is indicated by a super-index. For example, the super-index (12) refers to the second node of the first layer.

The parameters involved are, $\boldsymbol{w}_{\cdot}^{(h)}$, $\boldsymbol{w}_{\cdot}^{(x)}$, $b_{\cdot}$, where $\cdot$ is c, i, f, and o. A cell takes the prior output of all the other sibling cells in the layer. Given that the layer size is m, the prior output from the layer cells will be an m-vector $\boldsymbol{h}_{t-1}$ and, therefore, the $\boldsymbol{w}_{\cdot}^{(h)}$ has the same length m.

Within the cells, each time-step iteration takes in one time-step $\boldsymbol{x}_t$ and performs some operations to compute the output $\boldsymbol{h}_t$. Like the other RNNs, the hidden output $\boldsymbol{h}_t$ is transmitted to the next iteration and returned as a cell output.

In addition, LSTMs provide a distinctive property called *state* $\boldsymbol{c}_t$. It preserves information from the present to the past. Because of this, it is easier to detect patterns and links by having current and distant memories. In the diagram in Figure 2.6, the cell state is the horizontal line across the top of the diagram. This was not possible with former RNNs due to the gradient problem that quickly vanishes the intermediate outputs $\boldsymbol{h}_t$. Instead, the cell state stabilizes gradient by maintaining the memory.

Overall, the cell processes one observation at a time in a timed sequence window $\{\boldsymbol{x}_{T-\tau}, \boldsymbol{x}_{T-\tau+1}, \dots, \boldsymbol{x}_T\}$. $\boldsymbol{x}_t$ flows into the cell as input, it is processed along the paths in the presence of the previous output $\boldsymbol{h}_{t-1}$ and cell state $\boldsymbol{c}_{t-1}$, and yields the updated output $\boldsymbol{h}_t$ and cell state $\boldsymbol{c}_t$.

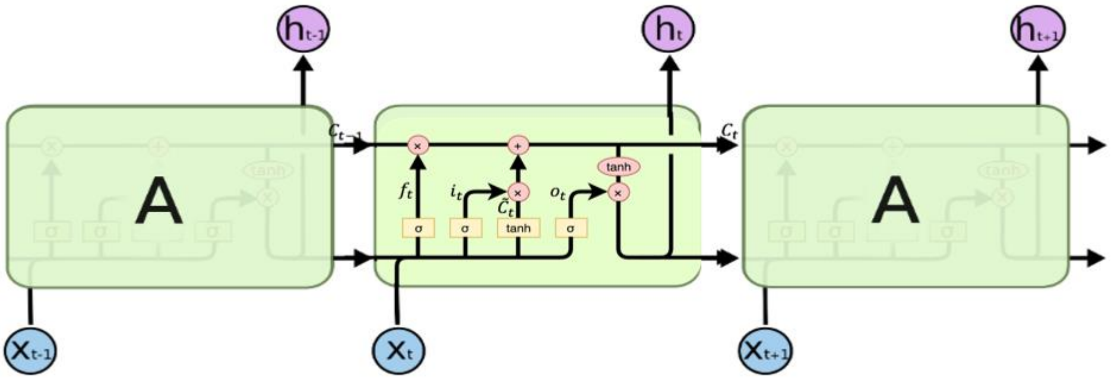


Figure 2.6. Time-step iterations in an LSTM cell (Olah, 2015).

2.6.1.3 LSTM Operations

The operation of an LSTM is divided into four sequential parts (Ranjan, 2020) in which the different components of the cell are applied. First, due to the new information that enters with $\boldsymbol{x}_t$, some of the previous memory may become irrelevant. In that case, that memory may be forgotten.

This forgetting decision is made in Equation (2.12):

It yields an indicator between 0 and 1 because of the sigmoid activated function. If the indicator is close to zero, the previous memory is forgotten. In this case, the information in $\boldsymbol{x}_t$ will replace the past memory. On the contrary, it means that the memory is still relevant and should be carried forward. This does not necessarily indicate that the information in $\boldsymbol{x}_t$ will not enter the memory.

$$f_t = \sigma\left(\boldsymbol{w}_f^{(\boldsymbol{h})}\boldsymbol{h}_{t-1} + \boldsymbol{w}_f^{(x)}\boldsymbol{x}_t + b_f\right) \tag{2.12}$$

The next step is to decide what new information at the time-step input $\boldsymbol{x}_t$ should be learned by the cell state. Equation (2.13) finds the relevant information in $\boldsymbol{x}_t$:

$$\tilde{c_t} = tanh\left(\boldsymbol{w}_{\tilde{c}}^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{w}_{\tilde{c}}^{(x)}\boldsymbol{x}_t + b_{\tilde{c}}\right) \tag{2.13}$$

where a *tanh* activation function is applied. This activation has negative and positive values in between -1 and 1.

It is possible that $\boldsymbol{x}_t$ has information, but it is redundant in the presence of the information already present with the cell of the previous $\boldsymbol{x}$. Therefore, Equation (2.14) is calculated with a sigmoid σ to have a value between 0 and 1:

$$i_t = \sigma\left(\boldsymbol{w}_i^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{w}_i^{(x)}\boldsymbol{x}_t + b_i\right) \tag{2.14}$$

A value closer to 0 would mean the information is irrelevant.

Equation (2.13) and Equation (2.14) find the information in $\boldsymbol{x}_t$, its relevance, and the memory requirement, respectively. These are combined to update the cell memory in Equation (2.15), adding the first component that determines whether to carry on the memory from the past:

$$c_t = f_t c_{t-1} + i_t \tilde{c_t} \tag{2.15}$$

The output gate in Equation (2.16) acts as a scale with value between 0 and 1:

$$o_t = \sigma\left(\boldsymbol{w}_o^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{w}_o^{(h)}\boldsymbol{x}_t + b_o\right) \tag{2.16}$$

In the last step, the output cell $\boldsymbol{h}_t$ is determined in Equation (2.17):

$$h_t = o_t tanh(c_t) \tag{2.17}$$

This output is based on a filtered version of the cell state c_t.

2.6.1.4 Bi-directional LSTM

In recent years, several improvements have been introduced in LSTMs to increase their performance. Schuster & Paliwal (1997) made a significant contribution by proposing a bi-directional RRN. As opposed to traditional approaches that process data in a forward direction, these models incorporate information from both past and future contexts by processing the input sequence in both directions simultaneously.

A bi-directional RNN (Salehinejad et al., 2017) considers the entire input sequence available both in the past and future for estimation of the output vector. For this purpose, one RNN processes the sequence from the beginning to the end in a forward temporal direction. Another RNN processes the sequence backwards, from the end to the beginning in a negative time direction. The outputs of the forward states are not connected to inputs of backward states and vice versa. Thus, there are no interactions between the two types of state neurons.

2.6.1.5 LSTM Autoencoder

An autoencoder processes flat data, but sometimes it is necessary to handle sequences. Therefore, an autoencoder can use LSTM units to deal with sequential information. Recurrent units in the encoder and decoder allow the autoencoder to capture the time dependencies that are present. Srivastava et al. (2016 is one of the early applications of this method.

An LSTM autoencoder is an unsupervised learning framework for sequential data that uses an encoder-decoder LSTM architecture. Each element of the sequence is processed one at a time, and the LSTM units retain memory of the previous elements. The encoder, consisting of LSTM cells, processes the input sequence, and compresses it into a lower-dimensional latent space representation. The LSTM cells in the encoder capture the temporal dependencies in the input sequence. Then, the latent space representation is passed through the decoder, which also consists of LSTM cells. The goal of the decoder is to reconstruct the original input sequence from the latent space representation.

Like the non-temporal autoencoder, the one that incorporates LSTM cells can identify anomalies. In this case, by analyzing sequences that deviate from the learned patterns.

Within the field of intrusion detection, Fernandez Maimo et al. (2018) develop a novel architecture composed by a low-level and high-level module to identify cyberthreats in 5G mobile networks. The first one uses Stacked Autoencoders (SAE) for anomaly symptom detection, followed by the second component for network anomaly detection. The latter is a LSTM that receives several streams of symptoms.

According to Clausen et al. (2021), recent evaluations show that the current anomaly-based network intrusion detection methods fail to reliably detect remote access attacks. This paper presents a model based on a bi-directional LSTM that is specifically designed to detect such attacks as contextual network anomalies.

It is also interesting combining neural network approaches with other types of machine learning methods for anomaly detection. Said Elsayed et al. (2020) propose a hyper approach based on LSTM Autoencoder and OC-SVM to detect anomaly-based attacks in an unbalanced dataset, by training the models using only examples of normal classes.

2.6.2 Gated Recurrent Unit

Due to its powerful capacity, LSTM has become the center of attention in DL and has been applied to multiple sequential tasks. The learning capacity of the LSTM cell is superior to that of the standard RNN cell. However, the additional parameters increase computational overhead. Therefore, the gated recurrent unit (GRU) was introduced by Cho et al. (2014).

To reduce the number of parameters, the GRU cell integrates the forget gate and input gate of the LSTM cell as an update gate. The GRU cell has only two gates: an update gate and a reset gate.

The update gate in a GRU determines how much of the previous hidden state should be retained and combined with the new input, as shown in Equation (2.18).

$$z_t = \sigma\left(\boldsymbol{w}_z^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{w}_z^{(x)}\boldsymbol{x}_t + b_z\right) \tag{2.18}$$

It considers both the current input and the previous hidden state, and outputs a value between 0 and 1 for each element in the hidden state. A value close to 1 indicates that the corresponding element in the hidden state should be updated, while a value close to 0 suggests that it should be mostly ignored.

On the other hand, the reset gate in Equation (2.19) determines how much of the previous hidden state should be forgotten or reset.

$$r_t = \sigma\left(\boldsymbol{w}_r^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{w}_r^{(x)}\boldsymbol{x}_t + b_r\right) \tag{2.19}$$

Like the update gate, it takes the current input and the previous hidden state as inputs and produces a value between 0 and 1 for each element in the hidden state. A value close to 1 suggests that the corresponding element should be preserved, while a value close to 0 indicates that it should be reset.

The cell output $\boldsymbol{h}_t$ is a linear interpolation, as shown in Equation (2.21) between the previous one $\boldsymbol{h}_{t-1}$ and the candidate $\tilde{\boldsymbol{h}}_t$. The latter is computed similarly to the one of the LSTM in Equation (2.20) but using the reset gate.

$$\tilde{h}_t = tanh\left(\boldsymbol{w}_{\tilde{h}}^{(h)}(r_t\boldsymbol{h}_{t-1}) + \boldsymbol{w}_{\tilde{h}}^{(x)}\boldsymbol{x}_t + b_{\tilde{h}}\right) \tag{2.20}$$

$$h_t = (1 - \boldsymbol{z}_t)\boldsymbol{h}_{t-1} + \boldsymbol{z}_t\tilde{\boldsymbol{h}}_t \tag{2.21}$$

Because of having two gates, one gating signal and its associated parameters are avoided. Consequently, GRUs have a simpler architecture which makes them easier to understand and implement. A reduced complexity leads to faster training and inference times, as well as lower memory requirements. However, since one gate is missing, the single GRU cell is usually less powerful than the original LSTM, although this may vary depending on the specific task.

Xu et al. (2018) compare different types of neural networks and RNNs using several datasets. The paper shows that GRU is more suitable as a memory unit for IDSs than LSTM. Moreover, its bi-directional version reaches the best performance compared with other methods.

2.7 Explainable Artificial Intelligence

In recent years, the widespread adoption of AI systems has raised concerns about their lack of transparency, leading to the emergence of eXplainable artificial intelligence (XAI). This focuses on developing techniques that provide understandable explanations for AI decisions, bridging the gap between the complex inner workings of algorithms and the need for human understanding and trust.

When talking about XAI, different terms are used that seem to be interchangeable at first. However, they refer to different concepts that help describe the characteristics of these systems. Barredo Arrieta et al. (2020) establish a terminology clarification of the two most used names in this field: explainability and interpretability. A model is considered to be interpretable if its design is itself understandable for a human. On the other hand, explainability is associated with the notion of explanation as an interface between human beings and a decision maker. It is linked to *post-hoc* explainability, as it encompasses the techniques used to convert an uninterpretable model into an explainable one.

Model accuracy is usually the main factor in evaluating and choosing a model. However, an improvement in understanding of a system can lead to correcting its deficiencies. There are three reasons to adopt XAI techniques in AI systems, according to Barredo Arrieta et al. (2020). First, interpretability helps to ensure fairness in decision making, i.e., to detect and correct biases in the training dataset. Second, it facilitates the provision of robustness by highlighting possible adverse perturbations that could change the prediction. And third, it can act as an insurance that only significant variables infer the outcome.

In addition to the advantages of its use in decision making, the current legislation establishes a regulatory framework for its implementation. The European Union's General Data Protection Regulation (GDPR) emphasizes the right to explanation for automated decisions, further underscoring the importance of XAI. This recognizes the potential risks associated with opaque systems and asserts the need for individuals to understand the significance of automated decisions affecting their lives.

Furthermore, the first European law on AI will be approved by the end of 2023. This legislation will be a stricter version of the initial proposal outlined by the European Commission in 2021 and will cover generative AI systems and the use of AI for biometric surveillance systems. Consequently, XAI becomes not only a technical consideration, but also a legal and ethical imperative.

The development of explainability within AI implies the possibility of its use in industry where decisions have a great impact and are critical. Its application has gained great popularity in several sectors facing the challenges associated with opacity. In healthcare, it is being used to improve clinical decision support systems, enabling doctors to understand the reasoning behind AI-generated diagnoses or treatment recommendations. In finance, XAI techniques are used to provide interpretable explanations of activities such as credit scoring, minimizing bias. In addition, XAI finds uses in autonomous vehicles, where interpretable explanations of AI-driven decisions are crucial for safety and public acceptance.

Gunning (2017) proposes creating a suite of machine learning techniques that produces more explainable models, while maintaining a high level of learning performance, as well as enables human users to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners. This trade-off between accuracy and interpretability, illustrated in Figure 2.7, is a fundamental consideration in the development of the models. Very complex models often achieve remarkable accuracy on a variety of tasks. However, their intrinsic operations are often opaque and challenging to interpret. On the other hand, simpler models are more interpretable. This allows humans to understand the decision-making process, but they may lose some level of accuracy.

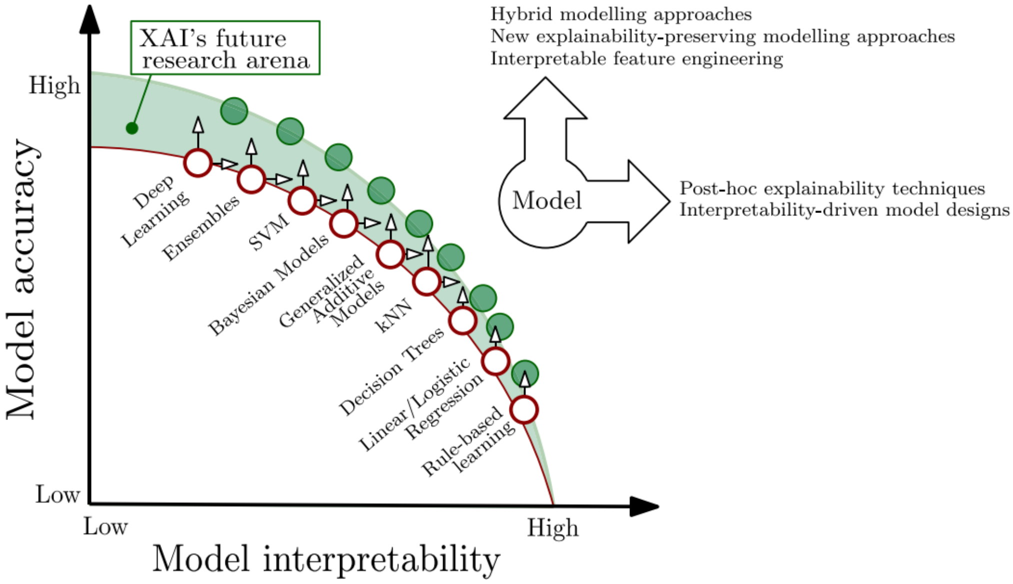

XAI's future research arena
High
Model accuracy
Low
Low
High
Model interpretability
Deep Learning
Ensembles
SVM
Bayesian Models
Generalized Additive Models
kNN
Decision Trees
Linear/Logistic Regression
Rule-based learning
Hybrid modelling approaches
New explainability-preserving modelling approaches
Interpretable feature engineering
Model
Post-hoc explainability techniques
Interpretability-driven model designs

Figure 2.7. Trade-off between model interpretability and accuracy (Barredo Arrieta et al., 2020)

Efforts are underway to develop techniques that strike a balance between accuracy and interpretability. These include *post-hoc* explanations and the use of hybrid models that combine the advantages of black-box models and transparent techniques. *Post-hoc* interpretability (Dosilovic et al., 2018) extracts information from already learned models and it does not precisely depend on how it works. The advantage of this approach is that it does not impact performance.

There are few contributions to explain recurrent models such as LSTM and GRU. The studies can be divided into two groups: explainability by understanding what a RNN model has learned and explainability by modifying RNN architectures to provide insights about the decisions they make. Some of the most common techniques in the first group are those called gradient-based methods. They can be used for feature relevance explanations as they analyze the contribution of each individual input and layer in the network tracing the gradients to the final output. On the other hand, a RNN considered as a black box can be explained by associating it a more interpretable model. Transparent methods, such as decision trees, can approximate the outputs of the network by learning a set of rules based on the input features. These rules provide a clear representation of how different variables and their specific values influence the final decision.

Chapter 3

Contribution Methodology

3.1 Network Traffic Data

Network traffic refers to the flow of data packets over a computer network. In a network, devices communicate with each other by exchanging these data packets, which contain information such as the source and destination addresses, payload data, and other information.

The TCP/IP model (Forouzan, 2002), also known as the internet protocol suite, is a conceptual framework that defines how data is transmitted and received over a network. It is named after two of its key protocols: the transmission control protocol (TCP) and the internet Protocol (IP). As illustrated in Figure 3.1, the TCP/IP model is composed of four layers, each responsible for specific functions in the communication process:

1. The application layer represents the interface between the network and the application software. It includes protocols such as hypertext transfer protocol (HTTP), file transfer protocol (FTP), simple mail transfer protocol (SMTP), and domain name system (DNS). These protocols enable tasks like web browsing, file sharing, email transmission, and domain name resolution.
2. The transport layer handles the reliable delivery of data between devices. The most prominent protocol in this layer is TCP, which provides connection-oriented and error-checked transmission. User datagram protocol (UDP) is another transport layer protocol that offers connectionless and unreliable transmission, suitable for applications where real-time data delivery is crucial.
3. The internet layer is responsible for addressing and routing packets across different networks. It uses IP to assign unique addresses to devices and facilitate packet routing. IP addresses, both IPv4 and the newer IPv6, are used to identify source and destination devices and enable communication across the Internet.
4. The network access layer deals with the physical transmission of data over the network medium. It includes protocols such as Ethernet, Wi-Fi, and point-to-point protocol (PPP). The link layer encapsulates the data into frames and handles the addressing of devices within a local network.

Various protocols work together across these layers to ensure proper communication and data transfer between devices. For example, when a user accesses a website, the HTTP protocol in the application layer is used to request web pages, which are then transported via TCP in the transport layer. The IP protocol in the Internet layer ensures the packets are correctly routed to the destination, and the link layer protocols handle the actual transmission over the physical medium.

Due to the limitations of publicly available network traffic datasets, the evaluation of IDSs has routinely faced some challenges. Until the mid-2000s, many of the existing datasets did not adequately represent modern network environments. In addition, they did not cover current attack scenarios (Moustafa & Slay, 2016).

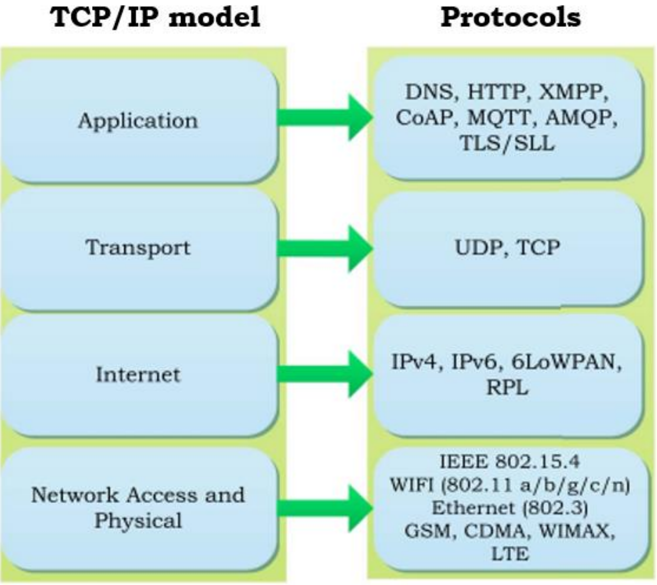


Figure 3.1. TCP/IP model and associated protocols.

Datasets play an important role in the testing and validation of any intrusion detection method. Therefore, this discrepancy makes it difficult to accurately assess the performance of these systems. A few datasets are publicly available for testing and evaluation of intrusion detection. The most widely used evaluation datasets in the last decades have been the KDD Cup 1999 and its modified version, the NSL-KDD dataset (Gogoi et al., 2012).

Both the KDD Cup 1999 and the NSL-KDD are evaluation datasets. The records in the dataset may be very different from the actual network traffic data. In addition, the nature of attacks and normal cases may change dynamically. One of the most important weaknesses of the KDD dataset is the large number of redundant records, which causes the learning algorithms to be biased towards frequent records and thus prevents them from learning from infrequent records, which may be more detrimental to the health of the network. Furthermore, the existence of these repeated records in the test set causes the evaluation results to be positively biased towards methods that have better detection rates on frequent records.

3.1.1 UNSW-NB15 Dataset

UNSW-NB15 Dataset (Moustafa & Slay, 2015) was created to address these limitations. It is produced by establishing the synthetic environment at the UNSW cybersecurity lab. The tool employed provides the capability to generate a modern representative of the real normal communications and the synthetical abnormal network. In contrast with the previous datasets, this one has a higher number of labeled attacks and updated traffic information.

The dataset contains 2,218,761 (87.35%) benign flows and 321,283 (12.65%) attack ones captured over 31 hours of simulation on three networks with 45 different computer addresses.

Some tools and algorithms process raw network packets and generate attributes of network flows. As a result, there are 47 features, described in Appendix A. These variables include a variety of packet-based features and flow-based features. The former help in the examination of the payload along with the packet headers. In contrast, for flow-based features only connected packets of network traffic are considered to keep computational analysis low rather than looking at all packets. The features from 1-35 represent the integrated gathered information from data packets and the rest are additional attributes from the matched ones.

The features are categorized into five groups (Moustafa & Slay, 2016):

1. Flow features: includes the identifying attributes between hosts (e.g., client-to-serve or server-to-client).
2. Basic features: involves attributes representing protocols connections.
3. Content features: encapsulates the attributes of TCP/IP; they also contain some HTTP service attributes.
4. Time features: contains the time attributes (e.g., inter-packet arrival time, packet start/end time and TCP round-trip time).
5. Additional generated features: this category can be further divided into two groups: general purpose features, where each feature has its own purpose, and connection features that are built from the flow of 100 record connections based on the sequential order of the last time feature.

Regarding data labelling, two attributes are provided: *label* whose value is 0 for normal behavior and 1 otherwise, and *attack_cat* that represents the nine attack categories and the normal connections described below.

1. Normal: natural transaction data.
2. Fuzzers: an attack in which the attacker attempts to discover security holes in a program, operating system, or network by feeding it massive random data input to cause it to crash.
3. Analysis: a type of variety intrusions that penetrate the web applications via ports (e.g., port scans), emails (e.g., spam), and web scripts (e.g., HTML files).
4. Backdoors: a technique of bypassing a stealthy normal authentication, securing unauthorized remote access to a device, and locating the entrance to plain text as it is struggling to continue unobserved.

5. DoS: an intrusion which disrupts the computer resources via memory, to be extremely busy to prevent the authorized requests from accessing a device.
6. Exploits: a sequence of instructions that takes advantage of a glitch, bug, or vulnerability to be caused by an unintentional or unsuspected behavior on a host or network.
7. Generic: a technique that establishes against every block-cipher using a hash function to collision without respect to the configuration of the block-cipher.
8. Reconnaissance: can be defined as a probe; an attack that gathers information about a computer network to evade its security controls.
9. Shellcode: an attack in which the attacker penetrates a slight piece of code starting from a shell to control the compromised machine.
10. Worms: attack whereby the attacker replicates itself to spread to other computers. Often, it uses a computer network to spread, relying on security flaws in the target computer to gain access to it.

3.2 Proposed framework

As described in Section 1.2, the main problems of anomaly detection at present are the high transmission rate associated with the new networks, the nature of the anomalies, the large number of false alarms and the poor interpretability and explainability of the methods used. These problems do not allow IDSs to function properly. It is therefore necessary to find suitable methods to solve each of these drawbacks.

After an analysis of the current state of the art of IDS and the different algorithms used, a series of specific solutions for the problems explained above are proposed. These solutions are materialized in the form of a global architecture and as specific components of it.

For the problem of high transmission rate, the aim is to work with variables specific to communications flows. In this way, it is not necessary to analyze the content of each of the packets, which means a higher processing overhead. Given the dataset proposed in Section 3.1.1 for the training and evaluation of the system, we finally work with both types of attributes, i.e., from flows and packets. Consequently, we have a better perspective of the network activity and can detect attacks that depend heavily on the content of the packets. In fact, many modern network security solutions employ a combination of these two approaches to improve their discovery capabilities (Andreas et al., 2021).

Another factor that determines the performance of IDSs is the type of attack to be detected. Depending on the characteristics of these, it is necessary to use one technique or another whose design is adjusted to the behavior of the attack. Therefore, considering the attacks present in the dataset, a relationship is established in Figure 3.2 with the types of anomalies described in Section 2.3. For example, in the case of collective anomalies, these represent a deviation from normal behavior when analyzed as a whole, being normal separately. This is the case for DoS attacks, where numerous requests for connection to a web server imply a collective anomaly, but a single one is legitimate. On the other hand, Probe (Reconnaissance) attacks are contextual in that they are based on a specific intention to obtain information about network security.

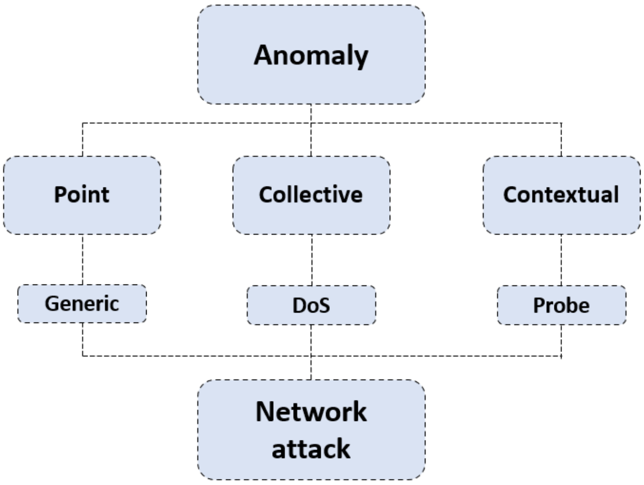


Figure 3.2. Mapping of some attacks with anomalies.

In order to obtain better performance in detecting non-point attacks, the aim is to use methods that have a memory of previous states, i.e., that can remember previous instances when looking for patterns. This helps to manage anomalies that are prolonged over time. A suitable solution is the RNN, and its more modern variants explained in Section 2.6, which are currently in demand in natural language processing. They allow processing sequential data, in this case network traffic.

On the other hand, another problem affecting the performance of anomaly detection IDSs is having false positives since they consider deviations from normal behavior as an attack. Moreover, this problem is aggravated by the lack of explanations accompanying the alerts. This prevents security analysts from certifying the veracity of alerts through their causes. Hence, the priority is the interpretability of decisions and if this is not possible, *post-hoc* explainability. Probabilistic graphical models such as BNs are an efficient approach to work with uncertainty, bringing interpretability to the processes without losing precision, as illustrated in Figure 2.7. On the other hand, as RNNs are characterized by being black boxes, explanatory methods are necessary to meet this objective. In addition to *post-hoc* techniques, association by means of transparent models helps in this task.

To combine the models described and justified above, a metaclassifier is chosen that fits specifically to the characteristics of the network traffic. Attacks occurring in real-world communications account for less than 1%. Most of the time no anomalies occur at all. This is also reflected in the datasets that show a significant imbalance, i.e., there is much more normal traffic than attacks. As indicated above, only 12% instances from the UNSW-NB15 dataset are attacks.

Likewise, this imbalance is also present in the set of attacks where some are more frequent than others. For example, generic attacks account for 67% of the 9 categories in the dataset as illustrated in Table 3.1. Consequently, it is necessary to set up a system to separate most of the normal traffic from suspicious observations and then analyze them in depth.

Type of attack	Percentage
Generic	67.07%
Exploits	13.87%
Fuzzers	7.55%
DoS	5.09%
Reconnaissance	4.35%
Analysis	0.83%
Backdoor	0.72%
Shellcode	0.47%
Worms	0.05%

Table 3.1. Percentage of type in the testing set of UNSW-NB15 dataset.

Cascading (Alpaydin, 2020) is a type of ensemble learning in which there is a base sequence of computer classifiers in terms of complexity. After the first classifier, the following classifiers capture those instances of uncertainty, e.g., the log-likelihood given by a BN is lower than a given threshold. However, since there is a large imbalance in the field of intrusion detection and false alarms are common in anomaly detection, it is decided to change the approach. Those instances that are suspected of being attacks are passed to the next more complex classifier for analysis. In addition, it is important that this initial filtering is efficient. Therefore, it is intended to use a first algorithm that is simple and with a low computational cost.

3.2.1 Approach A

Approach A, shown in Figure 3.3, follows the criteria explained above about the framework. First, it is composed of a component dedicated to point anomaly detection, which is exactly a Bayesian network. This model learns the normal behavior of the network traffic and calculates the log-likelihood for the instances that receives through the learned structure. This value represents the probability of observing a network flow given the BN parameters. If this value is lower than a previously defined threshold, the window in which the instance is in a certain position is sent to the second component. Otherwise, it is defined as normal behavior.

The second component aims to validate the instances classified by the first component by adding a different approach for the detection of collective and contextual anomalies. Since the first component presents a more flexible threshold, i.e., anomaly detection to discard most normal traffic, the second block performs a more accurate detection to determine if the instance is indeed anomalous. For this, it is proposed to use an advanced RNN autoencoder, e.g., LSTM, bi-directional and, GRU that is trained in an unsupervised manner through normal datasets. Therefore, the autoencoder is operated by advanced RNN cells and can receive flow windows of size *n* to consider previous observations. The autoencoder after encoding and decoding the window, produces a new sequence that is compared with the initial one received. This results in a reconstruction error for each of the elements of the buffer.

Depending on the criteria followed, the error of a particular window element is chosen and compared with a threshold. This limit is previously computed on the basis of the training. If the error is higher than the threshold, then the observation is considered an attack and a notification is generated. If not, it will be indicated as normal. This component, lacking interpretability, incorporates methods of explainability. On the one hand, it uses the layer-wise relevance propagation technique to identify feature importance. On the other hand, a decision tree is used as a transparent design method to show the factors that may have influenced the decision of the component. This tree is trained from the different categories of attacks in order to perform multiclassification tasks.

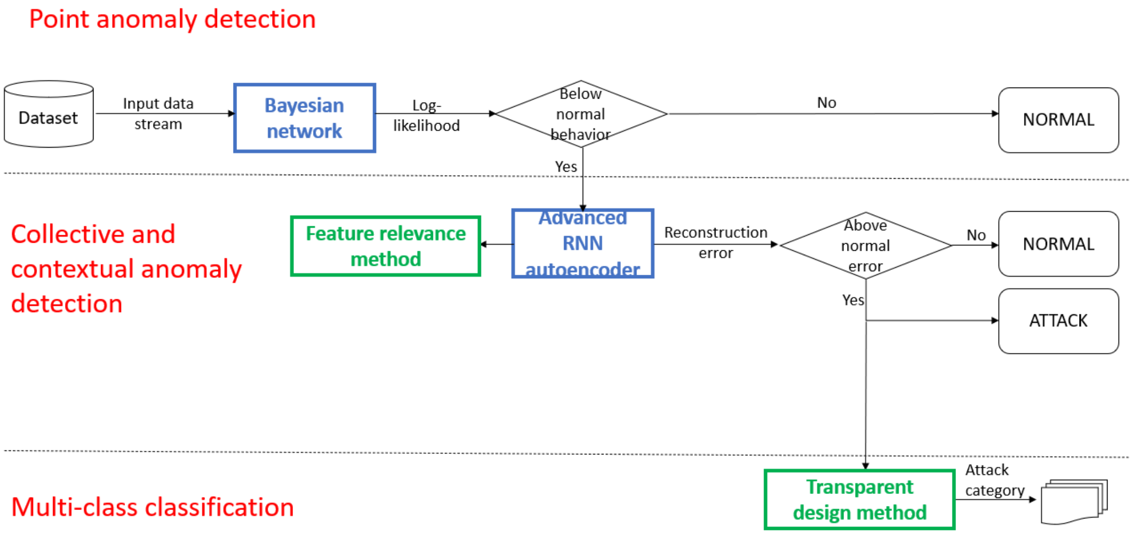


Figure 3.3. Approach A architecture.

3.2.1.1 Point Anomaly Detection

The first component of Approach A is a module of several BNs that receives network traffic and generates a log-likelihood value for each instance. As explained, this value is the probability of observing an instance given the BN parameters that are obtained during the training process using only normal network traffic data.

The initial idea to start the development of the system was to build a single BN that represents, consequently, all the normal traffic of the network. This first approach assumes that all communications are practically identical or very similar. However, there are differences in the normal traffic based on the type of connection made. Therefore, a single structure does not allow to generalize all the normal states and it is convenient to group the traffic according to certain variables. The protocols associated with the transport layer, TCP and UDP, differ fundamentally in their connection orientation and degree of reliability. This results in differences in network traffic that can be observed in several aspects. For example, the packet size in TCP is usually longer due to additional information to control the connection. Or, for example, in UDP the nature of the traffic tends to be more burst in comparison as it is used in real-time apps where timeliness is more important. Also, there are some differences between flows of the same transport layer protocol due to the application layer. This refers to the task for which the communication is intended. For example, HTTP, that is used for web browsing, involves small data payloads and short-request response cycles. On the other hand, FTP, that is used for file transfers, results in long connections with a high volume of traffic.

Therefore, in order to represent normal traffic more accurately, one BN is used for each type of transport protocol (TCP and UDP) and application (HTTP, FTP, FTP-Data, SMTP and DNS). This means that for each new flow received, its protocols are analyzed through their corresponding variables to send it to the corresponding BN of the module.

The selection of variables for the BNs is a very important process in determining whether a connection is anomalous or not. There are many options such as correlation analysis to eliminate highly related variables that provide redundant information.

However, this depends again on the protocols, so it is proposed to choose those variables that have a greater difference between their anomalous and normal distribution. For example, for a certain transport and application protocol flow, the marginal distributions of their variables are generated. On one side, for the traffic labeled as normal and on the other, the anomalous. A comparison is made between the two and those with the greatest difference are selected. In the variables of Figure 3.4, attributes A, B and D would be chosen since C is similar and would not contribute added value to the objective.

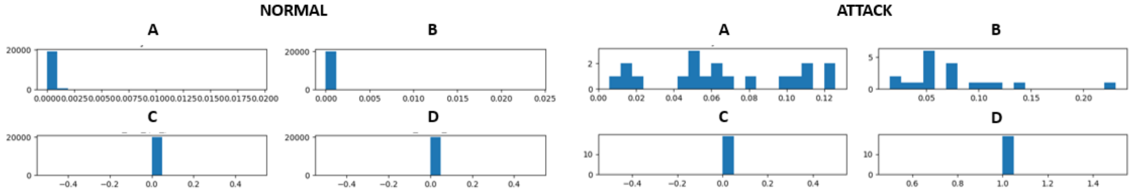


Figure 3.4. Example of the distribution of four variables in normal data and in attacks.

Once the most relevant variables for each BN are known, we proceed to their training using a Python package called PyBNesian (Atienza et al., 2022a). It is important to know the characteristics of the data when selecting the type of BN. The data is not homogeneous since they can be continuous or discrete and Gaussianity cannot be guaranteed. Consequently, an SPBN is the most suitable probabilistic model.

To create the SPBN structure, it is necessary to use a learning algorithm. HC is chosen among other options as the PC algorithm because the former tends to be more computationally efficient in performing local search operations. In addition, it can handle both discrete and continuous variables and the PC is less flexible. Furthermore, the structure score is another very important parameter in the case of score-and-search algorithms such as HC. Since the aim is to evaluate the goodness of a BN with respect to the data, the log-likelihood score is the most appropriate criterion. It seeks to maximize this goodness without considering other factors such as the complexity of the network.

During the learning process, a series of learning operators are also employed to perform small, local changes to the BN structure. Some make changes to the arcs and others apply changes to the nodes related to their type. Finally, it is necessary to learn the CPDs to make inferences based on the normal behavior of the network.

The distinction between anomalies and normal data is performed by a threshold given the log-likelihood of the flow. Each of the BNs of the module has a threshold determined by the transport and application protocols. From the same sample of normal traffic used for training, the log-likelihoods of the flows are obtained for the corresponding BNs. The threshold for each set of protocol is the smallest value obtained. This value approximates how far a normal connection can be from the trained model. Any value below this threshold is considered anomalous and is sent to the next component.

3.2.1.2 Collective and Contextual Anomaly Detection

The second component of the system is an autoencoder with recurring units such as LSTM or GRU. It allows to detect both point and collective or contextual anomalies. Hence, it is used to discard false positives of the first anomaly detection and to certify that they are indeed anomalous communications.

Unlike the previous component, only a single autoencoder is used for all traffic without grouping network traffic by protocols. Therefore, the present module makes use of a single set of variables. The attributes used are all those that are numerical since they are considered to provide value on the operation of the network. No other variable selection techniques are used prior to model training since this type of model provides significant improvements in dimensionality reduction as it is designed for that purpose.

A very important step in the data preprocessing at the time of training and testing is standardization. This involves transforming the input features of the dataset to have zero mean and unit variance. This makes all the variables have a similar scale avoiding that those with a larger scale dominate the training producing a bias. Also, this process mainly improves the convergence of the optimization algorithm by preventing it from being slow and unstable.

On the other hand, when preparing sequential data for modeling RNN architectures, temporalization of the data is necessary. A usual dimensionless input, also referred as planar data, does not directly provide time windows so it does not allow to extract temporal patterns. Therefore, it is decided to apply a sliding window approach whereby one starts at the beginning of the data and slides the window progressively along the data one at a time. This approach creates overlapping sequences of equal size.

The window size is also a fundamental factor when organizing the data. It determines the amount of past information the model has access to for predictions. The frequency and duration of patterns in the data must be considered when choosing the right size. There are attacks that may be short with abrupt changes while others persist over longer periods of time. Another very important factor is the computational cost since larger windows result in a larger number of parameters and consequently longer training and detection time. During the system evaluation process different window sizes are tested for comparison.

The architecture of the RNN autoencoder also influences the anomaly detection performance. It consists of an encoder and a decoder of equal size that produce a mirror effect structure where the size of the input layer and output layer is the same. The model is built using the free and open-source deep neural networks library TensorFlow (Martín Abadi et al., 2015). For the selection of the number of hidden layers and neurons a grid search of powers of 2 is used. It is observed that there are hardly any differences between complex autoencoders with several hidden layers and many units and other simpler models. However, there is a considerable increase in computation time in relation to the complexity of the network. Therefore, the architecture shown in Figure 3.5 is chosen. The input layer and output layer size are 37. This is the number of features used. Both the encoder and the decoder have two hidden layers of 128 and 64 units, respectively. The size of the latent representation layer also influences the performance. A larger latent size allows capturing more complex patterns but can lead to overfitting. In addition, it tends to require more computational resources, so a reduced latent size of 5 is chosen. It is important to note that all these values are multiplied by the window size as the model receives a sequence of flows. Additionally, a dropout of 0.2 is used as a regularization technique to avoid overfitting during training and to improve the generalization of the model.

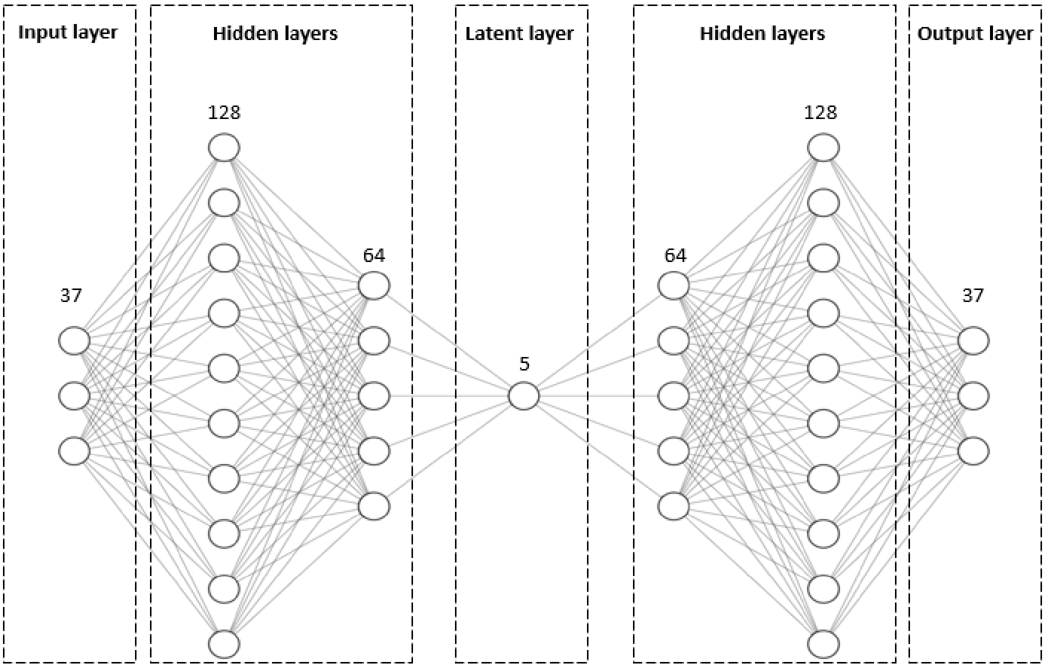


Figure 3.5. Advanced RNN autoencoder architecture.

During training, the advanced RNN autoencoder aims to minimize the reconstruction loss using the mean squared error (MSE) as the loss function. The Adam optimizer (Kingma & Ba, 2014) is employed to update the model parameters. The training process involves iteratively feeding the input data to the network, computing the reconstruction error, and backpropagating the gradients to update the weights of the model. The number of times the entire dataset is passed through the autoencoder during training is usually around 50 epochs. Moreover, a batch size of 64 is chosen, which strikes a balance between computational efficiency and model convergence.

The determination of a flow as an anomaly is done by using a threshold that marks the limit between the reconstruction errors of normal connections and attacks. There are several ways to obtain an optimal threshold. On the one hand, as in the first component of the system, the reconstruction errors for the normal flows used during training can be calculated. In this case, the threshold would be the maximum of those obtained or that corresponding to a very high percentile. However, it is decided to use a stochastic procedure (Aygun & Yavuz, 2017) that searches for the threshold with the best performance in the detection of anomalies. This function checks the false positive rate (FPR) and false negative rate (FNR) for a set of thresholds and chooses the one that produces the lowest FPR given a maximum FNR. This maximum is normally set to 0.05. The evaluation of the LSTM autoencoder through the range of thresholds is performed using a validation set with both kinds of flows i.e., normal, and anomalous. Algorithm 2 shows a pseudocode of this procedure.

Algorithm 2 Advanced RNN Autoencoder threshold algorithm

```
INPUT: Training set $ts$, validation set $vs$, validation set labels $lb$, LSTM autoencoder $ae$, minimum threshold $th_a$, maximum threshold $th_b$, maximum FNR $m$
OUTPUT: Threshold $best\_th$
  train $ae$ with $ts$
  $errors$ ← validate $ae$ with $vs$
  for $th$ ← $th_a$ to $th_b$ do
    $fpr$ ← calculate_fpr by using $errors$ , $lb$, and $th$
    $fnr$ ← calculate_fnr by using $errors$ , $lb$, and $th$
    if $fnr$ is less than $m$ and $fpr$ is less than $best\_fpr$ then
      $best\_fpr$ ← $fpr$
      $best\_fnr$ ← $fnr$
      $best\_th$ ← $th$
    end
  end
```

3.2.1.3 *Post-hoc* Explainability

In order to add explainability to the decisions made by the recurrent units that make up the autoencoder, a gradient-based method was chosen. One of the most commonly used is called gradient attribution (Ancona et al., 2017). This method consists of calculating the gradients of the model output with respect to the input features. In the case of RNNs, these gradients can be calculated with respect to the input sequence at each time step.

The gradients of the model output are computed once it is trained with respect to the input features. Then, they must be normalized to ensure that relevance scores are comparable between different features or time steps. Then, these values can be interpreted as feature relevance scores. Higher positive scores indicate features that contribute positively to the model outcome, while lower or negative scores indicate features that have less impact or even suppress the outcome.

In addition to the feature relevance methods, explanability can be added to the RNN autoencoder by associating a more interpretable model. Among the different possible options, the decision tree is chosen because it is usually one of the best performing and provides transparency in decision making. It adds explainability to the system by classifying instances that the autoencoder considers anomaly by showing the rules that condition the decision. It is also characterized by its scalability since its time complexity is generally logarithmic in the number of training samples. Therefore, it is suitable for contexts with a large number of instances.

The two components of the system only indicate whether a given network flow is anomalous or not. Then, it is convenient to include a model for multiclassification that indicates the specific type of attack when the communication is anomalous. A decision tree naturally handles this type of tasks by straightforward branching, so it is very adequate for this case.

For the model training it is necessary to use a sample with normal and anomalous behavior of the network. A set comprising all the attacks in the dataset is obtained. However, this set is unbalanced since it does not have the same number of classes within the attacks. Therefore, as a first preprocessing step it is necessary to use an oversampling technique to address the class imbalance. Specifically, SMOTE, explained in Section 2.3.1, is applied since it preserves the information and diversity of the classes. Subsequently, it is proceeded to the selection of variables because, unlike the previous preprocessing, this procedure generates improvements in this case. Accuracy is improved by eliminating redundant variables and, most importantly, it promotes a better generalization to unseen data by simplifying the structure and reducing complexity. It also enhances the interpretability of the model to gain insights into underlying relationships between features and the target variable. The selection is performed by recursive feature elimination (RFE) also using a decision tree as a base estimator. A grid search of 5 to 10 variables is employed to find the optimal number of variables while maintaining a low dimensionality that favors interpretability.

The construction of the model is performed using the *DecisionTreeClassifier* from the scikit-learn library (Pedregosa et al., 2011) that is capable of performing multi-class classification.

It is important to note that the decision tree is also trained with normal network behavior in addition to the different types of attacks. Although the second component of the system sends only instances considered as attacks, the present classifier can also define the received instances as normal. This may result in a discrepancy between the outputs of the autoencoder and those from the decision tree as a transparent, multi-classifier method. Therefore, there are two approaches to handle this problem. On the one hand, preserving the decision of the RNN autoencoder and, on the other hand, setting as final label the decision of the classification tree when it indicates that it is a normal communication. The latter is indicated in Algorithm 3. The labels included in *lb_old* and *lb_new* are either 0 if the instance is normal or 1 if it is anomalous.

Algorithm 3 Balancing algorithm between RNN autoencoder and decision tree

```
INPUT: Predicted labels $lb\_old$, anomalous predicted flows $\mathcal{D} = \{x_1, \dots, x_N\}$
OUTPUT: New labels $lb\_new$
  $lb\_new \leftarrow lb\_old$
  for $i$ to $N$ do
    if $lb\_old_i$ is equal to 1 then
      $attack\_cat_i$ ← predict_label by using $x_i$
      if $attack\_cat_i$ is equal to 'Normal' then
        $lb\_new_i \leftarrow 0$
    else then
      $attack\_cat_i$ ← 'Normal'
    end
```

Additionally, the decision tree may receive unknown types of attacks due to the anomaly-based approach of the system. In this case, it will label them by the closest matching class. Therefore, it is important to mention that this classification is only for guidance and explanatory purposes for the analysts.

3.2.2 Approach B

The main advantage of our system is its ability to detect unknown anomalies. This is very important nowadays due to the high speed at which technology and, consequently, cybercrime are advancing. However, its accuracy is not always high enough due to false alarms. Sensitive sectors such as this require high performance. Therefore, Approach B, which maintains the global architecture, but adds a modification in the second block, is proposed.

In Approach B illustrated in Figure 3.6, the second element is trained in a supervised way. The autoencoder is omitted in order to use only an advanced RNN. It is trained to classify instances into normal or attacks. The RNN generates a probability distribution based on these two classes. To handle the problem of false positives that may be produced by the anomaly detection models, all those instances to which the RNN does not assign a very low probability of being malicious are considered attacks, maintaining the decision of the SPBN. Otherwise, the instance is labeled as normal. As in Approach A, explainability methods are employed to help understand the neural network decisions.

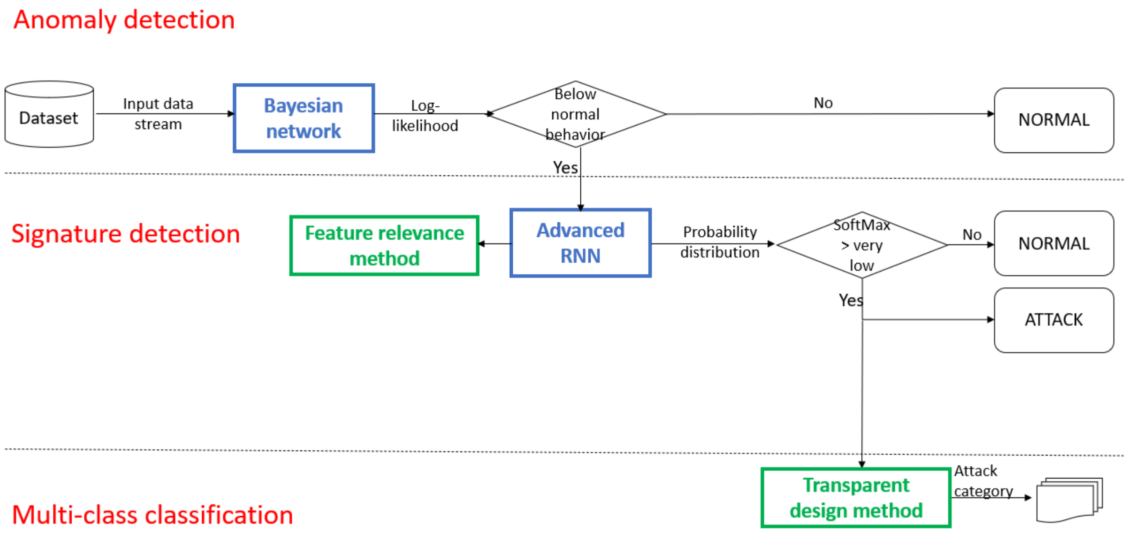


Figure 3.6. Approach B architecture.

3.2.2.1 Signature Detection

As in the second component of the anomaly-based version where a RNN autoencoder is used (Approach A), a data preprocessing is performed, and it coincides with this approach in the following aspects. On the one hand, the feature selection is conducted by discarding all those variables that are categorical. Also, data standardization is used to improve model performance. Since a RNN is employed again, the temporalization of the data is applied through sliding windows with overlapping, experimenting with different sizes during the testing.

The network architecture is the main difference since it is not based on anomaly detection and the structure is not encoder and decoder. As illustrated in Figure 2.4, the size of the hidden layers is reduced until ending in an output layer that performs the prediction. The function used for this depends on the type of problem. In Figure 3.3, binary classification is used to distinguish between a normal flow or an attack. In this case, the sigmoid function is used that squashes the output into a range between 0 and 1, providing a probability distribution interpretation for binary class.

In order to handle the distinction of attack types directly, the second component of this approach could also be an advanced RNN trained as a multi-class classification. This involves the use of a SoftMax activation function for the output layer. It receives an input vector and transforms it into a probability distribution over multiple classes.

For the latter model it is also required to apply the *OneHotEncoding* method during data preprocessing. By this, the classes are transformed into a numerical format for use in the RNN.

After an initial analysis, the sizes of the hidden layers are again 128 and 64 when using RNN units. A dense layer of size 16 is also included before the output one. The rest of the hyperparameters are the same as those used with the RNN autoencoder. TensorFlow is employed as in Approach A.

Moreover, a bi-directional approach, explained in Section 2.6.1.4, is evaluated to further improve the performance of this component.

3.2.3 System Operation

As explained above, the second component works by sequences of elements. Therefore, it is necessary to work through windows from the beginning of the analysis process. For this purpose, a buffer (window) of size *s* is created, which receives observations of the network traffic until it is full. At this point, based on an established criterion, an element is chosen from this buffer. In the case of Figure 3.7, the last element of the window of size 3 is always obtained. The first element chosen corresponds to that of index (2) and is sent to the SPBN. As shown, the BN indicates that it is a normal connection, and it does not continue in the classifier cascade. At this point, the oldest window element is deleted. This case corresponds to index (0). Then, a new element is added, index (3), and sent to the SPBN. In this case the model considers it to be an anomaly. Therefore, the complete window is sent to the advanced RNN Autoencoder which analyzes it taking into account each previous element in the window.

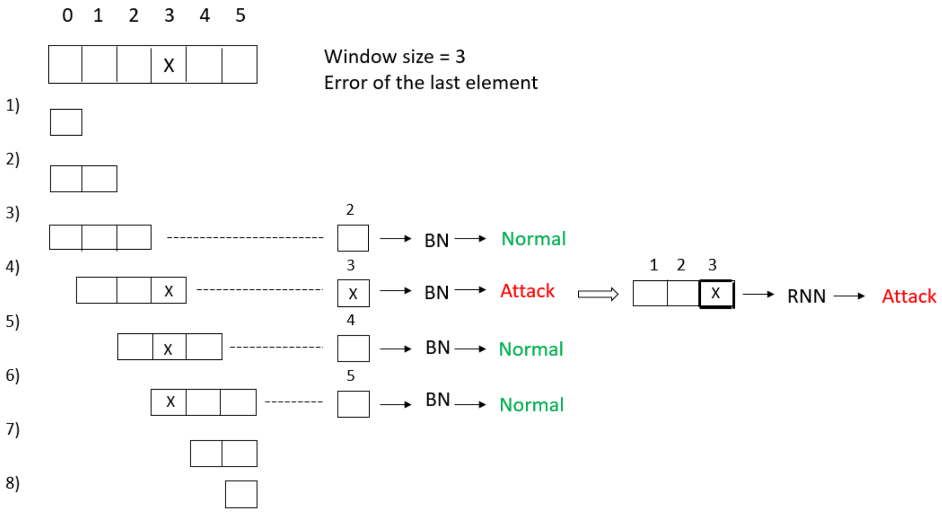


Figure 3.7. Example of the system operation.

This second model can use several criteria to define the degree of anomaly of the element being analyzed. Some of them are the reconstruction error of the element itself, the average of the errors of the elements of the same window or a weighted sum of each of them. In the example, the error of the last element is applied, effectively notifying that it corresponds to an attack. In the next iteration, the instance of index (1) is eliminated, and a new instance of index (4) is added, which is analyzed following the same process.

It is important to note that depending on the position of the element in the window being analyzed, the previous or subsequent flows may remain unstudied. In the example given, the last element of the buffer is always scanned without examining the first two of index (0) and (1). In general, if this criterion is followed, the first n-1 elements are not analyzed. This is insignificant for a system that has to process hundreds of thousands of flows per day, where almost always 100% of them are not anomalous.

3.2.4 Implementation

The implementation of the proposed cascading system involves utilizing specific code environments, programming languages, and libraries to develop and evaluate the contribution.

Two development environments, Jupyter Notebook and Google Colab, are used during different stages of the implementation. Jupyter Notebook is used for BN training, data preprocessing and full system integration. On the other hand, Google Colab is employed for all DL related tasks.

Python serves as the foundation for implementing the system. To handle the data through exploration and preprocessing, the Pandas library is used. Additionally, Matplotlib is utilized to create visualizations for result analysis and presentation purposes. Regarding the graphs of the BNs, a package called NetworkX is used for the creation and study of the structure. Numerical operations and array processing are performed using NumPy.

PyBNesian, TensorFlow and scikit-learn are the libraries for the creation of the models as it is mentioned in the corresponding sections.

Chapter 4

Results

The evaluation of the results obtained by the proposed methodology is performed using the dataset explained in Section 3.1.1. It is a relatively modern dataset that includes a wide variety of attacks and correctly simulates the normal operation of the network. Therefore, the test results correspond closely to those that would be obtained in a real case. Additionally, the fact that attacks rarely occur in communications results in a large imbalance in the dataset simulating a hypothetical real situation. This makes the training and evaluation more challenging.

4.1 Evaluation Metrics

In binary classification problems, the result of a classification can be correct or incorrect, and all possible results can be divided into the following four conditions:

- True positive (TP): actual attacks are classified as attacks.
- True negative (TN): actual normal records are classified as normal.
- False positive (FP): actual normal records are classified as attacks. This condition is also known as false alarm.
- False negative (FN): actual attacks are classified as normal records.

Accuracy is a widely used metric in classification tasks that measures the proportion of correctly classified instances over the total number of instances. However, when the data is unbalanced, accuracy is not considered an adequate metric to evaluate. In this case, it can be misleading since the system can classify all instances as normal and would achieve high accuracy due to the prevalence of these types of observations. Therefore, other metrics that are not influenced by this problem should be used to evaluate the model.

Some of these are precision and recall defined in Equation (4.1) and (4.2), respectively. Precision measures the proportion of correctly identified malicious instances out of the total instances classified as malicious. On the other hand, recall measures the proportion of correctly identified malicious instances out of the total actual malicious instances. Achieving a balance between these two metrics is essential to create an effective system that can successfully detect malicious activity while maintaining a low false alarm rate. This trade-off is indicated by the F1-score, Equation (4.3), which is the harmonic mean of precision and recall.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.3}$$

These metrics are assigned to each of the classes. Because of the imbalance between them, it is important to choose an appropriate averaging that results in a fair metric for the model. On the one hand, in a weighted metric, each class is assigned a weight proportional to its representation in the dataset. On the other hand, a macro average treats all labels equally and provides a balanced evaluation between all of them. Consequently, to avoid assigning a higher contribution to the class with more examples in the dataset, macro averaging is chosen.

In the context of network intrusion detection, the management of false positives is very important since most of the traffic is normal. This problem is usually accentuated when using anomaly-based methods. Therefore, reducing false positives avoids false alarms that affect the correct operation of the system. For this purpose, the False Positive Rate (FPR) and False Negative Rate (FNR) metrics are also used. FPR, Equation (4.4), represents the proportion of normal traffic that the system incorrectly classifies as intrusions. Conversely, FNR, Equation (4.5), is the proportion of intrusions that the system incorrectly classifies as normal.

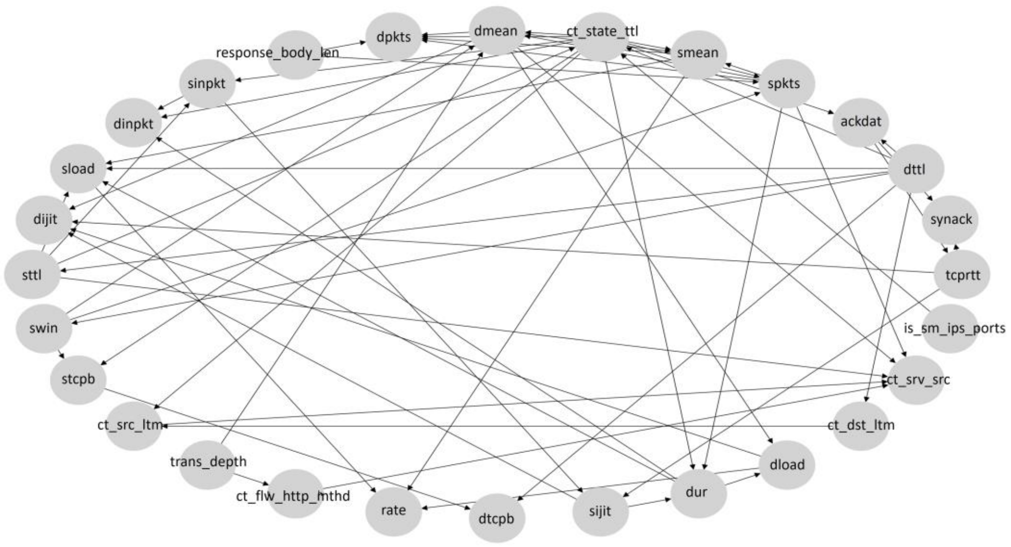
$$FPR = \frac{FP}{FP + TN} \tag{4.4}$$

$$FNR = \frac{FN}{FN + TP} \tag{4.5}$$

Additionally, a commonly used table in evaluating models is the confusion matrix. It provides a summary of the predictions by comparing them with the actual class labels. It is especially useful in multi-class classification problems, as it provides information on the strengths and weaknesses of the model when categorizing instances of various classes. This information can be useful in identifying specific areas for improvement.

4.2 Point Anomaly Detection

As explained in Section 3.2.1.1, point anomaly detection is performed by BNs that capture communications and generate log-likelihoods based on the normal operation of the network. During the initial experimentation, only a single model is built for all connections. This results in lower performance since it cannot generalize the behavior for different values of the network parameters.



Also, looking for a greater generalization, it is necessary to use a large part of the variables in the dataset trying to improve performance. In this case, as a result, the resulting graph is characterized by a high number of arcs linking the different nodes as illustrated in the full data SPBN graph of Appendix B. This graph is quite large and close to being complete, so it is not easy to interpret and decreases the comprehensibility of the decisions. It has 30 variables out of a total of 47. Those that are not included are eliminated because they are not relevant according to correlation analysis, in addition to eliminating the 8 variables that are categorical.

This particular SPBN model with this number of variables is the one that achieves the best performance using all the data without grouping by protocols. This is an FPR of 0.54 while maintaining an FNR less than 0.05. If the number of features is reduced to improve interpretability, the performance worsens considerably further.

It was therefore decided to group the data according to the different transport and application protocols. All resulting graphs are shown in Appendix B. For example, Figure 4.2 shows the structure of the SPBN trained using normal flows data whose transport protocol is UDP and its application protocol is unknown. The feature selection is done by the procedure described previously and note that the resulting structure is much simpler and easier to understand. Only four nodes (*sbytes, dttl, dbytes, ct_srv_dst*) are part of the connected network, besides one node (*sttl*) that is independent and not shown.

According to this structure, the bytes sent from the source to the destination (*sbytes*) depend on two factors in the normal operation of UDP communications. First, the time to live (TTL) from the data destination to the initial source and, second, the bytes sent from the destination to the source. Additionally, the destination address and the application protocol of the flows are determined by these two variables in the UDP context.

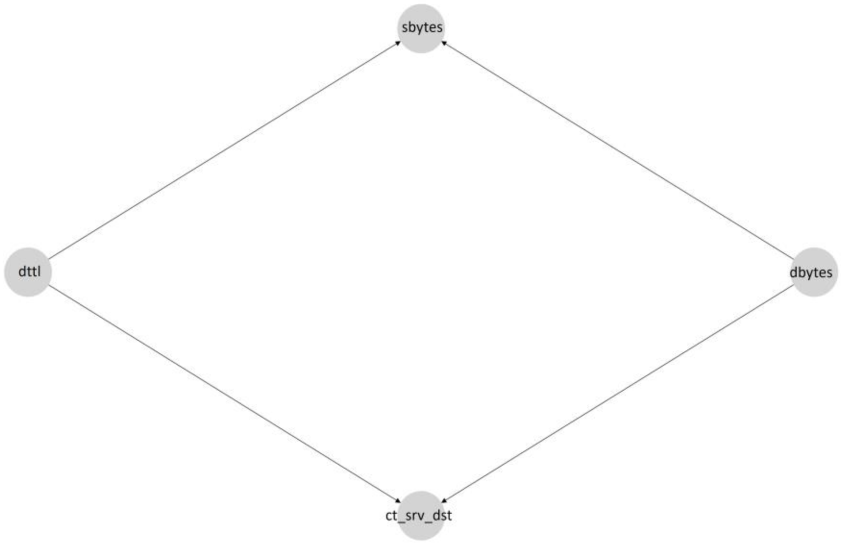


Figure 4.2. SPBN structure of UDP and None data.

When the SPBN receives a flow, it determines the probability with which it conforms to the parameters and structure learned through normal traffic. To do this, a log-likelihood is generated which is normally close to 0 for normal communications and moves away from this value for anomalies. Since a threshold is used to detect attacks, it is important that there is a significant difference between the average log-likelihood for the two possible classes. This helps differentiate more clearly between the two cases avoiding false predictions.

	Log-likelihood	
	Normal	**Attack**
Full	-56.71	-88.75
UDP	-8.20	-3895150.60

Table 4.1. Median log-likelihood for the full and only UDP SPBN models.

As shown in Table 4.1, grouping the network traffic according to the protocols allows to better adjust the representation of the normal behavior. The median loglikelihood for normal flows (-8.20) is much lower in the SPBN of the UDP protocol than the one using a single model for all (-56.71). Also, the disparity between normal traffic loglikelihood (-88.75) and attacks one (-3895150.60) within the specific SPBN is infinitely larger. This allows the threshold to be set correctly. The remaining median log-likelihoods for each of the BNs according to the protocols are given in Appendix C.

Figure 4.3 shows an example of the first 1000 log-likelihoods generated by the SPBN of UDP flows. Since there is a very large difference between these values, only those streams that are considered anomalous can be observed in the form of a bar. According to this graph approximately 30% of these communications are predicted to be attacks and should continue in the cascading system. In addition, it should be noted that among the flows whose index is approximately 500/600, the majority are considered anomalies and may be collective.

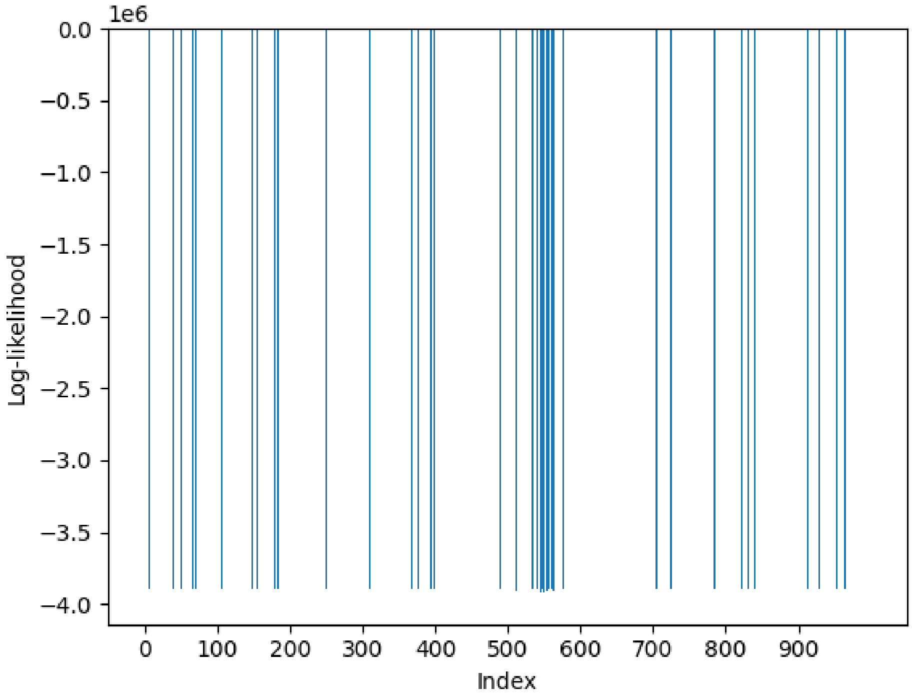


Figure 4.3. Log-likelihoods of the first 1000 UDP flows.

This is consequently reflected in the evaluation metrics, illustrated in Table 4.2, for anomaly detection using only this component. Using FNR and FPR as a reference, the new approach resulted in a drastic reduction of false alarms. For example, for the UDP model shown above, false positives were reduced by more than 80% compared to the full model. However, it fails to reduce false alarms in DNS protocol flows that are intended to resolve domain names to IP addresses. Therefore, it is necessary to use a second component that analyzes traffic from another perspective to improve performance.

Another factor influencing performance is the threshold that marks the boundary between normal and anomalous flows. This is calculated as explained in Section 3.2.1.1.

	FPR	**FNR**
Full	0.54	0.05
TCP/None	0.05	0.00
TCP/HTTP	0.04	0.00
TCP/FTP	0.03	0.00
TCP/FTP-Data	0.00	0.00
TCP/SMTP	0.00	0.00
UDP/None	0.09	0.00
UDP/DNS	0.51	0.00

Table 4.2. FPR and FPR of the different SPBN models.

4.3 Collective and Contextual Anomaly Detection

There are many factors that determine the performance of deep neural networks. In the case of the recurrent ones, the window size is one of them since the data must be temporalized given the design of these. Thus, different sizes are tested, and the performance and computation time are analyzed before training the model.

To do this, an LSTM autoencoder is used as the base model. Unlike the first component, the flows are not grouped by protocols and a single classifier is therefore used. This is trained by normal behavior and the first 1000 flows of the testing set are used to evaluate it. The architecture is the one explained previously. FPR is employed as a metric to decide the best size, while a threshold is set to limit the FNR to 0.05 at most. An odd window size is used, and it is progressively increased from 5 to 11.

Window size	**FPR**
5	0.19
7	0.25
9	0.31
11	0.47

Table 4.3. FPR of LSTM autoencoder based on different window sizes.

A hypothesis could be that a longer lookback could capture more predictive patterns and improve accuracy. However, Table 4.3 shows how the performance worsens as the window size increases. The number of false alarms is doubled by also doubling the size of the window. As a result, it is decided to use a smaller window with a maximum of 5 elements.

It is important to state that the number of parameters in an LSTM layer does not increase with the lookback. Jozefowicz et al. (2015) state that if the retrospection period is long, the LSTM cell states merge the information over a wide window. Because of this, the extracted information is blurred. Consequently, extending the waiting period does not always work with RNNs.

Another factor influencing performance is the threshold that marks the boundary between normal and anomalous flows. As indicated in Algorithm 2, a procedure is used to obtain the optimal threshold given a limiting FNR. This is affected by the distribution of reconstruction errors generated by the autoencoder. In an ideal situation there is a clear division between attacks and regular traffic.

The first 50,000 flows of the testing set are used to check the distribution of the errors by grouping them according to their label. This experiment is performed by the LSTM Autoencoder using the architecture explained during the methodology. It can be seen in Figure 4.4 that most of the normals (purple) obtain an error between 0.2 and 0.3, while the few attacks (pink) acquire a higher value. According to the graph, the optimal threshold would be around 0.4 although some erroneous prediction would occur.

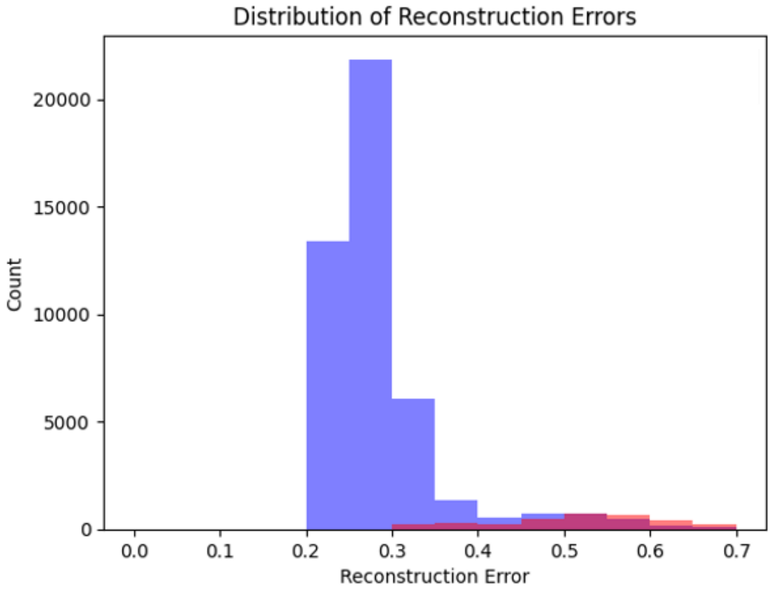


Figure 4.4. Distribution of Reconstruction Errors by LSTM Autoencoder.

Different RNN models are trained to integrate in the system and the most suitable threshold for each of them is obtained. An LSTM autoencoder, a bi-directional LSTM autoencoder and a GRU-based LSTM autoencoder are evaluated using the window size previously calculated by the procedure. The first 20,000 flows of the testing set are used. Since this second component receives mostly anomaly traffic, a threshold that generates at most an FNR of 0.05 is looked for. According to the results shown in Table 4.4, the most optimal threshold is between 0.45 and 0.50 depending on the model.

In addition, the FPR obtained using the data without filtering by the initial component is shown. This value is strongly increased by limiting the number of false negatives. Thus, it becomes evident the need for a cascade system that not only improves the efficiency, but also the performance.

Model	**Threshold**	**FPR**
LSTM	0.45	0.77
Bi LSTM	0.50	0.85
GRU	0.45	0.85

Table 4.4. Threshold determination for three advanced RNN autoencoders.

4.4 Signature Detection

As observed in the results of the unsupervised version of the advanced RNNs, they usually suffer from a high percentage of false positives. This problem is partly solved using SPBNs as an upstream component. However, due to this less than optimal performance, several supervised models are trained for Approach B and evaluated to improve attack detection. Likewise, the window size used is 5 given the experiments performed previously. The same first 20,000 flows of the dataset are used again for the evaluation.

The models used are the same, but in their signature detection version, i.e., without the autoencoder. Again, we limit the FNR to 0.05 and look for the configuration that gives the best FPR as a result. As illustrated in Table 4.5, there is a great improvement and false alarms are greatly reduced. The bi-directional version of the LSTM is the best performing recurrent network with 0.07 FPR. The simplified GRU version shows similar results also offering higher efficiency.

Model	**FPR**
LSTM	0.17
Bi LSTM	0.06
GRU	0.07

Table 4.5. Performance for three advanced RNN.

However, the LSTM model still maintains a high FPR despite the drastic reduction compared to its anomaly-based version. Sutskever et al. (2014) found that reversing the order of elements in all source sequences significantly improved the performance of the LSTM. Doing this introduces many short-term dependencies between source and target. Therefore, the model is tested to process the input sequence backward. The result is a reduction of the FPR to 0.12. This is 30% less than the version that does not process the input backwards. Hence, this new version is introduced in Approach B.

4.5 *Post-hoc* Explainability

When analyzing the relevance of features using gradient-based methods, there are several visualizations that provide more information to understand the decisions made by the RNN model. These are based mainly on the importance of the variables and their behavior throughout the predictions.

Figure 4.5 shows the feature distribution obtained by averaging the time steps of all correctly classified samples. This distribution represents the impact of the features on the outcome of the LSTM autoencoder model. In particular, the scores associated with the features show significant variations. Some features, such as *ct_state_ttl*, *ct_dst_src_ltm*, *sttl* and *dttl*, show large sums, indicating a large influence on the outcome. In contrast, most of the features have smaller scores, suggesting minimal importance in terms of outcome. It is important to note that some of the most determinant variables in this model are also present in the SPBN structures of the first component. For example, *ct_srv_dst* is the third most relevant feature employed in the second component and it is used in 4 of the 6 BNs: TCP and HTTP, TCP and FTP-Data, UDP and None, and UDP and DNS as can be seen in Appendix B.

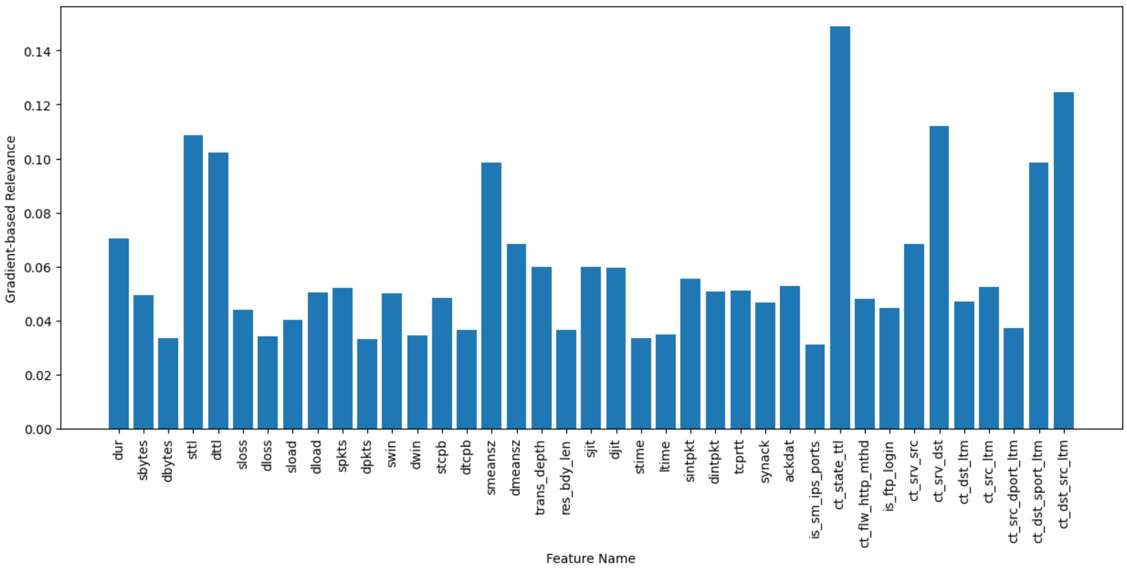


Figure 4.5. Average gradient-based relevance scores for each feature employed in the second component.

Analogous to what is done when averaging the relevance scores for each feature, a total of these values is performed to show the added relevance scores for the time steps. A small sample of 1000 instances is used to facilitate understanding. As illustrated in Figure 4.6, during the first 200 flows the mean scores of all variables are reliably constant and around 0.15. However, then there is a drastic drop until timestep 400.

This coincides with the fact that more than 70% of the anomalies in this sample are found in the first 200 communications and hardly any occurs after that up to timestep 400. During the rest of the analysis the weight assignment is more irregular because there are no long patterns of normal behavior or attacks.

In addition, the degree of relevance that the model assigns on average to each window element out of the five that receives is explained. For this purpose, a heat map, shown in Figure 4.7, is created to display the scores across the sequence data positions.

Looking at the horizontal axis corresponding to each of the positions within the buffer, the scores are quite similar in general. Darker colors are seen in the central elements of the window indicating less relevance to the model. However, the first and the last flow are more important as they are lighter. Additionally, the variables of indexes 4, 5 and 29 stand out in the fourth element with a much higher score than the rest. It can be seen that this coincides with what is shown in Figure 4.5.

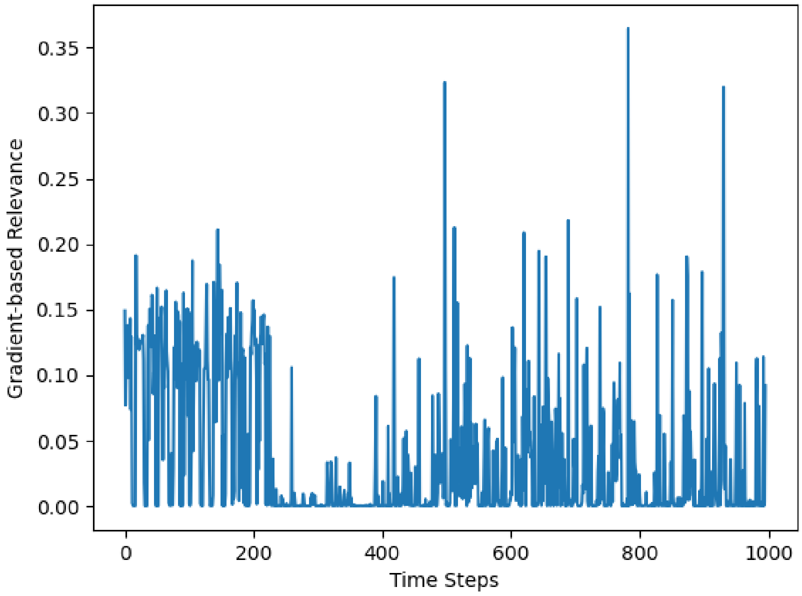


Figure 4.6. Gradient-based relevance distribution over time steps.

As explained above, the degree of anomaly of the last element based on its error is used to determine whether a window contains an attack. Then, this decision is also assigned to this last element for comparison with the actual label. As a result of this last explanation, this decision is correct since the model assigns a higher relevance to the fifth component of the window.

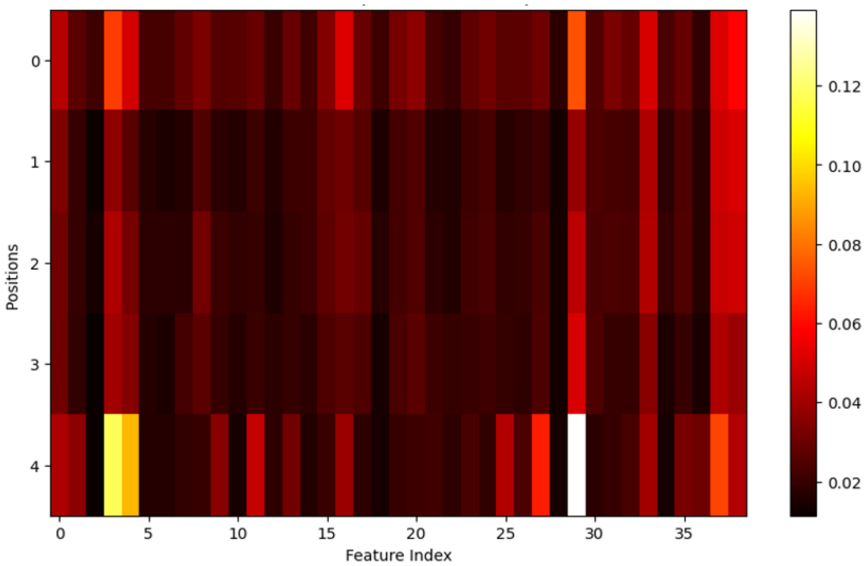


Figure 4.7. Gradient-based relevance of features across positions in sequence data.

Apart from explainability with the latter method to understand the influence of the features on final decisions of the RNN, a different approach is used that seeks to add information to the outputs of the system. For this purpose, a decision tree is used which, due to its transparent design, balances the null interpretability of the black boxes.

As a result of the feature selection method, the following 10 variables are used to train the model: *sbytes, dbytes, sttl, sload, spkts, smeansz, dmeansz, trans_depth, ct_srv_dst*, and *ct_dst_sport_ltm*. The result is an easy-to-understand tree structure that adds insight into why the received instances are considered anomalies by the recurrent model. Despite using a reduced number of attributes, the constructed tree is quite large and therefore not easy to visualize completely. To facilitate its visibility, the tree is pruned to a maximum depth of 3.

Figure 4.8 shows some of the decisions used to consider an instance as an anomaly and in turn a specific type of attack. This is the case of those classified as Backdoor, Generic, Shellcode and Reconnaissance. For the first one, the model establishes that it is sufficient to have a time to live from source to destination of less than 61 seconds. On the other hand, if it has a higher value, it can be of the rest of the anomalous classes. For example, if there have previously been more than 3 connections with the same source address and destination port, and the flow transmits more than 129 bytes to the destination, then it is classified as an attack of the Reconnaissance type.

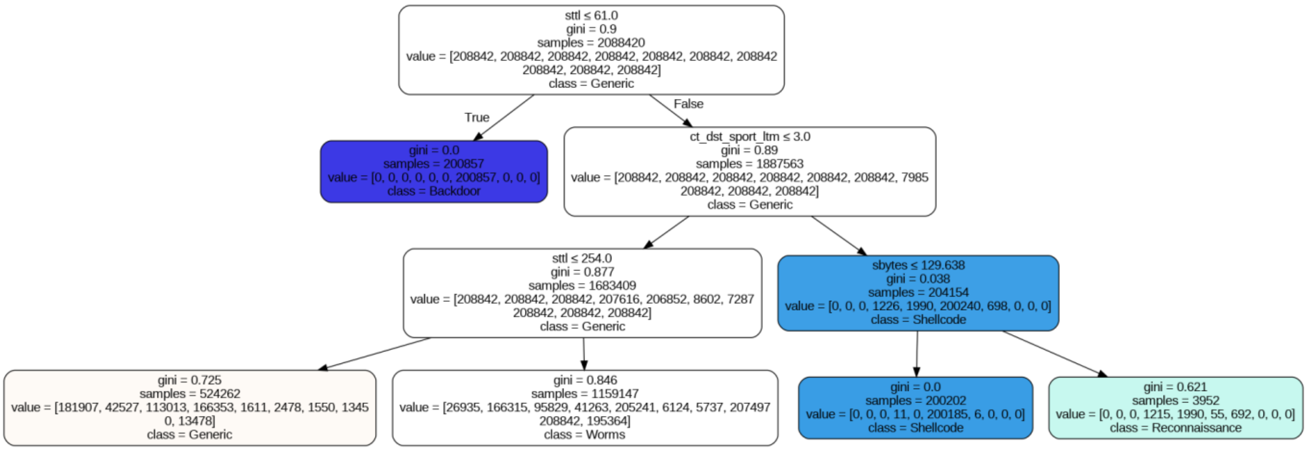


Figure 4.8. Decision tree with maximum depth of 3 as Transparent design method.

In addition to the explainability of the RNN's decisions offered by the model given its transparent design, the performance of the multi-class classification is analyzed. For this purpose, a normalized confusion matrix is used and the only instance corresponding to Analysis attack is eliminated. Therefore, there are 8 types of anomalies plus the normal flows.

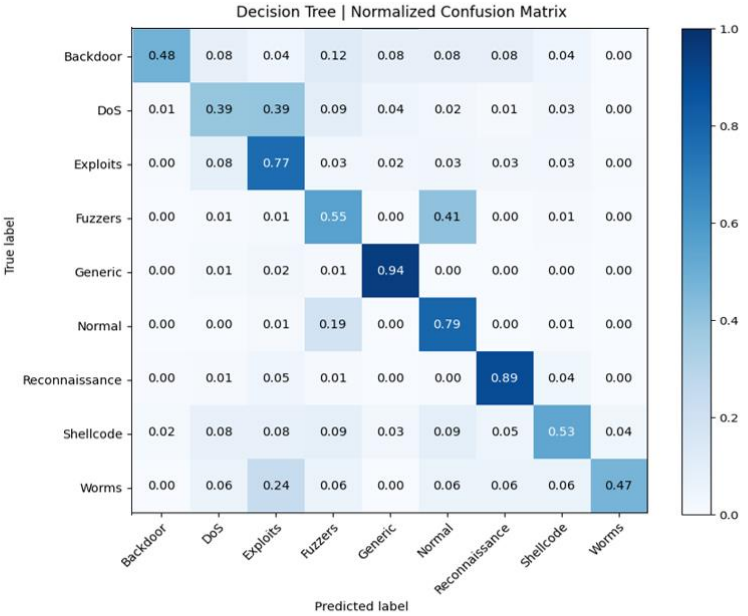


Figure 4.9. Normalized confusion matrix of decision tree results.

As shown in Figure 4.9, it performs a very good classification of the Generic attacks, being 94% of them correct. These are the most common within the dataset, since they represent almost 70% of the total attacks. It also performs a good prediction of almost 90% accuracy of Reconnaissance attacks despite being contextual anomalies.

It is worth noting that although it receives flows whose prediction is anomalous, the tree can also classify the instances as normal. Given the results, it can be observed that it detects false positives coming from the previous component in a remarkable way by classifying them as normal in 80%. However, if it is decided to apply Algorithm 3 with the objective of correcting false alarms, there is also an increase in false negatives as shown in Table 4.6. This is because a good part of Fuzzers attacks are predicted as normal as illustrated in the matrix. Therefore, as mentioned above, multi-classification should be used mainly as a value-added explanatory element.

FPR	**FNR**
0.15	0.01

Table 4.6. Performance of the system after applying Algorithm 3.

4.6 Approach A vs Approach B

The results of the two approaches are illustrated in Table 4.7. For each of them, the exact RNN used as the model of the second component is also indicated.

First, it can be observed that the performance is significantly improved compared to that achieved by the individual modules that constitute the complete structure. Primarily, the number of false alarms is reduced. For example, for Approach A with a bi-directional LSTM autoencoder, the FPR is reduced by almost 95% compared to the corresponding results in Table 4.4.

Among the models used in this approach, the one that uses a standard LSTM achieves the best performance with an F1-score of 0.88. Similarly, the lighter GRU model generates equally reliable results. One of the aspects to improve is the precision which is lower than the recall. This is because its value decreases for the positive class corresponding to the malicious activity.

Overall, the three models give almost identical performance. Since GRU is simpler and, consequently, more efficient, it is bolded as the best method of Approach A.

By using Approach B, which is a hybrid version combining anomaly detection and signature detection, very good performance is achieved. False alarms are again reduced compared to the first approach. The same models achieve the best results, but this time without the autoencoder architecture. In this case, LSTM is slightly superior to its simplified version with an F1-score of 0.95.

In short, the framework succeeds in meeting the performance criteria required by an IDS. The combination of the two components makes it possible to improve the capture of attacks. In addition, the supervised version achieves much higher accuracy.

	2nd Component	**Accuracy**	**Precision**	**Recall**	**F1-score**	**FNR**	**FPR**
Approach A	LSTM Autoencoder	0.96	0.83	0.97	0.88	0.02	0.04
	Bi-LSTM Autoencoder	0.96	0.82	0.95	0.87	0.02	0.05
	GRU Autoencoder	**0.96**	**0.83**	**0.97**	**0.88**	**0.02**	**0.04**
Approach B	**LSTM**	**0.98**	**0.92**	**0.98**	**0.95**	**0.01**	**0.02**
	Bi-LSTM	0.96	0.91	0.98	0.93	0.01	0.03
	GRU	0.98	0.92	0.98	0.94	0.01	0.02

Table 4.2. Evaluation metrics for Approach A and Approach B.

Chapter 5

Conclusions and Future Research

5.1 Conclusions

In this project, the development of a cascading explainable system for network intrusion detection is introduced, with the objective of improving the interpretability of the detection process while maintaining a high level of performance. The framework incorporates two main components: an initial anomaly-based detection component consisting of a module of several SPBNs and a secondary component based on a few types of RNNs. In addition, a feature relevance method and a decision tree are employed to generate decision explanations for the LSTM module.

Through extensive analysis and evaluation with a up-to-date network traffic dataset, the proposed system demonstrates remarkable performance in intrusion detection. The initial module provides an agile and accurate way of identifying potential anomalies by exploiting probabilistic relationships between features. BNs are based on the normal state of each transport and application protocol as it is difficult to generalize. When they receive new flows, they output log-likelihoods that determine the degree of abnormality using a specific threshold. As they are lighter models and work as first elements, the efficiency of the detection is improved by avoiding that all the traffic is analyzed by the neural network. In this way, the high levels of transmission that currently exist are correctly managed.

To further improve detection performance, we trained several RNNs autoencoders as anomaly-based models that are compared to their signature versions. Both are characterized by analyzing sequential patterns that allow to better capture those attacks that are not punctual, but occur in groups. This responds to one of the main issues based on the nature of the input data. Consequently, there are two different approaches for the same framework. The first method focuses on learning and reconstructing temporal patterns of normal network behavior, while the second uses labeled data to directly classify different type communications.

However, these models, as black boxes, are not interpretable. Therefore, feature relevance method is integrated that identifies the most influential inputs in the detection process. This allows to provide meaningful explanations of the decisions made by the recurrent module. As can be seen in the figures, there are some features that are very important for the detection of anomalies during processing by the RNNs, while most of them are not so decisive. Besides, the features at the extremes of the window are the most influential in the determinations.

Moreover, a decision tree is used to create interpretable rules that allow to know the reasons why the second component has determined that an instance is anomalous as well as to establish the specific type of attack. It is worth noting that the most prevalent attacks are classified with a high degree of accuracy and false alarm correction is again facilitated.

The evaluation results reveal that the cascading system outperforms the results obtained by the two components separately. In particular, from the point of view of the first component, the system significantly reduces false alarms by checking for instances that SPBN considers to be attacks. On the other hand, from the point of view of the second component, the fact that it receives already filtered anomalous traffic prevents it from processing a lot of normal traffic that may generate false positives. Therefore, one of the challenges that are part of the problem statement of this type of anomaly-based system is satisfied. This prevents the system from becoming unusable by flooding it with some irrelevant notifications.

Overall, this research contributes to the field of network intrusion detection addressing some of its main challenges. The proposed framework, with the combination of the strengths of BNs and DL, provides a practical solution. Also, the inclusion of feature relevance and decision tree-based explanations improve the transparency of the solution. This work opens avenues for future development in the domain of explainable network intrusion detection, facilitating a more reliable defense against malicious activities.

5.2 Future Research

The performance of the cascading system is generally good. The two approaches cover the main concepts within network intrusion detection. However, certain corrections can still be made to improve the behavior of the proposal. Therefore, several lines of future research are proposed ordered from highest to lowest priority.

1. The current system is validated using a dataset that is considered one of the best at present due to its wide variety of attacks and traffic fidelity. However, to perform a more reliable evaluation it is essential to test it on real network traffic. This can be challenging due to privacy issues and limited access to such data. Therefore, it is proposed to perform it with *ad hoc* network traffic using virtual machines that generate synthetic data representative of real-world network behavior. This data would enable to perform a better assessment of the system in a controlled environment.
2. As the field of DL continues to advance, transformer-based models have gained significant popularity and have achieved remarkable success in various domains such as natural language processing. They are very effective in capturing long-range dependencies and contextual information. Then, it is proposed to investigate the use of these modes for network attack detection with the goal of integrating them into the system. This would allow the proposal to capture more complex patterns,

leading to improved detection capabilities in identifying sophisticated intrusions.

3. Determining anomalies using a threshold is a very important factor in attack detection. Performance is significantly affected by the choice of such a boundary, so it is essential to perform this process correctly. Hence, it is proposed to use one-class models as they focus on learning the characteristics of the normal behavior of the network and identifying the cases that deviate from it.

   Leveraging the advanced RNN Autoencoder as the main model, the research will center on classifying instances as normal or anomalous based on reconstruction errors. These errors would serve as a representation of the deviation from the learned normal behavior of the network.

4. The SPBNs are responsible for identifying network flows that exhibit suspicious behavior. Rather than simply classifying flows as normal or anomalous, this research also proposes to extend the output of the BN to include a measure of the degree or severity of the anomaly. This would be communicated to the following component of the system. By providing the RNN autoencoder with this enhanced information, it would be able to better understand the severity of anomalies and adjust its reconstruction and prediction accordingly.

5. In traditional advanced RNN architectures, a fixed window size is typically used to encode and reconstruct sequences. However, network traffic patterns may exhibit variations in their temporal characteristics, where the duration of anomalous events may differ from normal behavior and between different types of attacks. To address this challenge, the use of a variable (non-constant) window size within the cascaded system is proposed as another future work. This would use an adaptive window sizing based on statistical measures that would dynamically focus on the relevant segments of the sequence.

6. As explained in Section 3.2.3 about the operation of the system, the last element of the window is always analyzed by its own reconstruction error for the detection of the anomaly. However, there are many other approaches that can be used. Therefore, it is intended to explore different alternatives such as the use of metrics that take into account the errors of the other elements of the window when determining the degree of anomaly of the last one.

   One option would be to assign weights to reconstruction errors based on their temporal proximity to the last element. This approach recognizes that recent events may have a greater impact in determining the degree of anomaly compared to earlier events.

7. One of the methods used to improve the explainability of the second component is characterized by pointing out the feature relevance during the decisions of this model. This results in a ranking of the most important variables. Therefore, as future work we can consider using these findings to eliminate those that are not important and observe how this affects the performance of the system. In this way, we can reduce the dimensionality of the data received by the RNN and improve computational times and interpretability.

8. In addition to the *post-hoc* explanatory techniques used to improve the understanding of RNNs, there are other different methods that also facilitate this task. This is the case of visual explanation techniques that help in the interpretation of black-box DL models. For instance, Karpathy et al. (2015) reveal the existence of interpretable cells that track long-range dependencies. Analyzing the internal states of these cells can help to understand how information flows and is processed over time. This helps to show which parts of the input are remembered or forgotten by the network.

Bibliography

Accenture. (2021). *Make the leap, take the lead: Tech strategies for innovation and growth.*

Aldini, A., Gorrieri, R., & Martinelli, F. (Eds.). (2005). *Foundations of Security Analysis and Design III* (Vol. 3655). Springer Berlin Heidelberg.

Alpaydin, E. (2020). *Introduction to machine learning.* The MIT press.

Ancona, M., Ceolini, E., Öztireli, C., & Gross, M. (2017). Towards better understanding of gradient-based attribution methods for deep neural networks. *6th International Conference on Learning Representations.*

Andreas, B., Dilruksha, J., & McCandless, E. (2021). Flow-Based and Packet-Based Intrusion Detection Using BLSTM. *SMU Data Science Review*, *3*(3).

Atienza, D., Bielza, C., & Larrañaga, P. (2022a). PyBNesian: An extensible python package for Bayesian networks. *Neurocomputing*, *504*, 204–209.

Atienza, D., Bielza, C., & Larrañaga, P. (2022b). Semiparametric Bayesian networks. *Information Sciences*, *584*, 564–582.

Axelsson, S. (2000). *Intrusion detection systems: A survey and taxonomy.*

Aygun, R. C., & Yavuz, A. G. (2017). Network anomaly detection with stochastically improved autoencoder based models. *Proceedings of the 4th IEEE International Conference on Cyber Security and Cloud Computing and 3rd IEEE International Conference of Scalable and Smart Cloud*, 193–198.

Bace, R., & Mell, P. (2001). *Intrusion Detection Systems.* U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115.

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys and Tutorials*, *16*(1), 303–336.

Bielza, C., & Larrañaga, P. (2021). *Data-Driven Computational Neuroscience. Machine Learning and Statistical Models.* Cambridge University Press.

Chandola, V. (2009). Anomaly Detection : A Survey. *ACM Computing Surveys*, *41*, 1–58.

Chawla, N. V, Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *Proceedings of {SSST}-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111.

Clausen, H., Grov, G., & Aspinall, D. (2021). CBAM: A contextual model for network anomaly detection. *Computers*, *10*(6), 79.

Debar, H., Dacier, M., & Wespi, A. (2000). Revised taxonomy for intrusion-detection systems. *Annals of Telecommunications*, *55*(7), 361–378.

Dosilovic, F. K., Brcic, M., & Hlupic, N. (2018). Explainable artificial intelligence: A survey. *Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics*, 210–215.

Fernandez Maimo, L., Perales Gomez, A. L., Garcia Clemente, F. J., Gil Perez, M., & Martinez Perez, G. (2018). A self-adaptive deep learning-based system for anomaly detection in 5G networks. *IEEE Access*, *6*, 7700–7712.

Forouzan, B. A. (2002). *TCP/IP Protocol Suite*. McGraw-Hill Higher Education.

Gogoi, P., Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2012). Packet and flow based network intrusion dataset. *International Conference on Contemporary Computing*, 322–334.

Goldstein, M., & Uchida, S. (2016). A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. *PLOS ONE*, *11*(4), e0152173.

Gunning, D. (2017). Explainable artificial intelligence (XAI). *Defense Advanced Research Projects Agency (DARPA), Nd Web*, *2*(2), 1.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107–116.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, 2342–2350.

Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). *Visualizing and Understanding Recurrent Networks*.

Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, *2*(1), 1–22.

Kingma, D. P., & Ba, J. L. (2014). Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations*.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

Kramer, M. A. (1992). Autoassociative neural networks. *Computers & Chemical Engineering*, *16*(4), 313–328.

Kruegel, C., Mutz, D., Robertson, W., & Valeur, F. (2003). Bayesian event classification for intrusion detection. *Proceedings - Annual Computer Security Applications Conference*, 14–23.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia Yangqing, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). *TensorFlow: Large-scale machine learning on heterogeneous Systems*.

Tom Gann. (2020). *The Hidden Costs of Cybercrime*. McAfee.

Microsoft. (2022). *Microsoft Digital Defense*. Microsoft.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *Proceedings of the 2015 Military Communications and Information Systems Conference*, 1–6.

Moustafa, N., & Slay, J. (2016). The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective*, *25*(1–3), 18–31.

Olah, C. (2015). *Understanding LSTM Networks*. Colah's Blog.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan kaufmann.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel V. and Thirion, B., Grisel, O., Blondel, M., Prettenhofer P. and Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Petsche, T., Marcantonio, A., Darken, C., Hanson, S., Kuhn, G., & Santoso, I. (1996). A Neural Network Autoassociator for Induction Motor Failure Prediction. *Advances in Neural Information Processing Systems*, *8*.

Ranjan, C. (2020). *Understanding Deep Learning: Application in Rare Event Prediction*. Connaissance Publishing Atlanta, GA, USA.

Said Elsayed, M., Le-Khac, N. A., Dev, S., & Jurcut, A. D. (2020). Network Anomaly Detection using LSTM based Autoencoder. *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, 37–45.

Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). *Recent Advances in Recurrent Neural Networks*.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681.

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP flow-based intrusion detection. *IEEE Communications Surveys and Tutorials*, *12*(3), 343–356.

Srivastava, N., Mansimov, E., & Salakhutdinov, R. (2016). Unsupervised Learning of Video Representations using LSTMs. *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, *37*(10), 843–852.

Stanford University. (2023). *Artificial Intelligence Index Report 2023*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *27*.

Wang, C., Wang, B., Liu, H., & Qu, H. (2020). Anomaly Detection for Industrial Control System Based on Autoencoder Neural Network. *Wireless Communications and Mobile Computing*, *2020*, 1–10.

Xu, C., Shen, J., Du, X., & Zhang, F. (2018). An intrusion detection system using a deep neural network with gated recurrent units. *IEEE Access*, *6*, 48697–48707.

Zavrak, S., & İskefiyeli, M. (2020). Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access*, *8*, 108346–108358.

Zhou, C., & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *Part F129685*, 665–674.

Appendix

A UNSW-NB15 Dataset Features

No.	Name	Type	Description
Flow features			
1.	*srcip*	Nominal	Source IP address.
2.	*sport*	Integer	Source port number.
3.	*dstip*	Nominal	Destination IP address.
4.	*dsport*	Integer	Destination port number.
5.	*proto*	Nominal	Transaction protocol.
Basic features			
6.	*state*	Nominal	The state and its dependent protocol.
7.	*dur*	Float	Record total duration (mSec).
8.	*sbytes*	Integer	Source to destination bytes.
9.	*dbytes*	Integer	Destination to source bytes.
10.	*sttl*	Integer	Source to destination time to live.
11.	*dttl*	Integer	Destination to source time to live.
12.	*sloss*	Integer	Source packets retransmitted or dropped.
13.	*dloss*	Integer	Destination packets retransmitted or dropped.
14.	*service*	Nominal	http, ftp, ssh, dns, ..., else (-).
15.	*sload*	Float	Source bits per second.
16.	*dload*	Float	Destination bits per second.
17.	*spkts*	Integer	Source to destination packet count.
18.	*dpkts*	Integer	Destination to source packet count.
Content features			
19.	*swin*	Integer	Source TCP window advertisement.
20.	*dwin*	Integer	Destination TCP window advertisement.
21.	*stcpb*	Integer	Source TCP sequence number.
22.	*dtcpb*	Integer	Destination TCP sequence number.
23.	*smeansz*	Integer	Mean of the flow packet size transmitted by the source.
24.	*dmeansz*	Integer	Mean of the flow packet size transmitted by the destination.
25.	*trans_depth*	Integer	the depth into the connection of http request/response transaction.

26.	*res_bdy_len*	Integer	The content size of the data transferred from the server’s http service.

Time features

27.	*sjit*	Float	Source jitter (mSec).
28.	*djit*	Float	Destination jitter (mSec).
29.	*stime*	Timestamp	Record start time.
30.	*ltime*	Timestamp	Record last time.
31.	*sintpkt*	Float	Source inter-packet arrival time (mSec).
32.	*dintpkt*	Float	Destination inter-packet arrival time (mSec).
33.	*tcprtt*	Float	The sum of *synack* and *ackdat* of the TCP.
34.	*synack*	Float	The time between the SYN and the SYN_ACK packets of the TCP (mSec).
35.	*ackdat*	Float	The time between the SYN_ACK and the ACK packets of the TCP (mSec).

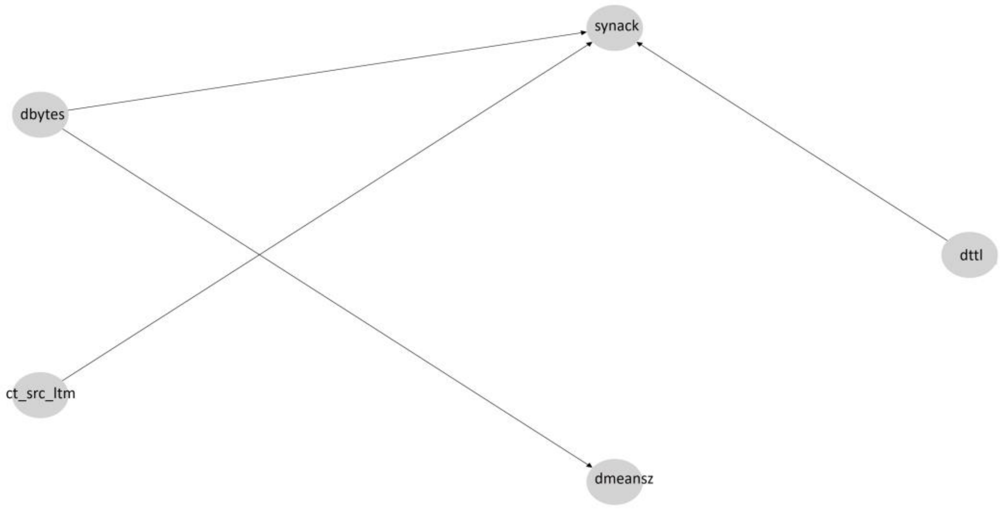
Additional generated features

36.	*is_sm_ips_ports*	Binary	If source (1) equals to destination IP addresses (3) and port numbers (2)(4) are equal, this variable takes value 1; else 0.
37.	*ct_state_ttl*	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
38.	*ct_flw_http_mthd*	Integer	No. of flows that has methods such as Get and Post in http service.
39.	*is_ftp_login*	Binary	If the ftp session is accessed by user and password, then 1; else 0.
40.	*ct_ftp_cmd*	Integer	No. of flows that has a command in ftp session.
41.	*ct_srv_src*	Integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
42.	*ct_srv_dst*	Integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43.	*ct_dst_ltm*	Integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44.	*ct_src_ltm*	Integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
45.	*ct_src_dport_ltm*	Integer	No. of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46.	*ct_dst_sport_ltm*	Integer	No. of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).

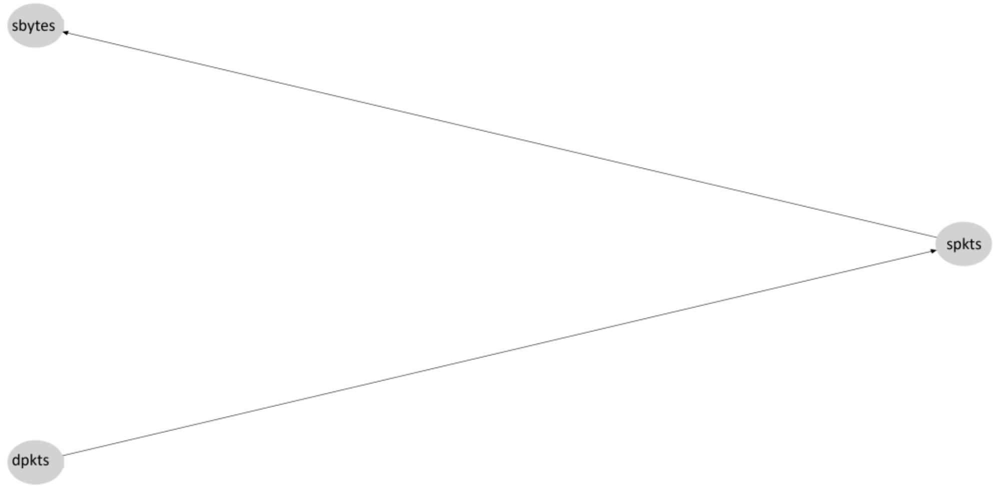
47.	*ct_dst_src_ltm*	Integer	No. of connections of the same source (1) and the destination (3) address in 100 connections according to the last time (26).

B SPBN Graphs of the First Component.

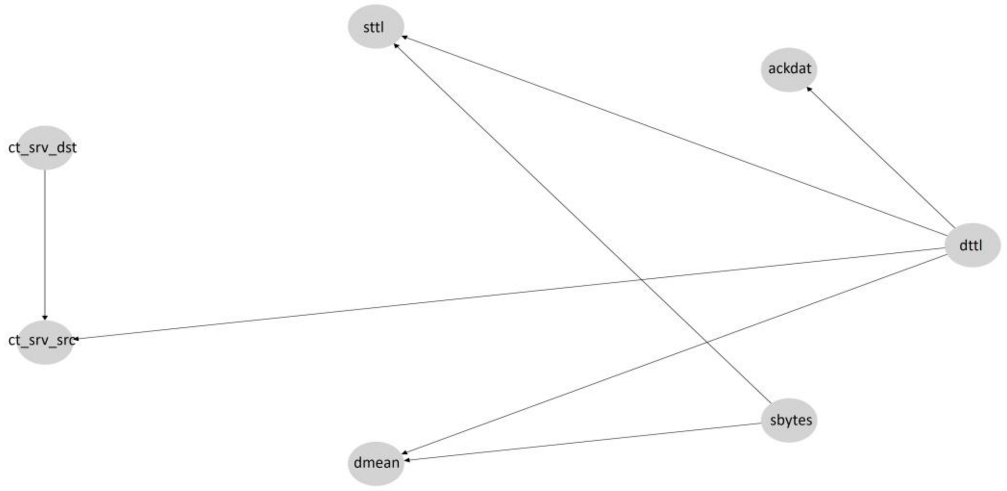
TCP and SMTP



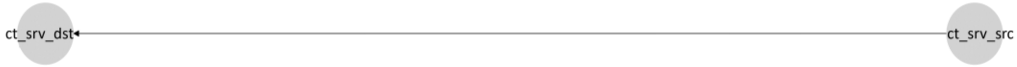
TCP and FTP



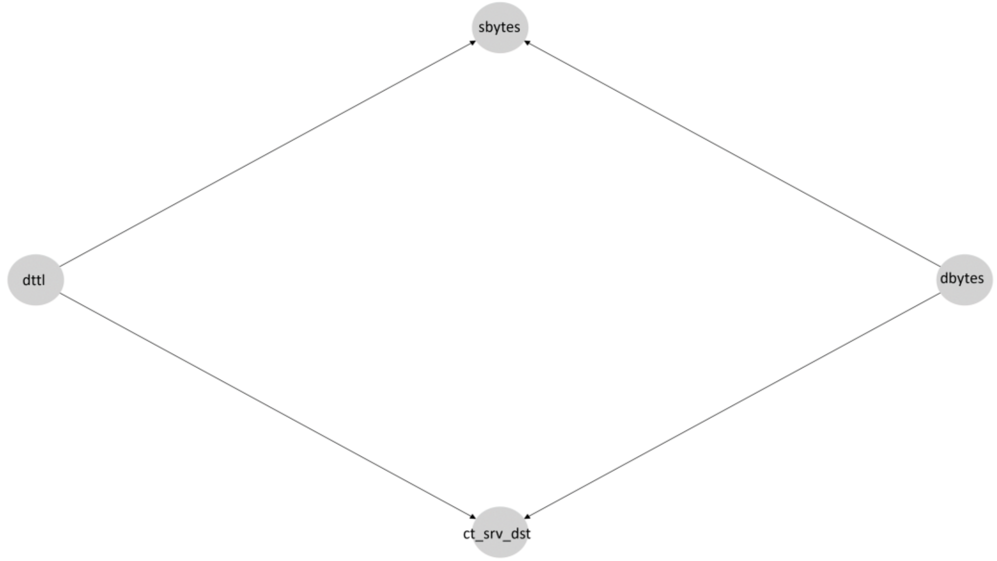
TCP and HTTP



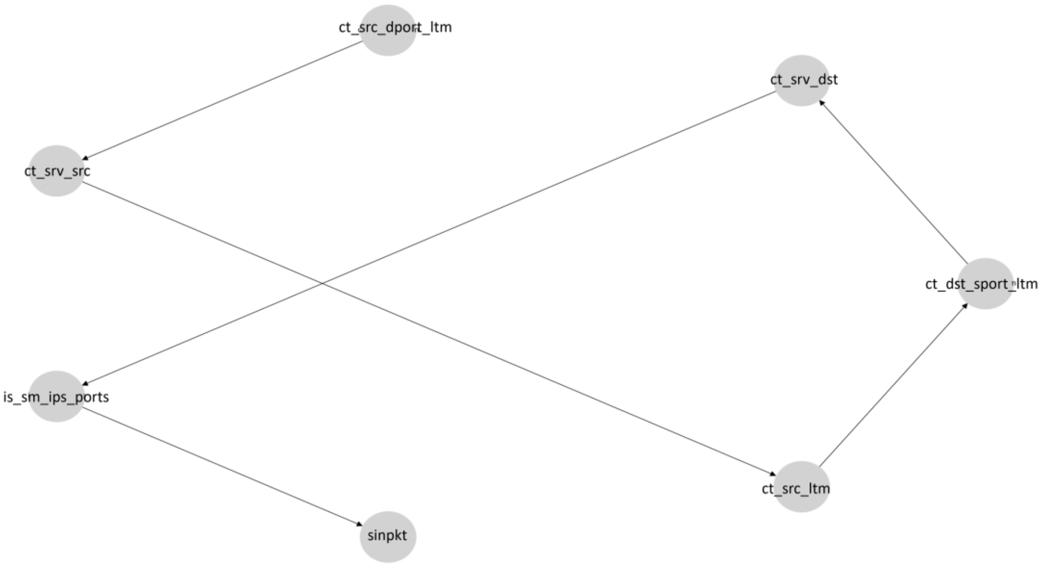
TCP and FTP-DATA



UDP and None



UDP and DNS


ct_src_dport_ltm
ct_srv_dst
ct_srv_src
ct_dst_sport_ltm
is_sm_ips_ports
ct_src_ltm
sinpkt

C Median log-likelihood for SPBN models.

	Log-likelihood	
	Normal	**Attack**
Full	-56.71	-88.75
TCP/None	-6.25	-47383399.27
TCP/HTTP	5.90	-673758269.25
TCP/FTP	-32764.38	-336867903.53
TCP/FTP-Data	0.85	-398388.87
TCP/SMTP	3.50	-332144669.83
UDP/None	-8.20	-3895150.60
UDP/DNS	-11447.8	-2920891437.78