Classifying Encrypted Network Traffic based on Deep Learning

Master Thesis

for obtaining the academic degree

Dipl. -ing.

as part of the study

Electrical Engineering and Information Technology

carried out by

Milos Crnjanski

student number: 01227434

Institute of Telecommunications

at TU Wien

Supervision:
Univ. Prof. Dipl.-Ing. Dr.techn. Tanja Zseby
Univ.Ass. Dott.mag. Maximilian Bachl

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, November 2019

Author's signature

Statement on Academic Integrity [1]

I hereby declare that this thesis is in accordance with the Code of Conduct rules for good scientific practice (in the current version of the respective newsletter of the TU Wien). In particular it was made without the unauthorized assistance of third parties and without the use of other than the specified aids. Data and concepts directly or indirectly acquired from other sources are marked with the source. The work has not been submitted in the same or in a similar form to any other academic institutions.

[1] Translation of the text above. The German version is the legally binding text.

Acknowledgements

First of all, I am grateful for the opportunity to study at Institute of Telecommunication, TU Vienna, and to all the professors for sharing a tremendous knowledge and spreading our professional horizons in many ways.

I would like to express my special appreciation to my supervisor, Professor Tanja Zseby, for giving me the opportunity to conduct a research on such an exciting topic, as well as excellent support, guidance and the given knowledge. A special thanks goes to my adviser Maximilian Bachl for continuous support, patience and all the helpful hints.

Thanks to my colleagues for all the good times spent together and for the support in a great number of situations when I thought that all was lost.

The biggest thanks goes to my family. Words cannot express how grateful I am for all the sacrifices that you have made on my behalf. Finally, I would like to express my deepest appreciation to my beloved wife Mirjana who spent sleepless nights by my side and was always supportive in the moments when there was no one to answer my queries.

"The only limit to AI is human imagination."

Chris Duffey, Superhuman Innovation: Transforming Business with Artificial Intelligence

Abstract

An enormous IP traffic growth in the last decade has resulted in new requirements regarding network security. With the traffic growth, the cybersecurity is also changing.It is difficult to apply security measures because of the bigger traffic amount and new applications and services. A large percentage of network traffic, as well as network attacks, is encrypted, and it is important to recognize an attack quickly to prevent any damage to the running system. With traditional methods of traffic classification, such as the port-based traffic detection and deep packet inspection, it is very difficult to follow the demand of the modern traffic classification.

In this thesis, machine learning is used as a solution to this problem. We developed a machine learning model based on binary classification which is able to detect attacks in encrypted network traffic. Our classification uses a new feature set, which consists of the following: the frame length, the time between packets in the flow and the direction of the flow. These are important features for us because their values do not change in encrypted traffic. The results open new discussions and change the view on today's traffic classification.

Zusammenfassung

Ein enormes Wachstum des IP-Verkehrs in den letzten zehn Jahren hat zu neuen Anforderungen an die Netzwerksicherheit geführt. Mit dem Netzwerkverkehrswachstum ändert sich auch die Cybersicherheit und es ist schwierig, Sicherheitsmaßnahmen anzuwenden, da der Netzwerkverkehr größer ist und neue Anwendungen und Dienste hinzukommen. Ein großer Teil des Netzwerkverkehrs sowie Netzwerkangriffe werden verschlüsselt, und es ist wichtig, einen Angriff schnell zu erkennen, um Schäden am laufenden System zu vermeiden. Mit traditionellen Methoden der Verkehrsklassifizierung, wie der Port-basierten Verkehrsdetektion und der Deep Packet Inspection, ist es sehr schwierig, den Anforderungen der modernen Verkehrsklassifizierung zu genugen.
In dieser Arbeit wird maschinelles Lernen als Lösung für dieses Problem eingesetzt. Wir haben ein maschinelles Lernmodell auf Basis der binären Klassifizierung entwickelt, das in der Lage ist, Angriffe im verschlüsselten Netzwerkverkehr zu erkennen. Unsere Klassifizierung verwendet ein neuartiges Feature-Set, das aus den folgenden Features besteht: der Packetlänge, der Zeit zwischen den Paketen im Flow und der Richtung des Flows. Dies sind für uns wichtige Features, da diese Werte auch bei verschlüsselten Daten noch verfügbar sind. Die Ergebnisse eröffnen neue Perspektiven und verändern den Blick auf die heutige Verkehrsklassifizierung.

Contents

1 Introduction

In this chapter, the overview of the thesis is given. It starts with the motivation and demand for research in this field. The next part is the aim of our work, where we discuss the key questions and goals of this thesis. In the final part we explain the structure of the thesis.

1.1 Motivation

During the last few years, the Internet traffic has experienced a tremendous annual growth rate of approximately 26 percent [19]. With the traffic growth, content of traffic is changing and the number of new applications is exponentially rising [19]. The philosophy of security threats is also changing as well. Cyberattacks are being directed more towards smaller companies [18] , and they are encrypted and more difficult to recognize.
That is why network traffic classification has become an important tool for recognizing the potential threats in traffic. It represents the foundation for important network activities, such as network planning, network security, network management, quality of service, accounting, etc. However, classification remains difficult because of the fast development of technologies, new protocols, applications and attackers hiding in those application, as well as because of our limited knowledge and technology.

There are different approaches towards network traffic classification. In [20], the author is providing the overview of traffic classification challenges. The oldest, and still very popular, approach is **port-based** classification, and it is based on the fact that most applications have an assigned port number. It is simple and fast, but not entirely reliable. New applications are using dynamic port assignment and attackers can easily brake the pattern by using another port and different tunnelling methods. A more reliable approach would be **payload-based inspection**, in which the content of a packet header is analyzed, and well known signatures are used to classify the traffic. This method is highly accurate but there are various downsides: it is more computationally expensive, the classification is not possible if the signature is unknown, it very difficult to perform in encrypted traffic, and, most importantly, it violates the privacy principle.

For those and many other reasons it is important to change the approach regarding the network traffic classification.

One new approach is to use **machine learning**. A huge growth of data collected through our daily network activities has created demand for developing new methods for analyzing data to recognize and learn about basic human behavior. The results of

the analysis would subsequently be used for commercial purposes. A subset of ML, deep learning provides a simulation of the human neural system, which can be applied to our problem.

Our research focuses on binary classification of the network traffic based on deep learning techniques to distinguish attack traffic from benign traffic. The data that we use contains normal and malicious traffic in the form of nine different cyberattack families. The data was processed to retrieve the desired features of the traffic and it was fed to a recurrent neural network. This specific method was used because it is better applied to sequential data and it fits well to an online classification. Different types of RNNs were tested, including long-short term memory cells, gated recurrent unit cells and different regularization techniques. The result was an accurate and fast network classification.

The contribution of our work is three-fold:

- a new feature set was tested. It consists of the packet length, packet inter-arrival time and flow direction. The main advantage of this set is that the features are available in both, encrypted and decrypted traffic
- an efficient machine learning model was developed, based on language processing using a recurrent neural network and long-short term memory cells. These techniques utilize specific cell structures which enable memorizing features of given traffic.
- different regularization methods have been analyzed and implemented, such as batch normalization and layer normalization which smooth the data input statistics and help the model learn more efficiently.

1.2 Aim of the work

The research, in general, proposes the answer to three important questions:

- *Q1: Are recurrent neural networks efficient in traffic classification*? By efficient model, we define a model which could achieve more than 96% of accuracy and 90% j-statistic (details are provided:4.4). During a traffic analysis, we deal with a huge amount of data. There are new and unknown attacks and threats, whose outcome is very difficult to predict.
- *Q2: Which features of data are most important for the encrypted traffic classification*? This research will introduce a new feature set based on the frame length, inter-arrival packet time and direction of the traffic in the flow.

- *Q3: What can we do to improve the process of deep learning*? Different regularization techniques, such as batch normalization and layer normalization, will be proposed and evaluated.

1.3 Structure

This research is structured into 5 chapters:

- Chapter 1 - **Introduction** - contains the motivation for researching the topic, the description of the methodology and the structure of the research. Some related work is described as well
- Chapter 2 - **Background** - provides the overview of the theory behind this research. It provides a short description of machine learning, statistical method used for supporting it, IT security, etc.
- Chapter 3 - The description of the **experiment** that was conducted. This part describes the feature set, preprocessing of the data and development of the recurrent neural network model.
- Chapter 4 - **Result** - The outcome of the experiment, optimization and regularization
- Chapter 5 - **Conclusion** - The final discussion regarding the research, future demand

2 Background

In this chapter, we are setting the theoretical background for this research. As we are trying to detect the security attacks in encrypted network traffic, in the first part of this chapter we are introducing the basic concepts of network security and network traffic encryption. A short overview of Intrusion Detection System (IDS) is given, in order to understand how the most used system for attack detection works and to determine the limitations which could be overcome by using our system.

The second part represents an introduction to different types of traffic classification. We will give a short overview of port-based, flow-based classification and deep packet inspection (DPI). Flow-based classification is explained more in details since this type is used in our research.

The third part of this chapter presents machine learning (ML). We are introducing ML and all the concepts that are used in our research. A summary of ML is given with the statistical concepts of the neural network and with the explanation of the recurrent neural network (RNN). The summary is followed by a theoretical overview of long-short term memory (LSTM) and gated recurrent unit (GRU) cells which are part of the models that are trained in our experiment.

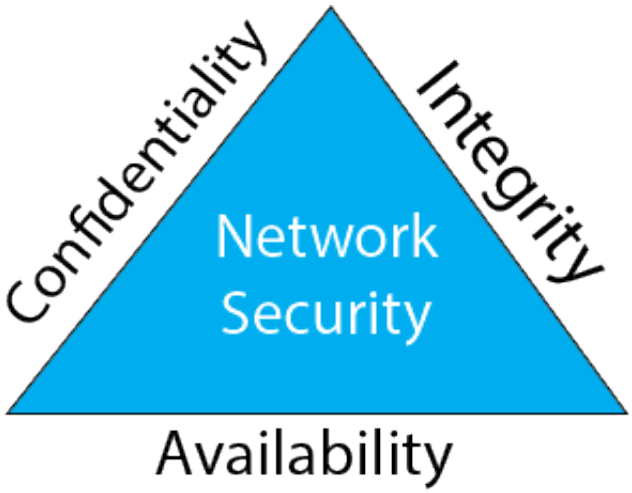

Figure 1: Network Security Principle

2.1 IT Security

2.1.1 Basic Concepts of Network Security

Network Security is a broad range of measurements and actions which are taken in order to protect the network infrastructure and to prevent an unauthorized access, misuse, modification and disruption of data, malfunction of the system, etc. In other words, Network Security is used to protect our network and data, their usability and integrity. The scope of network security covers software, hardware and firmware, as well as the information which is processed, stored and communicated. [21].

There are three basic concepts which make the core of Network Security: **Confidentiality**, **Integrity** and **Availability**, also known as the CIA principle. Those concepts define the most important security objectives for data and information services.

Confidentiality refers to concealing information or resources to protect secrecy. It could be interpreted as having 2 subsets, *data confidentiality*, which assures that the confidential information is revealed only to an authorized person, and *privacy*, which assures that the control or influence of the information is collected or stored only by the intended person. [44]
The method which is widely used to provide confidentiality is cryptography. Using a cryptographic key, encrypted data is scrambled in such a way that it cannnot be deciphered by an unauthorized person without the appropriate key.

Integrity refers to preventing an improper modification and unauthorized change. It can also be classified it into 2 subsets: *data integrity* assures that the information is

manipulated only in an authorized manner, whereas *system integrity* assures that a system performs intended functions, without any modification and biasing. [44]
The methods for preventing an information modification are:

- hashing - adds a number of bits, generated with secret key, to check if a message is altered
- message authentication - uses the secret key to ensures that a message is altered by an authorized person
- digital signature - ensures that the information is sent by a specific and authorized source

Availability refers to the proper functioning of the system, which means intended work of the system at a desired time. The loss of availability prevents us from using the information at the desired time and decreases system reliability. The recent times have seen the fast growing popularity of denial-of-service attack. As more and more web sites offer denial-of-service(DoS) attack on demand, availability faces the fastest growing vulnerability of all the concepts.

The order of importance of the concepts varies, depending on the purpose of a system and the applications. For example, in mail applications, confidentiality and integrity are the most important concepts. On the other hand, monitoring systems require availability and integrity. We can agree, however, that all three concepts are crucial for any network security application.

2.1.2 Intrusion Detection

Intrusion detection is a service which monitors and analyzes systems behavior in order to detect and provide a real time warning about an attempt to access security sensitive system in an unauthorized manner. [43] Intrusion detection represents second level protection and it is usually placed after the firewall to collect any intrusion that passes through the firewall.

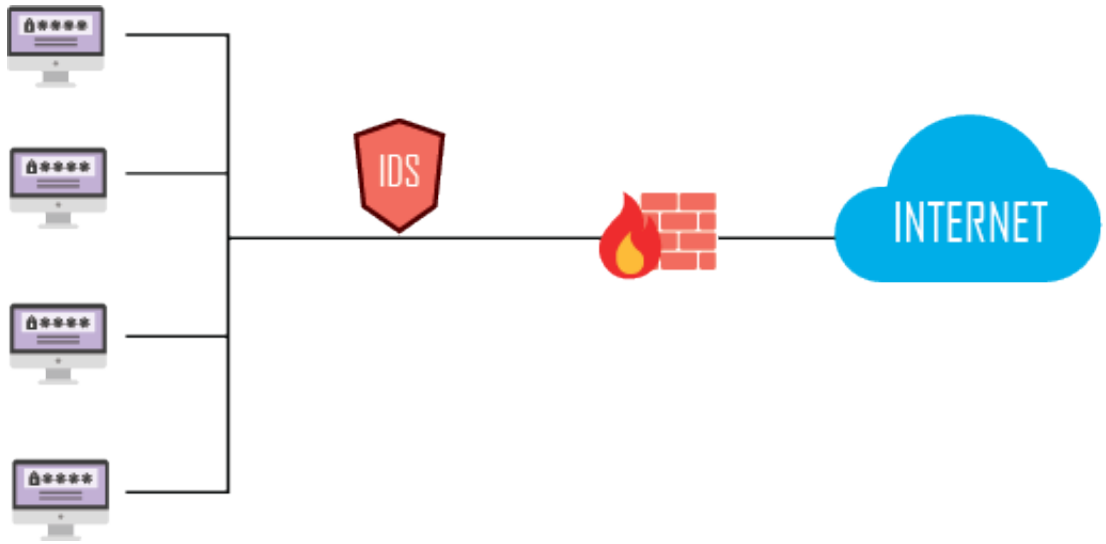

Figure 2: Intrusion Detection System

There are two different approaches used to analyze network data in order to detect intrusions:

- **Anomaly detection** - uses the collected data of a user's behavior to compare it with the set of expected values; it triggers a warning if it deviates too much
- **Signature** or **Heuristic Detection**, which uses a known malicious pattern for a comparison with the current behavior

The first phase of *anomaly detection* is modeling of the non-compromised user behavior by collecting and processing sensor data. This model is used afterwards for the comparison with the observed behavior to detect malicious activities.

There are different approaches to creating the models:

- **Statistical methods** - collect the data from sensors and use models with different methods of classifying the data (univariate, multivariate, time-series). The advantage of this approach is that it is simple and computationally efficient, but it is difficult to choose a proper metric and not all behaviors can be modeled.
- **Knowledge-based** approach consists of 2 phases: the *training phase*, in which the observed data is divided into different classes, and the *classification phase*, where the user behavior is classified based on those predetermined classes. The advantage is that this approach is robust and flexible, but it is difficult to determine an accurate model

- **Machine-learning** approach uses labeled training data to automatically develop the model. Developing the model requires some time and more computational power, but once it is developed, the model is fairly efficient. Various statistical methods used in this approach are: Bayesian networks, Markov models, neural network, fuzzy logic, generic algorithm, clustering and outlier detection.

The main limitation with anomaly detection, especially in the machine-learning approach, is that it only uses legitimate data, without any anomalies which is difficult to ensure for the modern network traffic.
We will try to address this issue in our research and propose different methods to detect and classify anomalies in the traffic.

Signature or *heuristic detection* is based on already known patterns. The user behavior is observed and compared with the existing signature database. It can be also configured to detect known penetrations or exploits, that is more specific to the system which is used. The advantage of this method is time efficiency and low computational complexity, but it is not able to recognize new threats without updating the signature database.

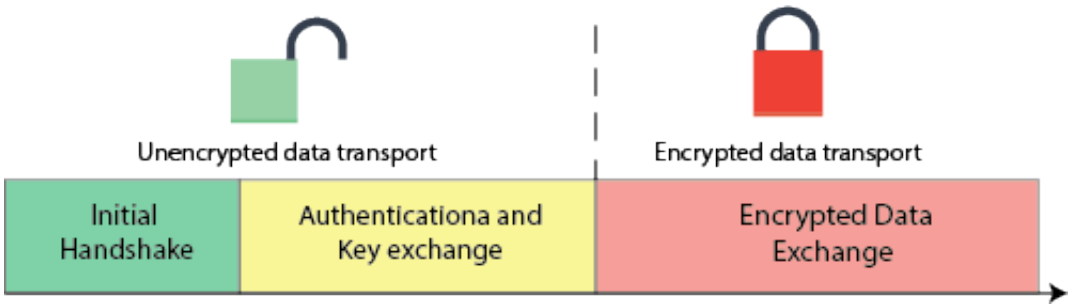

Figure 3: Data Encryption phases

2.2 Traffic Encryption

Encryption is used to provide confidentiality of the data, and most of those protocols also bring some level of integrity protection and authentication to the users. Protocols usually employ the secret key method of encryption, so the first phase is the initial handshake, the authentication of a user and key exchange. Fig: 3 shows encryption phases.

The protocols that are going to be described are: **IPsec** and **TLS**.

2.2.1 Internet Protocol Security

Internet Protocol Security (IPsec) [42] is a protocol that provides the authentication, encryption and integrity of the data on the network layer. It most commonly consist of two phases, the initiation phase, and the encryption phase. In the initial handshake phase, the authentication and a shared secret are established over UDP port 500. The main protocol used for the initial phase activities is the Internet Key Exchange Version 2(IKEv2) [15]. In the second phase the encryption of the data begins. IPsec uses **Encapsulating Security Payload(ESP)** to add header and trailer to each transferred packet, and everything in between is encrypted. ESP works in 2 modes: *transport mode*, where the original IP header is not encrypted, and *tunnelling mode*, where the original header is encrypted, and a new IP header is added.

It is worth mentioning that there is also the **Authentication Header(AH)** protocol which is used for authentication in IPsec. Since ESP also contains authentication feature, AH is rarely used.

The advantage of the IPsec is that it requires no change on the application level, but it is less flexible for applications which require some connection on TCP level.

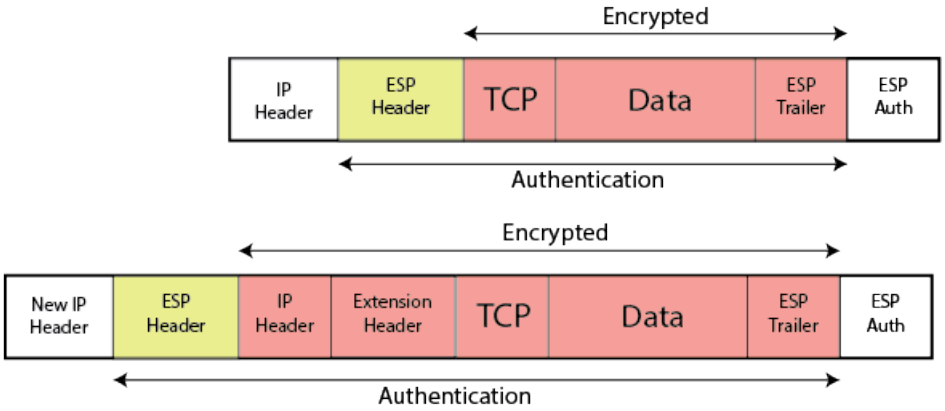

Figure 4: IPsec (a)transport mode, (b) tunnel mode

2.2.2 Transport Layer Security

Transport Layer Security (TLS) provides confidentiality, integrity, non-repudiation, replay protection and authentication with digital signatures. It is based on the Secure Socket Layer version 3 (SSLv3) and it provides a security level on top of TCP. It is the most commonly used protocol in network communication, especially in cooperation with VPN, Voip, HTML, FTP, etc

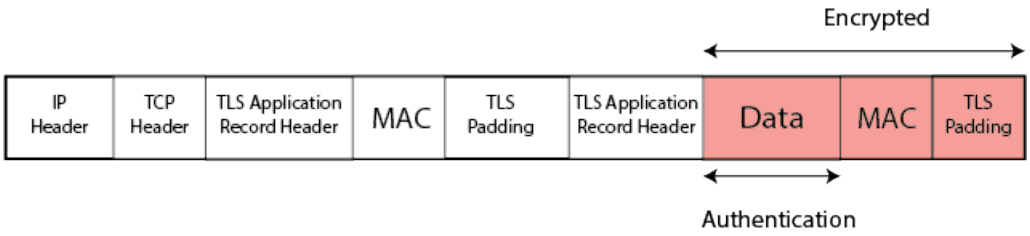

Figure 5: Transport Layer Security packet format

The first phase consists of the TLS handshake, in which the authentication takes place, and encryption and the session key are established. The encryption starts when the cipher suite is negotiated and the user authenticated. After the encrypting process, message authentication code (MAC) is added to provide data integrity protection.

Main advantage, that is important for our research is that in TLS the TCP header is not encrypted, while in IPsec the header is encrypted.

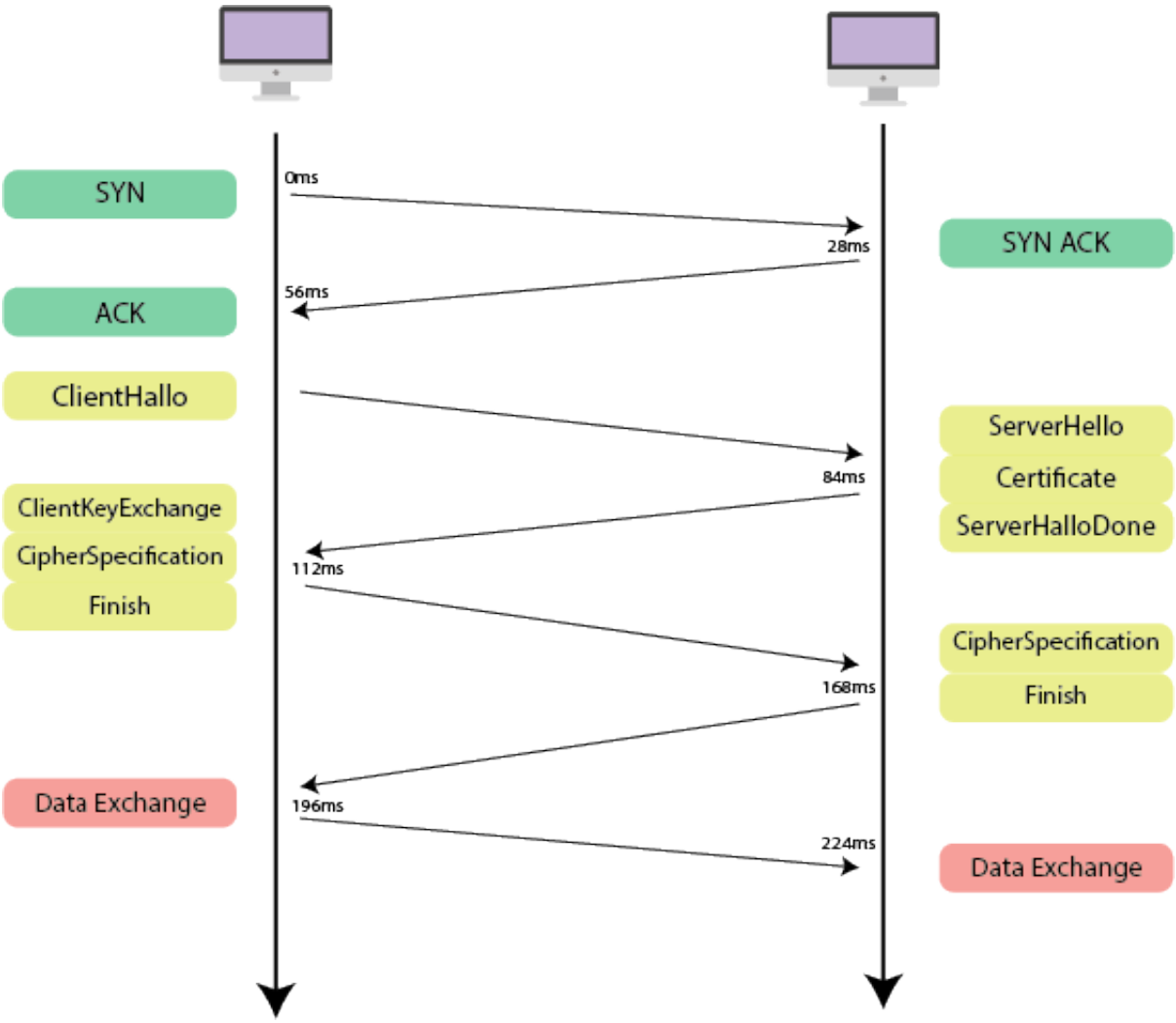

Figure 6: TLS initial handshake - process of establishing the TLS encryption. In the first phase, TCP handshake is established, then the version of TLS is exchanged and ciphersuite negotiated. After ciphersuite negotiation, the key parameters are exchanged, and after key has been proven, the encrypted data exchange can be started.

2.3 Network Traffic Classification

Network traffic classification plays a crucial role in network planning, network management, network security, network flow, etc. In the last decade, the amount of encrypted traffic has increased rapidly. More and more businesses are moving their services on-line and users are becoming aware of network security threats. Computational power of modern devices is much higher, which brings more users and applications to use encryption as a security measure. There has been a significant rise of encrypted data in modern communication and it is evident that the trend will continue.

Encrypted traffic experiences the growth of 90% every year and there are predictions that in 2019 around 80% of web traffic will be encrypted. Compared to the 2015 when there was only 20% of the traffic encryption, we would agree that the encryption experiences a big growth and it has tendency to continue growing. Due to an increasing rise of encrypted traffic, traffic classification is gradually becoming more difficult. [18]

Challenges which are introduced to the network traffic classification are numerous. There are difficulties to identify real-time applications such as P2P, Voip, online video in order to fulfill the QoS requirements. Another challenge would be to recognize and differentiate between malicious traffic and the regular network traffic. Nowadays, malicious traffic is usually encrypted, which helps it get through the firewall and IDS system undetected. For those reasons, there is a necessity for accurate traffic classification in order to apply some restrictions and blockage of unwanted application and services on the firewall.

In order to understand the difficulty which comes with encrypted traffic classification, we need to dive deeply into the well-known classification technologies. Network Traffic Classification is based on different features of traffic. The features change with encryption in various ways: namely content is moving from plaintext to ciphertext, statistics are becoming more random, and statistical properties on packet and flow level changes. [16]
The **port-based classification** relays on the fact that most applications have a known TCP or UDP port, assigned by IANA. [30] This represents the simplest method, which requires TCP/UDP port number from transport header to compare with port number in IANA database. Nowadays, it is not recommended to use this method because of a variety of reasons. First of all, some applications are using random or dynamic port numbers. Secondly, it is easy to spoof the port number and to disguise under known application. Furthermore, there are applications that do not have the number in the IANA database yet. Finally, due to encryption, port number could be hidden.

The **payload-based classification** method is also known as Deep Packet Inspec-

tion(DPI) and it is based on already known attack signatures and patterns. It analyses the payload of a packet and compares the information retrieved with the database of traffic signatures. This is a more efficient way of classification method in terms of accuracy, but it is computationally very expensive and also not reliable. The reasons for unreliability are: some of the signatures are not known, encrypting the payload makes this method very difficult and, most importantly, it violates the privacy principle.

The **statistical classification** is based on statistical properties of the traffic, as well as on the packet level and flow level. Different features are used in this method, starting with packet length, inter-arrival time, direction, etc., as well as more specific packet count, flow duration, packet ratio,etc. After the encryption, statistical features do change, but there are other (unencrypted) characteristics of traffic that can reveal information about the application, which makes this method very useful in modern traffic classification and more suitable for our research.

The **behaviour-based classification** or **host-based classification** analyses different application traffic based on host behavior. It analyses different connection patterns and attempts to classify the traffic into different classes. This method is rarely used since it gives very rough results, and the classification is impossible in the case of encryption and the usage of NAT.

2.3.1 Flow-based Classification

"*A flow is a sequence of packets sent from a particular source to a particular unicast, anycast or multicast destination that the source desires to label as a flow. A flow could consist of all packets in a specific transport connection or a media stream.*" [29]

In other words, a flow is a set of packets with some common properties which are passing an observed point in some time period. Properties are varying based on the intended use and could be based on packet header (source IP, destination IP, Port Numbers,etc.), packet characteristics (mpls label, flag, TTL,etc.) and packet treatment (next hop, interface,etc.). [28]
A flow is, traditionally represented by a 5-tuple:

- Source IP
- Destination IP
- Source Port
- Destination Port
- Transport Protocol

Due to different network and encryption requirements, some features may not be available so it often uses fewer parameters than usual.

The flow has its origin in circuit-switch communication. In this type of communication the connection between two hosts is established over dedicated resources, and it is not difficult to define the parameters of connection. In the packet-switch communication, the situation is a bit different. A message is fragmented and sent thorough the same communication resource, so, on the reception side, we have different packets from different sources, and it is important to define a set of parameter to identify the connection between users. *Destination* and *Source IP* are used for defining the end users, *Ports* are used to identify the application and it is also important to identify *transport protocol*.

Another important parameter is the direction of the flow, since most connections between two users are bidirectional. Many different applications and protocols have predetermined scenarios of exchanging the information between the sender and the receiver. This means that even when it comes to encrypted traffic, we can predict which application lies behind it, on the basis of the direction of traffic. A good example for this would be the initial handshake.

Aside from direction and the main 5-tuple parameters, it appeared to us that there are some statistical parameters in traffic flow which are unique for specific protocols and services, and even in encrypted data, it show a strong correlation with the original data. Some of these parameters are packet length, inter-arrival time, TTL, etc. The same principle applies to security attacks. Most of the attacks has a well-defined pattern of packets with specific length which are coming in specific time and are going in well-defined direction. This is very useful for the malicious traffic classification, with the help of machine learning algorithms. With a proper statistical method and a big data set, it is possible to develop a model which is able to classify malicious traffic based on its statistical characteristics and well-defined patterns, and with high accuracy.

In following figures, the attack patterns are presented. Those patterns are extracted from used data set, and plots are made based on three data set features: packet length, packet direction and packet inter-arrival time. All attacks in the same attack family have the equivalent attack patterns.

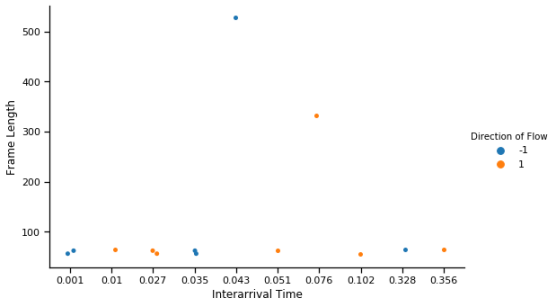

Figure 7: Exploits

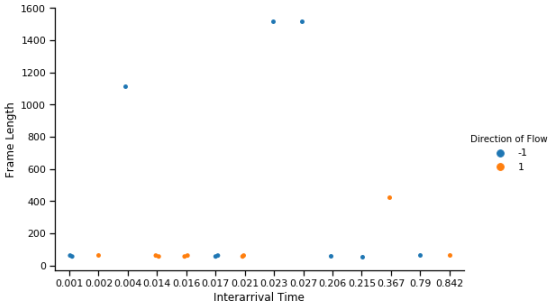

Figure 8: Generic

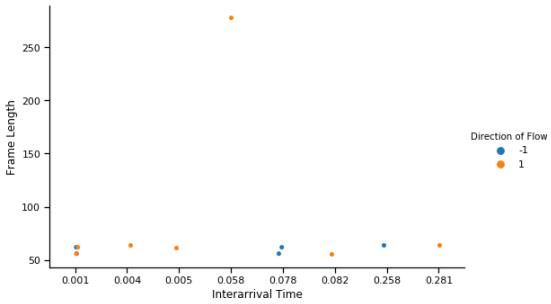

Figure 9: DoS

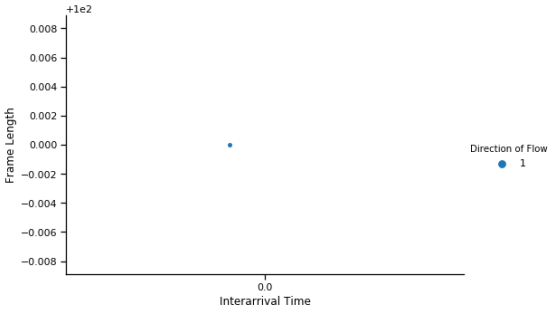

Figure 10: Reconnaissance

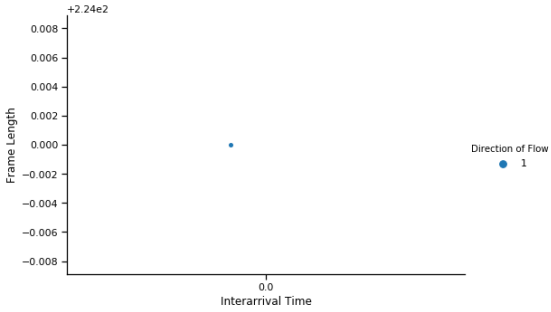

Figure 11: Shellcode

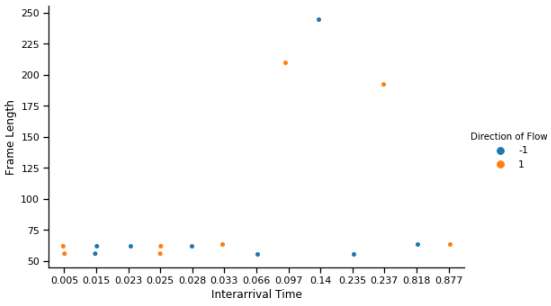

Figure 12: Fuzzers

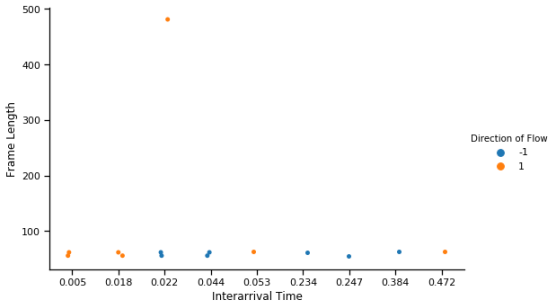

Figure 13: Worms

2.4 Machine learning

Due to the fast developing technologies, such as computers, mobile phones, the Internet, etc., the amount of data that we create is growing rapidly every day. Everything that we do today leaves a digital footprint, whether it is using the GPS to find a location, the communication with our friends, or interactions on social media. On top of this, industrial machines are generating huge amounts of data. Big data generation has brought the necessity of the fast and precise data processing, and the solution came in the form of machine learning, which is part of the already developed artificial intelligence.

We can describe machine learning with the following quote: " *A computer program is said to learn from experience* E *with respect to some class of tasks* T *and performance measure* P *if its performance at tasks in* T*, as measured by* P*, improves with experience* E*.*" [36]
In other words, machine learning is a set of machines or devices which have access to a big amount of data, and which are able to learn from this data in some programmed manner.
A machine learning algorithm can perform different tasks [23]. Some of them are:

- *Classification*: the process of assigning different input x_i to different classes k. The algorithm produces the function $f : R^n \rightarrow \{1....k\}$ and when on output $y = f(x)$, the input x_i is assigned to class k.
- *Regression*: the prediction of the output for the given input. To solve this, the algorithm produces the function $f : R^n \rightarrow R$
- *Transcription*: the process of observing a relatively untrusted representation of data in order to transcribe the data in textual format.
- *Machine Translation*: converts a sequence of symbols in one language into a sequence of symbols in another language.
- *Anomaly Detection*: the process of going through some data and trying to find an unusual or atypical behavior.

From the learning point of view, two main techniques can be distinguished **unsupervised** and **supervised** learning.

Unsupervised learning uses data samples with many features in order to learn and infer a structure from the data. The algorithm practically observes many samples of a random vector X and learns the probability distribution $p(X)$ or some other property

of the distribution.

Supervised learning uses labeled data samples and observes different features of data and labels in order to learn. In other words, the algorithm observes many samples of the random vector X and corresponding labels Y and learns according to conditional probability $p(X|Y)$.

Semi-supervised learning is a category which is in-between the supervised and unsupervised data. It uses a small portion of labeled data and a bigger portion of unlabeled data in the training process. This way the algorithm is able to produce better accuracy then by using unsupervised learning.

Reinforcement learning is a different category of machine learning which is based on the trial and error. The algorithm interacts with the environment and learns the ideal behavior through producing an action and discovering error or reward.

2.4.1 Deep Learning

Deep learning is a subcategory of machine learning based on the neural network and the conceptual brain model. This concept was presented by McCulloch in [34] for more than 70 years ago, but due to the limitations of technology, it did not have any practical application up to recently.

Neural Network

The brain uses million neurons structured in hierarchical networks and interconnected via axons, which are transporting the electronic signals called synapses from one neuron to another.
The same concept is represented by the neural network. It contains several layers with a number of neurons. A network neuron has a similar structure as a biological neuron.

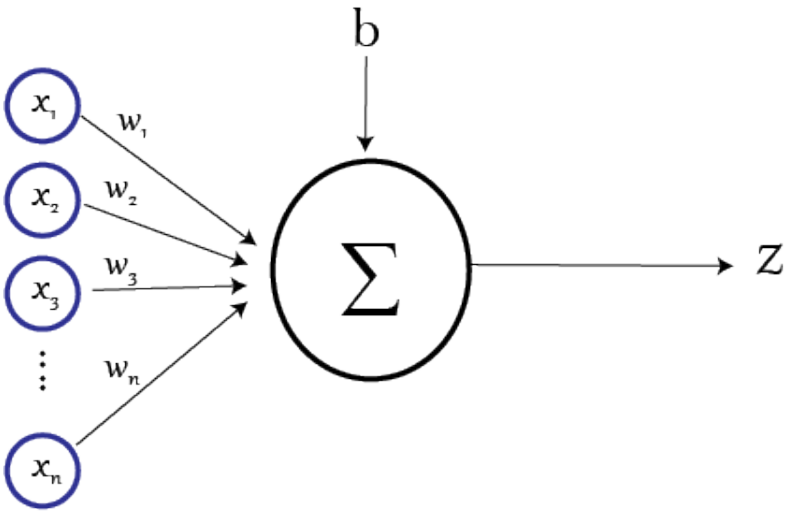

Figure 14: Neuron

Every neuron is fed with the input data $x_1, x_2, ...x_n$. Each input data is multiplied with weights $w_1, w_2, ...w_n$ and summed all together. The bias b is added to the sum of the weighted input data and on the output of the neuron we have:

$$z = (\sum_{i=1}^{n} w_i x_i) + b \tag{1}$$

Output z is passed through the activation function, and in the end we have:

$$\hat{y} = f((\sum_{i=1}^{n} w_i x_i) + b) \tag{2}$$

Weights and biases are playing a crucial role in supervised learning. They are random numbers which are adjusted during the training in order to minimize the error between the predicted values and real data. This process is called **backpropagation**, which we are going to explain in more details.
Another very important part is the **activation function**. The main responsibility of the activation function is to decide which information is important for learning and up to what extent. The output of the activation function is used as the input of another layer, so a carefully chosen activation function could change the accuracy of our learning model. In order to be able to learn complex models, the activation function needs to be non-linear.

Different activation functions that are usually used in machine learning are:

- **sigmoid** - smooth function and continuously differentiable, non-linear function. Values are between 0 and 1 and it has a S shape. Backpropagation is possible but it is not so popular because of : the vanishing gradient problem, output not zero centered(hard to optimize), sigmoid saturate and kill gradient, slow convergence.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

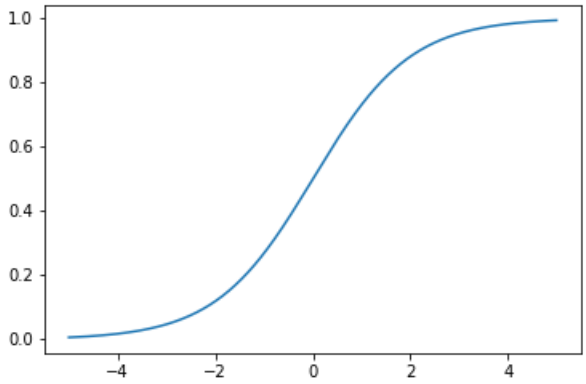

Figure 15: Sigmoid activation function

- **hyperbolic tangent** - scaled version of sigmoid. Values between -1 and 1, which outputs zero centered values.Non-linear function and error are easily backpropagated.

$$f(x) = 2sigmoid(2x) - 1 = \frac{2}{1+e^{-2x}} - 1 \tag{4}$$

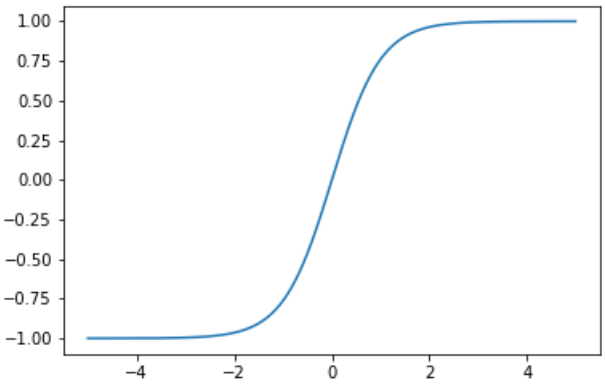

Figure 16: Tanh activation function

- **rectified linear unit(ReLU)** - non-linear function which makes error propagation easy, and it has multiple layers of neurons being activated. It improves the convergence by 6 times and it avoids and rectifies the vanishing gradient. Nowadays, this function is the one of the most employed ones, but it can only be used in hidden layers.

$$f(x) = max(0, x) \tag{5}$$

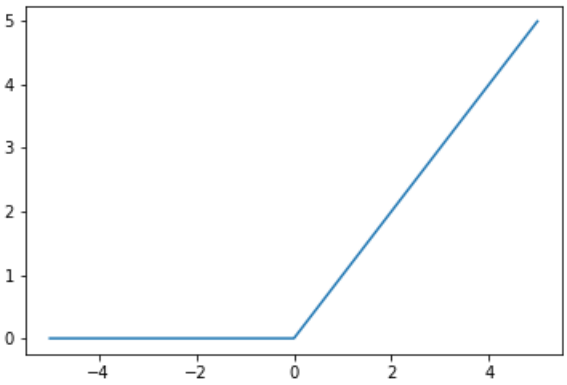

Figure 17: ReLU activation function

- **softmax** - a type of sigmoid functions, used in classification to output the probability of the class. The output of the sigmoid layer is probability between 0 and 1 divided by the sum of all outputs.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{J} e^{x_j}} \tag{6}$$

$$, i = 1...J$$

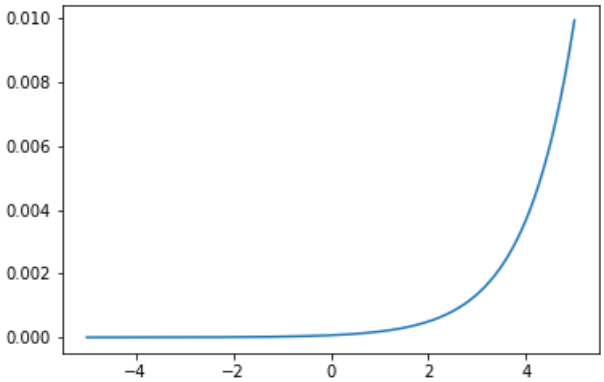

Figure 18: Softmax activation function

The neural network is represented by many connected neurons grouped in layers. We can see the representation of the neural network on the figure 19

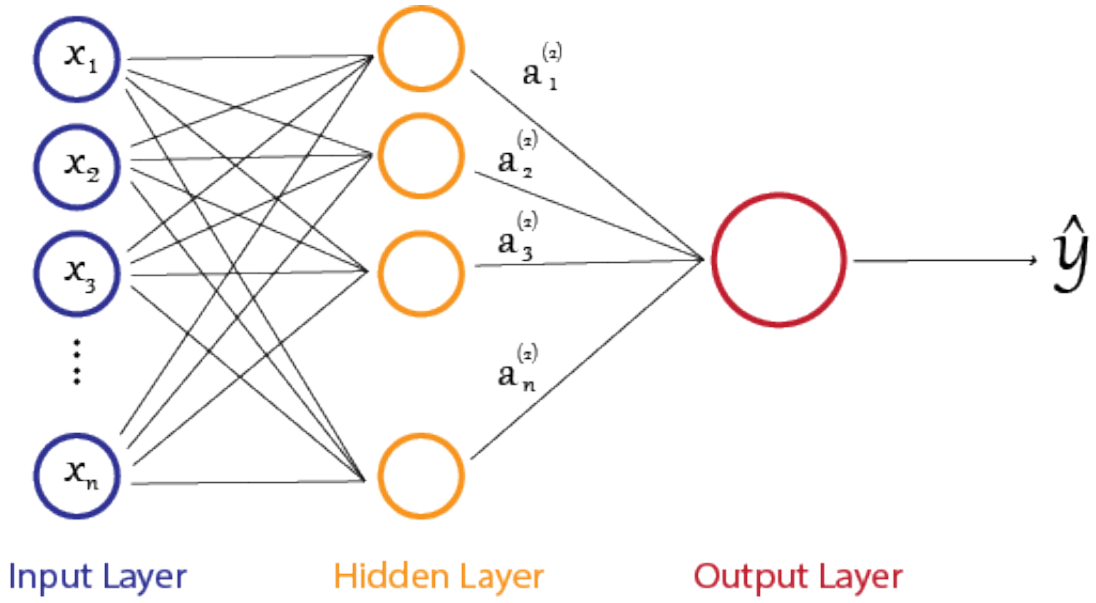

Figure 19: Neural Network [40]

Every neural network has many layers, and they are divided into groups of 3 layers: **Input layers** (leftmost,blue), **Hidden layers** (middle, orange), **Output layers** (rightmost, red). It is possible to have more than one hidden layer and they are connected to each other, so that the output of a previous layer is the input of the current layer. In the example on the Fig 19, we have $x_1, x_2, ...x_n$ which are passed to the hidden layer. The layers are denoted as L_l and for this example $l = 1$. Each input data x_i is connected to each neuron in the hidden layer L_1, and each data is weighted with $x^l_{i,j}$ and summed with the bias b_i. The output is passed through activation function of layer l, a^l, and in the end we have:

$$\begin{aligned}
a_1^{(2)} &= f(z_1^{(2)}) = f(w_{1,1}^{(1)}x_1 + w_{1,2}^{(1)}x_2 + w_{1,3}^{(1)}x_3 + ... + w_{1,n}^{(1)}x_n + b_1^{(1)}) \\
a_2^{(2)} &= f(z_2^{(2)}) = f(w_{2,1}^{(1)}x_1 + w_{2,2}^{(1)}x_2 + w_{2,3}^{(1)}x_3 + ... + w_{2,n}^{(1)}x_n + b_2^{(2)}) \\
a_3^{(2)} &= f(z_3^{(2)}) = f(w_{3,1}^{(1)}x_1 + w_{3,2}^{(1)}x_2 + w_{3,3}^{(1)}x_3 + ... + w_{3,n}^{(1)}x_n + b_3^{(3)}) \\
&\vdots \\
a_n^{(2)} &= f(z_n^{(2)}) = f(w_{n,1}^{(1)}x_1 + w_{n,2}^{(1)}x_2 + w_{n,3}^{(1)}x_3 + ... + w_{n,n}^{(1)}x_n + b_n^{(n)})
\end{aligned} \tag{7}$$

$$\hat{y}(x) = a^{(3)} = f(w_{1,1}^{(2)}a_1^{(2)} + w_{1,2}^{(2)}a_2^{(2)} + w_{1,3}^{(1)}a_3^{(2)} + ... + w_{1,n}^{(2)}a_n^{(2)} + b^{(3)}) \tag{8}$$

We can represent the output in the vector-matrix form:
Input data: $X^{1\times n} = \begin{bmatrix} x_1 & x_2 & x_3 & ... & x_n \end{bmatrix}^T$

Weights :$W^{l\times n} \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & ... & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{3,3} & ... & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & ... & w_{3,n} \\ ... & & & & \\ w_{n,1} & w_{n,2} & w_{n,3} & ... & w_{n,n} \end{bmatrix}$

bias: $B^{1\times n} = \begin{bmatrix} b_1 & b_2 & b_3 & ... & b_n \end{bmatrix}^T$

Activation: $A^{1\times l} = \begin{bmatrix} a_1 & a_2 & a_3 & ... & a_l \end{bmatrix}^T$

Neuron output: $Z^{1\times l} = \begin{bmatrix} z_1 & z_2 & z_3 & ... & z_l \end{bmatrix}^T$

We can write in more convenient vector form:

$$a^{(2)} = f(z^{(2)}) = f(W^{(1)}X + B^{(1)}) \tag{9}$$

$$\hat{y}(x) = a^{(3)} = f(z^{(3)}) = f(W^{(2)}X^2 + B^{(2)}) \tag{10}$$

and activation for next layer:

$$a^{l+1} = f(z^{(l+1)}) = f(W^{(l)}A^{(l)} + B^{(l)}) \tag{11}$$

The matrix representation allows to use linear algebra operations, which makes the calculation much faster.
The calculation correspond to *feedforward* networks or the *forward pass*. In order to optimize our network and to be able to perform learning, we need to backpropagate the error and to adjust the weights and biases.

Backpropagation algorithm

As already explained, for supervised learning we need to provide the input data: $X = \begin{bmatrix} x_1 & x_2 & x_3 & ... & x_n \end{bmatrix}^T$, and the corresponding label pairs: $Y = \begin{bmatrix} y_1 & y_2 & y_3 & ... & y_n \end{bmatrix}^T$
Gradient descent is usually used for training the network.
Gradient descent is an iterative algorithm which is used to find the minimum of a function. It starts with a random point on the graph and takes proportional steps in the direction of the negative gradient to find the local minimum of the graph. This basically means the following: if we have a function $F(x)$ which is differentiable in neighborhood of a, the function decreases faster if we move in the direction of $-\Delta F(a)$.

$$a_{n+1} = a_n - \alpha\Delta F(a_n) \tag{12}$$

In machine learning terms, gradient descent is an optimization algorithm which takes small steps, usually defined by the *learning rate* to minimize the cost.
For minimizing the cost, we often use *mean square error* cost function [40]

$$J(W,b) = [\frac{1}{m}\sum_{i=1}^{n}(\hat{y}(x^{(i)}) - y^{(i)})] + \frac{\lambda}{2}\sum_{i=1}^{l}\sum_{i=1}^{s_l}\sum_{i=1}^{s_l+1}(W_{i,j}^{(l)})^2 \tag{13}$$

The first term of equation represents an average square error and the second term is used for regularization, which tends to decrease the magnitude of weights in order to prevent overfitting.
Firstly, we initalize $W_{i,j}^{(l)}$ and $b_i^{(l)}$, with a random value near 0, in order to minimize the cost. It is important that the number is random, so that every layer can learn independent functions, in order to achieve non-linearity. Our main goal is to minimize the cost function $J(W,b)$ as a function of W and b. One iteration, used to adjust parameters, is stated below:

$$W_{i,j}^{(l)} = W_{i,j}^{(l)} - \alpha\frac{\partial}{\partial W_{i,j}^{(l)}}J(W,b)) \tag{14}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)) \tag{15}$$

In general, backpropagation is performed through the following steps:

1. for the input pairs (x,y), a forward pass is performed and we get the output $\hat{y}(x^{(i)})$

2. for each output unit i, in the layer l we calculate error δ_i^l

$$\delta_i^l = (\sum_{i=1}^{S_l+1} W_{i,j}^{(l)}\delta_i^{l+1})f'(z_i^{(l)}) \tag{16}$$

3. computing the partial derivation of the error in the function of weights and biases:

$$\frac{\partial}{\partial W_{i,j}^{(l)}}J(W,b) = a_j^{(l+1)}\delta_i^{l+1} \tag{17}$$

$$\frac{\partial}{\partial b_i^{(l)}}J(W,b) = a_j^{(l+1)}\delta_i^{l+1} \tag{18}$$

Overall, the iterations of gradient descent in the vector-matrix form can be implemented as follows:

1. set $W^{(l)} = 0$ and $B^{(l)} = 0$ for all l
2. perform iteration for m steps
   (a) use backwardpropagation to calculate the error derivation: $\Delta_{W^{(l)}} J(W, b)$ and $\Delta_{b^{(l)}} J(W, b)$
   (b) set:
   $$\Delta W^{(l)} = \Delta W^{(l)} + \Delta_{W^{(l)}} J(W, b) \tag{19}$$
   (c) set:
   $$\Delta b^{(l)} = \Delta b^{(l)} + \Delta_{b^{(l)}} J(W, b) \tag{20}$$
3. update weights and biases:
   $$W^{(l)} = \Delta W^{(l)} - \alpha[\frac{1}{m}(\Delta_{W^{(l)}}) + \lambda W^{(l)}] \tag{21}$$
   $$b^{(l)} = \Delta b^{(l)} - \alpha[\frac{1}{m}\Delta_{b^{(l)}}] \tag{22}$$

2.4.2 Recurrent Neural Network and LSTM

A recurrent neural network is based on the human memory and the basic principle of understanding the context of something that we read or hear. This network differs from other neural networks in neuron structure because each neuron has a loop which allows the information to persist and to be transferred to the next step.

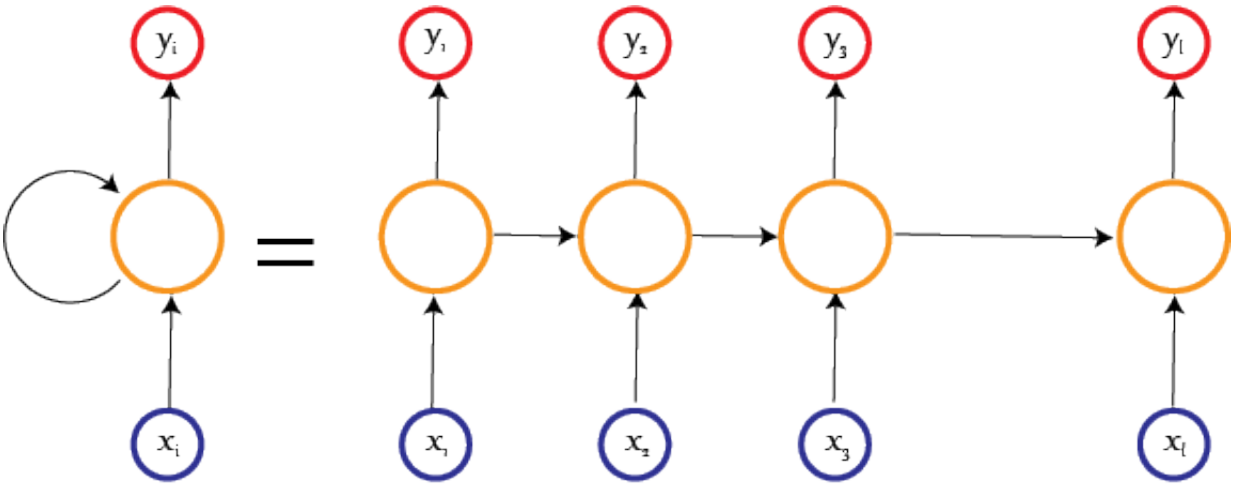

Figure 20: Recurrent Neural Network [41]

We can imagine a recurrent neural network as multiple copies of a same network, each passing the information to another. This approach is very useful in processing sequential data which requires some context, such as translation, speech recognition, language modeling, etc. Unfortunately, a classical RNN is able to handle only a short term dependency. The information that is passed to another layer, does not have much influence on learning after few steps, and the recent information has the most impact on learning. [13] This problem is solved by long-short term memory cells.
LSTM [24] cells are able to handle a long-term dependency by remembering the information which is easy to learn and forget the difficult one. LSTM have a chain structure, similar as RNN, and each cell in chain has four parts.

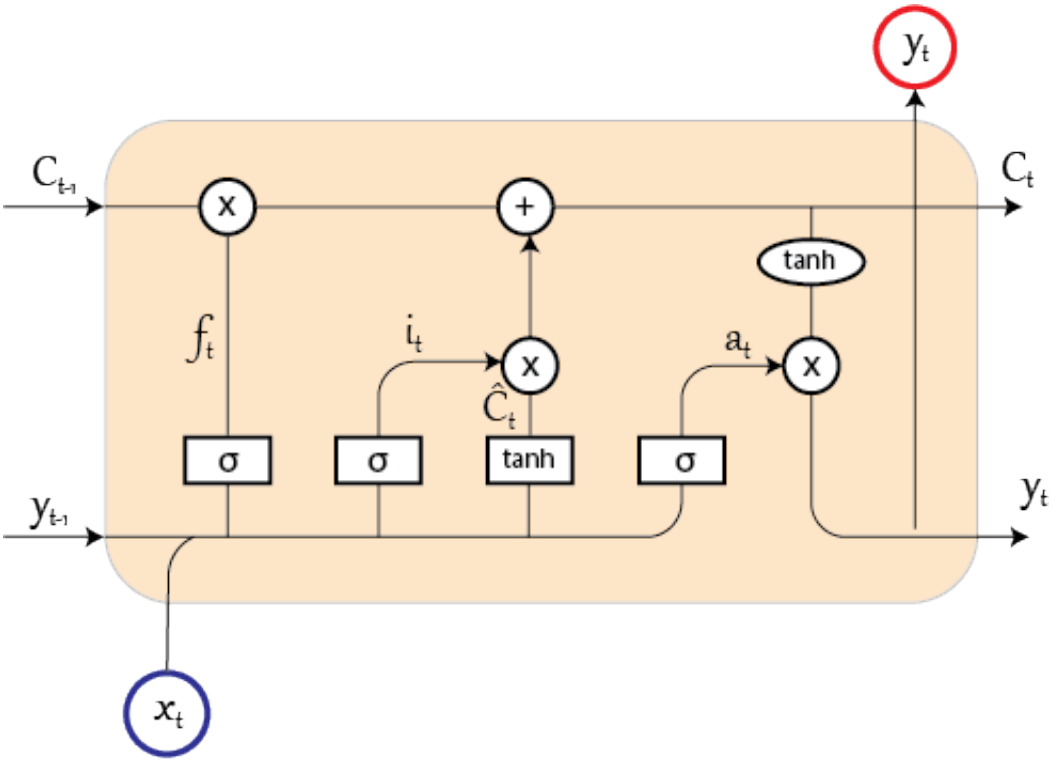

Figure 21: Long-short Term Memory cell [41] - shows 4 gate structure described in following text

The main part of the LSTM is the **cell state**. The cell state is running through the cell and it is updated with new information through a structure called **gates**. Gates are using $sigmoid$ layers or different multiplicative operations in order to decide which information is going to be passed to the cell state.

- the first layer is the **forget gate** layer and it decides which information should be forgotten . The input x_t and output of the last cell y_{t-1} are passing through $sigmoid$ function and according to the output the call state will be updated.

$$f_t = \sigma(W_f[y_{t-1}] + b_f) \tag{23}$$

- the second layer decides which information is going to update the cell state C_{t-1}. The $sigmoid$ layer is used to determine which value i_t is going to be updated, and the $tanh$ layer determines the candidate value $\tilde{C}_t$ for an update.

$$i_t = \sigma(W_i[y_{t-1}, x_t] + b_i) \tag{24}$$

$$\tilde{C}_t = tanh(W_C[y_{t-1}, x_t] + b_C) \tag{25}$$

- the third layer updates the old cell state C_{t-1} by multiplying it with f_t, and deciding what to forget. Then the new candidate value $i_t * \tilde{C}_t$ is added

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{26}$$

- the fourth layer decides what leads to the output of the cell y_t. Cell state is passing throughthe $sigmoid$ layer to decide which part is going to the output. The output of the cell state C_t is passed through the $tanh$ layer and multiplied with the sigmoid gate output a_t and then we get the filtered version of the current cell state.

$$a_t = \sigma(W_a[y_{t-1}, x_t] + b_a) \tag{27}$$

$$y_t = a_t * tanh(C_t) \tag{28}$$

Different variations of an LSTM are present. The one that we are going to use in the experiment is the gated recurrent unit(GRU). [17] This is a simpler version of LSTM which combines the forget and update gates and merges hidden state and cell state.

Gated Recurrent Unit

GRU consists of two layers called: the *update gate* and *forget gate*, and they decide what information would be passed to output. Similar as LSTM, this structure allows the information to remain in the cell state and to influence learning for longer period of time. This structure is used to successfully combat the vanishing gradient.

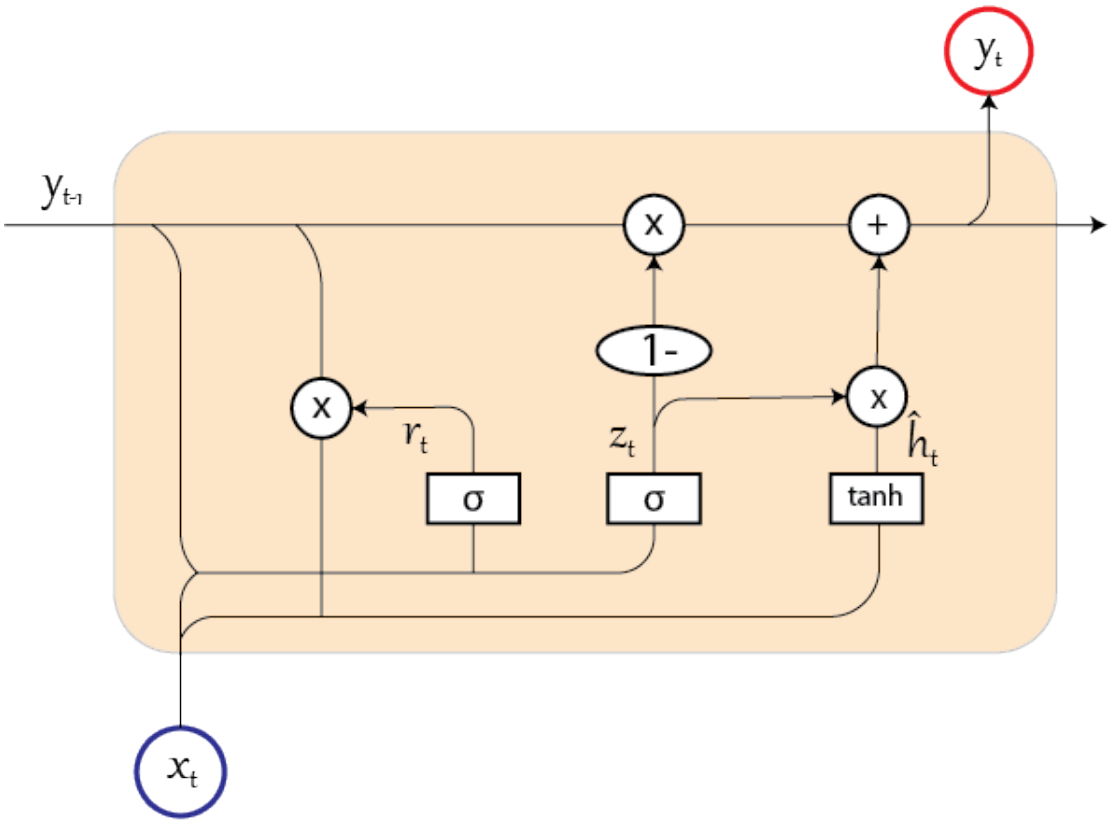

Figure 22: Gated Recurrent Unit [41] - similar structure as the LSTM, described in following paragraph

- firstly, we determine how much of the information should pass to the next step by calculating the update gate for time t:

$$z_t = \sigma(W_z[y_{t-1}, x_t]) \tag{29}$$

- secondly, we decide how much information from the previous layer is excluded by calculating:

$$r_t = \sigma(W_r[y_{t-1}, x_t]) \tag{30}$$

- thirdly, we calculate the current memory state:

$$\hat{h_t} = tanh(W[r_t * y_t, x_t]) \tag{31}$$

- finally, we calculate the output h_{t-1} which contains the current memory state and previous cell state:

$$y_t = (1 - z_t) * y_{t-1} + z_t * \tilde{y}_t \tag{32}$$

3 State of the art

Machine learning is already successfully used in network traffic classification. Numerous authors are proposing a variety of classification methods.

Moore et al. [37] used flow-based traffic and proposed supervised machine learning for data classification. The data was hand-classified into categories and categorized data sets were combined with other flow features(the flow length, port numbers, time between consecutive flows) and it were used to train the classifier. The experiment showed that, by using a naive Bayes classifier, it is possible to achieve a 65% accuracy when combining the data from the same time period, and a 95% accuracy with a number of refinements.

Zuev et al. [51] also used hand-classified data which was divided into flows with the naive Bayes statistical method. The authors used classes of traffic which were defined as common groups of users (bulk, database, interactive,attack,multimedia, etc). The flow was defined with 249 different discriminators, such as: the flow duration statistics, TCP port information, payload size statistics, Fourier transform of packet inter-arrival time discriminators, etc. The result showed that the correct classification was 66.7% . Some of the classes were classified with more success than others, e.g. services(DNS, X11) and bulk traffics(FTP) with around 90% of correctly-predicted flows.

The problem which could appear in the last 2 research papers is that the flow features, such as flow duration, time between flows, etc., are used. This implies that the flow needs to be finished in order to apply the accurate classification. This is usually not the case in real-time classification and it is difficult to handle, in case of long flows. Most of these methods use flow traffic which requires the whole flow. It is difficult and not suitable for on-line classification.
Our research tend to solve this problem problem by using packet based feature vector. This approach helps learning process to be more independent of network flow, which could be lengthy, and it is applicable in sequential and on-line data.

The possible solution for this problem was introduced by Bernaille et al. [14], with the theory that the first few packets, which come after the TCP 3-way handshake and the application negotiation phase, are good predictors of traffic classes. The researchers used unsupervised clustering to find the natural clusters in an unlabeled data set. Only the first five packets of the flow were considered, and this showed good performance with more than 80% of all data identified.
This theory is the core of our work and we will try to implement it with a slightly different feature set and modern Deep Learning framework.

Some recently performed research suggests using new concepts in deep learning, such as DNN, RNN, natural language processing (NLP), etc.
Tang et al. [45] proposed a flow based anomaly detection in Software Define Network (SDN) environment by using deep neural network. The classification was based on 6 different features (duration, protocol, src_bytes, dst_bytes, count, srv_count), and it performed with the accuracy of 75.75%. The performance was evaluated with other machine learning algorithms(NB, SVM, DT), and it showed satisfactory results.

In the research paper [31], the authors introduced using NLP in a intrusion detection, because the language model can learn the semantic meaning and interactions of each system call which was not possible with the existing methods. They used a RNN-based language model with LSTM units to enhance long-range dependencies. The result showed that the LSTM model with several hundreds of units performed a lot better than KNN and KMC algorithms. This method also proved to be more computationally efficient than other methods, as it had a smaller training overhead because it required no database to be built.
This work withdraw our attention, and we are trying to improve this idea on traffic classification and malicious traffic detection. We use different packet based feature vector in order to improve the model and provide efficient classification.

Another important research worth mentioning is [47]. The authors are used the end-to-end encrypted traffic classification method. The method was based on deep learning and it used a one-dimensional convolution neural network to learn the features from the raw traffic. The result showed the accuracy greater than 80%, and the better performance than the two-dimensional CNN.

Our research is also going in direction of this work, but tends to apply the traffic classification model on TLS encrypted data, since this protocol is not encrypting TCP header. Also, it is being assumed that since the traffic data is sequential, RNNs would give meter performance.

4 Research Methodology and Experiment

In this chapter we will discuss about the basic structure of the neural network model and the main checkpoints during the experiment. The first part explains the concept of the experiments with the main points of the research. The second part deals with the data set, and it describes the history, properties and the content of the data sets. The third part is dedicated to feature engineering, which represents the process of extraction and transformation of the desired features in order to prepare the data for the training process. The main technology of feature extraction, as well as the main difficulties and possible problems regarding this specific set, are explained. The fourth part deals with the training. A short overview of the technology is given and the models used in this experiment are described. In the last part, we go through the evaluation of our models and the methods used for this purpose.

4.1 Concept

The main objective of this research is to develop an efficient model which could recognize a security threat in encrypted network traffic. This is done by using recurrent neural network with different models and a different number of layers and cells. The layer number usually varies between 1 and 5 layers, and the cells used in the layers are mostly long-short term memory cells and gated recurrent unit cells. For this purpose, a labelled data set is used with a new feature set. Different models with different layer configurations are presented, and different regularization techniques are implemented in order to find the most efficient model with high accuracy and minimal loss.
The main checkpoints of the experiment are shown in Figure 23 :

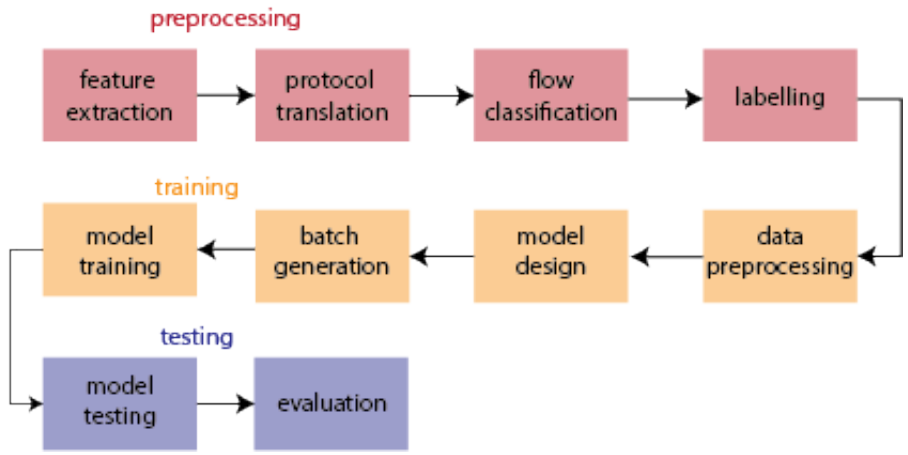

Figure 23: Experiment diagram - shows main checkpoints during our research (explained in details in: 4.2 and 4.3)

Data set

In the process of machine learning, an important step is choosing the appropriate data set and extracting the most important features from the data. For our experiment, the crucial characteristic of the data set is that it is big enough and up to date. For the network traffic classification and security threat recognition up-to-date means that it contains traffic with new applications and recently developed protocols, as well as current threats and attacks. There are a lot of possibilities of different data sets available for this purpose.

DARPA98 [1] was made by the Lincoln laboratories at MIT University in 1998. It represented a simulation of normal and malicious traffic in a military network environment. The data set contains 9 weeks of raw tcpdump [10] data, which is divided into seven weeks of training data with around 5 million connections, and 3 weeks of test data with around 2 million connections. **KDDCup99** [5] data set is an updated version of DARPA 98 with 41 features divided into three groups of features: intrinsic features, content features and traffic features. The set also contains 22 attack types. According to significant research based on this data set, its three main disadvantages are: TTL value does not correspond to the real world traffic characteristics, probability distribution is different in the training set and test set and the set is missing the recently reported low footprint attacks.

NSLKDD [46] data set is an updated version of KDDCup99, which endeavors to solve certain problems which came with the old data set. Firstly, it removes duplicated records in the training and data sets. Secondly, it randomizes data by picking up records from different points in the data set, and it removes the unbalanced number of records in the training and test set. The data set still does not include the modern low footprint attack scenarios which makes this data set unsuitable for the training models for the classification of modern network traffic.

UNSW-NB15 is a data set which is created by a cybersecurity research group at the Australian Centre for Cyber Security (ACCS). The IXIA Perfect Storm tool [4] was used to create normal and malicious network traffic, with an updated database of new attacks from the CVE web-site [3]. Three virtual servers were set, two producing normal and one producing malicious traffic. The generator architecture picture is as following:

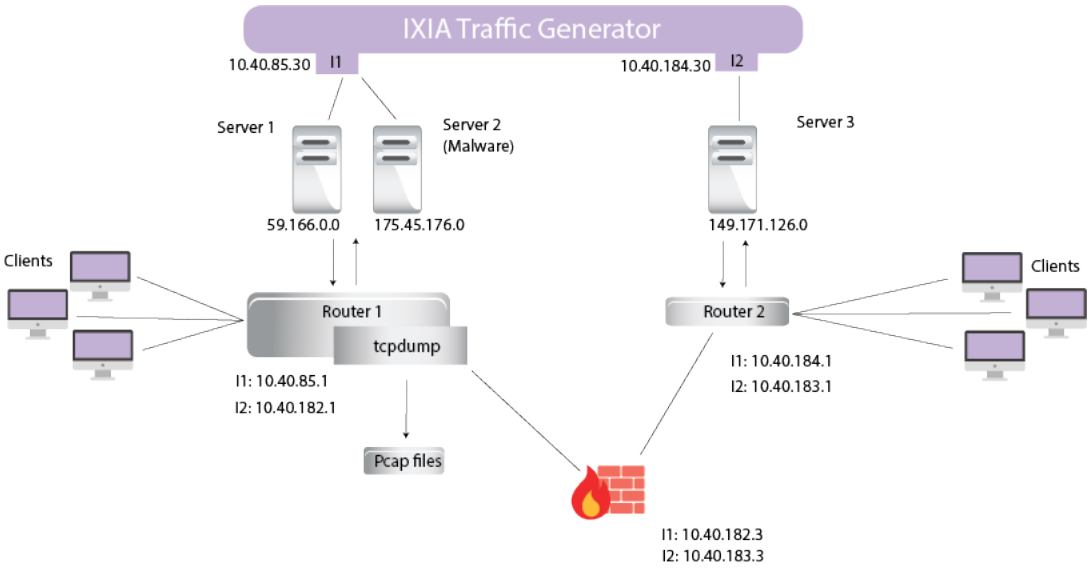

Figure 24: UNSW-NB15 data set generator architecture [38]

The traffic was captured with tcpdump, in duration of 16h on 22 January 2015, and 15h on 17 February 2015, and it contained around 100GB of data. It contains 49 features of traffic and 9 families of attack. The statistics of the data and attacks are shown in table:

Statistical features		16 hours on 22.1.2015	15 hours on 17.2.2019
No._ of_flows		987,627	976,882
Src_bytes		4,860,168,866	5,940,523,728
Des_bytes		44,743,560,943	44,303,195,509
Src_Pkts		41,168,425	41,129,810
Des_Pkts		53,402,915	52,585,462
Protocol types	TCP	771,488	720,665
	UDP	301,528	688,616
	ICMP	150	374
	Others	150	374
Label	Normal	1,064,987	1,153,774
	Attack	22,215	299,068
Unique	Src_ip	40	41
	Dst_ip	44	45

Table 1: UNSW-NB15 data set specifications

Compared to the previous data sets, UNSW-NB15 contains more data, more features,

more participants in communications, as well as different attack families which correspond to modern low footprint attacks. There are 9 different attack families:

- Fuzzers
- Analysis
- Backdoors
- DoS
- Exploits
- Generic
- Reconnaissance
- Shellcode
- Worms

For all the previously stated reasons, we are using this data set for research.

Feature engineering

Data sets are usually too big and not properly scaled for an intended purpose. The input that machine learning model uses is a numerical representation of data and it is called *features*.

" *Feature engineering is an act of extracting features from raw data and transforming them into a format that is suitable for the machine learning model.*" [50]
It is an important step of machine learning because properly chosen feature sets have a strong impact on the outcome of the experiment and could significantly improve the quality of the result.
There has been plenty of research on the importance of features for network traffic classification used for training various ML models. In research paper [35], author analyses different feature vectors: UNSW Argus/Bro Vector [39], CAIA Vector [49], Consensus Vector [22], TA Vector [25], AGM Vector [26]. The following table shows the corresponding feature sets:

UNSW Argus/Bro Vector	CAIA Vector	Consensus Vector	TA Vector	AGM Vector
object:	**object:**	**object:**	**object:**	**object:**
flows (bidirectional)	flows (bidirectional)	flows (bidirectional)	flows (unidirectional)	source hosts (unidirectional)
number of features:	**number of features:**	**number of features:**	**number of features:**	**number of features:**
45 (basic)	30 (basic)	19 (basic)	13 (basic)	8 (basic)/22 (aggregated inc.)
key:	**key:**	**key:**	**key:**	**key:**
srcIP, dstIP, srcPort, dstPort, protocol	srcIP, dstIP, srcPort, dstPort, protocol	srcIP, dstIP, srcPort, dstPort, protocol	srcIP, dstIP, srcPort, dstPort, protocol	srcIP
features:	**features:**	**features:**	**features:**	**features:**
srcPort, *dstPort*, protocol, state, duration, srcBytes, dstBytes, srcTTL, dstTTL, srcLoss, dstLoss, service, srcLoad, dstLoad, srcPkts, dstPkts, srcWin, dstWin, srcTcpb, dstTcpb, srcMeansz, dstMeansz, trans_depth, res_bdy_len, srcJit, dstJit, Stime, Ltime, srcIntpkt, dstIntpkt, tcprtt, synack, ackdat, is_sm_ips_ports, ct_state_TTL, ct_flw_http_mthd, is_ftp_login, ct_ftp_cmd, ct_srv_src, ct_srv_dst, ct_dst_ltm, ct_src_ltm, ct_src_dport_ltm, ct_dst_sport_ltm, ct_dst_src_ltm	protocol, duration, srcPkts, srcBytes, dstPkts, dstBytes, min_srcPktLength, mean_srcPktLength, max_srcPktLength, stdev_srcPktLength, min_dstPktLength, mean_dstPktLength, max_dstPktLength, stdev_dstPktLength, min_srcPktIAT, mean_srcPktIAT, max_srcPktIAT, stdev_srcPktIAT, min_dstPktIAT, mean_dstPktIAT, max_dstPktIAT, stdev_dstPktIAT, #srcTCPflag:syn, #srcTCPflag:ack, #srcTCPflag:fin, #srcTCPflag:cwr, #dstTCPflag:syn, #dstTCPflag:ack, #dstTCPflag:fin, #dstTCPflag:cwr	srcBytes, srcPkts, dstBytes, dstPkts, *srcPort*, *dstPort*, protocol, duration, max_srcPktLength, mode_srcPktLength, median_srcPktLength, min_srcPktLength, median_srcPktIAT, variance_srcPktIAT max_dstPktLength, mode_dstPktLength, median_dstPktLength, min_dstPktLength, median_dstPktIAT, variance_dstPktIAT	*srcPort*, *dstPort*, protocol, bytes, pkts, seconds-active, bytes_per_seconds-active, pkts_per_seconds-active, maxton, minton, maxtoff, mintoff, interval	#dstIP, *mode_dstIP*, pkts_mode_dstIP, #srcPort, *mode_srcPort*, pkts_mode_srcPort, #dstPort, *mode_dstPort*, pkts_mode_dstPort, #protocol, mode_protocol, pkts_mode_protocol, #TTL, mode_TTL, pkts_mode_TTL, #TCPflag, mode_TCPflag, pkts_mode_TCPflag, #pktLength, mode_pktLength, pkts_mode_pktLength, pkts

Figure 25: Feature set comparison [35]

The feature sets which were the object of the research performed well with various statistical methods such as: the naive Bayes classification, random forests, decision trees, etc.
Since those classifiers are different to those that we intend to use, we assume that those feature sets would perform poorly with a recurrent neural network. In addition to that, used data are not sequential, so it would not be appropriate for RNN use. Moreover, the feature sets are not suited for encrypted traffic classification, as there are many features from list there are not available due to encryption.

Due to those reasons, as well as some other ones, we decided to develop our own feature vector, which is more packet based. This means that our feature vector do not depend entirely on traffic flow. It contains **packet length**, **time between packets** arrived in the current flow and the **direction** of a packet in flow. Most of the protocols (including encryption protocols) are connection based, and they start with the

application negotiation phase. In this phase the flow direction and packet length of the first few packets in the flow play a very important part, since the establishment of the connection scenario between the users is predetermined. Our assumption is that packet based feature vector would be a good feature to feed during model training in order to achieve attack recognition in the early stage.
This would solve usual problem that most of the researchers have with the flow-based classification, and it is the fact that the entire flow need to be finished in order to classify the traffic.

4.2 Preprocessing

This subsection is dedicated to data preparation for feeding the RNN model in the process of training.
The first step of this process represents feature extraction. The UNSW-NB15 data set comes with different features sets (Argus/Bro), but since we were not using any of those sets, we needed to develop a script which would extract desirable features.
After analysing the .pcap files which came with the UNSW-NB15 data set, by using Wireshark [12], we have extracted following features:

- source IP
- destination IP
- source port for TCP and UDP traffic
- destination port for TCP and UDP traffic
- protocol
- packet length
- packet timestemp

The features were extracted as a .csv file, which was forwarded to next step.

The second step was protocol translation. Since the protocol was represented with protocol numbers, we translated it to a string ("UDP","TCP",...), because it was required for comparison in further steps (flow classification, labelling).
This was done by a Python script which compared the protocol number from the .csv file with the protocol number list made by IANA [8].
During the translation, we noticed that a great portion of protocols in data set is approximately as follows:

- TCP: 60%
- UDP: 39%
- others: < 1%

The third step was classifying the data set into a list of traffic flows. As the indicator of the flow, a 5-tuple is used, as is usual: Source IP, Destination IP, Source Port, Destination Port and Protocol.

Finding a good logic behind flow assignation was relatively tricky, since this part was significantly time consuming. The first idea was to use a double loop structure, in which the first packet was assigned to a new list and that was compared with all the other entries in the .csv file, in order to find a match. When the match between packets 5-tuple was found, the flow features list (packet length, timestamp) was updated with a new value and another feature, flow direction, was added. Since the .csv files are enormous, with around 2 million entries, this process was not time efficient and we needed to find another solution for this problem.
The solution came in the form of Python function dictionary. The 5-tuple was used as dictionary key, and two features(packet length and timestamp) were used as dictionary values. This decreased the time of processing because the script did not need to iterate through the whole data file. It only needed to check the unique 5-tuple pairs, which were not more then 30,000 per data file. When the 5-tuple match, the dictionary values are updated with packet length, timestamp, and flow direction. Flow direction was generated according to a *source-destination* relation, namely, if packet goes from source to destination, number 1 is assigned, and if packet goes other way (destination to source), -1 is assigned. Feature vector was created per packet and it contained packet length, timestamp and direction of packet. After flow processing, the dictionary values (list of flow vectors) had a 3-dimensional list structure which corresponded to a tensor. This kind of output was the most suitable one for TensorFlow, a machine learning framework, which will be explained in the next chapter. The flows were exported as a dictionary file and handed to the next step.

The fourth step of preprocessing was labelling. The data set that we used is not directly labelled. The set was coming with a separate *Ground Truth(GT).csv* file which contained a list of attacks. The GT file contained: the source IP, destination IP, source port, destination port, protocol, starting time, finishing time, attack name, attack description.
We developed a Python script which compared GT file entries with a recently created traffic flow dictionary. Since we already had problems with time consumption due to the multiple iterations, we decided to use again the dictionary logic again. Firstly, the GT file was converted to a dictionary, with the 5-tuple as the dictionary key and the

attack features (starting time, finishing time) as the dictionary value. The script was iterated through the flow dictionary and GT dictionary in order to find the match. When the match was found, another iteration was performed through the GT values in order to find whether the time of the attack also matched. If the time matched, the flow dictionary values were updated with the new value 1. If the two entries in the flow and GT dictionaries did not match, the flow dictionary value was updated with 0.
The last step of the preprocessing phase was calculating the time between the packets and exporting the data set into a .csv file. The calculation was done by subtracting the timestamp of the two subsequent packets. This was very useful to us because it could help us recognize a protocol connection pattern in network traffic. The idea came from language processing, which was the main strength of neural network machine learning. In language processing, the next word in a sentence or the next character in word depends the most on the last word or character. A model learns dependencies between various words and characters and tries to recognize some pattern how the sentence is formed. As this worked very well on language processing, we decided to apply this principle to our experiment. We find that the network traffic (as well as malicious attacks) has similar structure as language sentence, because it also follows some patters. The following experiment showed that this was a good idea.
After the time calculation, the data set was exported to a .csv file and the training phase could begin.

4.3 Training

After the preprocessing phase, the data was fed to the training phase, in which we developed the model by using TensorFlow. We used this model to learn from the data to distinguish normal from malicious traffic.

4.3.1 Training the model

The first step in training the model was to prepare the input data to be suitable for TensorFlow.
The input data was divided into **x_data**, which contained all the values and features of traffic, and **y_label**, which contained the labels of attack or non-attack traffic. The relation between traffic features and labels were based on instance number. We started with splitting the input data to the train set and test set. The train set contains 80% of the input data, and test set contained the remaining 20%. During the splitting, additional shuffling of the flows was used to undermine the possibility of some connection between the subsequent flows. Necessity for shuffling comes from the fact that the used data set was generate in laboratory, so, from our point of view, it is lacking the randomness in traffic generation. During a learning process, there is a possibility for model to recognize some pattern of generating the attacks, for example.

The data set and its component size are described in Table 2

Dataset size	Training data	Test Data
1,806,468	1,445,174	361,294

Table 2: Data set

The next step was to create a suitable model for learning. Different models were created to test different possibilities of deep learning within TensorFlow(table provided in Appendix):

1. **Basic LSTM model** (BLSTM) - we selected this model because it was simple and fast, and therefore suitable for testing different parameters. During the testing we trained this model with following parameters:
   - 1,3 and 5 layers of basic LSTM
   - different cell numbers in layer
   - different learning rates
   - different maximum batch sizes
2. **Batch Normalization LSTM model** (BNLSTM) - this model was selected because, as explained in following paragraph, the batch training is used, so this was logic step forward in terms of improvement of 1.model. We are expecting some improvements in performance using this model. Tested parameters:
   - 1 and 3 layers of basic LSTM
   - batch normalization layer
3. **Layer Normalization LSTM model** (LNLSTM) - this model was selected to test another similar normalization method. This method is more complex then batch normalization so we are expecting some improvement, with increase in training time compared to 2.model. Tested parametes:
   - 1 and 3 layers of basic LSTM
4. **Gated Recurrent Network model** (GRU) this model was selected because the cell structure is similar as LSTM cell in 1.model, so it is suitable for comparison with this model. According to [41], there is some improvement in training time and performance expected. Tested parameters:

- 1 and 3 basic GRU cells

There are different views on how to feed data into a model. The simplest way is to feed single values to the model. This method requires much more time and effort. A better solution lies behind the term **batch training**. It represents the division of data into small sets and feeding small sets to the model. An important parameter which could have more impact on the learning process is the batch size. A bigger size of the batch training is time efficient, but there is a possibility of overshooting or undershooting. The first idea was to use a fixed batch length, but it created many questions regarding the learning process. It required padding if there are not enough entries in the data set, which could probably confuse the training model.

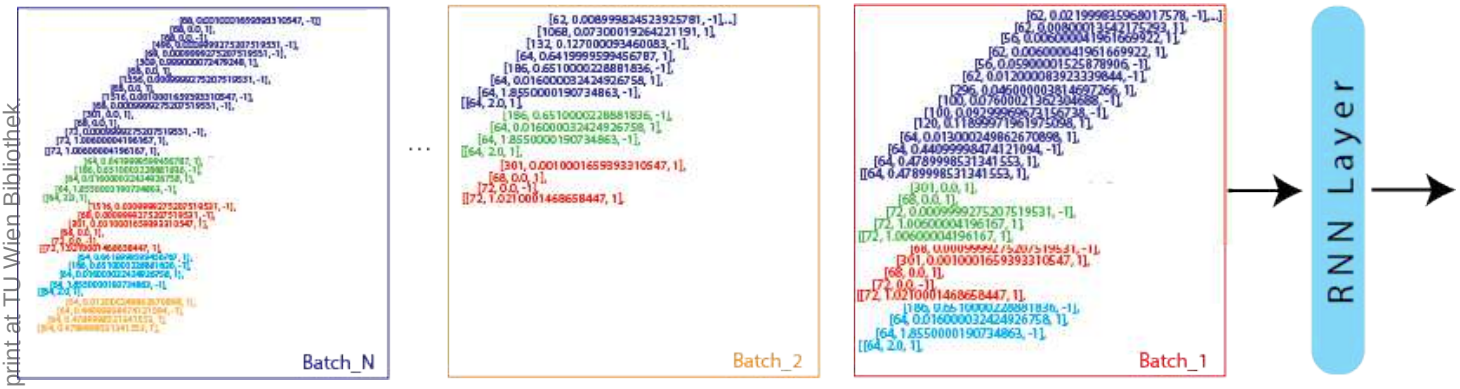

Figure 26: Batch Training example - different color rectangles represent variable size batches. Inside the batches, flows represented by their feature vectors are shown with different colors.

A possible solution to the problem, was to divide the data in the variable batches with the flow length as a parameter for this division. In this way, the flows with same characteristics would be fed into the model at the same time, which would, by our opinion, improve the learning process.
It is important to point out that, regarding the variable batch training, we have introduced another parameter, called **Maximum Batch Size**. As there is a lot of normal traffic with the same flow length, this could result in consumption of the entire RAM and crashing the model. The value that we are used was 1024 and it came from trial and error experiments. This value was a perfect combination of time consumption and model accuracy performance.

4.4 Evaluation

After the NN training, we move to the evaluation, where we use different methods to measure the performance of our model. This process gives us useful information about the performance of our models on unseen data.It also gives us a strong indication about which part of the model could be improved, and up to what extent the improvement should be implemented.
For the evaluation, we usually split the data into 2 parts: the training set and test set. The training set is used in the training process, in which the model uses the already explained methods to learn the data features and structure by adjusting the weights and biases. This set should be as large as possible and it usually takes more than 80% of the data set. The remaining data belongs to the test set, which is used to evaluate the classification of the model.
The next part of the evaluation regards the quantifying of the model performance. We need to choose the proper metric for the classification in order to get an accurate evaluation of the model. This is done by comparing the classified labels and the original labels.
During the classification there are 4 different output states, which are represented with a **confusion matrix** :

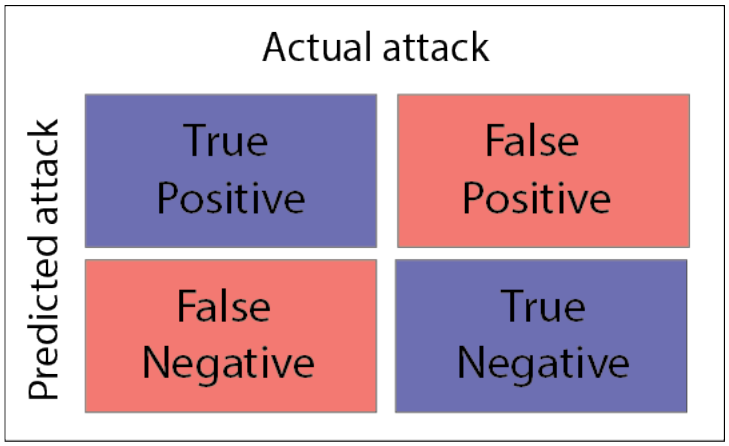

Figure 27: Confusion Matrix

- True Positive - the positive classified label matches the original label, in our case: the classified *attack* matches the *attack* in original data set
- True Negative - the classified *no attack* matches *no attack* in the original data
- False Positive - the classified *attack* does not match with *no attack* in the original data

- False negative - the classified *no attack* does not match with *no attack* in the original data

With a help of the confusion matrix, we can define different types of the metric that we are going to use:

- **Accuracy** - the number of correct predictions divided by the total number of predictions made:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Negatives + False\ Positives + False\ Negatives} \tag{33}$$

- **Precision** - the number of the correct positive predictions divided by the total number of the positive predictions:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{34}$$

- **Sensitivity** - the number of the correct positive predictions divided by the number of the original true labels:

$$Sensitivity = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{35}$$

- **Specificity** - the number of the correct negative prediction divided by the total number of the negative predictions:

$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives} \tag{36}$$

- **F1 Score** - the harmonic mean of Sensitivity and Precision. A higher number represents a better score

$$F1\_score = \frac{2}{\frac{1}{Precision} + \frac{1}{Sensitivity}} = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity} \tag{37}$$

- **Youden's J statistic** - the performance measure of a diagnostic test, where the outputs are values between 0 and 1. The value 0 represents a useless test where the classification gives the same results with or without a deep learning model,

i.e the same as random guessing. The value 1 represents a perfect test without false positives and false negatives.

$$J = Sensitivity + Specificity - 1 = \frac{True\ Positives}{True\ Positives + False\ Negatives} + \frac{True\ Negatives}{True\ Negatives + False\ Positives} \tag{38}$$

4.5 Normalization

After training and evaluating, we are comparing our models and looking for the ways to improve and optimize our code. The parameters that we considering for the optimization are: training accuracy, j-statistics, training time, etc. Training accuracy shows us how correct are our models classifying the traffic, and it is useful for comparison with similar experiments, since it is mostly used metric. Unfortunately, accuracy alone is not enough to evaluate performance of our model, since it uses sum of true positives and true negatives in nominator. J-statistic is giving precise performance evaluation, because it shows exact ratio of true positives and true negatives respectively. Training time is also one of universal metrics for model efficiency. Most of the experiments contain huge data sets and it is costly to lose time on training a not efficient models. During the neural network training, the main problem that occurs is a strong variation in input data distribution. This effect forces the optimization algorithm to adjust the weights and biases for every input step. The level of adjustments can be very high. High adjustments result in slow learning and a higher training error.
In order to avoid bad effects of input data distribution variation, we are using normalization methods. Those methods are utilized for scaling the numerical input data to a common range, which causes slight adjustments of parameters during the training and allows faster training and better precision. We are implementing two widely used methods for normalization: *batch normalization* and *layer normalization*

Batch Normalization

We are using batch training to achieve more efficiency in the training (more in the subsection 4.3.1). Firstly, the calculated loss over the batch is an estimate of the loss over the whole data set, and over time it improves with gradient descent. Secondly, batch training improves training efficiency by dividing the training set in smaller subsets and utilizing the parallelism of modern computing platforms.

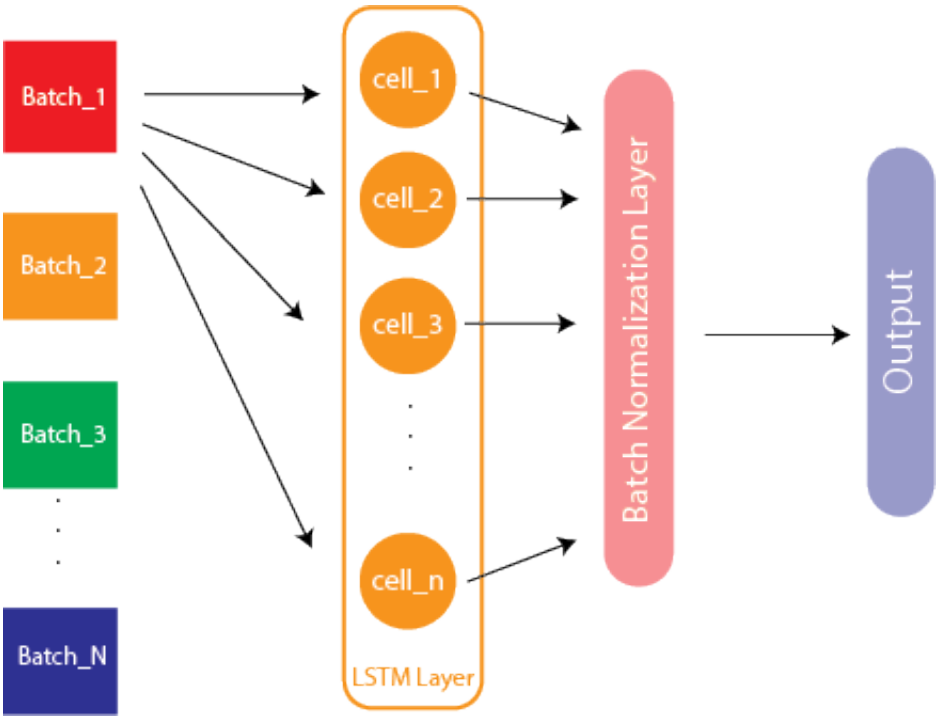

Figure 28: Batch Normalization

Although, the gradient is a simple and efficient way for training optimization, it requires carefully selected model parameters. Each parameter can influence the change of input distribution for each layer. Since the neural network adapts the weights and biases of each layer during the training, the activation of each layer would also change. Due to the change of input, each layer is forced to adapt constantly to new input distribution, which results in error propagation. It has significant consequences especially in case of more layers in the network.

The problem which occurs is called *covariate shift*, and it refers to a change of the input data distribution to a layer. This represents a problem, because we know that training converges faster if the input distribution is zero-mean and unit variance. [32] [48]

The goal of batch normalization is to reduce covariate shift in order to accelerate the training and improve precision. The reduction is achieved by normalizing the input data to the small range of values. For layer with input data, which is at the same time the output of activation function of the previous layer $a = (a^{(1)}, a^{(2)}, ...a^{(k)})$ we need to normalize each input: [27]

$$\hat{a}^i = \frac{a^{(i)} - E[a^{(i)}]}{\sqrt{Vat[a^{(i)}]}} \tag{39}$$

The shown normalization could change the expressiveness of the layer representation, as non-linear activation could become linear. To address this issue, we are adding two additional parameters $\gamma^{(k)}$ and $\beta^{(k)}$ for each activation $a^{(i)}$:

$$y^{(i)} = \gamma^{(k)}\hat{a}^i + \beta^{(k)} \tag{40}$$

The parameters scale the shifted values and represents the identity transform. Those parameters are learned during the training and they restore the representation power of the network.

Taking all the parameters into consideration, the batch normalization algorithm contains few steps:

- calculating the batch mean:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} a_i \tag{41}$$

- calculating the batch variance:

$$\delta_B^2 = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu_B)^2 \tag{42}$$

- normalization:

$$\hat{a}_i = \frac{a_i - \mu_B}{\sqrt{\delta_B^2 + \epsilon}} \tag{43}$$

- scale and shift:

$$y^{(i)} = \gamma\hat{a}_i + \beta \equiv BatchNormalization_{\gamma,\beta}(a_i) \tag{44}$$

A problem can appear with a small batch size, since the variance is close to zero and the estimates could be very noisy. There are also some problems with the recurrent neural network, since the activation for each time step has different statistics and is very sensitive to noise.

Layer Normalization

Layer normalization is fairly similar to batch normalizatio. The only difference is the object of normalization. In batch normalization we are performing normalizing across the *batch dimension*, whereas layer normalization normalizes the input across the *features*. This way of calculating allows the an independent calculation for each sample of data, as well as the use of an arbitrary batch size. Figure 29 shows the difference between the two normalization methods:

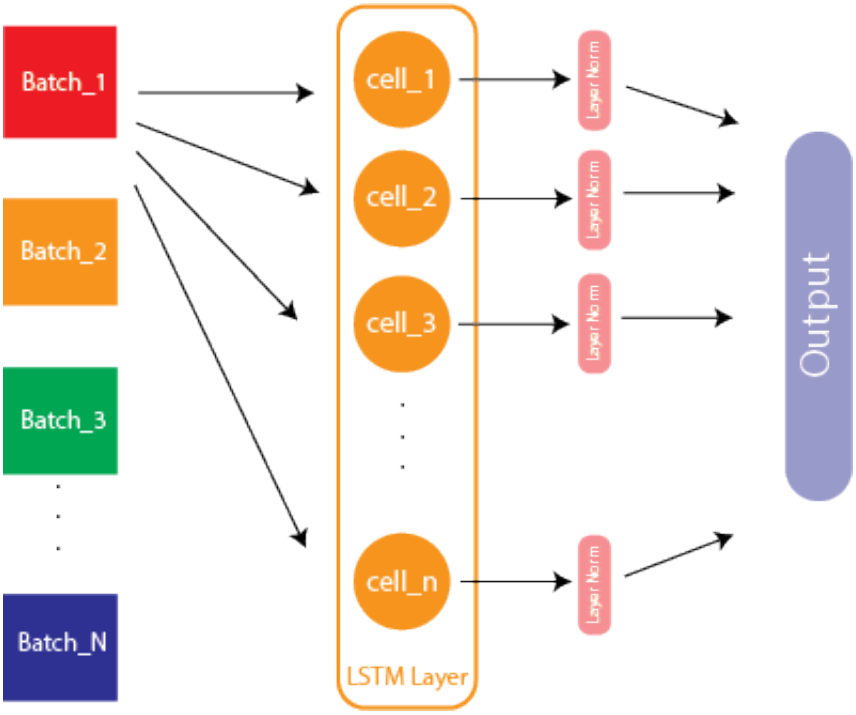

Figure 29: Layer Normalization

The mathematical representation is similar to batch normalization:[33]

$$\mu_i = \frac{1}{m}\sum_{i=1}^{m} a_{i,j} \tag{45}$$

$$\delta_j^2 = \frac{1}{m}\sum_{i=1}^{m}(a_{i,j} - \mu_i)^2 \tag{46}$$

$$\hat{a_{i,j}} = \frac{a_{i,j} - \mu_i}{\sqrt{\delta_i^2 + \epsilon}} \tag{47}$$

The Difference is in the input activation function $a_{i,j}$, where i represents a batch and j represents a feature.

5 Results

In this chapter, we are comparing the results that we got from the models that we developed. In general, we have 4 different models and they differ in the number of layers and number of cells in a layer. They also differ according to the kind of regularization techniques and with respect to their different learning rates and maximal batch size parameters.
In the first part of this chapter, we are introducing the hardware and software setups that were used for our experiments, including all their components.
The second part of the chapter is a discussion about the performance evaluation. We will compare different models in terms of the evaluating parameters, such as the number of true positives and false positives, true negative, false negatives, accuracy, precision, f1 score and j-statistics. This gives us an approximation of how good our models are, and, as a result, we can see which model would be the best one. We will discuss more about that in Chapter 5..
Aside from the common performance evaluation, we are also providing the probability of recognizing different attack categories, which is very important in model evaluation. The comparison between how different models recognize different attack categories is shown as well.
We will discuss the influence of other parameters of the deep learning model that we are using, such as the time of training, learning rate, max_batch size, etc. With this comparison we are finishing the performance evaluation and are getting the complete picture of our model, with the description of how good it is and what is it that we should do to make it better.
All evaluation results stated in this chapter will be used in the next chapter to present the best model for traffic classification by using deep learning. Future improvements will also be suggested.

5.1 Setup

First of all we need to introduce the hardware and software that we used.
In terms of the hardware setup, one custom made PC with the with following performances is used:

- AMD Ryzen R7 2700X with 16 cores
- 32GB RAM
- NVIDIA RTX 2060 with 1365 MHz, 6GB GDDR6, 1920 cores, 240 Tesor cores,

5.1.1 Deep learning framework

Nowadays, machine learning is one of the most talked about topic in data science. A lot of research have been made in this field and it is commercially present in the most industries. There are a lot of deep learning frameworks which could be used for generating a deep learning model. Most of them are open-source and they are supported by many programming languages, with Python, C++ and R at the top of the list. We will describe the most popular frameworks.

TensorFlow [11] is a framework developed by Google's AI department. It was recently launched as open-source platform, which has created a lot of attention in this field.
It is widely used because it supports multiple languages for creating a deep learning model, such as Python, C++ and R and it is available on many platforms, even on iOS and Android. TensorFlow allows us to use both the CPU and GPU processing powers, and, with today's multicore processors and parallel computing GPUs, deep learning is becoming more affordable and more available to everyone.
Tensors are multi-dimensional arrays which are used as the main core of the framework. The input data is reshaped in the tensors and fed into a computational graph, which allows us to take more features of the input data into account.

Pytorch [9] is the second most used framework, developed for Facebook, but it is also used for Twitter and Salesforce, which provides more flexibility and adaptation during computation. It uses Python as the backend, allowing for the GPU computation as well. Similar to TensorFlow, Pytorch is using also Tensors for computations. The basic advantage of TensorFlow is that this framework uses dynamic computational graphs, which change during the training.

Keras [6] is a popular API, which uses TensorFlow, Theano or Congruent Toolkit (described below) as the backend. It is very lightweight and applicable to any experiment, which makes this the best framework for beginners and for short experiments. It supports Convolutional and recurrent neural networks and it is able to run both CPU and GPU processing. A drawback of Keras would be that it is not very prone to complex models.

MXNet [7] is a highly scalable framework, developed by Apache and adopted by Amazon Web Services. It supports a great amount of languages, such as Python, C++, R, JavaScript, Julia, etc. The main advantage of this framework is that it has special features for GPU processing, which support parallel processing and multi GPU processing. It is very popular for speech and hand-writing recognition, natural language processing and forecasting.

CNTK [2] is yet another open-source framework, developed by Microsoft Research. It describes neural networks as series of computational steps through a directed graph. This framework supports the most popular model types, such as a convolutional neural network, Recurrent Neural Network, feed-forward DNN, etc.

For our experiment, we decided to use TensorFlow, since the 3-dimensional shape of tensors fits the best for our intended feature set shape. It also offers more flexibility in terms of the backend, API, etc.

The experiment is performed on Windows 10, using Anaconda to set the virtual environment. TensorFlow GPU 13.1 is used to support graphical processing. The GPU gives us the possibility of parallel computing, which is very popular among people who work on deep learning because it drastically decreases the computing time. During preprocessing, Pandas and NumPy are used for preparing the data for the training input.

5.2 Performance evaluation

In this part of the chapter, we are presenting an overview of the testing of our models after training. As explained in Chapter 4.3, we are training 4 different models with different layers and deep learning parameters in order to find the appropriate combination for the traffic classification. The experiments contain changes of layer number and cell number, learning rate, max_batch size, etc. The Main goal is to reduce the cost and avoid overfitting, as well as to get a high j-statistic and accuracy score for as little time as possible.

The design parameters for the models are the following:

- Activation function: ReLU
- Optimizer: Adam Optimizer
- Classification engine: softmax cross entropy between logits and labels
- Weight initiation: Xavier uniform initializer
- Bias initiation: Xavier uniform initializer

The characteristics of test data are shown in Table 3

Number of data	Number of attacks in test data	Percentage of attack in data
361,294	13,550	3.75%

Table 3: Test data specification

5.2.1 Model 1

Model 1 is built from basic LSTM cells with variable parameters. We are going to train 5 different variations of Model 1, which differ in the number of hidden layers, number of cells, learning rate, and max_batch value. This will show us the influence of some parameters on the deep learning model performance and lead us to the best model for traffic classification.

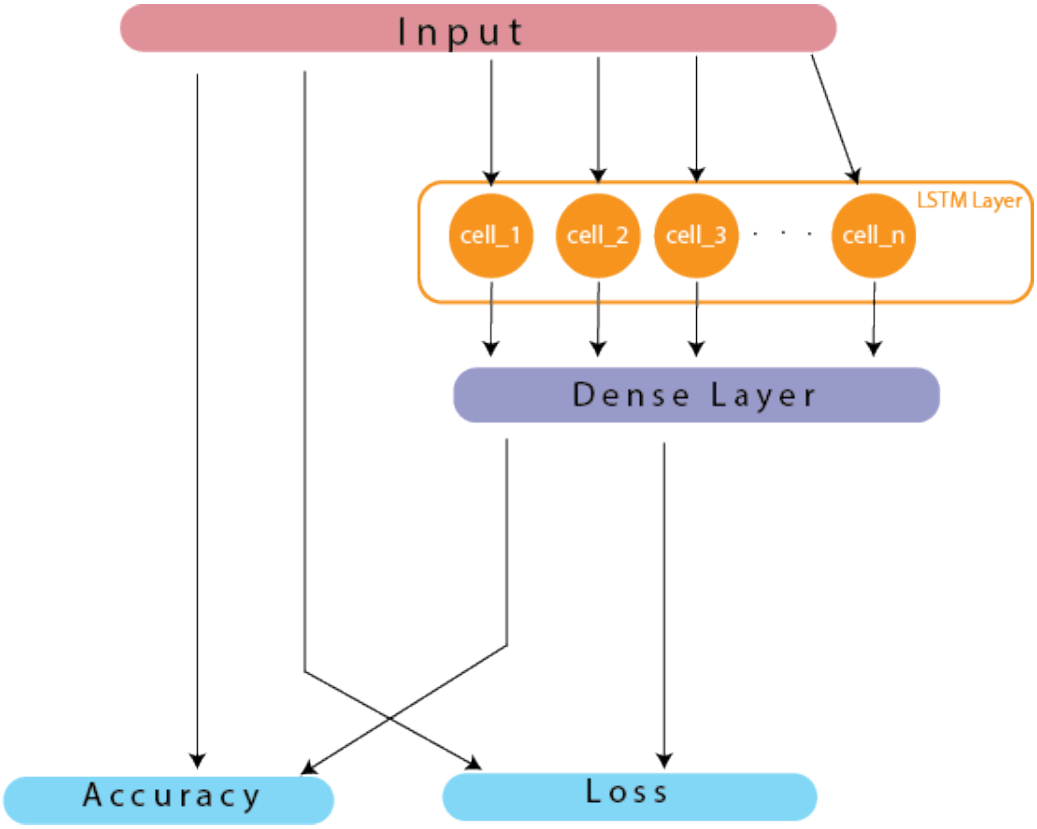

Figure 30: Model 1. BasicLSTM - consist of 1 layer of LSTM cells. The input data is passed through 1 layer of LSTM cells and passed to dense layer. Dense layer is fully connected layer which outputs single values per flow which is then used to compare with input data and to calculate loss and accuracy(reduce mean)

Number of Layers

This subsection contains the results from varying the Number of Layers in Model 1.

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1, 3, 5	128	LSTM	1	0.001	1024

Table 4: Basic LSTM Number of layers - Basic parameters

Table 5 shows the result of training Model 1 with 1, 3 and 5 layers of basic LSTM cells(128 units).

Evaluation metric	1 BasicLSTM	3 BasicLSTM	5 BasicLSTM
True Positives	11912	12637	11197
False Positives	1759	1753	2553
True Negatives	345985	345991	345191
False Negatives	1638	913	2353
Accuracy	99.05%	99.26%	98.64%
Precision	87.13%	87.81%	81.43%
Sensitivity	99.49%	93.26%	82.63%
F1 score	87.52%	90.45%	82.02%
J-statistic	87.40%	92.75%	81.90%

Table 5: Basic LSTM Number of layers - Performance evaluation

From the results we can see that the best score is achieved with 3 layers of basic LSTM cells. We can also see that, when using a 5 layer model, there is slight overfitting. Hence, there is no need to train the model with more then 5 layers because overfitting would be even worse.
In Table 6, the result of attack recognition accuracy are displayed. This can support the results from Table 5, and shows that the 3 layer model achieves the best score for any category of attacks. We can also noticed that the accuracy for some attack families are zero. This is because of the low appearance of attack in data set, and the model can not learn the pattern of attack.

Attacks	1 BasicLSTM	3 BasicLSTM	5 BasicLSTM
Generic	83.01%	88.49%	82.04%
DoS	87.35%	93.22%	81.87%
Exploits	91.06%	93.84%	81.46%
Fuzzers	86.46%	92.66%	83.41%
Reconnaissance	0%	0%	0%
Shellcode	0%	0%	0%
Backdoor	0%	0%	0%
Analysis	0%	0%	0%
Worms	0%	0%	0%

Table 6: Basic LSTM Number of layers - Accuracy of attack category classification

Learning rate

Another important parameter that we want to test is the learning rate. We are training 1 layer of basic LSTM cells(128), with learning rates of 0.01 and 0.001. The time of training is also presented.

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1	128	LSTM	1	0.001, 0.01	1024

Table 7: Basic LSTM learning rate - Basic parameters

The results are shown in Table 8

Evaluation metric	learning_rate = 0.001	learning_rate = 0.01
True Positives	11912	1482
False Positives	1759	1302
True Negatives	345985	346442
False Negatives	1638	12068
Accuracy	99.05%	96.29%
Precision	87.13%	53.23%
Sensitivity	99.49%	10.93%
F1 score	87.52%	99.62%
J-statistic	87.40%	18.14%

Table 8: Basic LSTM learning rate - Performance evaluation

From the results we can see that the best score is achieved with learning_rate = 0.001. There is a major difference between the given training learning rates, so we are not increasing the value of the parameter anymore. On the other hand, decreasing the

learning rate further would have a major impact on the time of the training, so we are taking the 0.001 value as the most efficient one. We can notice that there is a big difference in sensitivity and j-statistic between two compared experiments, while F1 score shows even improvements. This is due much worse value of true positives, which made a strong impact on sensitivity and j-statistic respectively.
Table 9, shows the results of the attack classification accuracy.

Attacks	learning_rate = 0.001	learning_rate = 0.01
Generic	83.01%	17.95%
DoS	87.35%	4.30%
Exploits	91.06%	2.46%
Fuzzers	86.46%	13.83%
Reconnaissance	0%	0%
Shellcode	0%	0%
Backdoor	0%	0%
Analysis	0%	0%
Worms	0%	0%

Table 9: Basic LSTM learning rate - Accuracy of attack category classification

Cell units

This subsection describes the results of training the model with 4 different numbers of cells in one LSTM layer. The purpose of cell variation is to find out the perfect number of the cells in the layer in order to avoid overfitting.

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1	128, 512, 1024, 1500	LSTM	1	0.001	1024

Table 10: Basic LSTM cell units - Basic parameters

Results shown on Table: 11

Evaluation metric	cell_num = 128	cell_num = 512	cell_num = 1024	cell_num = 1500
True Positives	11912	12055	12108	11999
False Positives	1759	1604	1578	1630
True Negatives	345985	346140	346166	346114
False Negatives	1638	1495	1442	1551
Accuracy	99.05%	99.14%	99.16%	99.11%
Precision	87.13%	88.25%	88.47%	88.04%
Sensitivity	99.49%	88.96%	89.35%	88.55%
F1 score	87.52%	88.61%	88.91%	88.29%
J-statistic	87.40%	88.51%	88.90%	88.08%

Table 11: Basic LSTM cell units - Performance evaluation

If we look on the accuracy and j-statistic, the best results are achieved with cell_num = 1024. Overall results show that cell_num = 128 is better solution for further experiments.
Table 12 shows the results of the attack classification accuracy.

Evaluation metric	cell_num = 128	cell_num = 512	cell_num = 1024	cell_num = 1500
Generic	83.01%	84.19%	84.83%	83.65%
DoS	87.35%	88.39%	89.43%	87.22%
Exploits	91.06%	92.46%	93.51%	92.13%
Fuzzers	86.46%	87.47%	86.99%	87.19%
Reconnaissance	0%	0%	0%	0%
Shellcode	0%	0%	0%	0%
Backdoor	0%	0%	0%	0%
Analysis	0%	0%	0%	0%
Worms	0%	0%	0%	0%

Table 12: Basic LSTM cell units - Accuracy of attack category classification

Max_batch

One of the deep learning parameters that we find very important is the maximum number of flows in a batch. We are using the variable batch size, and every batch can contain the flows with same length. It turns out that there is a huge amount of traffic with an equal flow length. This results in a few hundred thousand flows in one batch, which crash our model training because it requires an extreme amount of RAM. Aside from that, a large batch size could result in overshooting the result.On the other hand, a small number of batch size results in increasing the training time.

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1, 3, 5	128	LSTM	1	0.001	512, 1024, 5000, 10000,15000

Table 13: Basic LSTM max_batch - Basic parameters

Evaluation metric	mb = 512	mb = 1024	mb = 5000	mb = 10000	mb = 15000
True Positives	11957	11912	11948	12314	12090
False Positives	1382	1759	1798	1450	1645
True Negatives	346584	345985	345946	346294	346099
False Negatives	1371	1638	1602	1236	1460
Accuracy	99.23%	99.05%	99.05%	99.25%	99.14%
Precision	89.63%	87.13%	86.91%	89.46%	88.02%
Sensitivity	89.71%	99.49%	88.17%	90.87%	89.22%
F1 score	89.67%	87.52%	87.54%	90.16%	88.62%
J-statistic	89.31%	87.40%	87.66%	90.46%	88.75%

Table 14: Basic LSTM max_batch - Performance evaluation

According to the results, the best score is achieved with max_batch = 10000. However if we look at the accuracy, it is not possible to detect regular increasing patterns, so we can assume that the model ends up overfitting. Table 15, shows the results of the attack classification accuracy, and it also shows confusing patterns of increasing accuracy values for different attack categories.

Evaluation metric	mb = 512	mb = 1024	mb = 5000	mb = 10000	mb = 15000
Generic	83.33%	83.01%	82.04%	85.05%	83.33%
DoS	91.58%	87.35%	88.52%	90.87%	89.70%
Exploits	93.81%	91.06%	91.40%	94.45%	93.07%
Fuzzers	83.67%	86.46%	86.49%	88.84%	87.40%
Reconnaissance	0%	0%	0%	0%	0%
Shellcode	0%	0%	0%	0%	0%
Backdoor	0%	0%	0%	0%	0%
Analysis	0%	0%	0%	0%	0%
Worms	0%	0%	0%	0%	0%

Table 15: Basic LSTM max_batch - Accuracy of attack category classification

5.2.2 Model 2

As we found a good combination of the parameters for the basic LSTM model, we are trying to boost our results with the use of regularization techniques. Model 2 contains batch normalization layers, which should increase the accuracy and precision of our model, and will allow us to use higher learning rates the same or small amount of time. In this section, the results of using batch normalization are shown. A comparison between the basic LSTM model and model using batch normalization layer is given. Besides that, another comparison is made between the models using LSTM and GRU cells, and different learning rates are tested.

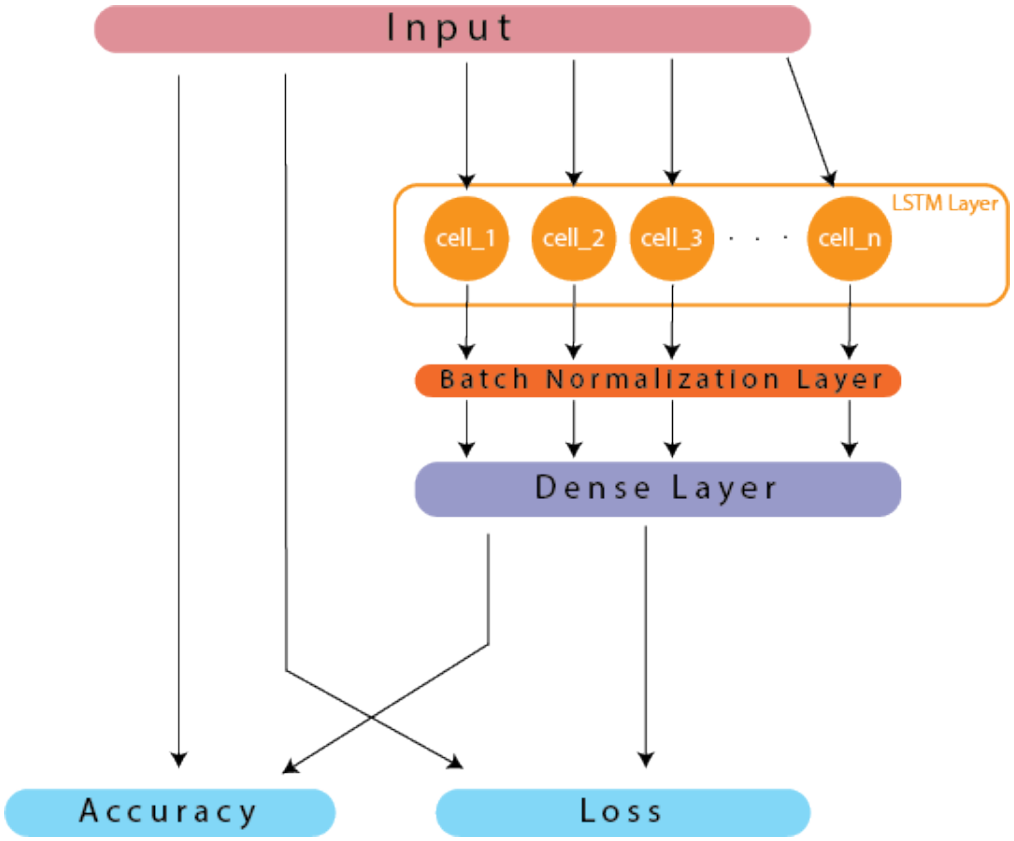

Figure 31: Model 2.Batch Normalization - consist of 1 layer of LSTM cells with one batch normalization layer between LSTM and dense layer

LSTM with Batch Normalization

This subsection shows the comparison between the models with 1 and 3 layers of basic LSTM cells(128), and the same models with a batch normalization layer. Results are shown in Table 17

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1, 3	128	BatchNorm , LSTM	1	0.001	1024

Table 16: LSTM with Batch Normalization - Basic parameters

Evaluation metric	BatchNorm	LSTM	3xBatchNorm	3xLSTM
True Positives	12346	11912	12331	12637
False Positives	1716	1759	1572	1753
True Negatives	346028	345985	346172	345991
False Negatives	1204	1638	1219	913
Accuracy	99.19%	99.05%	99.22%	99.26%
Precision	87.79%	87.13%	88.69%	87.81%
Sensitivity	91.11%	99.49%	91.00%	93.26%
F1 score	89.42%	87.52%	89.83%	90.45%
J-statistic	90.62%	87.40%	90.55%	92.75%

Table 17: LSTM with Batch Normalization - Performance evaluation

From the results, we can see that the best score is achieved with the Basic LSTM cells with the batch normalization layer. There is a slight difference in accuracy and using batch normalization showed success, which could be seen in first column. On the other hand, using more LSTM layers with batch normalization showed slightly worse performance then basic LSTM model with same number of layer, shown in third column. This is probably due to complex connection between layers.
Table 18, shows the result of attack classification accuracy.

Evaluation metric	BatchNorm	LSTM	3xBatchNorm	3xLSTM
Generic	86.55%	83.01%	87.09%	88.49%
DoS	90.61%	87.35%	89.70%	93.22%
Exploits	93.94%	91.06%	94.77%	93.84%
Fuzzers	89.32%	86.46%	88.81%	92.66%
Reconnaissance	0%	0%	0%	0%
Shellcode	0%	0%	0%	0%
Backdoor	0%	0%	0%	0%
Analysis	0%	0%	0%	0%
Worms	0%	0%	0%	0%

Table 18: LSTM with and without Batch Normalization - Accuracy of attack category classification

Learning rate

The main goal of this section is to show that by using Batch Normalization, learning rate could be decreased without significant difference in results.
Results are shown in Table 20

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1, 3	128	BatchNorm	1	0.001, 0.001	1024

Table 19: Batch Normalization learning rate - Basic performance

Evaluation metric	BatchNorm(0.01)	3xBatchNorm(0.01)	BatchNorm(0.001)	3xBatchNorm(0.001)
True Positives	2920	1121	12346	12331
False Positives	1991	868	1716	1572
True Negatives	345753	346876	346028	346172
False Negatives	10630	12429	1204	1219
Accuracy	96.50%	96.31%	99.19%	99.22%
Precision	59.45%	56.36%	87.79%	88.69%
Sensitivity	21.54%	8.27%	91.11%	91.00%
F1 score	31.63%	14.42%	89.42%	89.83%
J-statistic	20.97%	8.02%	90.62%	90.55%

Table 20: Batch Normalization learning rate - Performance evaluation

Training the model with learning_rate = 0.01 is resulting in much worse performance then learning_rate = 0.001. It showed even worse results than training model 1 with learning_rate = 0.01. There is not any possibility to decrease the learning rate using Batch Normalization Layer, according to the results.
The Table 21, shows the result of attack prediction accuracy. The results follow the same trend like in previous table. It is also worth mentioning that the results for 3 layers of LSTM with batch normalization showed worse result, probably due to the layer complexity.

Evaluation metric	BatchNorm(0.01)	3xBatchNorm(0.01)	BatchNorm(0.001)	3xBatchNorm(0.001)
Generic	25.91%	13.22%	86.55%	87.09%
DoS	17.73%	4.30%	90.61%	89.70%
Exploits	14.71%	0.98%	93.94%	94.77%
Fuzzers	24.23%	10.42%	89.32%	88.81%
Reconnaissance	0%	0%	0%	0%
Shellcode	0%	0%	0%	0%
Backdoor	0%	0%	0%	0%
Analysis	0%	0%	0%	0%
Worms	0%	0%	0%	0%

Table 21: Batch Normalization learning rate - Accuracy of attack category classification

5.2.3 Model 3

Another technique for regularization that we are using is Layer Normalization. As already mentioned in Chapter 4.5, Layer Normalization should provide us with additional

efficiency during the training. We are comparing models with Layer Normalization, Batch Normalization layers and basic model.

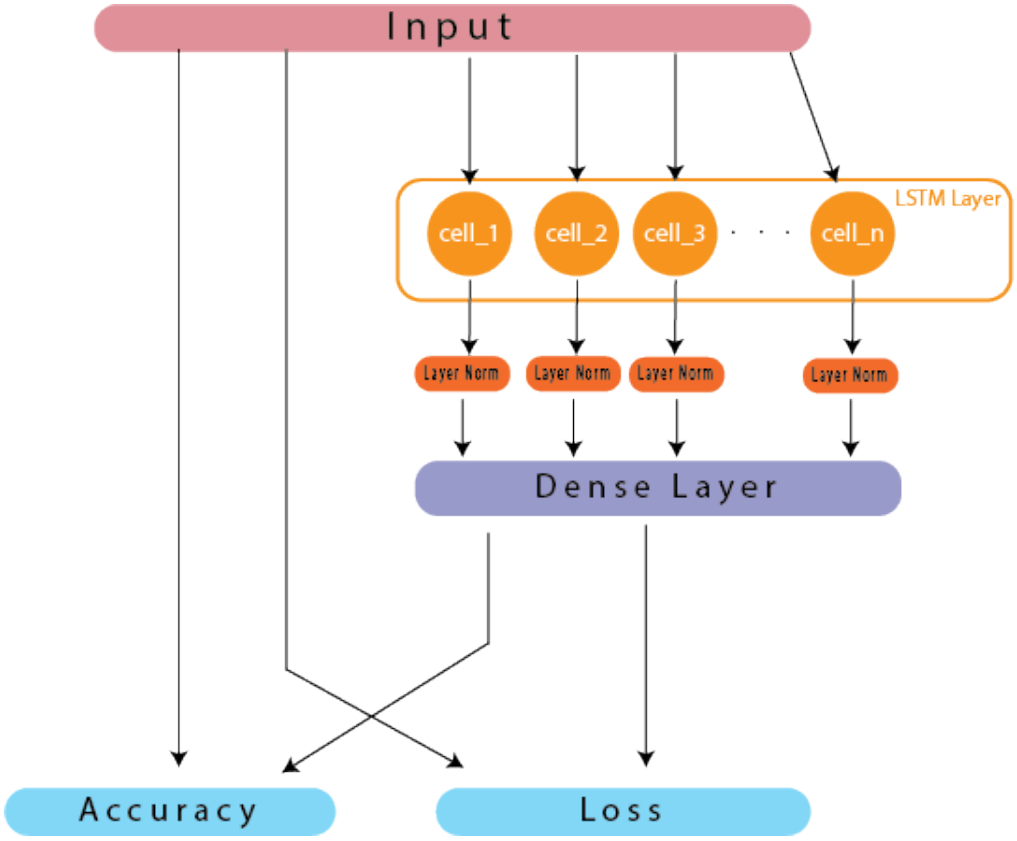

Figure 32: Model 3.Layer Normalization - consist of 1 layer of LSTM cells followed by layer normalization layer, which is connected to dense layer

Layer Normalization Layer vs Batch Normalization vs LSTM

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1, 3	128	LayerNorm, BatchNorm , LSTM	1	0.001	1024

Table 22: LSTM with Layer Normalization Layer and Batch Normalization - Basic parameters

Results are shown in Table 23

Evaluation metric	LayerNorm	BatchNorm	LSTM
True Positives	11770	12346	11912
False Positives	5225	1716	1759
True Negatives	342519	346028	345985
False Negatives	1780	1204	1638
Accuracy	98.06%	99.19%	99.05%
Precision	69.25%	87.79%	87.13%
Sensitivity	86.86%	91.11%	87.91%
F1 score	77.06%	89.42%	87.52%
J-statistic	85.36%	90.62%	87.40%

Table 23: LSTM with Layer Normalization Layer and Batch Normalization - Performance evaluation

After comparing the results, we can conclude that using layer normalization did not help us to provide better clasification performance to our model. We would not guide our experiment in this direction any more since our results have not shown expected improvement of performance.

Evaluation metric	LayerNorm	BatchNorm	LSTM
Generic	83.97%	86.55%	83.01%
DoS	87.22%	90.61%	87.35%
Exploits	89.15%	93.94%	91.06%
Fuzzers	84.49%	89.32%	86.46%
Reconnaissance	0%	0%	0%
Shellcode	0%	0%	0%
Backdoor	0%	0%	0%
Analysis	0%	0%	0%
Worms	0%	0%	0%

Table 24: LSTM with Layer Normalization Layer and Batch Normalization - Accuracy of attack category classification

5.2.4 Model 4

The Last model that we are testing is made of GRU cells. We would like to test performance efficiency of GRU cells in comparison with already proven LSTM model. Since GRU has some improvements in cell design, we are expecting better efficiency.

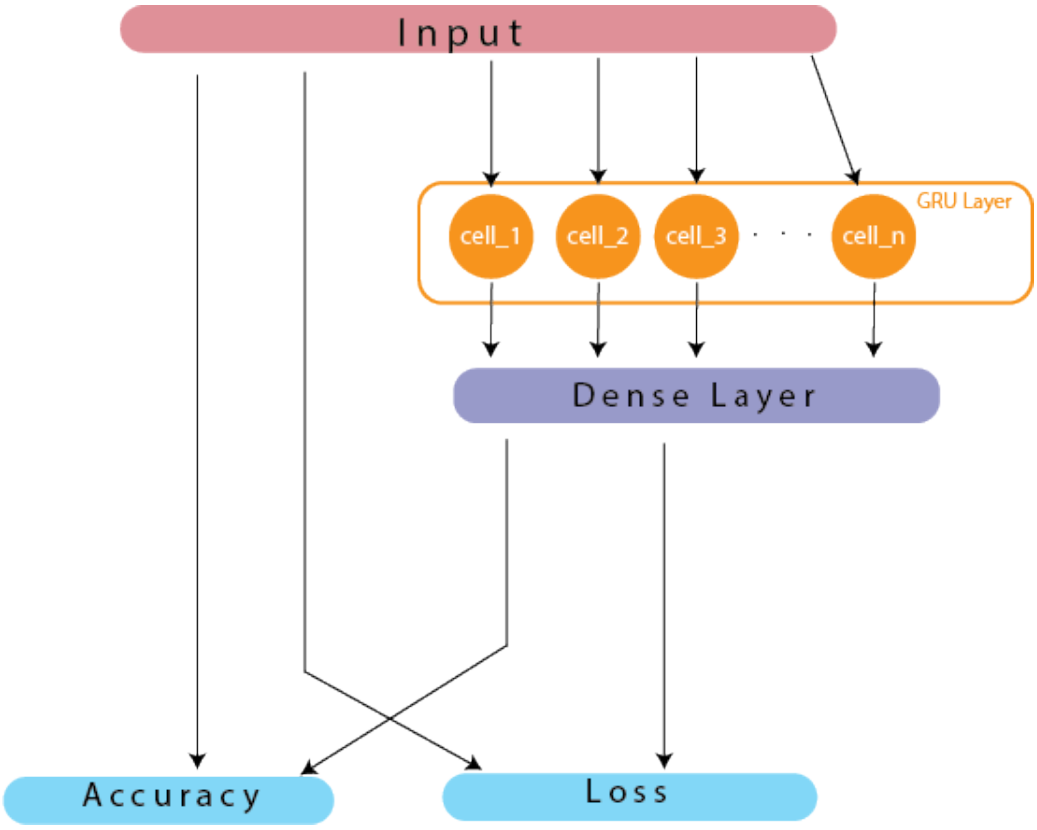

Figure 33: Model 4. GRU cell Layer - consist of 1 layer of GRU cells instead LSTM cells connected to dense layer

GRU vs LSTM

Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
1, 3	128	LSTM, GRU	1	0.001	1024

Table 25: GRU vs. LSTM - Basic parameters

Results are shown in Table 26

Evaluation metric	LSTM	GRU	3xLSTM	3xGRU
True Positives	11912	12036	12637	11494
False Positives	1759	1516	1753	2646
True Negatives	345985	346228	345991	345098
False Negatives	1638	1514	913	2056
Accuracy	99.05%	99.16%	99.26%	98.69%
Precision	87.13%	88.81%	87.81%	81.28%
Sensitivity	99.49%	88.82%	93.26%	84.82%
F1 score	87.52%	88.82%	90.45%	83.01%
J-statistic	87.40%	88.39%	92.75%	84.06%

Table 26: GRU vs. LSTM - Performance evaluation

From the results we can see that GRU is giving better performance then LSTM. On the other hand, increasing the number of layers of GRU cells ends up in overfitting and in the long run, LSTM is a better choice.
The Table: 27, shows the result of attack prediction accuracy.

Evaluation metric	LSTM	GRU	3xLSTM	3xGRU
Generic	83.01%	84.30%	88.49%	82.47%
DoS	87.35%	87.22%	93.22%	81.35%
Exploits	91.06%	92.36%	93.84%	83.64%
Fuzzers	86.46%	87.68%	92.66%	85.20%
Reconnaissance	0%	0%	0%	0%
Shellcode	0%	0%	0%	0%
Backdoor	0%	0%	0%	0%
Analysis	0%	0%	0%	0%
Worms	0%	0%	0%	0%

Table 27: GRU vs. LSTM - Accuracy of attack category classification

5.3 Summary

Based on the stated results in previous subsection, we can conclude that the best performance is given by Model 1, the Basic LSTM model. Our experiments showed that increasing the number of layers to 3 made an improvement in performance, although increasing it more then 3 ended up in overfitting. Learning rate which showed the best results with all the tested models was 0.001. Increasing the number of cells in layer to 1024 by model 1 showed increase in accuracy, j-statistic and precision, but overall the best score was achieved by cell number 128. Experimenting with maximal

batch size ended up in overfitting, so the optimal values would be 1024.

Testing a different regularization techniques, ended up worse then we expected. Layer normalization technique was achieved worse result compare to the batch normalization model and basic LSTM model. Batch normalization showed improvement in all performance metric parameters. Moving towards using more LSTM layers with batch normalization layer, showed worse performance than multi-layer LSTM without batch normalization layer.

Another useful experiment using model 4 with GRU cells, showed grate improvement compared to LSTM cell model. Unfortunately, increasing the number of layers it all the performance metric parameters badly decreased.

The best model with efficient values of parameters:

Model	Layer number	Cell numbers	Type of Cells	Epochs	Learning Rate	max_batch
Model 1	3	128	LSTM	1	0.001	1024

Table 28: The most efficient model according to our experiments

6 Conclusion

During the research, our main goal was to classify the encrypted network traffic using deep learning methods. This was done by using a recurrent neural network with long-short term memory cells and gated recurrent unit cells. The performance evaluation was performed in order to prove that those methods, usually used in language processing, are also suitable for traffic classification.

For that purpose, we used the UNSW-NB15 data set with more than 100GB data and 2 million flows for the training and around 350000 flows for testing. The deep learning platform TensorFlow was used for model development. It was the right choice because it suited the shape and configuration of data.

As stated in Section 1.2, we sought the answer to our main question Q1. During the training phase, we employed 4 different models to find out the answer on this question, which were built with LSTM and GRU cells. We used variable layer numbers, cell numbers, learning rates, max_batch sizes, etc. The models were evaluated by using different performance metric including accuracy, precision, sensitivity, f1 score, as well as j-statictic. Our evaluation showed good results. All models displayed the accuracy of more then 99% and a j-statistic of 88-90%. The final results showed that the results with a higher number of cells in a layer and larger max_batch size improved the performance of the model. The LSTM with 3 layers showed the best results on smaller number of cells and max_batch size. A slight improvement in the performance is noticeable on a single layer GRU, but overall, the multilayer LSTM showed the best score. The LSTM showed a good performance especially for attacks with a large number of packets. Some attacks(Reconnaissance, Shellcode, Backdoor, Analysis, Worms) were not recognized because of low number of appearance in dataset.

The second important challenge was to focus on traffic features. We attempted to discover the features of traffic that are not changing during the encryption. We also tried to train the model with this data features. This model was expected to be able to accurately classify the encrypted traffic and to differentiate between malicious traffic and normal traffic. The first attempt included using the packet length, direction of packet and time difference between the first and following packets in the flow. The main idea behind using the time difference was to utilize the LSTM and to test the depth of memory. This ended up with less than 96% accuracy, which is worse than random guessing, because our data set contains less then 4% of attacks and even with random guessing, we could achieve result which is more then 96%.
Another attempt included using the packet length, direction of packets and packet inter-arrival time (time between the packets). The idea behind this approach came from language processing. Different packets in flow represent words, and a whole flow

represents the sentence. During the training, the model learns that a next packet with specific features (packet length and direction) is coming after certain amount of time (inter-arrival time). This concept was proven as a good approach. After switching to a new feature set, our performance experienced a significant improvement.

The third important question that we needed to clarify in this research regarded the implementation of different regularization techniques in order to improve the performance of the models. The first method we used was batch normalization, which showed a significant improvement in the performance on a single layer. However, when the number of the layers increased, it resulted in overfitting and ended up behind multi-layer LSTM. Layer normalization was a more complex technique that we used for the regularization. It showed a noticeable increase in the time consumed for the training and a decrease in accuracy and the j-statistic.

Therefore, we suggest to use the Basic LSTM model with 3 layers, 128 numbers of cells in layer, maximum batch size with 1024 flows and to train with learning rate 0.001.

6.1 Future Work

After our research question were successfully answered, many new questions appeared. There is a plenty of room for another types of testing which could improve efficiency and performance of our model. This could be done by improving either the data or the model.

A data improvement is the first and maybe the most important topic for future work. The data set that we used was armed with new and up-to-date attacks, but there were many deficiencies while traffic flow classification, there is small number of attacks, attacks are generated in lab environment, etc. The next step could be testing the model with the already known KDD data set, which is more used in this type of experiments. The best way to proceed with the experiment would be to create another data set with live data collected with darkspace or honeypot. These data contains up-to-date network traffic,as well as known and unknown attacks. This kind of data set could be used for supervised or unsupervised learning which could improve intrusion detection systems.

Model improvements would be the next step. Firstly, unsupervised learning could represent another important topic for research, since it represent real intrusion detection system more closely. Secondly, implementing new machine learning models, such autoencoders,as well as sequence-to-sequence models would be another logical step forward. This models showed good results in other fields of research, so it would be interesting to see how they performed on classifying the network data. Thirdly, mov-

ing focus to a specific attack classification or unknown attack classification would be also very useful. This topic should bring a many improvements to the network security industry and to commercial use.

7 References

[1] 1998 DARPA INTRUSION DETECTION EVALUATION DATASET. https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset.

[2] CNTK. https://archive.fo/UWjbl.

[3] CVE Mitre Web site. https://cve.mitre.org.

[4] IXIA Perfect Storm. https://www.ixiacom.com/products/perfectstorm.

[5] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[6] Keras. https://keras.io/.

[7] MXNet. https://mxnet.apache.org/.

[8] protocolnumbers. https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml.

[9] PyTorch. https://pytorch.org/.

[10] tcpdump. https://www.tcpdump.org/manpages/tcpdump.1.html.

[11] Tensorflow. https://www.tensorflow.org/.

[12] tshark. https://www.wireshark.org/docs/man-pages/tshark.html.

[13] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[15] Y. Nir P. Eronen T. Kivinen C. Kaufman, P. Hoffman. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, October 2014.

[16] Zigang Cao, Gang Xiong, Yong Zhao, Zhenzhen Li, and Li Guo. A survey on encrypted traffic classification. In *International Conference on Applications and Techniques in Information Security*, pages 73–81. Springer, 2014.

[17] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[18] CISCO. Encrypted Traffic Analytics. https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf, 2019.

[19] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 1, 2018.

[20] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. Issues and future directions in traffic classification. *IEEE network*, 26(1):35–40, 2012.

[21] Computer Security Department. Glossary of key information security terms. *NIST Internal/Interagency Report NISTIR 7298*, 2013.

[22] Daniel C Ferreira, Félix Iglesias Vázquez, Gernot Vormayr, Maximilian Bachl, and Tanja Zseby. A meta-analysis approach for feature selection in network traffic research. In *Proceedings of the Reproducibility Workshop*, pages 17–20. ACM, 2017.

[23] Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning, 2018.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] Félix Iglesias and Tanja Zseby. Time-activity footprints in ip traffic. *Computer Networks*, 107:64–75, 2016.

[26] Félix Iglesias and Tanja Zseby. Pattern discovery in internet background radiation. *IEEE Transactions on Big Data*, 2017.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[28] B. Claise J. Quittek, T. Zseby and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, oct 2004.

[29] B. Carpenter J. Rajahalme, A. Conta and S. Deering. IPv6 Flow Label Specification. RFC 3697, mar 2004.

[30] Markku Kojo Kumiko Ono Martin Stiemerling Lars Eggert Alexey Melnikov Wes Eddy Alexander Zimmermann Brian Trammell Joe Touch; Eliot Lear, Allison Mankin and Jana Iyengar. Service name and transport protocol port number registry. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, 2009.

[31] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and Sungroh Yoon. Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. *arXiv preprint arXiv:1611.01726*, 2016.

[32] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[33] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[34] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[35] Fares Meghdouri, Tanja Zseby, and Félix Iglesias. Analysis of lightweight feature vectors for attack detection in network traffic. *Applied Sciences*, 8(11):2196, 2018.

[36] Tom Michael Mitchell. *The discipline of machine learning*, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning . . . , 2006.

[37] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM, 2005.

[38] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[39] Nour Moustafa and Jill Slay. 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, 2016.

[40] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[41] Christopher Olah. Understanding lstm networks–colah's blog. *Colah. github. io*, 2015.

[42] S. Krishnan S. Frankel. IP Security (IPsec) and Internet Key Exchange (IKE). RFC 6071, February 2011.

[43] R. Shirey. Internet Security Glossary. RFC 2828, May 2000.

[44] William Stallings and Lawrie Brown. *Computer security: principles and practice*. Pearson Education Upper Saddle River (NJ, 2018.

[45] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep learning approach for network intrusion detection in software defined networking. pages 258–263, 2016.

[46] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6. IEEE, 2009.

[47] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48. IEEE, 2017.

[48] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pages 657–665, 2011.

[49] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16, 2006.

[50] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. " O'Reilly Media, Inc.", 2018.

[51] Denis Zuev and Andrew W Moore. Traffic classification using a statistical approach. In *International workshop on passive and active network measurement*, pages 321–324. Springer, 2005.

A Appendix

Models	Number of Layers	Number of cells in layer	Type of Cells	Epochs	Learning Rate	max_batch
Model 1	1, 3, 5	128, 512, 1024, 1500	LSTM	1	0.001, 0.01	512, 1024, 5000, 10000, 15000
Model 2	1, 3	128	BatchNorm, LSTM	1	0.001, 0.01	1024
Model 3	1, 3	128	LayerNorm, BatchNorm, LSTM	1	0.001	1024
Model 4	1, 3	128	GRU, LSTM	1	0.001	1024

Table 29: Training parameters

Table of Abbreviations

IDS Intrusion Detection System

GPS Global Positioning System

ML Machine Learning

DPI Deep Packet Inspection

DNN Deep Neural Network

KNN k-Nearest Neighbors

NLP Natural Language Processing

RNN Recurrent Neural Network

LSTM Long-Short Term Memory

GRU Gated Recurrent Unit

NB Naive Bayes

SVM Support Vector Machine

DT Decision Tree

KNN K-Nearest Neighbor

KMC K-Means Clustering

CIA Confidentiality Integrity Availability

DoS Denial of Service

IP Internet Protocol

TCP Transport Control Protocol

UDP User Datagram Protocol

IANA Internet Assigned Numbers Authority

DoS Denial of Service

GT Ground Truth

QoS Quality of Service

CoS Class of Service

SDN Software Define Network

SCADA System Control and Data Acquisition

TU Wien Vienna University of Technology

ACCS Australian Centre for Cyber Security

List of Figures

List of Tables