UNIVERSIDADE FEDERAL FLUMINENSE

EDUARDO DE OLIVEIRA ANDRADE

Malware Classification Using Word Embeddings Algorithms and Long Short-Term Memory Networks

NITERÓI

2019

UNIVERSIDADE FEDERAL FLUMINENSE

EDUARDO DE OLIVEIRA ANDRADE

Malware Classification Using Word Embeddings Algorithms and Long Short-Term Memory Networks

Dissertation presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Field: Systems and Information Engineering.

Advisor:
JOSÉ VITERBO FILHO

Co-advisor:
CRISTINA NADER VASCONCELOS

NITERÓI

2019

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

A553m Andrade, Eduardo de Oliveira
Malware Classification Using Word Embeddings Algorithms and Long Short-Term Memory Networks / Eduardo de Oliveira Andrade ; José Viterbo Filho, orientador ; Cristina Nader Vasconcelos, coorientadora. Niterói, 2019.
77 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2019.

DOI: http://dx.doi.org/10.22409/PGC.2019.m.13457297746

1. Aprendizado de máquina. 2. Segurança da Informação. 3. Produção intelectual. I. Viterbo Filho, José, orientador. II. Vasconcelos, Cristina Nader, coorientadora. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

EDUARDO DE OLIVEIRA ANDRADE

Malware Classification Using Word Embeddings Algorithms and Long Short-Term Memory Networks

Dissertation presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Field: Systems and Information Engineering.

Approved in February 2019.

APPROVED BY

D.Sc. José Viterbo Filho, UFF

D.Sc. Cristina Nader Vasconcelos, UFF

D.Sc. Flávia Bernardini, UFF

D.Sc. Luis Kowada, UFF

Dr.-Ing. Fábio Borges, LNCC

Niterói

2019

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

Alan Turing

Acknowledgements

I thank my family, friends, fiancée and advisors for the support and contribution that gave me the necessary strength for the conclusion of this thesis.

I thank all those who have made available essential content for the work even without knowing them.

Thank you so much for everything.

Resumo

O número de *softwares* maliciosos, mais conhecidos como *malwares*, aumenta a cada ano e seu desenvolvimento torna-se mais sofisticado à medida que novas técnicas são utilizadas para contornar a verificação de programas, como os antivírus. Com isto, a área de aprendizado de máquina surge como um novo meio para a identificação destas ameaças. Muito tempo e dinheiro já estão sendo investidos por governos e empresas na coleta de dados para algoritmos de aprendizagem na procura por novas soluções que envolvam tecnologias capazes de lidar com uma vasta quantidade de dados. O aprendizado profundo consiste em uma abordagem de aprendizado de máquina que possui um alto nível de abstração, sendo capaz de obter bons resultados com grandes conjuntos de dados. As arquiteturas mais conhecidas de aprendizado profundo são as redes neurais e uma das redes mais utilizadas é a rede neural recorrente (RNN). Neste trabalho, são propostos alguns modelos de redes neurais baseados em um tipo de RNN, a rede *long short-term memory* (LSTM) para os cenários de classificação binária e multiclasse, analisando dados não estruturados de *malware* em um novo conjunto de dados balanceado criado especificamente para este trabalho e disponibilizado publicamente. Três modelos de LSTM são propostos e dois deles foram desenvolvidos em conjunto com outros dois algoritmos não supervisionados de vetorização de palavras, *global vectors* (GloVe) e word2vec. Por fim, estes modelos são comparados e a acurácia da LSTM com o word2vec apresenta os melhores resultados: 88.94% para a classificação binária e 75.13% para a classificação multiclasse, incluindo seis classes com cinco diferentes tipos de malware.

Palavras-chave: Classificação de Malware; Aprendizado Profundo; Redes Neurais.

Abstract

The number of malicious softwares, more commonly known as malwares, increases every year and its development becomes more sophisticated as new techniques are used to bypass scanning of programs, such as antivirus. Thereby, the machine learning area emerges as a new way for the identification of these threats. Much time and money are already being invested by governments and companies in data collection to help learning algorithms in the search for new solutions that involve technologies capable of handling a vast amount of data. Deep learning consists of a machine learning approach that has a high level of abstraction and can achieve good results with large datasets. The most well-known architectures of deep learning are neural networks and one of the most used networks is the recurrent neural network (RNN). In this work, some models of neural networks based on a type of RNN, the long short-term memory network (LSTM), are proposed for the binary and multiclass classification scenarios, analyzing unstructured malware data in a new balanced dataset created specifically for this work and made publicly available. Three LSTM models are proposed and two of them have been developed in conjunction with two other unsupervised word embeddings algorithms, global vectors (GloVe) and word2vec. Finally, these models are compared and the accuracy of LSTM with word2vec shows the best results: 88.94% for binary classification and 75.13% for multiclass classification, including six classes with five different types of malware.

Keywords: Malware Classification; Deep Learning; Neural Networks.

List of Figures

List of Tables

List of Acronyms

ANN	— Artificial Neural Network
BCE	— Binary Cross-Entropy
CBOW	— Continuous Bag-of-Words
CCE	— Categorical Cross-Entropy
CNN	— Convolutional Neural Network
CPU	— Central Processing Unit
DLL	— Dynamic-Link Libray
ESN	— Echo State Network
FNR	— False Negative Rate
FPR	— False Positive Rate
GloVe	— Global Vectors
GPU	— Graphics Processing Unit
GRU	— Gated Recurrent Unit
LSTM	— Long Short-Term Memory
ME	— Mutation Engine
NLP	— Natural Language Processing
ODL	— Online Deep Learning
PE	— Portable Executable
PReLU	— Parametric Rectified Linear Unit
ReLU	— Rectified Linear Unit
RNN	— Recurrent Neural Network
SGRAM	— Skip-Gram
SVM	— Support Vector Machine
TFIDF	— Term Frequency-Inverse Document Frequency
TPR	— True Positive Rate

Contents

Chapter 1

Introduction

The number of malicious software, more commonly known as malwares, increases every year [34, 54]. In one day, 230.000 new malware samples are produced[1] and the continuous growth of information security spending underscores its importance [35]. One of the most damaging causes for people and businesses is trojan ransomware, a type of malware that can encrypt the victim's data so that they can no longer be accessed or even block access to a particular system. In this way, the victims usually need to pay a certain amount of money for the attacker to unblock their data or to re-release the system access. This is just one example of malware whose estimated collection for criminals is around 1.4 billion in the year 2018 [37].

Complementing the issue of information security, the number of data generated has increased dramatically in recent years, e.g., the number of emails sent and received that reached 196 billion in 2014 [56]. There is a trojan called TibsPK[2] that is responsible for most of the malicious emails that circulate on the network and after its execution, installs itself in the Windows system folder and opens a door to other malware.

There are numerous examples of threats that can infect multiple systems. There is a lot of talk today about the term big data, because we are in an era characterized by the great volume of data, whether they are in transition, as in the case of exchanging emails or stored, as in a possible source of infection by a ransomware. In any case, security is still a priority issue and the damage that can be caused by malware is a major source of concern [14].

Through all this amount of data and malware that comes up every time [53], the analysis of this data is a complex process and multiple studies are needed to characterize

[1] `https://goo.gl/sMGMQg`
[2] `https://goo.gl/5xJrsY/`

and differentiate a cleanware file (short for clean software) from a malware file. This is known as a binary malware classification, and when malware is divided into classes determined by its subtypes (trojan, virus, worm, etc.) or families (confiker, cryptowall, nivdort, etc), the task is a multiclass malware classification, disregarding that malware may have more than one subtype (ransomware trojan, for example).

The machine learning area has been very promising in terms of malware classification [16] with better results over time, using computational techniques based on mathematical models that aim to carry out a learning process through the available data, and thus performing a classification task according to some characteristics of this data. Learning essentially has three approaches: supervised, semi-supervised and unsupervised. The supervised and semi-supervised approaches deal with a labeled dataset, but in the case of semi-supervised learning, generally only part of the dataset is labeled. In the unsupervised case it is different, the data involved are not labeled and it is sought to group them into distinct sets according to characteristics that are similar.

When dealing with a large amount of data, the deep learning area, which is a machine learning subarea, is superior in achieving better results than traditional approaches [21], using artificial neural networks (ANNs) that can be based on any of the three forms of learning. About malware classification, ANNs are also used and show good results [58, 20, 36].

The unsupervised approach is used by word embeddings algorithms that return words represented as mathematical objects, usually vectors. Each dimension value of these vectors considered corresponds to a characteristic that can have a semantic meaning or even a grammatical interpretation [60]. In the case of cleanwares or malwares, you can consider the machine instruction codes, for example, as the words that make up software "language" [46].

1.1 Problem Definition

The main use of artificial neural networks was initially in relation to image classification. Some works even use convolutional neural networks (CNNs) to classify malware, usually performing the conversion of some characteristics to images [18, 63]. However, the use of these neural networks has been expanded to several areas and to achieve this purpose, one must deal with different details present in each domain.

One of these domains is in language modeling [59], where the application of deep

learning and the recurrent neural networks (RNNs) has been widely used. The states of these networks aid in the understanding of sentences, in which so-called "cells", in the case of long short-term memory neural networks (LSTMs) [26], a specific type of RNN, process a word a time and return the probabilities of possible values that the next word may have in one sentence. The "deep" aspect of neural networks represents the transformation process that such data undergoes, which becomes more abstract as they reach higher levels in theses networks. In this way, some details of the data end up being more relevant, while other details become less important and are disregarded.

In this work, we try to explore the understanding of malwares by reading their hexadecimal codes as sentences. This approach was proposed by [32], while malware samples were obtained from VirusShare[3], a repository of malware collected from various Windows systems. The native hexdump tool of Unix systems was responsible for generating the text files containing the hexadecimal representations that were used to serve as input to the LSTM models developed in this master's thesis.

In addition, each type of malware has its peculiarities [11], even in the case of "language", since some machine instructions, for example, are common in certain types than others [10]. Therefore, the approach of this work also tries to deal with the multiclass scenario, whose classification is important to differentiate the types of malware that present a higher level of threat and how users protect themselves in the best way[4].

1.2 Objective

The main objective of this thesis was to obtain good classification results with the MC-dataset (abbreviation for malware and cleanware), which was created specifically for this work and can be divided into two parts: the MC-dataset-binary[5] for the binary case (classification task between two distinct classes) and the MC-dataset-multiclass[6] for the multiclass case (classification task between several distinct classes). Apart from this, there is a hypothesis that was considered in this research:

"Does the use of word embeddings algorithms with neural networks improve the results of malware classification?"

To answer this question, the LSTM models proposed in this work were executed

[3] `https://virusshare.com/`
[4] `https://goo.gl/ShUAoZ/`
[5] `https://goo.gl/8AhXuy/`
[6] `https://goo.gl/YkiY7h/`

with and without the main unsupervised word embeddings algorithms: word2vec [40] and GloVe [45], an abbreviation for global vectors. In addition, the MC-dataset has been made publicly available so that other users can use it. We did not find another dataset with the same number of samples and malwares types included into MC-dataset-multiclass like this work. This was another goal, because there was no dataset with a good size for deep learning, including sufficient cleanwares samples, and a contribution, as well as the development of the neural network models for the classification.

1.3 Methodology

A classification based only on GloVe and word2vec was carried out without the use of the neural networks. The first model named only "LSTM" was the only one executed on a "home machine" because it required a smaller amount of RAM than in the other models. This model did not use word embeddings algorithms as a preprocessing.

The remaining two models were run on DGX[7], a supercomputer provided by NVIDIA for research at our university. This was required by the amount of RAM required by word2vec and GloVe. The second and third models, named "word2vec+LSTM" and "GloVe+LSTM", respectively, present a similar architecture as the "LSTM", but using word embeddings algorithms as preprocessing for the neural networks. Thus, through the comparison of the output values obtained from the neural network models, we tried to corroborate the classification results.

1.4 Organization

The structure of this work is divided into 8 chapters. In Chapter 2, an overview about the malwares, describing its main aspects, types, and most current and commonly used countermeasures against these threats. After this, Chapter 3 presents a review of the literature about deep learning, including basics concepts of ANNs, LSTMs and the word embeddings algorithms, word2vec and GloVe.

In Chapter 4, we have the related work of the malware classification area in the binary and multiclass scenarios. After this, Chapter 5 presents the methodology adopted in the creation of the MC-dataset, which is separated into MC-dataset-binary, relative to binary classification and MC-dataset-multiclass, relative to the multiclass classification.

[7] `https://www.nvidia.com.br/object/prbr_080311.html/`

In Chapter 6, the implementation of the three proposed LSTM models in this master's thesis, LSTM, word2vec+LSTM and GloVe+LSTM. The results of the experiments with the binary and multiclass scenarios, discussing the classification values returned in the tests are in Chapter 7. Lastly, Chapter 8 presents the conclusions of this work, describing some limitations of the research and future work.

Chapter 2

Malicious Software

This chapter begins by describing malicious software since the emergence of the term malware to the techniques and strategies developed to evade the current scanners. In addition, the various types of malware are presented with details that differentiate them. Finally, countermeasures with the advantages and disadvantages, including the static and dynamic analysis used for the detection and classification of malware, are explained in the last section.

2.1 Overview

The term malware, a combination of the words malicious and software, was first mentioned by professor Yisrael Radain in 1990 [23]. The main function of malware is to cause harm to a computer, mobile device, data or even people. This can occur in a variety of ways, such as deleting, stealing, or changing data, monitoring users without permission, etc.

Since the emergence of the first virus in 1982 called the Elk Cloner [55], millions of other types of malware have been created, with about 12 million of malwares being reported per month today only for Windows[1]. The principle of virus replication, like the Elk Cloner, remains the same until today but now it propagates through various other systems and services. However, practically all types of malware are now being developed with much more complex techniques and strategies that seek to evade detection tools [7], like obfuscation, encryption, oligomorphism, polymorphism and metamorphism.

- **Obfuscation:** A set of techniques used to evade signature-based detection methods, such as antivirus. Usually involves the implementation of unnecessary instructions

[1] `https://goo.gl/KBY2yP/`

or even changing the order in which they are executed but maintaining the behavior. In Figure 2.1, there is a simple dead-code insertion, in which there is the addition of inefficient code only to make it difficult to be detected by a signature scan [68].

00401005	8BF0	MOV ESI, EAX	00401005	8BF0	MOV ESI, EAX
00401007	3E:8A00	MOV AL, BYTE PTR DS:[EAX]	00401007	3E:8A00	MOV AL, BYTE PTR DS:[EAX]
0040100A	84C0	TEST AL, AL	0040100A	84C0	TEST AL, AL
0040100C	74 46	JE SHORT Test.00401054	0040100C	74 46	JE SHORT Test.00401054
0040100E	53	PUSH EBX	0040100E	53	PUSH EBX
0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]	0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]
00401016	D3DB	RCR EBX, CL	00401016	D3DB	RCR EBX, CL
00401018	0FCB	BSWAP EBX	00401018	0FCB	BSWAP EBX
0040101A	68 56104000	PUSH Test.00401056	0040101A	68 56104000	PUSH Test.00401056
0040101F	5B	POP EBX	0040101F	5B	POP EBX
00401020	3E:8903	MOV DWORD PTR DS:[EBX], EAX	00401020	3E:8903	MOV DWORD PTR DS:[EBX], EAX
00401023	43	INC EBX	00401023	43	INC EBX
00401024	0FBDC2	BSR EAX, EDX	00401024	0FBDC2	BSR EAX, EDX
00401027	A9 46A978DC	TEST EAX, DC78A946	00401027	A9 46A978DC	TEST EAX, DC78A946
0040102C	8BC2	MOV EAX, EDX	0040102C	8BC2	MOV EAX, EDX
0040102E	52	PUSH EDX	**0040102E**	**90**	**NOP**
0040102F	B6 86	MOV DH, 86	**0040102F**	**90**	**NOP**
00401031	B3 27	MOV BL,27	**00401030**	**42**	**INC EDX**
00401033	B8 7CFAA17E	MOV EAX, 7FA1FA7C	**00401031**	**52**	**PUSH EDX**
00401038	EB 01	JMP SHORT Test.0040103B	**00401032**	**FE0C24**	**DEC BYTE PTR SS:[ESP]**
0040103A	90	NOP	**00401035**	**90**	**DEC EDX**
0040103B	0FBCC2	BSF EAX, EDX	00401036	0FBCC2	MOV DH, 86
0040103E	3E:C705 FC8841	MOV DWORD PTR DS:[4188FC], 0	00401038	3E:C705 FC8841	MOV BL, 27
00401049	2D 210DE8B9	SUB EAX, B9E80D21	0040103A	2D 210DE8B9	MOV EAX, 7FA1FA7C
0040104E	69DA E577D49D	IMUL EBX, EDX, 9DD477E5	0040103F	69DA E577D49D	JMP SHORT Test.00401042

Figure 2.1: An example of a dead-code insertion.

- **Encryption:** Generally, malware encryption involves two sections, one with the body of the code and another with the decryption loop. Again, there is an attempt to escape signature-based detection methods because the encrypted code snippet is only decrypted when malware is executed, before that the code has no meaning, making it difficult to be detected by a scanner. In Figure 2.2 a code snippet is displayed before and after decryption [47].

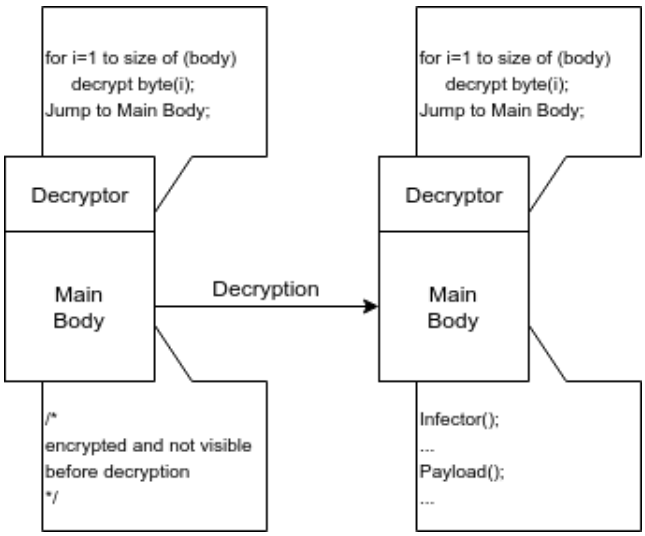


Figure 2.2: Encrypted code snippet of a malware.

- **Oligomorphism:** It also makes use of encryption as a defense mechanism, but in this case, the oligomorphic malware is able to change its encryption routine to a limited number of times. If there is more than one decryption routine, malware can be considered as oligomorphic. As we can see in Figure 2.3, malware presents itself differently in each new infection.

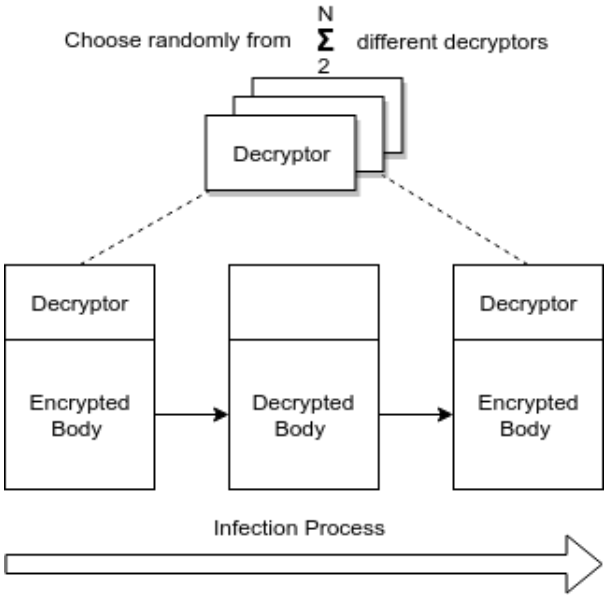


Figure 2.3: Process of infection by an oligomorphic malware.

- **Polymorphism:** It follows the same principle of oligomorphic malware but can generate infinite decryptors using various obfuscation techniques. The mutation engine (ME) tool is responsible for introducing obfuscated snippets of code and generation of different decryptors, as represented in Figure 2.4. Generally. the

default function of polymorphic malwares is not changed every time a decryption occurs, remaining basically the same.

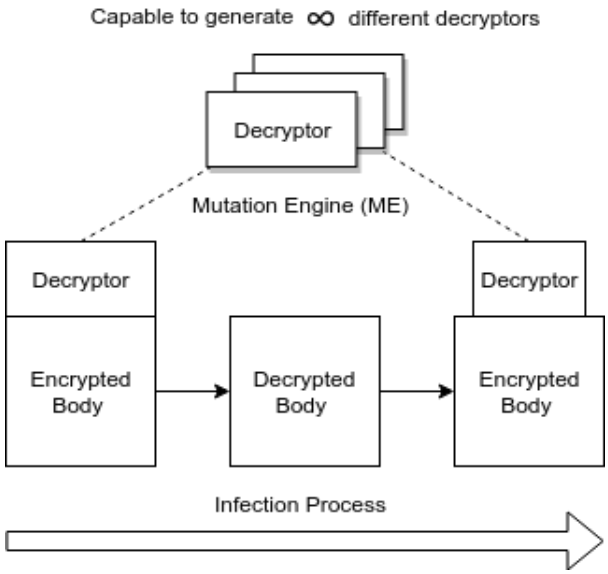


Figure 2.4: The mutation engine and the process of infection by polymorphic malware.

- **Metamorphism:** Metamorphic malwares can change their internal structure completely every time an infection process occurs. Unlike polymorphic malware, a decriptor is not required in this strategy. In Figure 2.5, we observe the generation through ME with malware containing completely different code segments from one to another. Despite this, the malware behavior remains the same for each created M [62].

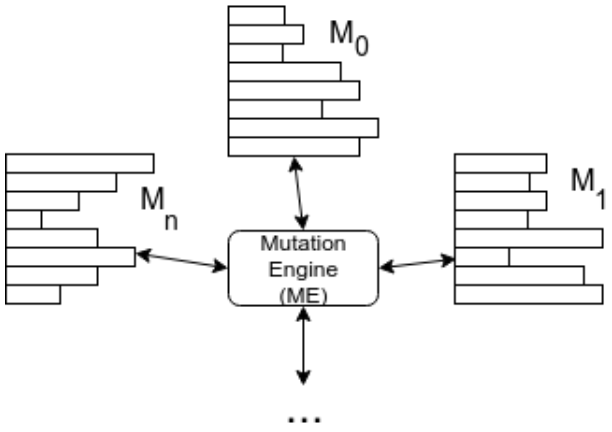


Figure 2.5: The creation of different code segments by the mutation engine in a metamorphic malware.

2.2 Types

Malware can be divided into several types that are not mutually exclusive depending on the function for which a particular malware was developed. Five types of malware were included in the dataset used in this work but there are several other types of malware.

- **Backdoor:** This type of malware was developed to bypass authentication procedures, being able to compromise multiple systems. After the bypass, more backdoors can be installed, allowing future access to the attacker without detection by the user.
- **Rootkit:** The implementation of this malware is intended to make it "invisible" to users and security softwares. It tries to take control of a computer or gain remote access in this way.
- **Trojan:** The trojan horse or just trojan is probably the most common type of malware. This malware through a "camouflage" tries to pass as a conventional cleanware or some program that the user is convinced to download. Unauthorized access is granted to the computer on which the trojan is installed and can be used to monitor user activities, steal data, modify files, make the computer part of a botnet, and so on.
- **Virus:** The virus is able to replicate itself several times. It can spread across multiple connected computers on the same network, for example, and be activated by running software that the user does not know that is infected.
- **Worm:** One of the most dangerous types of malware, the worm is able to spread rapidly across a computer network exploiting operating system vulnerabilities. The worm is also able to replicate itself, but does not need to a program running by the user as in the case of the virus for this to occur.

2.3 Countermeasures

Basically, there are two types of malware analysis: dynamic (also called behavioral analysis) and static (also called code analysis). The hybrid analysis is simply the use of parts of each of the two analyzes. Generally, dynamic analysis consists of monitoring the behavior of malware in a controlled environment, such as a sandbox, preventing a system infection and observing the flow of information, instructions and function calls while the malware is running [61].

The problems of dynamic analysis are the time required for monitoring, the analysis environment might not be completely secure, and the difficulty in detecting multipath malware [41]. In contrast, dynamic analysis is the best solution for metamorphic malware. In this work, only the static analysis that seeks to detect the malwares without executing them was explored. Static analysis can be divided into techniques based on signature detection or heuristics.

- **Signature-based detection:** The most known technique of static analysis since it is a commonly used approach to antivirus software. In this case, the signatures may be pattern strings that are injected as malicious code that characterizes the malware, and being added to the antivirus databases in the updates. This procedure is less costly than other approaches, but frequent updates are necessary for the new malwares that emerge every day.
- **Heuristic-based detection:** Unlike signature-based detection, this technique looks for instructions or commands that are not present in conventional software, making detection better for new malwares. For example, you can search in a program for commands that are considered malicious, such as deleting some type of essential file and then classify this program as a malware. It is also possible to assign varying weights to certain commands that are considered malicious to classify software as malicious or not.

About obfuscation and other malware strategies that do not include metamorphism to evade detection tools, static analysis, depending on the way it is performed. is able to detect these threats (including the implemented models of this work). There are common code snippets even in the case of polymorphism that can be detected, which does not occur in metamorphic malwares, since the code is always different, using the most varied obfuscation techniques.

Considering the identification of malware, whether by dynamic or static analysis, it is important to verify not only the accuracy and the true positives, but also the false negatives and false positives. Classifying a malware as a cleanware is apparently the most dangerous case but also classifying a cleanware as malware can be quite damaging in financial terms for businesses and governments.

Chapter 3

Deep Learning

A review of the literature about deep learning is presented in this chapter since the development of the first artificial neural networks. The functions of activation, loss, gradient descent optimization algorithms, backpropagation, and dropout are described, which are usually present in most neural networks. We also present the architectures and characteristics of the long-term memory networks and then the unsupervised word embeddings algorithms, word2vec and GloVe, which were important for the models implemented in this work.

3.1 Artificial Neural Networks

The first neural network model created with electrical circuits and considered the beginning of the development of ANNs occurred in 1943 [38]. The main idea behind the elaboration of ANNs is the creation of models that simulate biological nerve systems, such as the brain, for instance, and thus be able to learn from the data [18].

The simplest architecture of a neural network is the feedforward neural network [8], as we can visualize in Figure 3.1 and which is best explained in [3], another work published by the same authors of this master's thesis. In its smallest form, this architecture is represented by an input layer, a hidden layer and an output layer, and has no cycles.

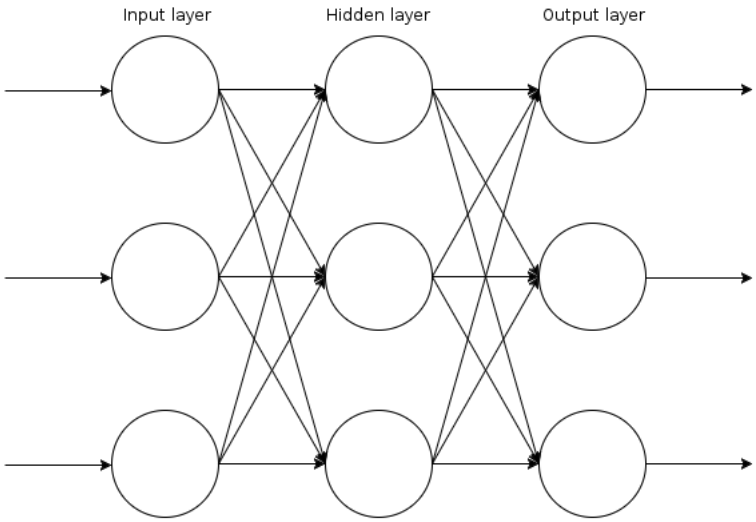


Figure 3.1: Representation of a feedforward neural network with three layers

Given an input vector $\boldsymbol{i}$ (represented by the values assigned to the neurons of the input layer), the weight matrix $\boldsymbol{G_{ih}}$ (represented as a set of transformations occurring in the neurons of the input layer destined to the neurons of the hidden layer) and the activation function f_h (responsible for the activation of the neurons of the hidden layer), we have the Equation 3.1 representing the vector of the hidden layer $\boldsymbol{h}$.

$$\boldsymbol{h} = f_h(\boldsymbol{iG_{ih}}) \tag{3.1}$$

Following the previous reasoning, given an input vector $\boldsymbol{h}$ (represented by the values assigned to the neurons of the hidden layer), the weight matrix $\boldsymbol{G_{ho}}$ (represented as a set of transformations occurring in neurons of the hidden layer destined to the neurons of the output layer) and the activation function f_o (responsible for the activation of the neurons of the output layer), we have the Equation 3.2 representing the vector of the output layer $\boldsymbol{o}$.

$$\boldsymbol{o} = f_o(\boldsymbol{hG_{ho}}) \tag{3.2}$$

Years later, the first neural network consisting of a supervised learning algorithm, the perceptron [49], was developed. Its architecture can be seen in Figure 3.2. The perceptron works as a basic ANN, being able to receive several input values $(i_1, i_2, ..., i_k)$ and producing output o through the product with the weights $(g_1, g_2, ..., g_k)$. This produced output is unique, being represented by the Equation 3.3, in which the bias b is added to allow the classifier to change its decision boundary to either right or left, for example.

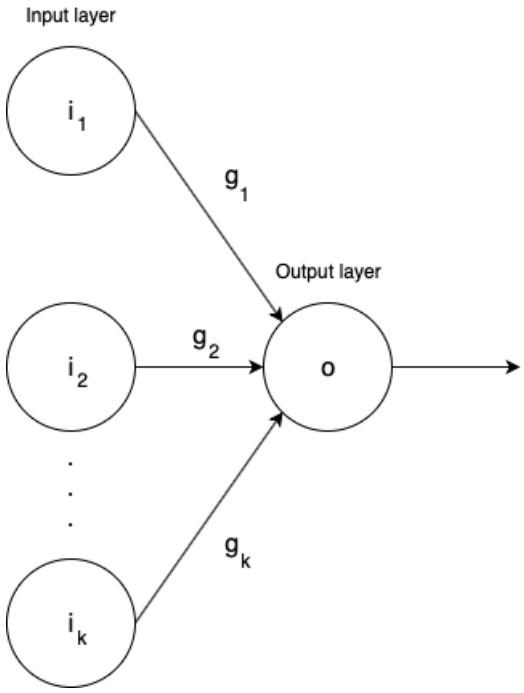


Figure 3.2: Architecture of the perceptron neural network.

$$f(i_1, i_2, ..., i_k) = \begin{cases} 1, & \text{if } \sum i_k g_k + b \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

The perceptron problem is its classification capacity, being limited to outputs 0 or 1, making it impossible to work with problems involving more classes. A classic example is the problem of XOR [67] that was solved with the multilayer perceptron, which gave rise to the current neural networks, which mostly have a much more complex architecture. This increased complexity of neural networks is one of the characteristics of deep learning, also characterized by the implementation of multiple hidden layers, providing a higher level of abstraction and making deep neural networks a more specific approach to machine learning and its conventional algorithms.

3.1.1 Activation Functions

The insertion of the activation functions in the neural networks had as purpose to make the networks more powerful to deal with a higher level of abstraction, being able to work with more complex problems and data. This occurs mainly with the use of non-linear functions, which can return real values, for instance, which does not occur in the case of the perceptron because changes in weights and bias, even though small changes in values,

may change completely the network that can assume only an output of value 0 or 1. The most known non-linear activation functions are the sigmoid, hyperbolic tangent and rectified linear units (ReLU).

- **Linear:** It is the activation function used in the perceptron and can be seen in Figure 3.3 and Equation 3.4. In relation to deep learning, linear function is the worst option for dealing with most complex problems with many parameters. It has an output in the $(-\infty, \infty)$ range, and a major drawback is that every time a backpropagation is performed, the gradient ends up continuing the same and the error is not improved. In addition, the "deep" characteristic of the neural network is lost because no matter how many layers with linear transformations the network has, the final output remains a linear transformation of the input, all layers could be reduced to a single layer with neurons whose activation function is linear.

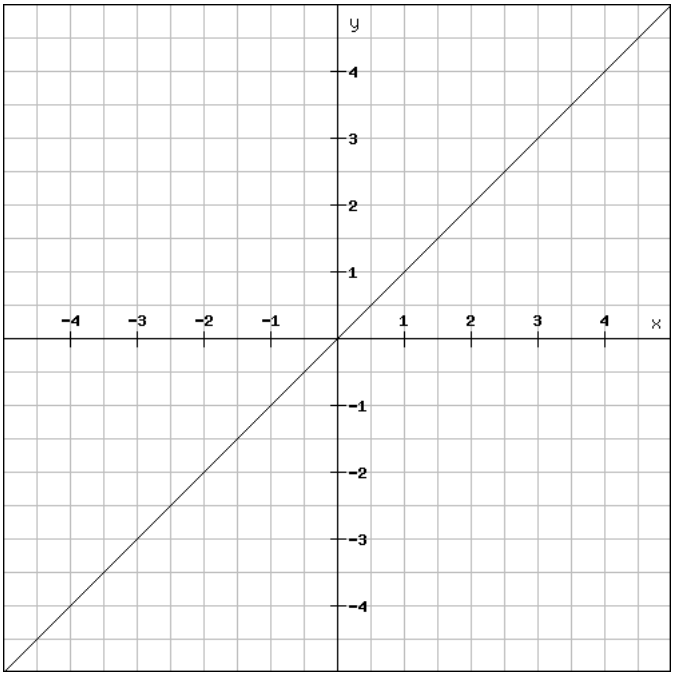


Figure 3.3: Linear function graphic.

$$f(x) = x \tag{3.4}$$

- **Sigmoid:** The sigmoid or logistic function can be seen in Figure 3.4 and Equation 3.5. As advantages, the sigmoid is able to always return values between the $(0, 1)$ range, unlike the linear function, for example, which may have an output in the interval $(-\infty, \infty)$. Therefore, it is able to work with non-binary activations and its gradient is smooth, with no abrupt changes as in the perceptron.

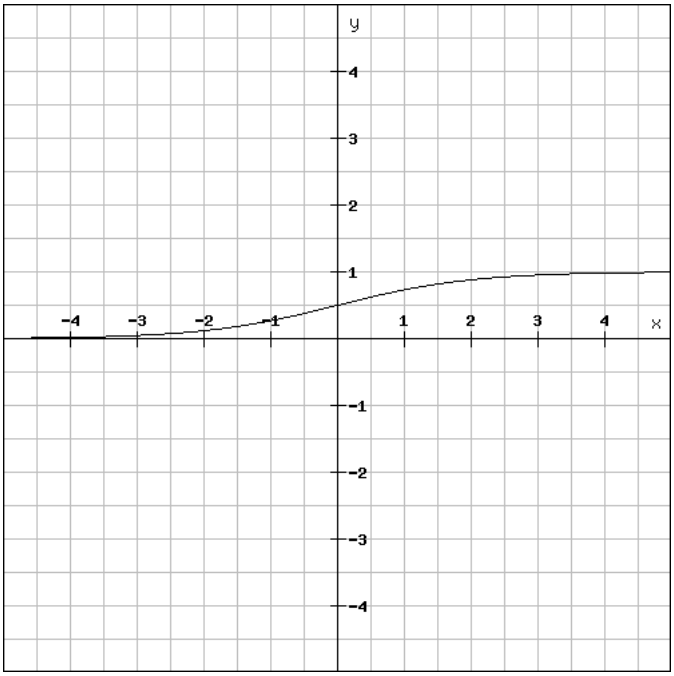


Figure 3.4: Sigmoid function graphic.

$$S(x) = \frac{e^x}{e^x + 1} \tag{3.5}$$

- **Hyperbolic tangent:** The hyperbolic tangent or tanh has a sigmoid-like trend and can be seen in Figure 3.5 and Equation 3.6. A tanh can have outputs between the $(-1, -1)$ range and so you can choose the tanh instead of the sigmoid. In fact, tanh is no longer a sigmoid by its S-shape, but the logistic function is more commonly called the sigmoid. Other reasons for the use of tanh rather than sigmoid are the strongest gradients from tanh whose derivatives are larger and bias in the gradients is also avoided [33].

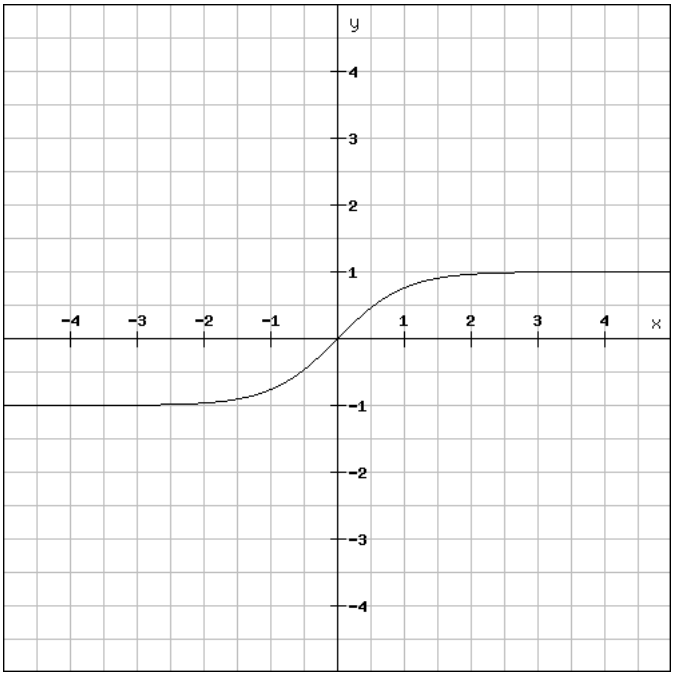


Figure 3.5: Hyperbolic tangent function graphic.

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.6}$$

- **ReLU:** In Figure 3.6 and Equation 3.7, we can observe the function ReLU that simply returns a value x, case x is positive and 0 otherwise. The disadvantage of using ReLU for some problems is that all negative values become zero, since $(0, \infty)$ is the range covered by the function, decreasing the learning ability of the network. There is another version of the ReLU function called Leaky ReLU, which instead of the zero gradient, uses a very small gradient, allowing the network to continue learning [66]. In any case, ReLU is the most used activation function in deep learning, demanding a computational cost less than the sigmoid and tanh and trying to solve the vanishing gradient problem [25] by obtaining more sparse representations [19].

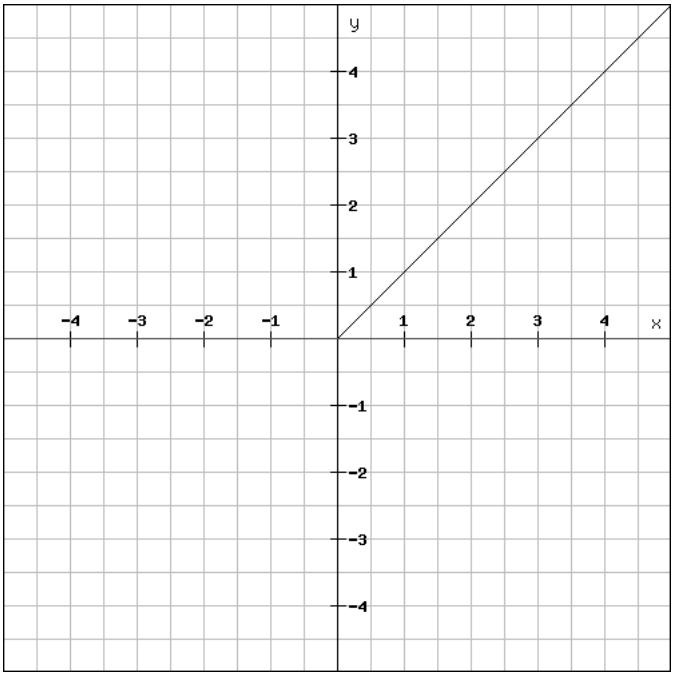


Figure 3.6: ReLU function graphic.

$$R(x) = max(0, x) \tag{3.7}$$

3.1.2 Loss Functions

The loss function, objective function or cost function, is intended to minimize the calculated error. In Equation 3.8, we can visualize one of the most commonly used loss functions, in which MSE represents the mean squared error. MSE is the sum of squared distances, considering our target variable and predicted values.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n} \tag{3.8}$$

In the case of neural networks, backpropagation is responsible for this error calculation, in which the error usually traverses a path back through the previous layers of the network. As this backpropagation occurs, the weights and biases are updated, trying to minimize the error. There are several loss functions that can be used in neural networks. Choosing this function for your neural network depends on the problem you want to solve. However, some are more used and known, such as the classification loss functions used in this work, binary cross-entropy (BCE) and categorical cross-entropy (CCE).

- **Binary cross-entropy:** Before the BCE receives its input, there must be a pre-

ceding sigmoid function that gives its output values to the BCE. This choice of activation function occurs because the sigmoid can return outputs of value either 0 or 1 and as the name itself says, this cross entropy aims to perform a classification of the binary type. The Equation 3.9 is relative to the loss function BCE for N training examples.

$$BCE = -\frac{1}{N}\sum_{j=0}^{N} o_j log(\hat{o}_j) + (1 - o_j) log(1 - \hat{o}_j) \tag{3.9}$$

- **Categorical cross-entropy**: The CCE is similar to BCE but aims to perform a classification that involves several classes and not just two classes. For this to occur, the Equation 3.10 differs as it traverses all output nodes through the sum that considers M and $t_{j,m}$ is 1 if sample j is in class m and 0 otherwise. The predicted probability $p_{j,m}$ reffers to the change of sample j be in the class m. Preceding the CCE, we have the softmax activation function[1] that can be seen in the Equation 3.11 and is responsible for the input values in the CCE.

$$CCE = -\frac{1}{N}\sum_{j=1}^{N}\sum_{m=1}^{M} t_{j,m} log(p_{j,m}) \tag{3.10}$$

$$\sigma(x_m) = \frac{e^{x_m}}{\sum_{j=1}^{J} e^{x_m}} \tag{3.11}$$

3.1.3 Gradient Descent Optimization Algorithms

The gradient descent optimization algorithms are intended to assist in the search for a path that minimizes the objective function by updating network parameters in the opposite direction of the calculated gradient. There are various forms of gradient descent whose characteristic that differs is the amount of data used to compute the objective function gradient. The most commonly used form is the mini-batch gradient descent that performs an update for each mini-batch of any number of training examples, as described in Equation 3.12, where η is the learning rate that determines the number of steps to arrive at a local minimum, θ are the parameters (such as weights, for example) with $x^{(j)}$ and $y^{(j)}$ being each training example and label, respectively. Complementing the Equation 3.12, ∇_θ represents the gradient associated with the parameters and J the Jacobian matrix.

[1] `https://goo.gl/5Sfq3V`

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; x^{(j)}; y^{(j)}) \tag{3.12}$$

In terms of optimization algorithms, the most used are those of adaptive learning that, unlike those with a constant learning rate, such as those based on the standard stochastic gradient descent, try to update the learning rate per parameter, higher depending on the behavior of previous gradients. The adam [31] is one of the examples of adaptive learning gradient descent optimization algorithm and can be seen in the Equation 3.13, where t represents a step in time, w_t is the first moment, v_t the second moment and generally $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon} \frac{w_t}{1-\beta_1^t} \tag{3.13}$$

3.1.4 Dropout

A serious problem that can occur in neural networks is when some models learn a lot about training data, including details and noises. This can negatively impact learning on new data and accuracy in terms of classification. The name of this modeling error is overfitting. Some of the ways to mitigate the overfitting problem include penalties for loss functions and weights, such as stopping training as the results of the validation set begin to worsen [43]. For example, in logistic regression there are the known penalties L1 (Laplacian) and L2 (Gaussian).

Another technique that tries to solve the overfitting problem is called dropout. The main idea of the dropout is to randomly drop some of the network's neurons along with their connections, preventing these neurons from "learning too much" [57]. Generally, a probability of $p = 0.5$ is used for the drop of the neurons because this value is closer to the optimal one that covers a larger number of networks and tasks that can be executed. In the Figure 3.7, we can observe a simple example of execution of the dropout technique.

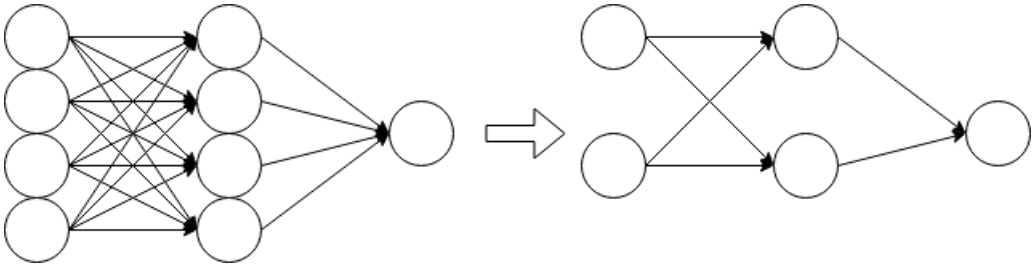


Figure 3.7: Dropping of neurons in a neural network with one hidden layer and $p = 0.5$.

3.2 Long Short-Term Memory Networks

The main characteristic that differentiates an RNN from other neural networks is the formation of direct cycles between their neurons. This provides time processing and sequential learning to the network with the creation of internal states. These details and the complexity of the RNNs usually cause them to require a greater amount of memory and a longer execution time.

The internal states in the RNNs keep part of the activation of the previous entries and perform a feedback of the calculations of the network with each step of time. Defining $\boldsymbol{G_{hh}}$ as the weight matrix of the hidden layer represented by the set of transformations that occur in the hidden layer neurons destined to the neurons of the hidden layer itself, we have the Equations 3.14 and 3.15.

$$h(t) = f_h(i(t)\boldsymbol{G_{ih}} + h(t-1)\boldsymbol{G_{hh}}) \tag{3.14}$$

$$o(t) = f_o(h(t)\boldsymbol{G_{ho}}) \tag{3.15}$$

The neural network LSTM is a subtype of RNN that seeks to solve the problem of long-term dependency [27]. The memory cells present in the LSTMs were implemented to solve this problem because they can preserve the states against long periods of time. We can observe the basic architecture of a block or unit of LSTM memory in Figure 3.8. We can have several of these units that connect to a standard structure of an RNN or other units.

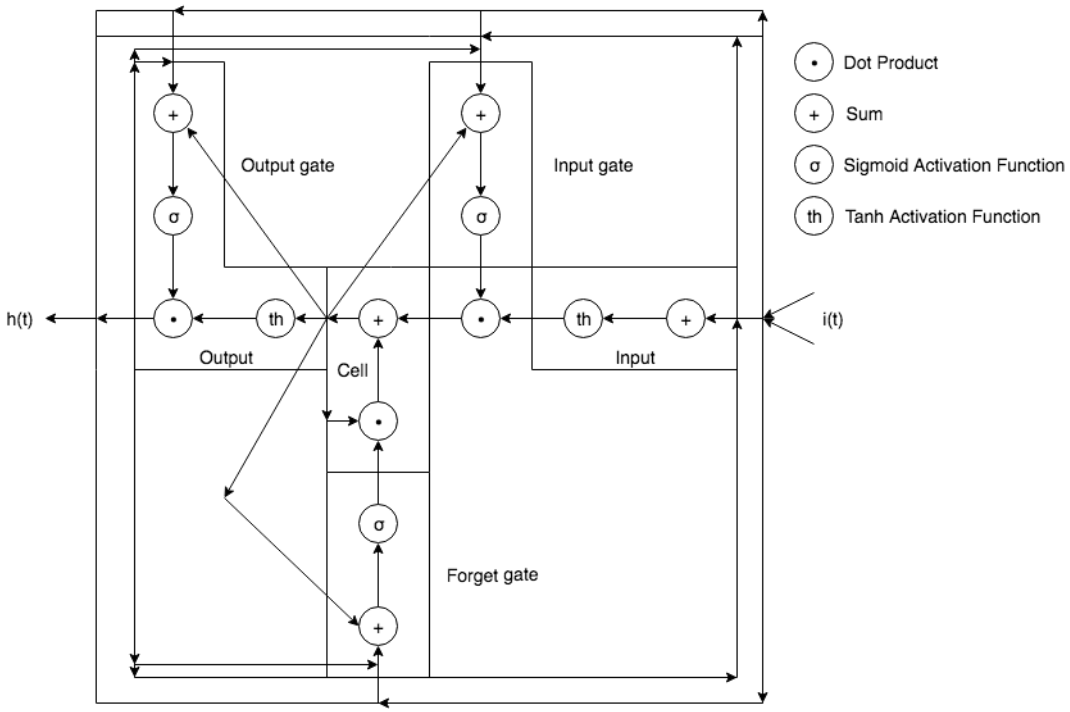


Figure 3.8: Architecture of an LSTM memory block. The neural network is usually composed of several of these blocks.

Memory cells through self-connections store the network time state. Also in each unit there are 3 gates, the forget gate, the input gate and the output gate. The forget gate acts by using the internal state of the cell before adding it back to the cell as input. This happens with the use of a self-recurring connection, adaptively redefining the memory cells in an adaptative way. The input gate attempts to control the input activations in relation to the memory cell. The control of the output of the activation cells is responsibility of the output gate. Finally, the peephole connections (the connections that leave the memory cell destined to the gates) are an attempt to learn the precise timing of the outputs, but they do not greatly increase the performance and are sometimes are omitted in the LSTM units' representations [6].

Defining $\boldsymbol{y}$, $\boldsymbol{x}$, $\boldsymbol{w}$, and $\boldsymbol{z}$ as the forget gate, input gate, output gate, and cell activation vectors respectively, $\boldsymbol{G_{ix}}$ as the weight matrix of the input vector destined to the input gate, $\boldsymbol{G_{hx}}$ as the weight matrix of the hidden layer destined to the input gate, $\boldsymbol{G_{zx}}$ as the weight matrix of the cell vector destined to the input gate, $\boldsymbol{G_{iy}}$ the weight matrix of the input vector destined to the forget gate, $\boldsymbol{G_{hy}}$ as the weight matrix of the hidden vector destined to the forget gate, $\boldsymbol{G_{zy}}$ as the weight matrix of the cell vector destined to the forget gate, $\boldsymbol{G_{iz}}$ as the weight matrix of the input vector destined to the cell, $\boldsymbol{G_{hz}}$ as the weight matrix of the hidden vector destined to the cell, $\boldsymbol{G_{iw}}$ as the weight matrix of the

input vector destined to the output gate, $\boldsymbol{G_{hw}}$ as the weight matrix of the hidden vector destined to the output gate and $\boldsymbol{G_{zw}}$ as the weight matrix of the cell vector destined to the output gate, we have the Equations 3.16, 3.17, 3.18, 3.19, and 3.20.

$$x(t) = \sigma(\boldsymbol{G_{ix}} i(t) + \boldsymbol{G_{hx}} h(t-1) + \boldsymbol{G_{zx}} z(t-1) + b_x) \tag{3.16}$$

$$y(t) = \sigma(\boldsymbol{G_{iy}} i(t) + \boldsymbol{G_{hy}} h(t-1) + \boldsymbol{G_{zy}} z(t-1) + b_y) \tag{3.17}$$

$$z(t) = y(t) z(t-1) + x(t) \tanh(\boldsymbol{G_{iz}} i(t) + \boldsymbol{G_{hz}} h(t-1) + b_z) \tag{3.18}$$

$$w(t) = \sigma(\boldsymbol{G_{iw}} i(t) + \boldsymbol{G_{hw}} h(t-1) + \boldsymbol{G_{zw}} z(t) + b_w) \tag{3.19}$$

$$h(t) = w(t) \tanh(z(t)) \tag{3.20}$$

3.3 Word Embeddings Algorithms

Unsupervised word embeddings algorithms are a distributed representation of text that has the idea of mapping words into vectors of real values in continuous or embedded representations. The concept of embedded word consists of a concept of learning through a text in which words that have the same meaning present a similar representation. The values present in the vectors are learned as a conventional neural network and the embedded words can serve as a pre-training for a neural network such as an LSTM. There are two unsupervised word embeddings algorithms that are the most commonly used and known [2], which are covered in this work, word2vec and GloVe.

3.3.1 Word2vec

The introduction of word2vec was quite impressive in the natural language processing (NLP) scenario. When we deal with NLP, words are usually treated as discrete atomic symbols. It is easier to differentiate between horse and trojan related images than both represented as words in a malware context, for example. Working as a simple neural network, the performance of word2vec was superior to other word embeddings algorithms that

were published until its release, such as the neural network language model (NNML) [9].

In Figure 3.9, we can observe the example in word2vec for the input vector corresponding to the word horse. The hidden layer in this case has a linear activation function for its neurons and in the output layer the softmax function is used. The vocabulary consists of all the words that appear in the text that we are considering or in the words that appear at least a certain number of times. Thus, the input vector is represented by n components that constitute the total number of words present in the vocabulary.

The component with value 1 associated in Figure 3.9 represents the word horse in that position of the input vector. Meanwhile, all other words are represented by 0 in their positions in this same vector. In the output, we have a vector with the same number of components of the input vector but with values corresponding to the probabilities that the words appear next to the word horse, for example. Therefore, the word trojan would have a value between 0 and 1 which would correspond to the probability that it appears next to the word horse. It is important to note that the sum of all the probabilities returned in the output layer must result in the value 1.

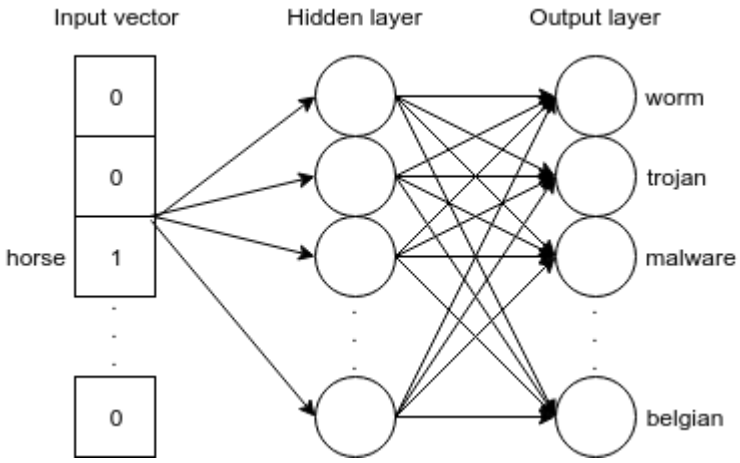


Figure 3.9: Architecture of the word2vec considering the input vector for the horse example.

This closeness between words in a given text can be considered through a window of size c. An example with $c = 1$ for the case of the word horse would be the probability that the word trojan would appear at a distance of 1 for horse in terms of words, either forward or back in text. This can be seen in Figure 3.10, in which two prediction models used in word2vec are observed, the skip-gram (SGRAM) and the continuous bag-of-words (CBOW) with the Equations 3.21 and 3.22 being relative to these models, respectively and in which $v_1, v_2, ..., v_T$ represent the sequence of words.

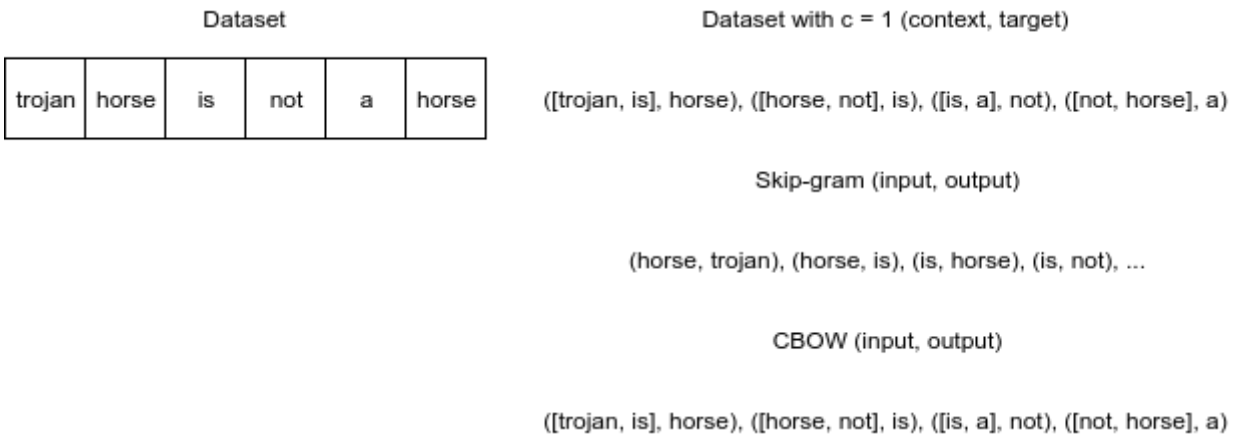


Figure 3.10: Example of a dataset using the skip-gram and CBOW models

$$SGRAM = T^{-1} \sum_{t=1}^{T} \log p(v_t | \sum_{-c \leq j \leq c, j \# 0} v_{t+j}) \tag{3.21}$$

$$CBOW = T^{-1} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \# 0} \log p(v_{t+j} | v_t) \tag{3.22}$$

We can notice a reversal of the pair (context, target) in skip-gram, which does not occur in CBOW. This change interferes with the input and output of word2vec. In the case of skip-gram, the architecture of word2vec remains similar to Figure 3.9. However, when we consider CBOW, the architecture becomes slightly different with the inclusion of more input vectors in the network for each word related to the context. The probabilities can be obtained by the softmax activation function adapted to the skip-gram in Equation 3.23, where u_v is a target embedding vector and d_v is a context embedding vector for v, swapping both vectors in the case of CBOW.

$$p(v_c | v_t) = \frac{e^{d_{v_c}^T u_{v_t}}}{\sum_{v=1}^{V} e^{d_v^T u_{v_t}}} \tag{3.23}$$

The inversion may seem to show no significant change between the prediction models but this is not true. The skip-gram gives better results [39] by treating each pair (context, target) as new samples but CBOW has a faster execution. However, when dealing with large datasets, we must also be aware of the problem of high dimensionality [65] and sparsity, since not always a larger number of samples returns more precise results.

Finally, the purpose of word2vec is to learn about words that appear in similar contexts. In the case of the words horse and trojan, the algorithm could help in the classifica-

tion of a text among the categories information security or traction animals, for example. Some analogies could also be made, such as $horse - animal + trojan = malware$.

3.3.2 GloVe

GloVe is a counting-based model, different from word2vec, which is a predictive model. The implementation of GloVe was an attempt to use the advantages of counting methods and linear extraction of meanings that occurs in word2vec, for example. The name of the algorithm, global vectors, expresses the ability of the algorithm to directly capture the statistics of an entire text or dataset.

In Table 3.1, we can see an example of how GloVe works for trojan and worm, demonstrating the frequency between words present in rows and columns within the same context. The last column of Table 3.1 could be any random word in the text or dataset that had a minimal degree of relationship with trojan and worm.

Table 3.1: Co-occurrence between words considering the examples trojan and worm.

Probability and ratio	$s = horse$	$s = replication$	$s = malware$	$s = belgian$
$p(s\|trojan)$	high	low	high	low
$p(s\|worm)$	low	high	high	low
$\frac{p(s\|trojan)}{p(s\|worm)}$	high	low	~ 1	~ 1

Defining D_{qs} as the count of the instances in which q is in the context of s, the terms b_q and $\tilde{b_s}$ as the respective biases to q and the statistical mean of s, v_q representing the vector of the main word and v_s the vector of the probed words, we have the Equation 3.24. The same reasoning can be seen in the Equation 3.25 only by changing the variable s to r, where v_r is the vector of the context word and b_r the bias.

$$log(D_{qs}) = v_q^T \tilde{v_s} + b_i + \tilde{b_s} \tag{3.24}$$

$$log(D_{qr}) = v_q^T v_r + b_i + b_r \tag{3.25}$$

The f weighting function is used to avoid learning only extremely common word pairs. The authors of GloVe chose $a_max = 100$ and empirically $\alpha = 0.75$ as the most appropriate value for the incognito in the exponent of the Equation 3.26. Finally, considering a possible set of words like s, $q = trojan$ and $r = worm$, we finally have E as the cost function

expressed in the Equation 3.27, where U is the size of the vocabulary.

$$f(a) = \begin{cases} (\frac{a}{a_{max}})^{\alpha}, & \text{if } a < a_{max} \\ 1, & \text{otherwise} \end{cases} \tag{3.26}$$

$$E = \sum_{q=1}^{U} \sum_{r=1}^{U} f(D_{qr})(v_q^T \tilde{v_s} + b_q + \tilde{b_s} - log(1 + D_{qs}))^2 \tag{3.27}$$

As we can see, Table 3.1 is formulated through the implementation of a co-occurrence matrix, in which each entity in a row (words) is paired with the entities present in the columns (context). In the example, it is noted that the words malware and belgian are not good for discriminating trojan and worm, unlike horse and replication.

It is expected that the co-occurrence matrix will be filled with zeros when we consider a whole dataset. However, we have the factorization of the matrix for vectors of words and vectors of contexts (features) in the search for dimensionality reduction, and this occurs with the attempt to minimize the reconstruction loss function. There is a normalization of counting and exponential-logarithmic smoothing, in which the most recent observations carry a greater weight than the older ones, technique also used in predictive models, such as word2vec. In the GloVe article, it is stated by the authors that the results are obtained faster and are better than those returned by word2vec, regardless of the velocity [45].

Chapter 4

Related Work

This chapter follows a chronological order on related work, since more papers with different approaches about viruses and other types of malware started to be published. The main aspects of the papers are discussed in relation to the implemented models of this work. Lastly, the adversarial examples are also considered because of their current relevance for the malware classification.

4.1 Binary Approach

The first mention of computer viruses with an emphasis on dealing with these threats in a comparative way with viruses in humans occurred in 1988 [42]. An epidemiological analysis is carried out and situations of isolation, quarantine, immunization, among others, are described. All of these situations are approached analogously between viruses in humans and computers, providing ideas that protect computer systems and networks.

Years later [30] still under the biological aspect, is designed the first neural network for defense against computer viruses. In this case, this antivirus technology had as main objective to detect boot viruses that at that time were responsible for 80% of the incidents. The boot sector was a small code sequence of 512 bytes in size. The number of existing viruses was also very small (about 4000 viruses, including 250 of boot) compared to the current amount. Despite the simplicity of the neural network model and the small number of virus samples, the problem has already been addressed in the form of n-grams[1] (3-byte strings or 3-grams). A training set with 150 samples included 76.500 3-grams, in which 25.000 were distinct. Only the most frequent 3-grams were considered, and the network had only one layer and 50 weights, so they tried to reduce these 25.000 3-grams to a

[1] `https://web.stanford.edu/~jurafsky/slp3/3.pdf`

window of size 50. The result between the virus and non-virus classes was a rate of about 15% of false negatives and 0.02% of false positives with an estimated accuracy of 85% for new boot viruses.

Different types of features have been proposed in another work [52]. Three features were used and one of them was the portable executable (PE) headers, that are a Windows relevant file format for executables, dynamic-link libraries (DLLs), object file, etc. The headers are responsible for informing how the respective PE mapping to the dynamic linker is done. The remaining features were the n-grams strings (obtained by extracting strings from binaries, such as "kernel" and "advapi", for example) and byte-sequences (using the hexdump tool, transforming the binary files in hexadecimal). The dataset consisted of 4.266 files in total, including 3.265 malware and 1.001 cleanwares, all labeled as malicious or benign. All malicious executables were obtained from various FTP sites and the labels were assigned by a commercial virus scanner that was not specified in the article, with 5% of the malware in the dataset consisting of trojans and the remainder by viruses. The cleanwares were mostly collected from Windows 98 and others downloaded from the Internet. The URL to the dataset is no longer available. The machine learning algorithms used were an inductive rule-based model (RIPPER classifier) for the PE headers, probability-based model (naive bayes) for the n-grams strings and multi-naive bayes model for the byte-sequences. The best result of accuracy was the multi-naive bayes with 97.76%. The naive bayes had the lowest false positive rate at 3.80%. There was not an approach mixing the different types of features and the dataset that served as input for the RIPPER model was smaller, with only 244 files, making impossible an accurate evaluation in relation to the other classification algorithms.

The features approach of this master's thesis was mainly based on a work published in the last decade [32]. Again, using the hexdump tool, each executable of the dataset was converted to hexadecimal codes and n-grams were made from the combination of each 4-byte sequence, resulting in 256.000.000 features. The dataset consisted of 3622 executables, including 1651 malwares and 1971 cleanwares. The cleanwares were collected from Windows 2000, XP and SourceForge[2]. The malwares composed of viruses, worms, and trojans were obtained from VX Heavens[3] and from forensic computer experts at MITRE Corporation[4]. Several classifiers were used, such as term frequency-inverse document frequency (TFIDF), naive bayes, decision tree (J48), support vector machine (SVM) and

[2] `https://sourceforge.net/`
[3] `http://83.133.184.251/virensimulation.org/`
[4] `https://www.mitre.org/`

boosted [15]. Because of the large number of features, there was a reduction for the 500 most relevant n-grams. The training and test sets were cross-validated using 10-fold cross-validation, which consists of randomly partitioning the executables into 10 disjoint sets of the same size, choosing one of them to be test set and the other 9 combined to be the training set. This procedure is repeated 10 times, choosing a different test partition every time. The classification experiments performed occurred in the larger dataset of 3622 executables and in another smaller dataset. In relation to the largest dataset that returned the best result, an accuracy of 99.58% was obtained with the boosted J48. The authors report a high computational overhead and time to perform the experiment on the larger dataset but do not cite more details about the features used and runtime.

Three years later, another article [10] did a statistical analysis of the distributions of opcodes, which are the machine language instructions that specify the operations to be performed by the programs. Despite the small sampling of 67 malware executables that have been disassembled, that is, they have gone through a reverse engineering process using IDA PRO[5], more relevant opcodes to discriminate cleanwares from malwares have been discovered. The author found that in fact the rarer opcodes present in executables are a more important predictor for differentiating malware from cleanwares than the other way around. In addition, it is also possible to differentiate malware types through these rare opcodes often below 0.2% of the total opcodes. The most frequent opcodes were responsible for explaining only a variation between 5% and 15% in terms of prediction while the rarest had a variation between 12% and 63%. The approach of this master's thesis tried to implement this idea in terms of the n-grams considering the sequences of bytes, but the obtained results were inferior by the large number of features generated and rare n-grams were not a good option for differentiation.

Another work [51] with a huge dataset and false positive detection rate of only 0.1% was published in 2015. The dataset was created from benign and malicious binary files obtained from Invincea's[6] own computers and its client network. The total files included in the dataset were 431.926, with 350.016 labeled as malware and 81.910 as benignware. Labeling occurred through VirusTotal[7], a site made up of several antivirus products and which returns the results of these scans for uploading one or more files by users. Unlike this master's thesis, the authors labeled as malware files that were considered malware by 30% or more of antivirus scans and those with a malware alarm of 0% were labeled

[5] https://goo.gl/WHK8nq/
[6] https://www.sophos.com/en-us/lp/invincea.aspx/
[7] https://www.virustotal.com/

benignware. Files that were detected as malware by more than 0% and less than 30% by antivirus softwares have been discarded by the uncertainty of their malignant or benign nature. The experiments were run on an Amazon EC2 g2.8xlarge instance with 60 GB of RAM and four 1.536 CUDA core graphical processing units, but only one was used. The Keras[8] framework was used for the implementation of the neural network model proposed by the authors. Four types of features were considered for the creation of what served as input to the neural network. The first feature was obtained by binary values from a two-dimensional byte entropy histogram that modeled the distribution of bytes in the benign and malignant files. The second feature was derived from the input binary's import address table, constituting the initialization of an array with 256 integers with value zero and that can be incremented according to another table of DLLs, creating a model that tries to capture the semantics of external function calls. The third feature is similar to the extraction of the PE headers performed in [52] but in this case it was through the numeric fields present in the binary files and using the Python "pefile" parsing library, entering the names also obtained in an array of size 256 and with that, trying to help the neural network in learning signatures of malware. The last feature is just a concatenation of all the previous features in a large vector and according to the authors, the reduction to a smaller vector dramatically reduced memory usage and central processing unit (CPU) time to load and train the neural network without greatly harming the ending result of accuracy. The neural network model was elaborated with four layers, in which three were constituted by 1024 nodes, using the dropout technique and the parametric rectified linear unit (PReLU) [24], as activation function in the first two layers and the sigmoid function in the last hidden layer, being the fourth layer related to the prediction. The proposed features were analyzed together and separately, with the metadata extracted from the PE obtaining the best result. In the set with all features, the accuracy was 95.2%. The input of the neural network was reduced to 256 and 200 epochs were performed for training. Each epoch took about 15 seconds to be trained and the entire model was trained in about 40 minutes, and the training chart showed no signs of overfitting. The article in this experiment provided a link to the code of the neural network but unfortunately it is not possible to see it because it is protected by copyright terms and the dataset is also not accessible.

Despite the wide use of n-gram byte in numerous works, this approach was questioned in [48], which disagrees with [52] on increasing the amount of data necessarily improve the results. The authors have shown that there is always a tendency to overfitting even

[8] `https://keras.io/`

with the 6-gram that got the best results in the experiments and generated 1.6 million features. The dataset was constructed with 400.000 training samples, divided equally between benign and malignant files and 77.349 test samples, 40.000 of malignant files and 37.349 of benign ones. In addition, the memory consumption is very high, and it is better to include other features for classification as well. Therefore, the authors consider that the n-gram byte approach is overestimated and had never been questioned for static malware analysis. In this master's thesis, considering the context of deep learning and byte n-gram, we looked for the best that we can get from this approach to use it in conjunction with other features in other works, considering the help of word embeddings algorithms.

Three neural network architectures (LSTM, gated recurrent unit (GRU) [12] and CNN) are proposed in another work [5]. The LSTM with temporal max pooling and logistic regression returned the best result according to the authors, with the false positive rate being only 1%. They also state that features learned by the LSTM language model help improve performance when compared to other random-weighted-based and initialized architectures such as the echo state network (ESN) [28]. The dataset used in the experiments with the three neural networks was created from 50.000 files for training, 10.000 for validation and 15.000 for testing. The number of files malignant and benign was equal, that is, 37.500 of each type. As input to the LSTM with temporal max pooling, a dynamic analysis of the files was performed and a vector of 114 positions were created for the API calls found in the dataset. The second stage considered after the outputs returned by the LSTM, was constituted of the logistic regression as final classifier. The maximum number of epochs was 15, with the mini-batch used being 50 and 1.500 hidden units. As in this master's thesis, the Keras framework was also used for the implementation of the LSTM model.

4.2 Multiclass Approach

A malware family consists of malicious code that shares the same characteristics and behaviors [17] and an approach considering this was made in this decade [13] to solve the problem of sparse binary features with the implementation of a capable architecture to handle 2.6 million labeled samples, returning an error rate of 0.49% with a single neural network and 0.42% with a set of neural networks. To obtain good classification results, random projections that reduced dimensionality in the input space by a factor of 45 were used, allowing the neural network to be trained on a high-dimensional input data. The number of malicious samples present in the dataset was 1.843.359 and 817.485 were of

benign samples. Most of the files were manually labeled by analysts and the remainder by other sources, such as CERT[9]. Each malicious file was also associated with a malware family. From these families, a set of files belonging to the 134 most important families to be identified was selected. This was determined by analysts who also created another generic class to label the remaining malware files that were not considered to belong to these 134 families. Three different features were extracted that consisted of system strings for creating unknown malware files on a virtualized machine, denominated 3-grams of system API calls (three consecutive system calls), and a combination of API call with an input parameter (which assists in the individual identification of malware families). These features were extracted in real time with the dynamic analysis of malware behavior. Two classification techniques were used, including logistic regression and neural networks. About the neural networks, a final softmax classification produced the probability that the file would belong to one of 136 classes, including 134 families of malware, a generic class of malware and a class corresponding to cleanwares. Several network architectures were used, and the authors concluded that more than one layer did not have many benefits, the neural network of a layer without pre-training returned the best results with an error rate of 9.53% for multiclass classification. The sparse binary inputs were 179.000, followed by 4.000 linear units corresponding to the random projection. After that, 1.536 sigmoid hidden units and to finish, the final layer corresponding to the softmax classifier. The momentum value used for training was 0.9 and learning was 0.3, using a NVIDIA C2075 graphics processing unit (GPU) and taking about three hours. Despite the robust work, the dataset used and the code on the different neural network architectures implemented were not made available. A dynamic analysis was also carried out, differently from this master's thesis, with an enormous quantity of samples that requires resource and time, even with manual labeling of the samples by several analysts.

In 2015, the Microsoft malware classification challenge[10] (BIG 2015) occurred. The dataset still available has about 400 GB, consisting of 9 different malware families with 21.741 samples, 10.868 for training and 10.873 for testing. The number of samples of each class is not uniform in this dataset, which presents almost 3.000 instances of the Obfuscator.ACY family and less than 500 of the Vundo family, for example. The winner of the challenge made a feature extraction engineering composed of opcodes count, 4-gram byte, single byte frequencty, and so on. The modeling was done with XGBoost (the machine learning library focused on gradient boosted trees[11]) and the accuracy obtained

[9] `https://goo.gl/pHFQMW/`
[10] `https://www.kaggle.com/c/malware-classification/`
[11] `http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/`

was 99.87%. This work was very relevant due to the large number of different features that were considered but the focus was on obtaining the best possible classification for the Microsoft dataset to overcome the challenge, not being necessarily a better approach to other malware classification scenarios.

Using the same Microsoft dataset, [1] managed to get an accuracy of 99.77%. Executing only static analysis, hexadecimal-based features of malware files were extracted, including 1-gram, 2-gram, metadata, entropy, strings histograms and grayscale representation of images. Other features were extracted from the assembly of these same files (through IDA PRO), including frequency of symbols, opcodes, registers, etc. The feature fusion occurred with the creation of a long vector, representing all this extracted information. The authors also used XGBoost and a laptop with a quad-core processor (2 GHz) and 8 GB RAM. 2-gram featured more than 65.000 features and took about 10.123 seconds to extract. Because of this amount of time, 3-gram and 4-gram were discarded. The accuracy obtained was lower than the work that won the challenge, but the complexity and computational resources used were smaller.

4.3 Adversarial Examples

One of the most important current issues in deep learning and information security are the adversarial examples, which are made up of entries created purposely to misclassify the results of deep neural networks [44] according to the goals of the attacker. This can occur in the form of white-box, when there is prior knowledge of how the deep neural network was modeled or black-box, which is otherwise, when the model is unknown.

In the malware analysis, the problem is also pertinent [22], in which it is possible to do a misclassification by classifying a benign file as malignant and vice versa, for example. In the case, adversarial examples are applied in a malware detector for the classification of dataset DREBIN [4], which contains files from Android system. DREBIN consists of 123.453 benign and 5.560 malware files. A malware detector was trained specifically for the problem that considered as features, vectors composed of binary indicators of system file calls, whether they execute these calls or not, similar to the entries of [5]. Also, the case considered as the worst by an attacker in this article, was the total knowledge about the model and training data, that is, completely white-box. Then we modify the vectors that can only have binary values in a minimal way that is able to change the classification, like adding system calls that do not occur, however in a way that is

not abrupt and influences in too many intermediate changes of the gradients. It was considered at most a change of 20 system call features to create the adversarial examples in the problem. For these modifications to preserve the functionality of the applications intact, it was only possible to change some files originating from Android itself. The deep neural network architecture considered for classification in DREBIN by the authors consisted of two hidden layers with 200 neurons in each. Of all the features, only 0.0004% or 89 system calls were specifically used to deceive the classifier. Finally, considering a black-box situation for the models implemented in this master's thesis and described in Chapter 6, probably a problem could be the increase of the computational resources necessary for the appropriate classification of the malwares. In the case of white-box, the problem could be a more serious misclassification, with a new interpretation of the inputs by the network models.

Chapter 5

The MC-dataset

In this chapter, the methodology adopted for the creation of the MC-dataset used in this master's thesis is discussed. The dataset, besides being elaborated exactly for the purpose of this work, also had the objective of public disclosure to other users and researchers to use the available data. Considering the number of samples and the exclusivity of the classes present (especially the cleanwares), the MC-dataset at the time it was launched, presented itself as the largest public labeled dataset available.

5.1 Overview

The dataset was divided into two parts, one part related to the binary (MC-dataset-binary) case with the cleanware and malware classes, and the other part related to the multiclass case (MC-dataset-multiclass) with the classes backdoor, cleanware, trojan, rootkit, virus and worm. Both parties followed practically the same methodology for creation, which can be visualized in Figure 5.1. The MC-dataset was created in a totally balanced way in relation to the number of samples in each scenario and this determined the choice by which the dataset was divided. In MC-dataset-binary, 5.740 samples of each class are present and in MC-dataset-multiclass, 3.290 samples are also present for each class. The total size of each compressed dataset is 1.87 GB for MC-dataset-binary and 2.91 GB for MC-dataset-multiclass.

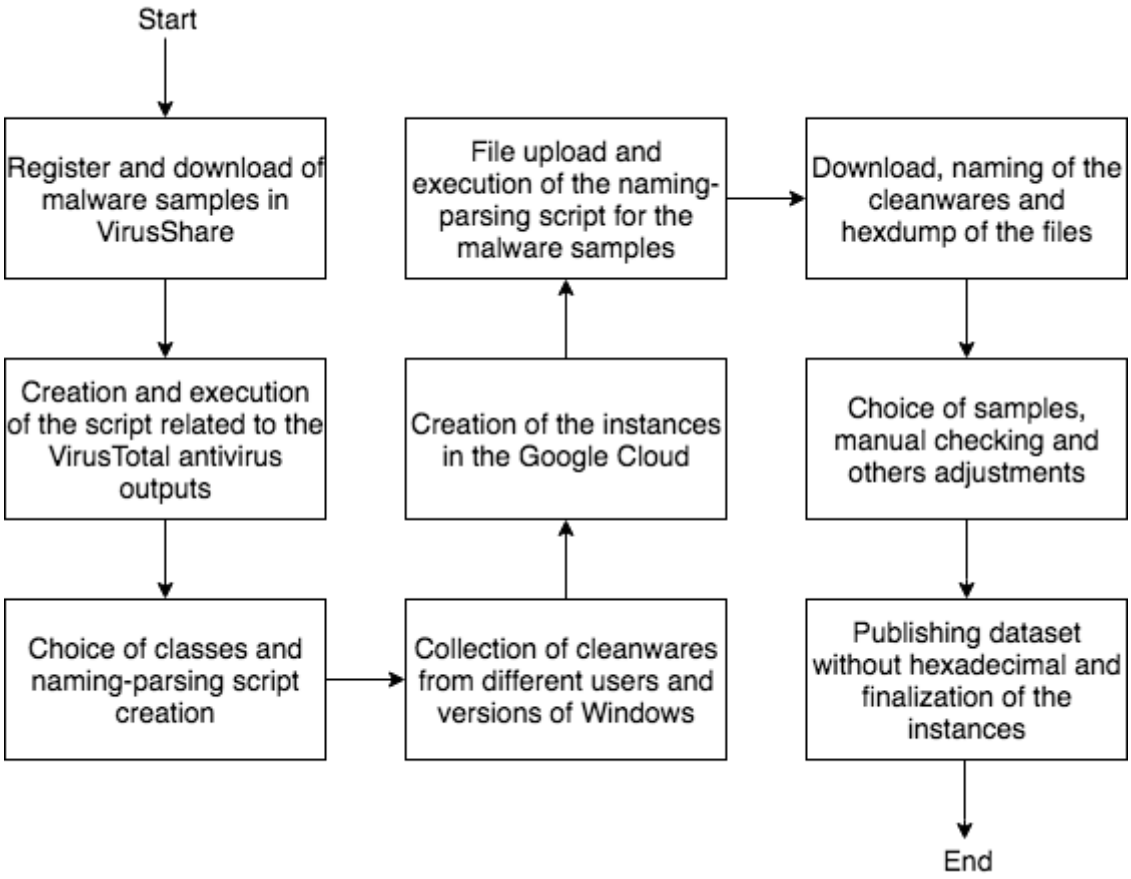


Figure 5.1: Schematization of the methodology for the creation of the MC-dataset.

It is important to note that the files selected for the dataset, both cleanwares and malwares, do not exceed something around 3.2 MB. We did this for a better fit in LSTM models and to maintain a reasonable size for uploading the dataset itself. Another characteristic that needs to be mentioned in our dataset are the 32-bit and 64-bit files. It is not an easy task to check if a file is 32-bit or 64-bit in a set of files from different sources. We tried to put half of the cleanwares samples as 32-bit files and the other half as 64-bit, but we know that this amount was not perfectly exact.

Most malware analysis papers usually deal with Windows executables only. However, malwares obtained from VirusShare contained different media types[1] (also known as MIME types). This is not entirely reliable because malwares can use a variety of techniques that "trick" this simple identification based on headers and some other parameters. Steganography [29] and encryption [47] are examples of techniques that can easily fool this check. Anyway, files were inserted into the MC-dataset with similar MIME types and Windows executable files were the vast majority, both for cleanwares and malwares. Basically, 5010 cleanwares and 5056 malwares inserted into the MC-dataset-binary were from

[1] `https://goo.gl/8oeynU/`

Windows executable files. In MC-dataset-multiclass, were inserted 2995 cleanwares and 14894 malwares (considering the five distinct malware classes as one) of these executable files. Again, it is not possible to state precisely these numbers in relation to malwares because of the techniques that can deceive this simple file type verification.

The approach proposed by [32] was followed, in which the *hexdump* of the malware sample files is used to generate text files containing the bytes of the original files, represented by hexadecimal pairs. We discard information such as memory address or checksum that are sometimes present. Once this is done, the hexadecimals are converted to numbers that serve as input to our LSTM models. These numbers are read in sets of sixteen hexadecimal digits, which corresponds to a 32-bit words pair or just a 64-bit word. The *hexdumps* were not included in the dataset for size reasons and because they were an approach created exclusively for our work. The organization of the directories and files of the dataset can be seen in Figure 5.2.

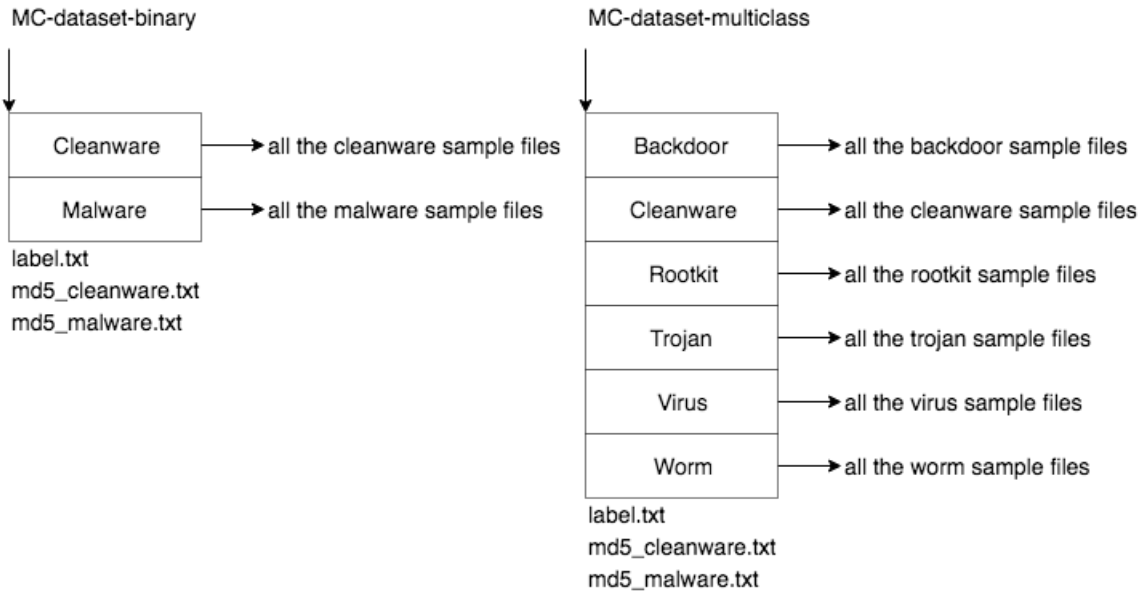


Figure 5.2: MC-dataset file structure.

- **label.txt:** This file contains all samples of the dataset labeled row by row. Firstly, the file name appears and then, its class, with the separation made by a ";". Examples: "cleanware-b221fbcd761bf3ee1d6d9f0968abb9c7;cleanware" and "malware-4719a04edaf693b42be3f8b352682500;trojan" (the insertion of "cleanware" and "trojan" into the names was just a way of making the files easier to handle).

- **md5_cleanware.txt:** Here are all the MD5 hash codes that are part of the name of the cleanware files. It was also done for the purpose of hiding the original name of the files. This separate file assists in checking the originality of the cleanwares,

even though the files name already contains the hash codes. The malware samples downloaded from VirusShare are accompanied by a similar file. Example of a line in the file: "b221fbcd761bf3ee1d6d9f0968abb9c7".

- **md5_malware.txt:** Same idea as the file *md5_cleanware.txt* but for the samples of malware. Example of a line in the file: "4719a04edaf693b42be3f8b352682500".

5.2 Data Gathering

The first step in building the MC-dataset was the register on VirusShare. An account must be requested from the repository administrators along with a justification for accessing torrent files containing malware samples of various types and systems, but most of them are trojans designed for Windows. The size of the torrent files in VirusShare is quite variable. The largest downloaded torrent size for this work was 106.99 GB with 131.072 samples and the lowest was 5.01 GB with 65.536 samples. The number of downloaded malware samples was 7.340.032, totaling 2314.34 GB.

The cleanwares were obtained from various Windows systems of different collaborating users and from software downloads on the Internet, such as The Portable Freeware Collection[2]. All cleanwares underwent a check of some antivirus as well to avoid incorrect labeling. For this entire collection process, mainly malware, additional machines were required and for this, Google Cloud[3] was used to create instances of virtual machines. In Section 5.3, this creation is best detailed.

Comparing the size and quantity of downloaded samples with the final version of MC-dataset, we have a huge difference. This was because of the dataset balancing, which was based on the number of avaliable cleanwares. The cleanwares were harder to obtain and the restriction of the size of the files was approximately 3.2 MB maximum. In addition, the amount of trojans present in the torrent files is much higher than the other types in VirusShare and the excess was discarded.

5.3 Data Labeling

Although a small piece of code has already been created for collecting cleanwares on the computers of collaborating users, the first script was written for VirusTotal. This script

[2] `https://www.portablefreeware.com/`
[3] `https://cloud.google.com/`

was responsible for generating a JSON with the information of all samples of malware downloaded to this work through VirusShare. The information was simply text files for each torrent with the outputs that are shown when the MD5 hashes of the malware files are inserted into VirusTotal, which includes the results of about 50 antivirus scanners for each file.

With this, the criterion adopted for MC-multiclass-dataset was the count of the results of all antivirus for each malware and classified them according to the class returned as the majority. For example, we could imagine the malware file named "malware-1edb420700aa68aba62122b69b805853" classified as trojan by 30 antivirus scanners and 20 classified as backdoor. In this case, we consider malware as belonging to the trojan class. In the case of a tie, we simply did not put the malware in the dataset. All files were named this way, in both datasets, the word cleanware or malware attached to the respective md5 of each file. Other approaches have been considered for labeling in other works, such as [51], but there is no consensus on the best choice. Although we know of the existence of hybrid malwares, we do not consider such cases that would be more related as a multi-label problem.

The second and most important script was the one that made this choice for a particular class through the outputs returned from VirusTotal. As previously stated, cleanwares were crucial for balancing the dataset but there were other types of malware that could be inserted into the dataset. Adwares[4] or ransomwares could be considered, for example. In the case of Adware, there is an uncertainty in the information security community about its malignancy[5]. Some adware programs are considered only as "annoying" and others as malware in fact. In addition, they are more confused with trojans than otherwise. The ransomware sample quantity present in the torrents is small, appearing more in the most recent VirusShare torrents and by the results of antivirus scanners, many are also considered trojans since the ransomware-trojan hybrid case is very common.

To speed up the download process of VirusShare torrents and execute the naming-parsing script several times, four instances of virtual machines were created and ran in parallel. The hardware configuration of the instances was pretty basic, using the free credits provided by Google Cloud for a newly created account. More precisely, each virtual machine was created with 200 GB of standard persistent disk, 4 vCPUs n-1-standard-4 with 15 GB of RAM and Ubuntu 16.04 LTS 64-bit[6]. Then for each instance, the following

[4] `https://www.avast.com/pt-br/c-adware/`
[5] `https://goo.gl/BjjAo3/`
[6] `https://cloud.google.com/compute/docs/machine-types/`

order was followed: installation of the necessary dependencies for Ubuntu, upload of a VirusShare torrent, upload of the JSON corresponding to this torrent, extraction of the compressed file from the torrent and execution of the naming-parsing script to select the samples, label and name them. Finally, the files were downloaded to an external hard drive.

5.4 Handling

Shortly after downloading samples of malware from virtual machine instances, it was also necessary to name cleanwares files and organize them along with malwares. Some samples were discarded because there were incongruities, such as duplicates and very small file sizes. A manual check was done after the hexdump of the files to look for some type of irregularity too. For example, some files hidden in the directories were unbalancing the dataset because the hexdump command considered these files at the time of hexadecimals generation. After verifying these details, all files were organized as in Figure 5.2 and sent to the DGX used for some of the experiments of this work.

The publication of the dataset occurred without the hexadecimal of the samples, as explained in Section 5.1. MC-dataset-binary [**?**] and MC-dataset-multiclass [**?**] are in the FigShare[7] data repository and can be downloaded and used by any user. After the MC-dataset has been uploaded to FigShare, all instances used to create the labeled malware samples have been terminated. This process with the virtual machines running took about three months to be finalized. In DGX, the dataset still had to be inserted into a Docker[8] container because of the access restrictions on the machine, only then was possible to run the experiments.

[7] `https://figshare.com/`
[8] `https://www.docker.com/`

Chapter 6

Model Implementation

Three neural network models were proposed for this master's thesis. The implementation of each model is explained in this chapter, from the input of the data to the final classification. These models were developed for the classification of the MC-dataset that was also created for this work, but could be used in other datasets with specific adjustments, if necessary. The LSTM was used in all models, two of which were also composed by the word embeddings algorithms, word2vec and GloVe.

6.1 LSTM

One of the concerns with the LSTM was the cost-benefit regarding the time and configuration of the machine for the experiment. We can say that the chosen architecture is relatively simple and has been tested a lot of times until reaching the desired final version. In Section 7.3 the values chosen for the parameters after the numerous tests are best discussed. Our model for the binary classification consists sequentially in four layers, an input layer, followed by an LSTM layer (hidden layer), a dropout layer, a dense layer and its design can be seen in Figure 6.1.

The terms architecture and model for neural networks are used in a confusing way by many authors. In this work, the architecture refers to the configuration of the network in terms of numbers of neurons, connections between them, layers. The term model refers more to the programming used to implement the different neural networks and others parameters chosen, such as batch size and learning rate, for example. The programming language for the model implementation was Python with the framework Keras, derived

from the open-source machine learning software library called TensorFlow[1].

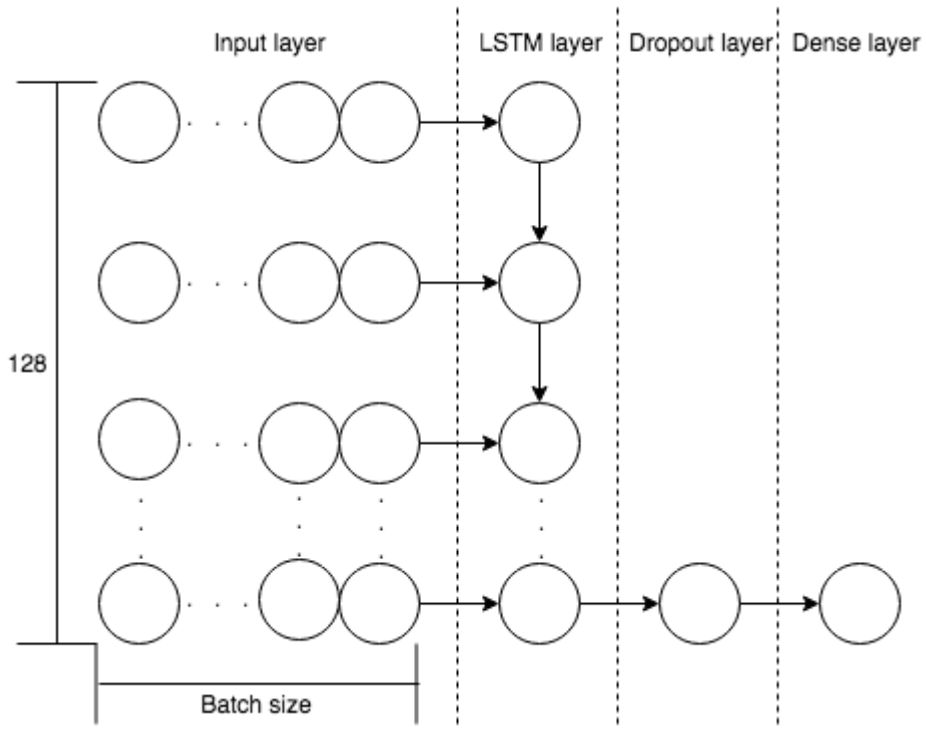


Figure 6.1: The LSTM architecture design for the binary classification.

The input layer receives the hexadecimal samples of cleanwares and malwares converted to numerical values and passes it on to the LSTM layer. Time steps were suitable to fit the LSTM's reading of 2048 hexadecimals for each sample, which represents the most frequent 128 64-bit words and 256 32-bit words presents in each file, because the dataset presents both 32-bit and 64-bit files, as explained in Section 5.1. The use of less frequent words was not a viable approach and has returned worse results, even addressing them in conjunction with the more frequent ones.

The entries in Keras must have a fixed size and a zero padding to the right has been done without impacting the performance of the network. Another important detail is that the model works as a stateful LSTM, which helps in searching for dependencies between the words of the samples and improved the results. In the Figure 6.2 is schematized the process from the hexadecimal generated by the hexdump of a dataset file to the input layer.

[1] `https://www.tensorflow.org/`

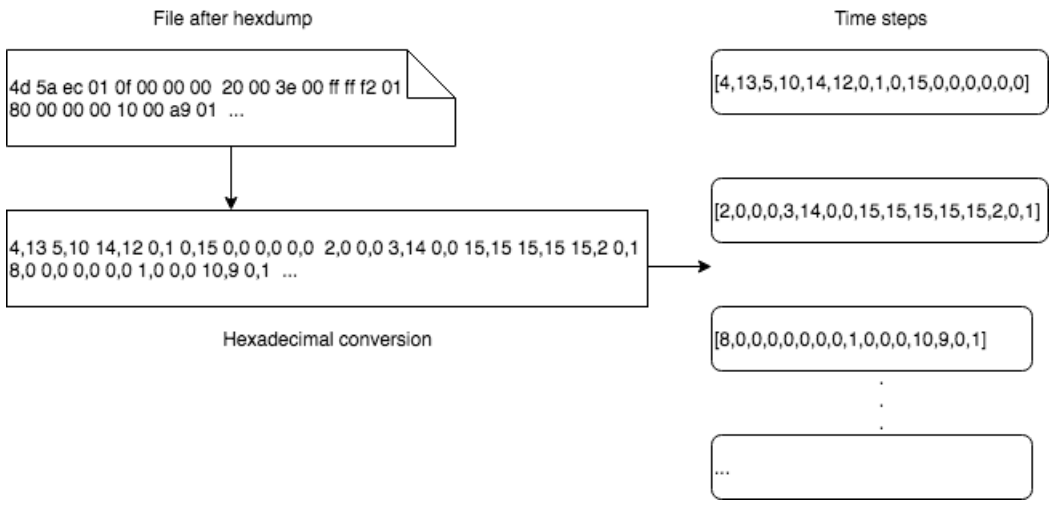


Figure 6.2: Example of converting a sample hexadecimal file to the time steps corresponding to the input layer.

The LSTM layer has about 128 cells that use tanh as an activation function and are responsible for receiving the inputs of the input layer and produce outputs for the dense layer. However, after the LSTM layer and before the dense layer, there is a dropout. Keras defines dropout as a layer but it only consists in randomly setting a fraction rate of input units to zero at each update during training time, which helps prevent overfitting. The value chosen for the dropout was the default 0.5 found in most papers, since there were no different results with other values. In this case, the unit set to zero has as input, the output of the cell. Finally, the dense layer has the sigmoid as activation function for the binary classification.

For the multiclass classification, the architecture is the same, but the dense layer has the softmax as activation function. Briefly, the dense layer returns six values corresponding to the probabilities of the sample belongs to a given class. In binary classification, the idea is the same with only two values. Therefore, the sample is associated with a given class by the highest value returned. In the case of two identical values of probability, the choice considered was alphabetical order of class names, but this never happened in the experiments. The tests for this LSTM model were conducted at a computer with a 2.2 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 of RAM without using GPUs.

6.2 Word2vec+LSTM and GloVe+LSTM

The architecture chosen for the two neural networks with the word embeddings algorithms was similar, with the difference that the embedding layer was seeded with the word2vec

word embedding weights for one of the networks and GloVe word embedding weights for the other. Both models for the multiclass classification consists sequentially in eight layers, an embedding layer (input layer), a dropout layer, an LSTM layer (first hidden layer), another dropout layer, another LSTM layer (second hidden layer), the last dropout layer, a flatten layer, a dense layer and its design can be seen in Figure 6.3.

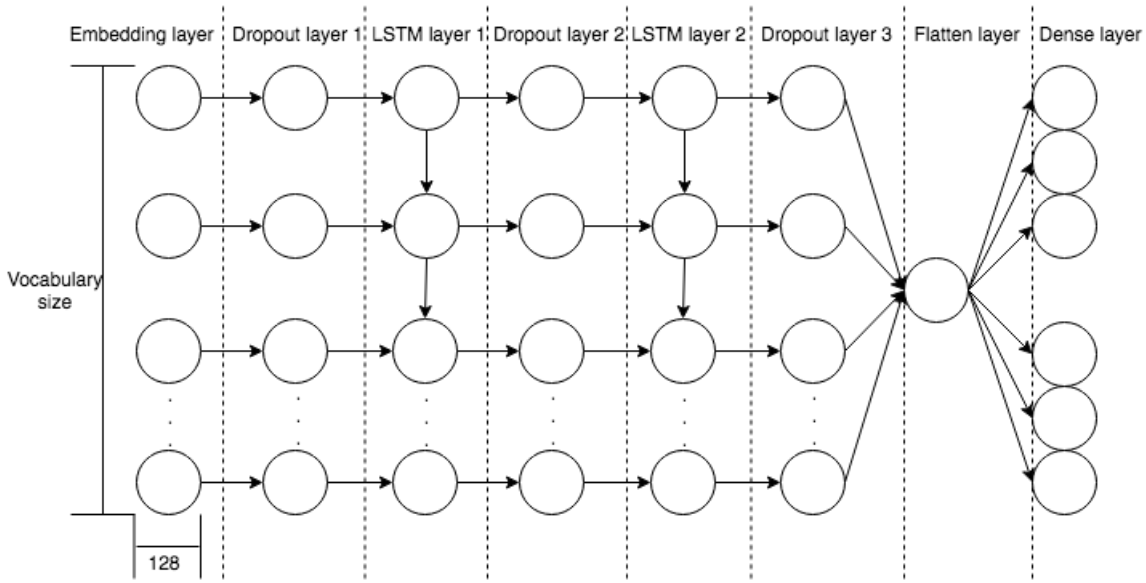


Figure 6.3: The LSTM architecture design with the word embeddings weights for the multiclass classification.

The embedding layer in the models works as a table in which each word in the vocabulary has an index and is represented by a vector of 128 positions. This can be seen in Figure 6.4. These 128 positions have numbers that correspond to the final weights after the execution of word embeddings algorithms. As a result, we have a matrix of weights already trained in the input layer for the samples. Finally, by converting each word from the samples to numeric values and associating it as a dictionary, we get the outputs that serve as input to the first LSTM layer.

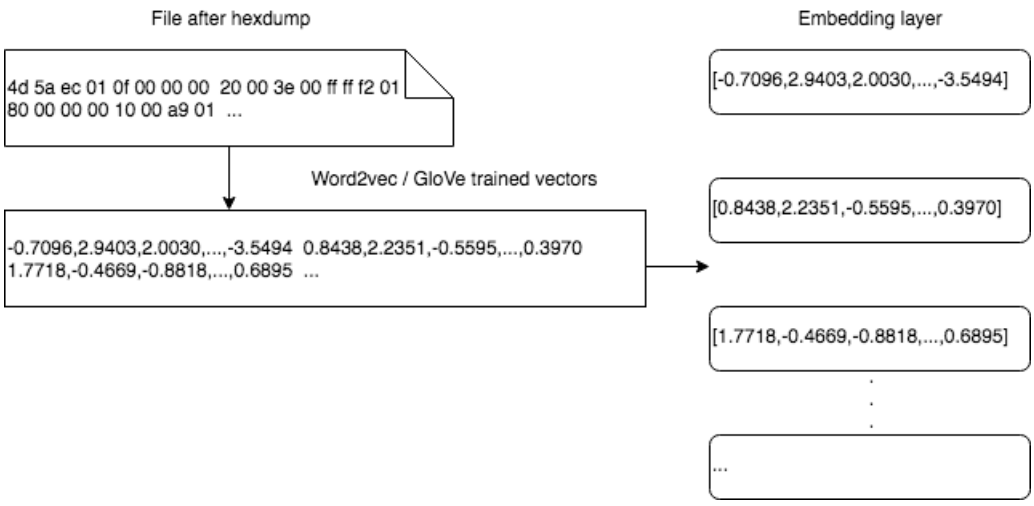


Figure 6.4: Example of the vector training process for words in a sample hexadecimal file to the embedding layer.

A dropout layer with a default value of 0.5 precedes each LSTM layer. As in the Section 6.1 model, the two LSTM layers present 128 cells that use tanh as the activation function. The only difference in this case is that they are two layers instead of one, increasing network complexity and the "return_sequences" flag is set to "true" in Keras. This is necessary when there is more than one LSTM layer, because the output sequence for the next layer will be of the same size.

Since the output of the second LSTM layer is not just a vector, it is necessary to have a flatten layer to do a reshape for the one dimension of the dense layer. For binary classification, unlike the multiclass classification illustrated in Figure 6.3, only two values are required to discriminate the classes with the sigmoid activation function in the dense layer, so just a single neuron is needed. Again, the softmax activation function is responsible for the classification among the six possible classes in the multiclass classification.

Word2vec and GloVe needed a lot of memory because of the size of the vocabulary, more than the processing of GPUs. For a vocabulary of 10 million words and dimension 128 for word2vec, for example, we would have $2 * 4\ bytes\ per\ float * 10000000 * 128 = \sim 10\ GB$ of memory usage. The vectors returned by the two algorithms used for the embedding layer are different in their training method, but they work in the same way as weights for the neural network models. The tests for these LSTM model with word embeddings algorithms were run on DGX with 2 CPUs 20-core IntelXeon E5-2698 v4 2.2 GHz, 512 GB of RAM and 8 GPUs Tesla P100.

Chapter 7

Experimental Results

The experiments that were conducted with the use of the neural networks models proposed in this work for the MC-dataset, are described in this chapter. The outputs and parameters adopted for the binary and multiclass scenarios are discussed after the various tests performed. The distinct results that were obtained are compared and graphics were elaborated for the best visualization of the final classification of the classes considered in each scenario.

7.1 Binary Classification

The number of cleanware and malware present in the MC-dataset-binary is 11480. The randomly split of these samples was done as follows: 6888 samples for the training set and 2296 samples for the test set and validation set. A mini-batch of 14 was used and each network was run 20 times to obtain the results described in this section, not including the intermediate experiments, with other parameters and reduced number of samples.

The stopping point chosen was the highest validation accuracy value and this occurred with network training snapshots. Thus, the networks were executed more than once for each of the 20 considered times. All tests took place up to the 30th training iteration for a better visualization of the graphics and because all the networks had overfitting before that, following what was discussed in [48]. We can observe these training graphics for the accuracy and loss in Figure 7.1 and Figure 7.2. The comparative charts of true positive rate (TPR), false positive rate (FPR) and false negative rate (FNR), can be seen in Figure 7.3, Figure 7.4 and Figure 7.5, respectively. Finally, the confusion matrices of each network can also be seen in Table 7.1, Table 7.2, and Table 7.3.

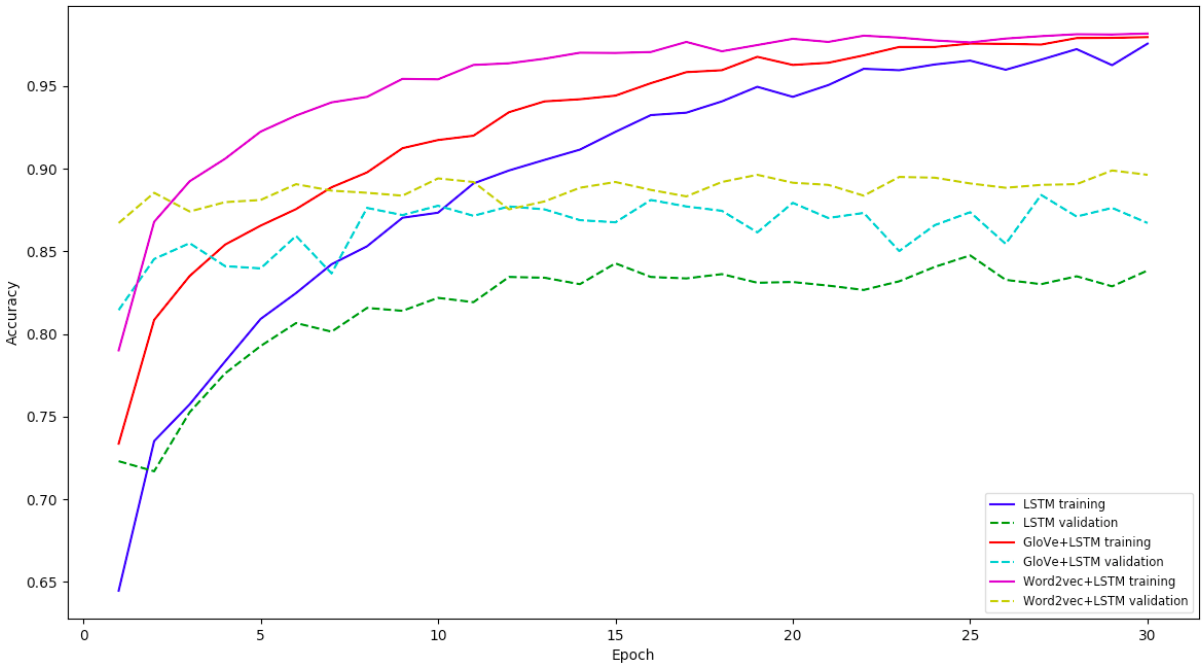


Figure 7.1: Training epoch & validation accuracy for the binary classification.

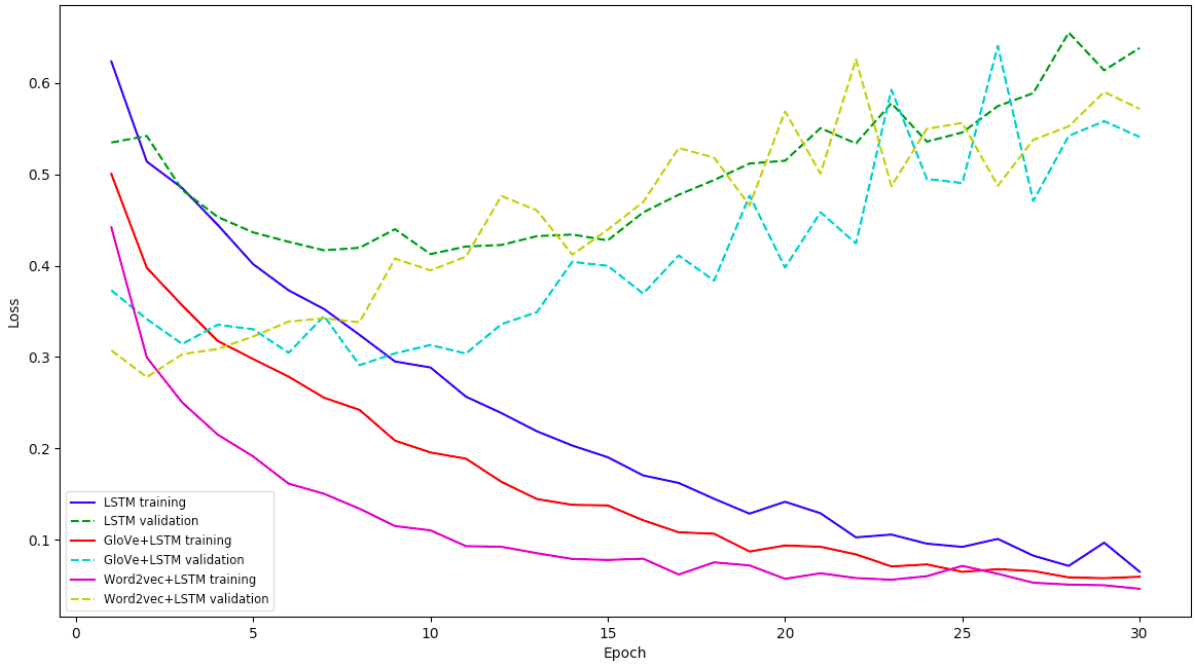


Figure 7.2: Training epoch & validation loss for the binary classification.

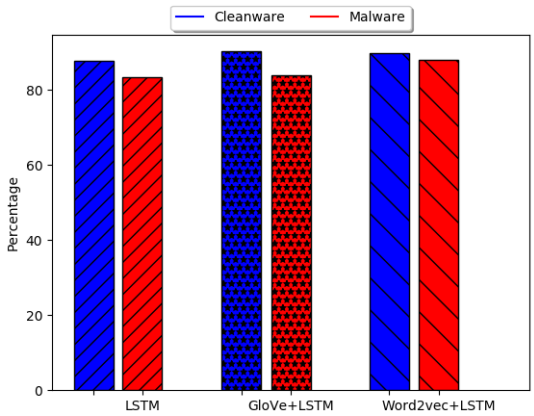


Figure 7.3: TPR for the binary classification.

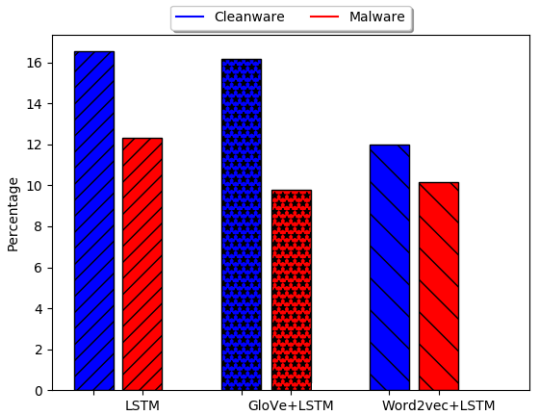


Figure 7.4: FPR for the binary classification.

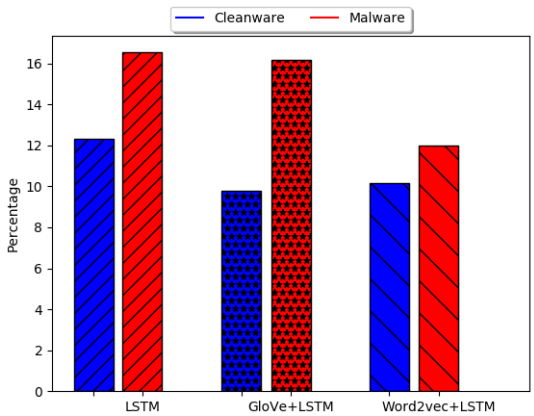


Figure 7.5: FNR for the binary classification.

Table 7.1: LSTM confusion matrix for the binary classification.

	cleanware	malware
cleanware	969	197
malware	136	994

Table 7.2: GloVe+LSTM confusion matrix for the binary classification.

	cleanware	malware
cleanware	952	201
malware	103	1040

Table 7.3: Word2vec+LSTM confusion matrix for the binary classification.

	cleanware	malware
cleanware	1028	138
malware	116	1014

7.2 Multiclass Classification

The number of backdoors, cleanwares, rootkits, trojans, viruses and worms present in the MC-dataset-multiclass is 19740. The randomly split of these samples was done as follows: 11844 samples for the training set and 3948 samples for the test set and validation set. A mini-batch of 14 was used and each network was executed 20 times too. It is important to note that in addition to the stopping point by the validation accuracy, the training with the highest accuracy was adopted. Therefore, there was no type of average between the 20 considered times.

As the binary classification, all tests took place up to the 30th training iteration too. We can observe these graphics for the accuracy and loss in Figure 7.6 and Figure 7.7. The comparative charts of TPR, FPR, and FNR, can be seen in Figure 7.8, Figure 7.9 and Figure 7.10, respectively. Lastly, the confusion matrices of each network can also be seen in Table 7.4, Table 7.5, and Table 7.6.

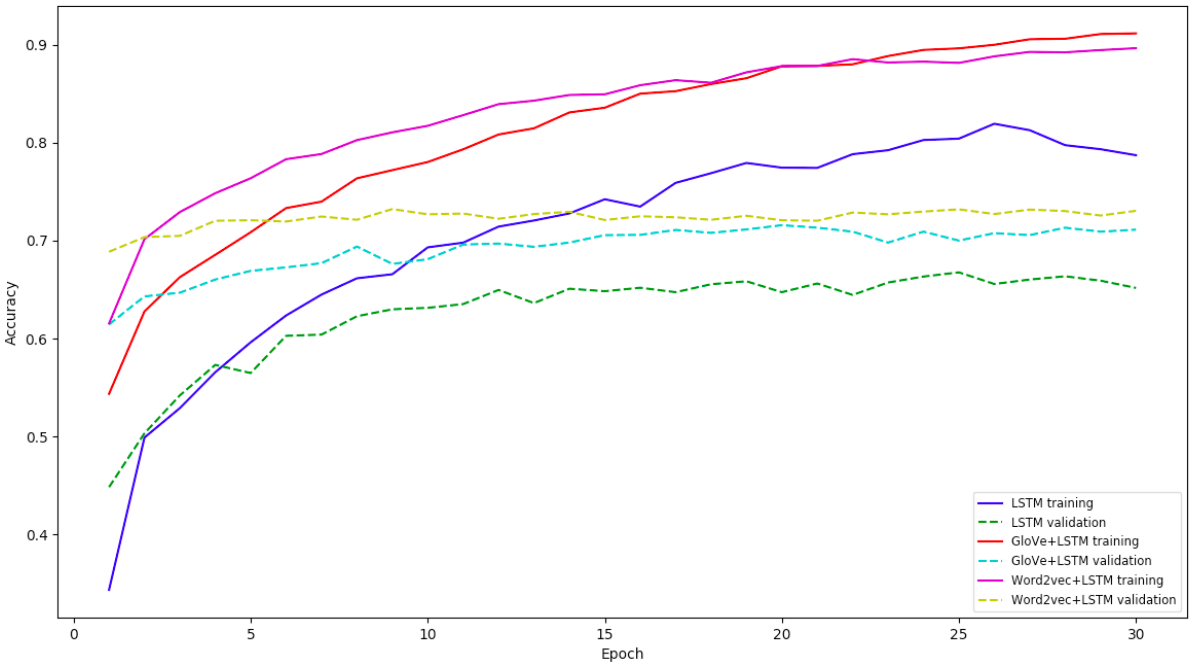


Figure 7.6: Training epoch & validation accuracy for the multiclass classification.

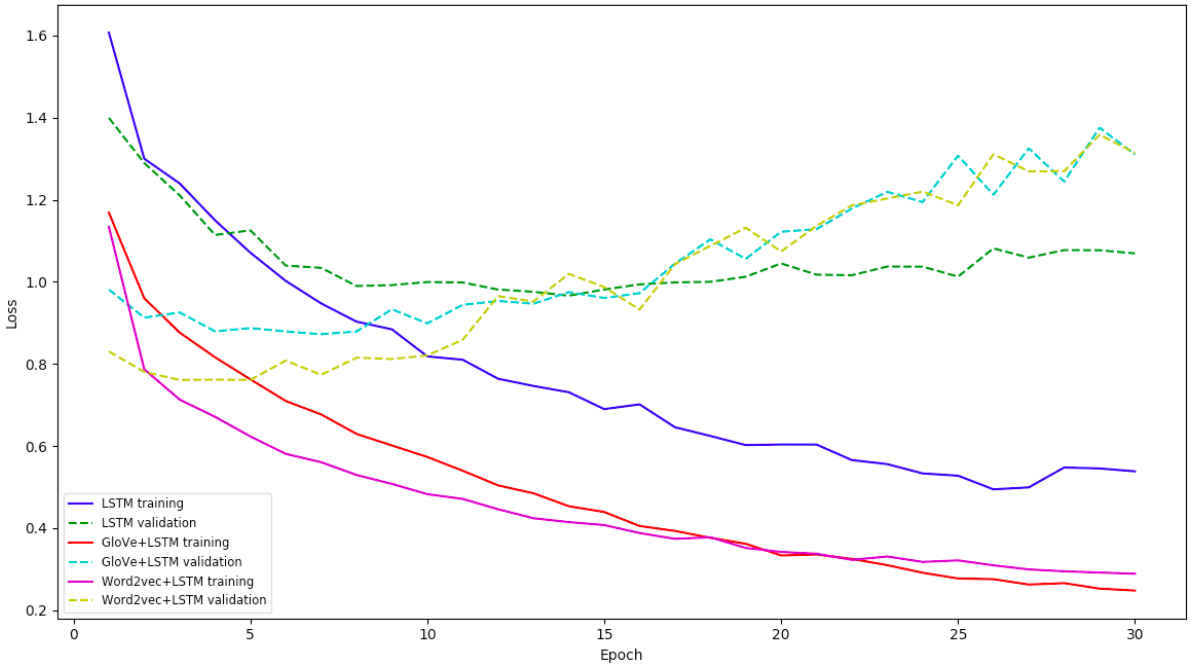


Figure 7.7: Training epoch & validation loss for the multiclass classification.

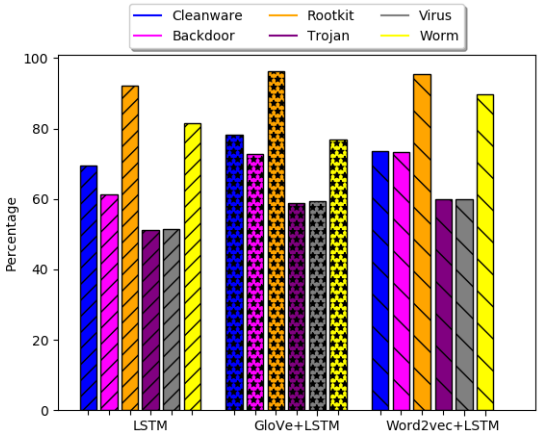


Figure 7.8: TPR for the multiclass classification.

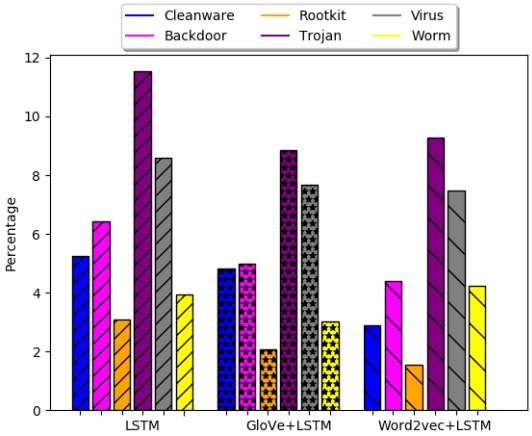


Figure 7.9: FPR for the multiclass classification.

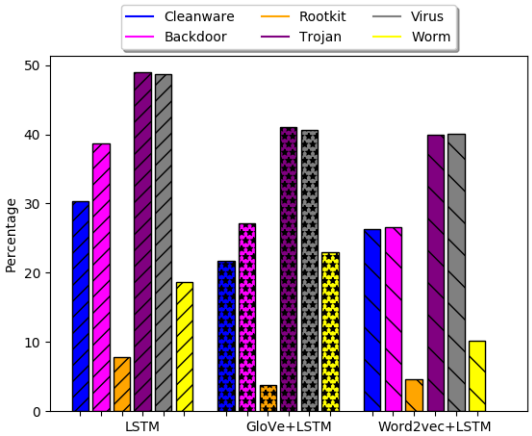


Figure 7.10: FNR for the multiclass classification.

Table 7.4: LSTM confusion matrix for the multiclass classification.

	backdoor	cleanware	rootkit	trojan	virus	worm
backdoor	422	28	8	82	61	31
cleanware	22	451	5	33	106	7
rootkit	26	20	578	25	23	8
trojan	125	49	14	289	153	49
virus	56	87	16	87	386	29
worm	37	13	6	50	23	543

Table 7.5: GloVe+LSTM confusion matrix for the multiclass classification.

	backdoor	cleanware	rootkit	trojan	virus	worm
backdoor	486	25	1	59	37	42
cleanware	12	465	3	33	98	16
rootkit	24	6	615	14	18	7
trojan	78	35	9	344	110	66
virus	46	56	10	96	427	40
worm	21	7	1	38	30	573

Table 7.6: Word2vec+LSTM confusion matrix for the multiclass classification.

	backdoor	cleanware	rootkit	trojan	virus	worm
backdoor	488	26	8	71	31	8
cleanware	27	531	1	15	50	0
rootkit	9	5	629	14	21	2
trojan	82	49	10	370	131	37
virus	38	100	7	85	418	13
worm	21	10	4	61	46	530

7.3 Discussion

In both classifications, binary and multiclass, all neural network models implemented had overfitting and it happened before in the neural networks with word embeddings algorithms, since the pre-trained weights of these algorithms accelerated the learning process. In addition, the GloVe+LSTM and word2vec+LSTM networks obtained better results than LSTM in both scenarios, that is, word embeddings algorithms also helped to improve training in a way that LSTM could not reach. In contrast, more time and computational resources were needed, especially in terms of memory.

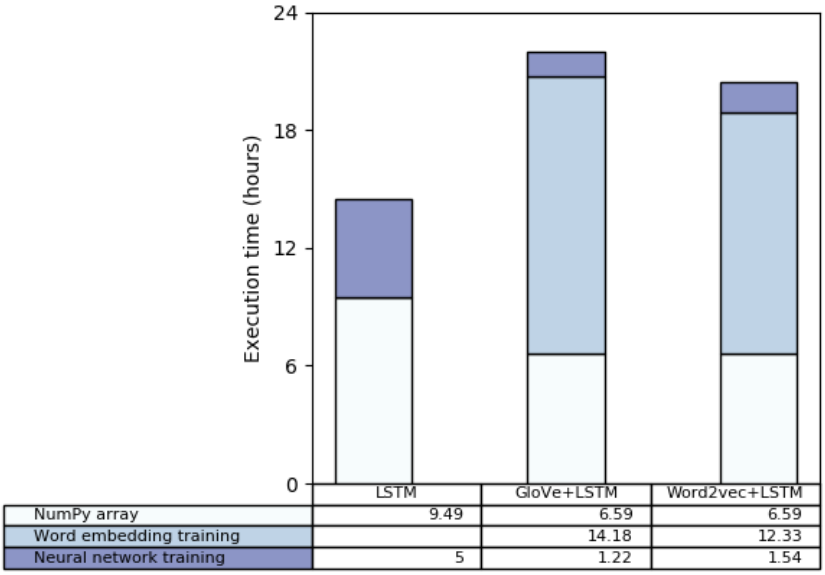
7.3.1 Execution Time and Parameters

Conversion process using our computer for hexdump and Unix time command, registered 22.31 minutes of elapsed real time for all samples of MC-dataset-binary and 38.40 minutes for MC-dataset-multiclass. Considering the NumPy array (n-dimensional array object) with numerical values corresponding to the hexadecimals that serve as input to the LSTM, including all the same samples, it took about 9.49 hours for the binary classification and 14.69 hours for the multiclass classification.

In the case of GloVe+LSTM and word2vec+LSTM for binary classification, it took 6.59 hours for the NumPy array, 12.33 hours for the word2vec training with 1000 iterations and 14.18 hours for the GloVe training with the same number of iterations. There was no overfitting in the word embeddings algorithms and more iterations also did not improve the results in the tests. We also performed the experiment with only word embeddings algorithms without the LSTM models, but the results were worse, with the accuracy reduced by about 10% for each algorithm.

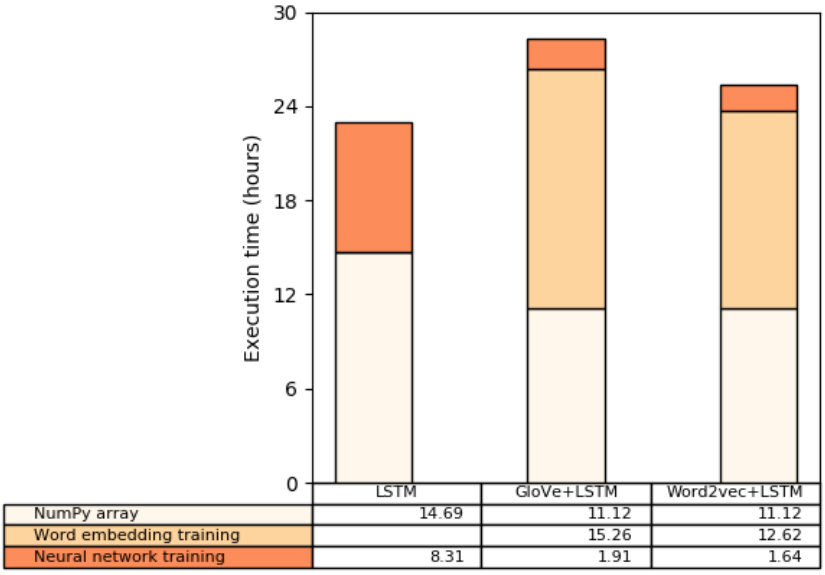
For the multiclass classification, GloVe+LSTM and word2vec+LSTM took 11.12 hours for NumPy array, 12.62 hours for word2vec training with 1000 iterations and 15.26 hours for GloVe training with the same number of iterations. For the training of the networks in the three models in the binary classification and with 30 epochs, it took 5 hours for the LSTM, 1.54 hours for the word2vec+LSTM and 1.22 for the GloVe+LSTM. It is important to remember that the LSTM was the only network that did not have outputs from the execution in the DGX, in which the only gain would be in relation to the time (tests with more complexity and other parameters were executed in the DGX for the LSTM model, but without better results).

The training of LSTM in the multiclass classification took 8.31 hours. Considering word2vec+LSTM, the training took 1.64 hours and for GloVe+LSTM, 1.91 hours. If a user wanted to insert a single sample to be classified with an already trained network, this would take a few seconds. The time of hexdump and NumPy array vary by sample size. In terms of test set, all networks did a fast classification of the samples and this would not be different with this single sample. Word2vec+LSTM, for example, took only 31 seconds to classify the 3948 multiclass samples. It should also be mentioned that in DGX there were other processes running together with the experiments of this work and this may have influenced in the execution times. The the execution time including each model, can be best seen in Figure 7.11 and Figure 7.12 for binary and multiclass classification, respectively.



	LSTM	GloVe+LSTM	Word2vec+LSTM
NumPy array	9.49	6.59	6.59
Word embedding training		14.18	12.33
Neural network training	5	1.22	1.54

Figure 7.11: Execution time for binary classification.



	LSTM	GloVe+LSTM	Word2vec+LSTM
NumPy array	14.69	11.12	11.12
Word embedding training		15.26	12.62
Neural network training	8.31	1.91	1.64

Figure 7.12: Execution time for multiclass classification.

Adam is a widely used method of stochastic gradient-based optimization, which is standard in Keras and has been selected for this work. The binary and categorical cross entropy are standard loss functions in Keras too and are adopted in the neural network models for binary and multiclass classifications, respectively. The learning rate adopted

was 0.00001, a value considered low and was chosen as an attempt to reduce the chance of overfitting. Other parameter choices such as number of layers and neurons occurred through the various tests until the best possible accuracy was obtained.

The size of the binary vocabulary was 606854 and the multiclass vocabulary, 949520. If we considered that all words were different among the 128 chosen from each sample, we would have 1469440 words for the binary vocabulary and 2526720 for the multiclass vocabulary. Thus, approximately 41.30% of the words that would be possible were sufficient to compose the vocabulary of binary classification and 37.58%, the vocabulary of multiclass classification. There were words among the 128 most frequent in the samples that appeared only once. However, reducing from 128 to fewer words or even increasing, the results were worse, even changing other parameters of the networks.

7.3.2 Comparison of Scenarios

The best accuracy values were obtained with the word2vec+LSTM network, 88.94% for the binary classification and 75.13% for the multiclass classification. However, in addition to the accuracy, it is necessary to observe other important statistical measures in the two scenarios, such as FPR and FNR. In the malware classification, FPR is a relevant measure because of the risks, especially when we have malware that are classified as cleanware. Therefore, the reduction of FPR should be a priority. In this case, considering the binary scenario, GloVe+LSTM is slightly better, with an FPR of 9.76% versus 10.14% of word2vec+LSTM.

Compressing the six classes in only two classes (cleanware and malware), GloVe+LSTM also has the lowest FPR (21.72%) compared to LSTM (30.40%) and word2vec+LSTM (26.35%). In general, LSTM had the worst results and an accuracy of 85.50% for binary classification and 67.60% for multiclass classification. The GloVe+LSTM presented an accuracy of 86.76% for the binary classification and 73.71% for the multiclass classification.

Looking only at the multiclass malware scenario, we observed that the trojan class achieved the worst results overall for all neural network models implemented. Probably this was because the trojan was the type of malware with the most subtypes. The best results were 60.06% for TPR and 39.94% for FNR with word2vec+LSTM. The FPR was 8.86% with GloVe+LSTM. Afterwards, we had the virus as the second worst class in terms of classification results.

Although the classification with the trojan and virus classes did not reach the best TPR, FPR, and FNR values, other two had excellent results. The rootkit class had a TPR of 96.24%, FNR of 3.76% with GloVe+LSTM, and FPR 1.55% with word2vec+LSTM. The other class was worm, with a TPR of 89.83%, FNR of 10.17% with word2vec+LSTM, and FPR 3.03% with GloVe+LSTM. Summarizing, the implemented models have obtained good classification results for some classes, such as rootkit and worm, and bad results for others, such as trojan and virus. In the binary scenario, word2vec+LSTM was able to classify the cleanwares better and GloVe+LSTM obtained a lower FPR in relation to malware misclassification.

7.3.3 Statistical Results

The Wilcoxon or Mann-Whitney [64] test was chosen because it is nonparametric and appropriate for data from repeated-measures with two conditions. The test was performed for each two of the three models of neural networks for the binary and multiclass scenarios. An alternative hypothesis $H_a : m > m_0$ was established ("*There is an increase in the performance of the classifiers in terms of accuracy?*"), in which m and m_0 correspond to the higher accuracy values obtained ($N = 10$) by the implemented models. The p-value was chosen as 2.5% of significance level.

Table 7.7: The Wilcoxon test for the neural network models considering the binary and multiclass scenarios.

$H_a : m > m_0$	p-value (binary)	p-value (multiclass)	H_a
GloVe+LSTM > LSTM	0.9986	> 0.9999	acceptance
Word2vec+LSTM > LSTM	> 0.9999	> 0.9999	acceptance
Word2vec+LSTM > GloVe+LSTM	0.9998	0.9955	acceptance

In Table 7.7, it is possible to view the p-values and results from the statistical test for each scenario. There are significant differences between the performance of the neural networks. Therefore, there is an increase in the classification accuracy considering the proposed models, especially in word2vec+LSTM. The medians in the binary and multiclass scenario, respectively, were 0.8324 and 0.6574 for LSTM, 0.8497 and 0.7027 for GloVe+LSTM, and were 0.8768 and 0.7234 for word2vec+LSTM.

Chapter 8

Conclusion

This master's thesis studies the problem of malware classification in two distinct scenarios, trying to understand the language that differs non-malicious from malicious files. The language consisted of the hexadecimal codes of these files read as sentences. The MC-dataset was created specifically for this work, with file samples for the binary and multiclass scenarios of malware. The classification was done through neural networks, which tried to learn about the hexadecimal codes of these samples.

The malware classification is not a trivial task because at every moment, new threats are created with new techniques to make it difficult to be detected by the maximum number of possible automatic tools. In addition, there is also a difficulty in finding large datasets available. Thus, one of the goals was to produce a dataset and make it publicly available for other researches. This has been successfully made, but the size of the dataset is still smaller than in some other works. However, if we consider the number of cleanware samples present, the MC-dataset is the largest available public dataset until the date of publication of this work.

The LSTM was the type of neural network chosen in conjunction with word embeddings algorithms for malware classification. This choice occurred because of the good recent results in other works of the area with the LSTM. The word2vec and GloVe were chosen because they are the most used word embeddings algorithms and have consistent implementations. The experiments demonstrated that these algorithms with the LSTM can actually improve the classification results.

At the end of the experiments, it was possible to conclude that the approach used in this work for the multiclass classification, for instance, served to classify some classes better than others. It was also found that some statistical measures were better with

GloVe+LSTM and others with word2vec+LSTM. However, even using different architecture and parameter settings, overfitting was a characteristic present in all tests. Even so, the results and experiments were a success, enabling a discussion about the different models and introducing new ideas that can be implemented.

8.1 Limitations

Some limitations were present in this work. The utilization of DGX for the experiments of GloVe+LSTM and word2vec+LSTM was sometimes hampered by the processes of other users occurring in parallel and using a large part of the machine resources. The power outages also disrupted the tests, forcing new executions of the neural network experiments and taking time of this master's thesis completion.

The labeling of MC-dataset was restricted to the analysis coming from VirusTotal scanners. A multi-label dataset would be closer to the real malware scenario, but this is more difficult and hard-working to do. Finally, the number of samples was also limited to the number of available cleanwares that were obtained from the computers of other users and from freewares downloaded from the internet, to maintain the dataset balanced, even knowing that the ideal dataset would be composed of more cleanwares than malwares.

8.2 Future Work

There are several improvements that can be done. One of them would be to expand the MC-dataset with the creation of a multi-label dataset and increase the number of samples. Tests with varying amounts of samples, such as a small, medium and large dataset sizes, could be performed to evidence improvement through increasing data too. Other types of neural networks could also be considered and even hybrid networks, like CNN-LSTM.

Adding new features may also improve the classification results. In addition to static analysis with only the understanding from the hexadecimal codes of the files, there are other approaches that could be considered together, such as dynamic analysis with the extraction of API calls most commonly used by malware, for example.

Another improvement that could be made is regarding scalability. New approaches to neural networks include learning on the fly. This open challenge is called Online Deep Learning (ODL), a promising area that has as input sequential data in a stream form [50] and can enhance the execution time and results of networks for malware classification.

References

[1] Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy* (2016), ACM, pp. 183–194.

[2] Alvarez, J. E.; Bast, H. *A review of word embedding and document similarity algorithms applied to academic text.* Tese de Doutorado, University OF Freiburg, 2017.

[3] Andrade, E. O.; Viterbo, J.; Nader, C. V. Um levantamento do uso de aprendizado profundo em análise de sentimentos. In *14th National Meeting on Artificial and Computational Intelligence (ENIAC)* (2017), Brazilian Conference on Intelligent Systems, pp. 85–96.

[4] Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (2014), vol. 14, pp. 23–26.

[5] Athiwaratkun, B.; Stokes, J. W. Malware classification with lstm and gru language models and a character-level cnn. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on* (2017), IEEE, pp. 2482–2486.

[6] Azzouni, A.; Pujolle, G. A long short-term memory recurrent neural network framework for network traffic matrix prediction. *arXiv preprint arXiv:1705.05690* (2017).

[7] Bazrafshan, Z.; Hashemi, H.; Fard, S. M. H.; Hamzeh, A. A survey on heuristic malware detection techniques. In *Information and Knowledge Technology (IKT), 2013 5th Conference on* (2013), IEEE, pp. 113–120.

[8] Bebis, G.; Georgiopoulos, M. Feed-forward neural networks. *IEEE Potentials 13*, 4 (1994), 27–31.

[9] Bengio, Y.; Ducharme, R.; Vincent, P.; Jauvin, C. A neural probabilistic language model. *Journal of machine learning research 3*, Feb (2003), 1137–1155.

[10] Bilar, D. Opcodes as predictor for malware. *International Journal of Electronic Security and Digital Forensics 1*, 2 (2007), 156–168.

[11] Chen, Z.; Roussopoulos, M.; Liang, Z.; Zhang, Y.; Chen, Z.; Delis, A. Malware characteristics and threats on the internet ecosystem. *Journal of Systems and Software 85*, 7 (2012), 1650–1672.

[12] Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[13] Dahl, G. E.; Stokes, J. W.; Deng, L.; Yu, D. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (2013), IEEE, pp. 3422–3426.

[14] Downer, K.; Bhattacharya, M. Byod security: A new business challenge. In *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on* (2015), IEEE, pp. 1128–1133.

[15] Freund, Y.; Schapire, R. E., et al. Experiments with a new boosting algorithm. In *Icml* (1996), vol. 96, Citeseer, pp. 148–156.

[16] Gardiner, J.; Nagaraja, S. On the security of machine learning in malware c&c detection: A survey. *ACM Computing Surveys (CSUR) 49*, 3 (2016), 59.

[17] Gennari, J.; French, D. Defining malware families based on analyst insights. In *Technologies for Homeland Security (HST), 2011 IEEE International Conference on* (2011), IEEE, pp. 396–401.

[18] Gibert, D. *Convolutional neural networks for malware classification*. Tese de Doutorado, MS Thesis, Dept. of Computer Science, UPC, 2016.

[19] Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), pp. 315–323.

[20] Gonzalez, L. E.; Vazquez, R. A. Malware classification using euclidean distance and artificial neural networks. In *Artificial Intelligence (MICAI), 2013 12th Mexican International Conference on* (2013), IEEE, pp. 103–108.

[21] Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[22] Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security* (2017), Springer, pp. 62–79.

[23] Hassan, S. S.; Bibon, S. D.; Hossain, M. S.; Atiquzzaman, M. Security threats in bluetooth technology. *Computers & Security 74* (2018), 308–322.

[24] He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1026–1034.

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

[26] Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[27] Hochreiter, S.; Schmidhuber, J. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems* (1997), pp. 473–479.

[28] Jaeger, H.; Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science 304*, 5667 (2004), 78–80.

[29] Johnson, N. F.; Jajodia, S. Exploring steganography: Seeing the unseen. *Computer 31*, 2 (1998).

[30] Kephart, J. O.; Sorkin, G. B.; Arnold, W. C.; Chess, D. M.; Tesauro, G. J.; White, S. R.; Watson, T. Biologically inspired defenses against computer viruses. In *IJCAI (1)* (1995), pp. 985–996.

[31] Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[32] Kolter, J. Z.; Maloof, M. A. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), ACM, pp. 470–478.

[33] LeCun, Y. A.; Bottou, L.; Orr, G. B.; Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[34] Lo, C. T. D.; Pablo, O.; Carlos, C. M. Towards an effective and efficient malware detection system. In *Big Data (Big Data), 2016 IEEE International Conference on* (2016), IEEE, pp. 3648–3655.

[35] Lowry, P. B.; Posey, C.; Bennett, R. B. J.; Roberts, T. L. Leveraging fairness and reactance theories to deter reactive computer abuse following enhanced organisational information security policies: An empirical study of the influence of counterfactual reasoning and organisational trust. *Information Systems Journal 25*, 3 (2015), 193–273.

[36] Makandar, A.; Patrot, A. Malware analysis and classification using artificial neural network. In *Trends in Automation, Communications and Computing Technology (I-TACT-15), 2015 International Conference on* (2015), IEEE, pp. 1–6.

[37] Mansfield-Devine, S. Extreme prejudice: securing networks by treating all data as a threat. *Computer Fraud & Security 2018*, 6 (2018), 16–20.

[38] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 4 (1943), 115–133.

[39] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[40] Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.

[41] Moser, A.; Kruegel, C.; Kirda, E. Exploring multiple execution paths for malware analysis. In *Security and Privacy, 2007. SP'07. IEEE Symposium on* (2007), IEEE, pp. 231–245.

[42] Murray, W. H. The application of epidemiology to computer viruses. *Computers & Security 7*, 2 (1988), 139–145.

[43] Nowlan, S. J.; Hinton, G. E. Simplifying neural networks by soft weight-sharing. *Neural computation 4*, 4 (1992), 473–493.

[44] Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; Swami, A. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on* (2016), IEEE, pp. 372–387.

[45] Pennington, J.; Socher, R.; Manning, C. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), pp. 1532–1543.

[46] Popov, I. Malware detection using machine learning based on word2vec embeddings of machine code instructions. In *Data Science and Engineering (SSDSE), 2017 Siberian Symposium on* (2017), IEEE, pp. 1–4.

[47] Rad, B. B.; Masrom, M.; Ibrahim, S. Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security 12*, 8 (2012), 74–83.

[48] Raff, E.; Zak, R.; Cox, R.; Sylvester, J.; Yacci, P.; Ward, R.; Tracy, A.; McLean, M.; Nicholas, C. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques 14*, 1 (2018), 1–20.

[49] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65*, 6 (1958), 386.

[50] Sahoo, D.; Pham, Q.; Lu, J.; Hoi, S. C. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705* (2017).

[51] Saxe, J.; Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on* (2015), IEEE, pp. 11–20.

[52] Schultz, M. G.; Eskin, E.; Zadok, F.; Stolfo, S. J. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on* (2001), IEEE, pp. 38–49.

[53] Sen, S.; Aydogan, E.; Aysan, A. I. Coevolution of mobile malware and anti-malware. *IEEE Transactions on Information Forensics and Security 13*, 10 (2018), 2563–2574.

[54] Sim, G. Defending against the malware flood. *Network Security 2018*, 5 (2018), 12–13.

[55] Spafford, E. H. Computer viruses–a form of artificial life?

[56] Spiekermann, S.; Acquisti, A.; Böhme, R.; Hui, K.-L. The challenges of personal data markets and privacy. *Electronic Markets 25*, 2 (2015), 161–167.

[57] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[58] Stopel, D.; Moskovitch, R.; Boger, Z.; Shahar, Y.; Elovici, Y. Using artificial neural networks to detect unknown computer worms. *Neural Computing and Applications 18*, 7 (2009), 663–674.

[59] Sundermeyer, M.; Schlüter, R.; Ney, H. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association* (2012).

[60] Turian, J.; Ratinov, L.; Bengio, Y. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics* (2010), Association for Computational Linguistics, pp. 384–394.

[61] Uppal, D.; Mehra, V.; Verma, V. Basic survey on malware analysis, tools and techniques. *International Journal on Computational Sciences & Applications (IJCSA) Vol 4* (2014), 103–111.

[62] Walenstein, A.; Mathur, R.; Chouchane, M. R.; Lakhotia, A. The design space of metamorphic malware. In *2nd International Conference on i-Warfare and Security (ICIW 2007). 2nd International Conference on i-Warfare and Security (ICIW 2007)(2007)* (2007), pp. 241–248.

[63] Wang, W.; Zhu, M.; Zeng, X.; Ye, X.; Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. In *Information Networking (ICOIN), 2017 International Conference on* (2017), IEEE, pp. 712–717.

[64] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics bulletin 1*, 6 (1945), 80–83.

[65] Wu, X.; Zhu, X.; Wu, G.-Q.; Ding, W. Data mining with big data. *IEEE transactions on knowledge and data engineering 26*, 1 (2014), 97–107.

[66] Xu, B.; Wang, N.; Chen, T.; Li, M. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853* (2015).

[67] Yanling, Z.; Bimin, D.; Zhanrong, W. Analysis and study of perceptron to solve xor problem. In *Autonomous Decentralized System, 2002. The 2nd International Workshop on* (2002), IEEE, pp. 168–173.

[68] You, I.; Yim, K. Malware obfuscation techniques: A brief survey. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on* (2010), IEEE, pp. 297–300.