AMERICAN UNIVERSITY OF BEIRUT

Malware Detection and Classification Using Recurrent Neural Networks

by
Wael Al Rahal Al Orabi

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science
to the Department of Computer Science
of the Faculty of Arts and Sciences
at the American University of Beirut

Beirut, Lebanon
February 2019

AMERICAN UNIVERSITY OF BEIRUT

Malware Detection and Classification Using Recurrent Neural Networks

by

Wael Mohammad Al Rahal Al Orabi

Approved by: Haidar Safa

Dr. Haidar Safa, Full Professor — Advisor
Department of Computer Science

Dr. Wassim El Hajj, Associate Professor, Chairperson — Committee Member
Department of Computer Science

Dr. Mohamed Nassar, Assistant Professor — Committee Member
Department of Computer Science

Date of thesis defense: June 21, 2018

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Al Rahal Al Orabi   Wael   Mohammad
Last   First   Middle

☑ Master's Thesis   ○ Master's Project   ○ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☑ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ✓ year from the date of submission of my thesis, dissertation or project.**
Two ___ years from the date of submission of my thesis , dissertation or project.
Three ___ years from the date of submission of my thesis , dissertation or project.

Signature   Date

Acknowledgements

I would like to thank my thesis advisor Prof. Haidar Safa for his excellent guidance, patience, persistent and unlimited support. He was always available whenever any help is needed. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

My sincere thanks goes to my committee members Prof. Wassim El Hajj, and Prof. Mohamed El Baker Nassar. Their help and support was very beneficial in order to complete my thesis.

Another special thanks must go to Prof. Wassim El Hajj for being a great academic advisor throughout my journey at American University of Beirut. He was not only my academic advisor, committee member in my thesis, but also the one who is always to help, share thoughts, and give advise in all situations.

I would like to thank my friends Omar Kayali and Chadi Helwe for being there throughout my master years. We are one working group aiming to provide help and support to each other in any academic and non-academic subject.

Finally, I must express my very profound gratitude to my parents and especially my brother Adel Al Orabi for providing me with unfailing support and continuous encouragement throughout my years of study and through the tough time of my thesis. This accomplishment would not have been possible without them.

One word to my father and mother: I owe it all for you. Thank You & Love You.

An Abstract of the Thesis of

Wael Al Rahal Al Orabi for Master of Computer Science
Major: Computer Science

Title: Malware Detection and Classification Using Recurrent Neural Networks

Malware detection and classification is becoming one of the hottest era of research due to the fact that the number of malware is increasing nowadays which raises many questions and concerns related to security. For example, recently ransomware is a malware that targeted huge companies and infected many computing systems. Over the years, researchers have focused on automating the process of detecting malware in computing systems by designing approaches that rely on data mining and machine learning methodologies. These approaches were proved to be efficient by achieving great results in terms of accuracy. On the other hand, one of their limitations is that they still being considered as shallow models compared to deep learning. Deep learning technologies rely on more complex computational architecture which needs more data. As the computational complexity of the model increases, a larger dataset is required to train, build, and validate it. To remedy the limitations of those shallow approaches, in this thesis we propose an automated solution for malware detection and classification in binary executable sequences based on deep learning. We define a new malware language which is designed with the concept of a vocabulary, documents, and words. Each malware assembly instance is a document, and each assembly action in the malware document is a word. Consequently, a malware vocabulary is defined as a set of malware documents. This language design is used to extract the features from executable binary sequences. We develop a hybrid classification model that consists of two main component: feature extraction and classification component. The feature extraction component is based on the predefined malware language. We have different architectures for the classification component such as Long Short Term Memory (LSTM), Gated Recurrent Units (GRU), 1 dimensional Convolutional Neural Networks (1DCNN), and a hybrid architecture that consists of 1D-CNN and LSTM. We validated our models empirically by running a set of experiments on Microsoft Malware dataset provided on Kaggle.

The hybrid architecture (1D CNN LSTM) reached the highest accuracy value of
99.31

Contents

Acknowledgements **v**

Abstract **vi**

1 Introduction **1**
1.1 Introduction to Machine and Deep Learning 1
1.2 Motivation 3
1.3 Problem Statement 3
1.4 Objectives and Contributions 5
1.5 Thesis Organization 6

2 Background and Basic Concepts **8**
2.1 Machine Learning 8
2.1.1 Definition and Overview 8
2.1.2 General Approach of Machine Learning 9
2.1.3 Types of Machine Learning 10
2.1.4 Machine Learning Algorithms 13
2.1.5 Neural Netowrks 14
2.2 Recurrent Neural Networks 15
2.2.1 RNN Mathematical Foundations 17
2.2.2 Variations of Recursive Neural Networks Models 20
2.2.3 RNN Mathematical Foundation 22
2.2.4 Back Propagation Through Time Algorithm 28
2.2.5 Recurrent Neural Networks Limitation 29
2.2.6 Long Short Term Memory (LSTM) 31
2.2.7 Long Short Term Memory Mathematical Foundations . . . 32

3 Literature Review **36**
3.1 Static Analysis Approaches 36
3.2 Dynamic Analysis Approaches 39
3.3 Deep Learning Approaches 41
3.4 Word Embedding 46

4 Proposed Approach **48**
4.1 High Level Architecture . . . 48
4.2 Word Embedding . . . 49
4.3 Feature Extraction Component . . . 50
4.4 Classification Component . . . 53

5 Performance Evaluation **58**
5.1 Dataset Description . . . 58
5.2 Experimental Results . . . 59
5.2.1 Parameters and Metrics . . . 59
5.2.2 Machine Learning Models . . . 61
5.2.3 Benchmark of Classification Models . . . 63
5.2.4 Parameters Validation . . . 67
5.2.5 Analysis . . . 74

6 Conclusions **80**

A Abbreviations **82**

B Mathematical Notations **84**

List of Figures

2.1 Folded and Unfolded Representation of RNN 16
2.2 Hyperbolic Tangent Activation Function 19
2.3 Softmax Activation Function 20
2.4 Derivative of Hyperbolic Tangent Function 30

3.1 Word Embedding Example 47

4.1 Model High Level Architecture 49
4.2 Malware Language Design 51
4.3 Representation of CNN-LSTM Architecture for Malware 55

5.1 Accuracy of Different Machine Learning Classifiers 62
5.2 Variation of Accuracy of Classification Models versus Number of Layers 63
5.3 Variation of Accuracy of Classification Models versus Dataset Size 64
5.4 Variation of Accuracy of Classification Models versus Dropout Percentage 65
5.5 Variation of Accuracy of Classification Models versus Sequence Length 66
5.6 Variation of Accuracy of Classification Models versus Number of Filters 66
5.7 Variation of Accuracy of Classification Models versus Kernel Size 67
5.8 Variation of Loss versus dataset size for multiple number of LSTM layers 68
5.9 Variation of Loss versus dataset size for multiple number of LSTM layers 68
5.10 Variation of Loss versus dataset size for different word embedding algorithms 69
5.11 Variation of Loss versus dataset size for different word embedding algorithms 69
5.12 Variation of Loss versus dataset size for multiple dropout percentage values 70
5.13 Variation of Loss versus dataset size for multiple dropout percentage values 70

5.14 Variation of Loss versus dataset size for multiple CNN filters . . . 72
5.15 Variation of Loss versus dataset size for multiple CNN filters . . . 72
5.16 Variation of Loss versus dataset size for multiple kernel size values 73
5.17 Variation of Loss versus sequence length for multiple kernel size values 73
5.18 Summary of Accuracy Values of Different Architectures of our Model 75
5.19 Learning Curve of Hybrid CNN LSTM Classifier 79

List of Tables

2.1 Supervised vs Unsupervised Machine Learning Algorithms 14
2.2 Comparison between different RNN Models 22

3.1 Static vs Dynamic Malware Detection and Classification Approaches 42
3.2 Deep Learning Malware Detection and Classification Approaches . 46

5.1 Summary of best values for different hyper-parameters of our classification model 73
5.2 Summary of Accuracy Values of Different Architectures of our Model 75
5.3 Execution time versus batch size 77

Chapter 1

Introduction

This chapter introduces machine and deep learning and motivation for addressing malware detection and classification research area. Also, it discusses the problem statement and our main objectives and contributions. Finally, it presents the organization of the thesis.

1.1 Introduction to Machine and Deep Learning

Machine Learning is the process of learning from data. A machine learning model learns the changes in a system that performs tasks associated with artificial intelligence. There are a lot of tasks that could be learned such as diagnosis, classification, planning, robot control, text and voice recognition, prediction of financial values and studies. These changes might enhance an already existing model or might be used to develop a new machine learning based model. One important class of machine learning is classification. In classification problems objects are classified into their correct classes. For example, detecting if an input file is a malware or not is considered a binary classification. On the other hand,

classifying a pre-known malware file into its correct class of malware is a multi-class classification problem where the set of malware families is greater than two.

Deep Learning is considered as a subset of machine learning in artificial intelligence (AI) that has a different and special architecture. Deep learning is composed of a set of neural networks that are capable of learning from unstructured and unlabeled data. It is also known as Deep Neural Learning or Deep Neural Network. High level abstractions in deep learning is based on the traditional algorithms used in machine learning using multiple nonlinear transformations on the level of the hidden layers. These algorithms are used to model high level abstractions in data through the use of architectures composed from multiple nonlinear transformations aiming for learning the representations of data.

Deep learning is mainly specified in building and training a set of neural networks which are considered very powerful in decision making problems. Not all the algorithms in the Learning field are classified as deep, but they should contain a series of nonlinearities or non-linear transformations before leading to the desired output. As an advantage of deep learning is that it removes the manual identification of features in dataset and relies on the training process to identify and discover useful patterns in the input samples. This will make the training of neural networks easier and faster and may lead to better results that advances the field of AI.

1.2 Motivation

In the internet age, the world is becoming totally connected to each other. As the internet is evolving and the computers are being the essential component of our daily life, the number of malware is increasing tremendously. This phenomenon raised many concerns to security issues in corporate and business and also at a personal level. Consequently, malware detection became a very important and essential area of research to be investigated. Researchers used different technologies to deal with malware detection and classification problem such as signature based, heuristic based analysis, data mining, machine learning, and deep learning.

Originally, researchers designed many solutions that rely on signature based and heuristic based analysis methodologies in order to investigate and solve malware detection and classification problem. The former methodology is based on recognizing patterns across the binary representation of malware executable while the latter is based on detecting recognizable patterns from malware execution traces. One limitation of those approaches is that they are capable of detecting a malware only after the system is infected by the malware and it is available for analysis.

1.3 Problem Statement

Over the years, many approaches were designed and implemented to solve malware detection and classification problem scenarios based on more complex technologies such as data mining and machine learning [1] [2]. In general, these

technologies share the same high level architecture of the model. The architecture is divided into two modules: feature extraction module and a classification module. Feature extraction module can be achieved using either static analysis methods such as [1] [3], or dynamic analysis methods such as [4][5]. The output of the feature extraction component (features mainly) is fed into the classification component. The latter is designed based on data mining, machine learning, or deep learning set of algorithms in order to process these features and classify the malware under investigation into its correct class.

Data mining and machine learning approaches were and still being used heavily in automating the process of detecting and classifying malware. These approaches were proved to be efficient by achieving great results in terms of accuracy [6][7][8]. On the other hand, one of their limitations is that they still being considered as shallow models compared to deep learning. This is due to the fact that approaches that rely on data mining and machine learning can be avoided using several techniques such as code obfuscation [9]. Deep Learning is a new technology that is based on machine learning set of algorithms that is capable of producing better and more accurate results in malware detection and classification contexts due to its complex structure. It is capable of processing data using more complicated paradigms.

1.4 Objectives and Contributions

In this thesis, we aim to develop a new deep learning based model for malware detection and classification that can outperform existing approaches in the literature. Our developed model is based on convolutional neural network and a special type of recurrent neural network defined as Long Short Term Mermory (LSTM). We define a new language for malware. We design a malware with a concept of a vocabulary, documents, and words. A malware vocabulary is defined as a set of malware documents. A malware document is defined as an assembly instance, and a malware word is an action in the malware document. We extract our features using a python parser based on our new malware language. Finally, we design, implement, and present our new hybrid CNN-LSTM based classification component along with a deep feed forward fully connected neural network.

Our contributions that are presented in this thesis can be summarized as follows:

- We built a new language design for malware detection and classification for executable binary sequences.
- We developed a python parser to extract the actions that a malware intends to perform in a running environment which are known as features to our classification component.
- We designed and implemented a new model for malware detection and

classification based on convolutional neural networks and a special type of recurrent neural networks known as LSTM and a deep feed forward fully connected neural network.

- The proposed approach is capable of detecting and classifying malware by viewing the malware as a sequence of assembly instructions. By studying and analyzing those instructions, the model is capable of knowing the intention of detecting and classifying the malware into its correct class.

- The proposed approach is capable of detecting the malware file during compilation rather than running time. In other words, the malware is detected before it infects the system.

1.5 Thesis Organization

The thesis is organized as follows: Chapter 2 presents basic concepts and background related to machine, deep learning, recurrent neural networks along with their mathematical formalization. Chapter 3 surveys existing solutions related to malware detection and classification in the literature. In this chapter, three approaches are discussed mainly static analysis approaches, dynamic analysis approaches, and deep learning based approaches. Chapter 4 presents our proposed approach. First, we describe the high level architecture of our solution. Second, we give an overview about word embedding concept in deep learning. Third, we present the details of our language based feature extraction component. Finally, we present our hybrid convolutional neural network (CNN) - Long Short Term

Memory (LSTM) based classification model along with the our developed algorithm. Chapter 5 presents our empirical evaluation process of our LSTM based classification model in comparison with other investigated models for malware detection and classification. Chapter 6 concludes this thesis and presents future work.

Chapter 2

Background and Basic Concepts

This chapter presents the background material, and some basic concepts needed for the rest of the thesis. It mainly discusses machine and deep learning technologies especially neural networks, recurrent neural networks, and their limitations. Also, it gives a mathematical overview of the aforementioned topics.

2.1 Machine Learning

This section discusses some key concepts of machine learning along with its set of algorithms especially neural networks. Also, it gives a basic intuition of the mathematical formulation related to machine learning.

2.1.1 Definition and Overview

Machine learning is defined as the process of learning patterns from data. It is considered as a subset of artificial intelligence. It brings the computer science field and mathematical probability to enhance the predictive power of the desired model. The idea behind machine learning is to train a model on some source of dataset known as training data and test it on a new dataset which is referred

to as testing data. The accuracy of the model can be generated based on some mathematical formulas that can be applied. The value of the accuracy and the prediction capabilities of the model change proportionally. Higher accuracy value leads to a higher and better predictive model and vice versa.

2.1.2 General Approach of Machine Learning

This section discusses the algorithm that should be applied in almost all machine learning problems. Mainly, it consists of the below six steps. The last two steps are executed multiple times and in a loop manner aiming for best accuracy values.

1. Define the problem: In this step, many questions should be answered prior to starting the technical work on the problem. These questions address relevant points such as: what will be predicted? what is the format of the output? what is the available data? what is the format of the input? and is there background knowledge needed to do this well?

2. Gather data: Once the problem is well defined, dataset should be gathered. Many preprocessing techniques can be applied on the dataset in order to have only the needed information and in the right format.

3. Decide on a machine learning algorithm: This is a very important and crucial step in order not to use the wrong algorithm. In this step, many questions arises such as what is the type of the problem (supervised or unsupervised, detailed below)? how simple is the algorithm to understand? how difficult is the algorithm to implement? and can it handle the existing

amount of data?

4. Split data into training and testing: The dataset should be divided into training dataset and testing dataset. This is essential since the model should be tested on dataset that is totally different from the dataset that was trained on. One technique to split the data is to use 60% as training data, 20% as testing data, and 20% as validation data.

5. Train: The model is trained on the training dataset multiple times until an acceptable performance level is reached. Based on the algorithm used, its hyper-parameters are twisted and modified aiming for a higher accuracy and consequently a better predictive model.

6. Test: In this step, the model is tested on the testing data and its performance is analyzed.

2.1.3 Types of Machine Learning

There are different types of machine learning classes: supervised, unsupervised, and semi-supervised.

Supervised Learning

Supervised machine learning is where there are the input variable X and its corresponding output Y and an algorithm is used to learn the mapping function between the input and the output. This is referred to as labeled data. The ultimate goal is to approximate the mapping function very good so that the

model is capable of predicting the right output for new input instances. It is called supervised since the learning process of the algorithm from the training dataset can be thought of as a teacher supervising the learning process. The correct output of a new instance variable is known, so the algorithm iteratively makes predictions on the training dataset which are corrected by the teacher. The learning process stops when the algorithm achieves an acceptable level of performance. Supervised machine learning problems can be divided into two classes, classification and regression.

1. Classification: The aim of such problems is to classify or predict the correct class of an input instance. The problem can be either a binary or multi class classification. In binary classification, there are two predefined output classes while in multi class classification, the set of predefined output classes contains three or more classes.

2. Regression: The aim of regression supervised machine learning problems is to predict the correct output of a new input instance where the output is real value.

Unsupervised Learning

Unsupervised machine learning is the process of learning complex patterns from the structure of distribution of the data where there exist the input variable X without its corresponding output class. The dataset is known as unlabeled data. This approach is called unsupervised since the learning process has no correct

answers and there is no teacher supervising it. Algorithms should discover and present interesting structure and patterns in the data alone. There are two types of unsupervised machine learning problems: clustering and association.

1. Clustering: The aim of such problems is to cluster data into groups. For example, in some business domains, customers might be grouped by their purchasing orders.

2. Association: The aim of such problems is to extract and identify rules that describe large portions of the dataset. For example, extracting a rule such that if a person is buying item X, then he/she will buy item Y.

Semi-supervised Learning

Semi-supervised learning lies between both supervised and unsupervised learning. In-semi supervised learning, only few of the input instances are labeled. Many real world problems lies in semi-supervised learning class. This is due to the fact that it is very expensive to label each input instance and on the other extreme it is very cheap to deal with unlabeled data. Unsupervised and supervised learning algorithms can be used in such problems. Unsupervised learning techniques can be used to discover and learn the structure of the input data variables, and supervised learning techniques can be used to make predictions about unlabeled data, feed that predication back to the supervised learning algorithm as training dataset and make prediction of new unseen data.

2.1.4 Machine Learning Algorithms

There is a set of machine learning algorithms such as Decision Trees [10], Naive Bayes Classification [11], Linear Regression [12], Logistic Regression [12], Ordinary Least Squares Regression [13], Support Vector Machines, Ensemble Methods [14], Clustering Algorithms [15], Principal Component Analysis [16], Singular Value Decomposition [14], Independent Component Analysis [16], Neural Networks [17], K-mean clustering [15], Random Forest [12], and their regularized and boosted versions [18]. Table 2.1 classifies the aforementioned algorithms into their correct class: supervised or unsupervised.

One thing to note is that neural network algorithm can be used in supervised and unsupervised applications. As discussed in section 2.1.3, it depends if the data is labeled or not. Neural networks are discussed in section 2.1.5.

Machine Learning Algorithm	Supervised	Unsupervised
Decision Trees [10]	X	
Naive Bayes [8]	X	
Linear Regression [31]	X	
Logistic Regression [31]	X	
Ordinary Least Squares Regression [9]	X	
Support Vector Machines [7]	X	
Ensemble Methods [20]	X	
K-mean clustering [20]		X
Principal Component Analysis [31]		X
Singular Value Decomposition [34]		X
Independent Component Analysis [41]		X
Neural Networks [12]	X	X
Random Forest [41]	X	
Apriori Algorithm [8]		X

Table 2.1: Supervised vs Unsupervised Machine Learning Algorithms

2.1.5 Neural Netowrks

Neural Networks can be defined as a set of algorithms that aim to identify patterns in a specific dataset using a set of processes that mimic the way a human brain operates. Neural networks have the ability to adapt changes in the input without changing the design of the output criteria. A neural network consists of

three layers: input layer, one or more hidden layers, and an output layer. The dimensions of these layers are changed based on the problem. The layers are interconnected such a connection can exist between any two nodes in the neural network. The input layer receives the input, and each hidden layer in the neural network is a mathematical function such as a linear regression that captures and recognizes information based on a predefined architecture. Each node feeds the signal generated by linear regression into an activation function that might be nonlinear. Finally, the output layer contains classifications or output signals to which input patterns should be mapped. Neural networks are being used in different application domains such as financial operations, trading, business analytics, product planning and maintenance. They are widely used in forecasting, risk assessment, and fraud detection problems. On the other hand, deep learning is defined as designing complex models based on neural networks such as recurrent neural networks which are presented in section 2.2.

2.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are defined as a set or group of neural network that are capable of processing sequential data. In other words, given an input vector $\boldsymbol{x(t)}$, an RNN processes this data to produce an output vector $\boldsymbol{o(t)}$. RNN has the advantage of passing information from one time step $\boldsymbol{t}$ to step $\boldsymbol{t+1}$.

Recursive neural networks are widely used in different application domains such as malware detection, machine transition, speech and handwriting recogni-

tion, images processing and description generation, image/video captioning, word prediction, and text translation [19].

RNN is different from multilayer neural networks in the sense that it takes advantage of one of the most important concepts in machine learning, which is known as parameter sharing. RNN shares parameters across different parts of the same model.

RNN can be modeled using two types of graphs: folded and unfolded. The unfolded version of an RNN shows the details of each time step of the sequence processing model while the recurrent version shows an abstraction of the model. Figure 2.1 depicts different representations of RNN [20].

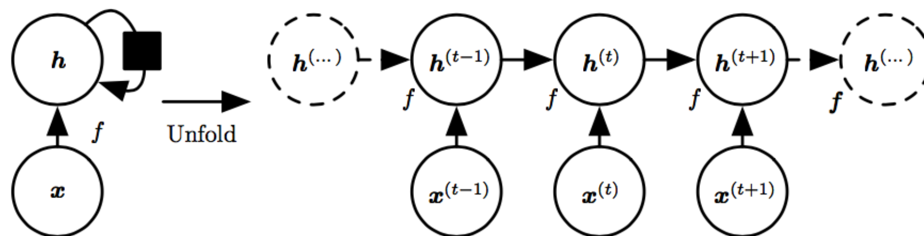

Figure 2.1: Folded and Unfolded Representation of RNN

The specifications of RNN can be summarized in the below three major points.

1. Parameter sharing: RNN shares parameters between different time steps of the model.

2. Model input size: The input size is the same during all the time steps of the sequence. This is due to the fact that the input is specified in terms of the transition from one state to another rather than the input training

dataset size.

3. RNN has the possibility of learning a unique transition function overall the model. This is pretty important since it reduces the cost of learning a new objective function at each time step of the sequence model.

4. From the previous points, we can infer that in some cases RNN will require less training data that other neural networks approaches.

2.2.1 RNN Mathematical Foundations

This section describes the mathematical formulas used by a standard recurrent neural network to calculate its output at each time step $\boldsymbol{t}$. This can be decomposed into two parts: feed forward propagation and back propagation through time. The latter is derived from back propagation generic algorithm with some modifications. A list of all mathematical notations used in this section is listed in Appendix B.

Feed Forward Propagation Formulas

The equations that are applied during the feed forward propagation algorithm. Equation 2.1 shows the $S^(t)$ which represents the hidden layer at time t.

$$s^{(t)} = tanh(b + Ws^{(t-1)} + Ux^{(t)}) \tag{2.1}$$

where b is a bias vector, W is the hidden layer to output layer matrix, U is the input layer to hidden layer matrix, $x^{(t)}$ is the input vector at time $\boldsymbol{t}$, and $s^{(t-1)}$ is the hidden layer at time ***t-1***.

$$o^{(t)} = C + Vs^{(t)} \tag{2.2}$$

where $o^{(t)}$ is given as the output vector of the recurrent neural network at time t, V is the hidden to hidden layers matrix, and C is a bias vector. Finally $\hat{y}^{(t)}$ which is given in Equation 2.3

$$\hat{y}^{(t)} = softmax(o^{(t)}) \tag{2.3}$$

Hyperbolic Tangent Activation Function

The hyperbolic tangent function is an old mathematical function. It is defined as the ratio between hyperbolic sine and hyperbolic cosine functions. The below function gives the mathematical intuition behind tanh function. It is a non linear function and used in deep learning to transform the output from a linear to non-linear one.

$$\begin{aligned} \tanh &= \frac{\sinh}{\cosh} \\ &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ &= \frac{e^{2x} - 1}{e^{2x} + 1} \\ &= \frac{1 - e^{-2x}}{1 + e^{-2x}} \end{aligned} \tag{2.4}$$

Figure 2.2 illustrates the hyperbolic tangent activation function [21].

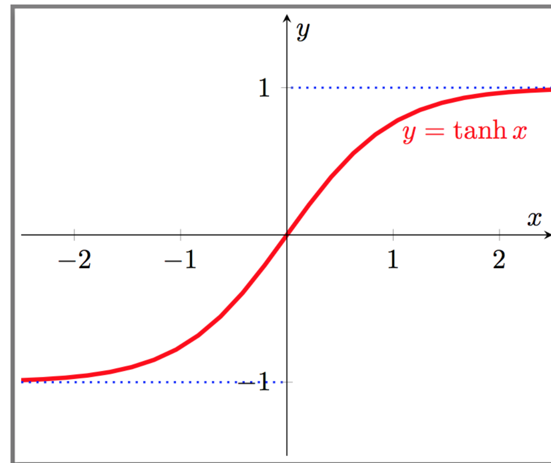

Figure 2.2: Hyperbolic Tangent Activation Function

Softmax Activation Function

The softmax function takes as input a vector of dimension N of arbitrary real values and gives another vector of the same dimension with real values in the range of (0,1) that sum up to 1. The actual per element formula given in Equation (2.5) as:

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \qquad \forall j \in 1 \therefore N \tag{2.5}$$

Figure 2.3 depicts the softmax activation function [22].

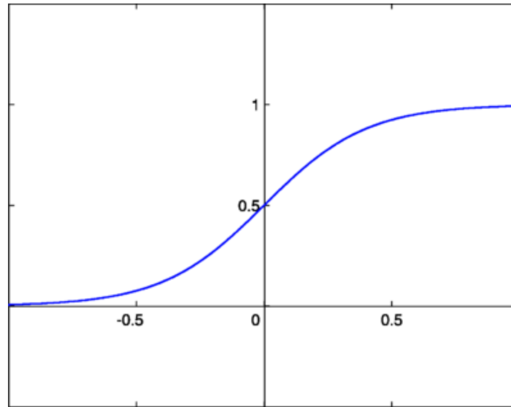

Figure 2.3: Softmax Activation Function

2.2.2 Variations of Recursive Neural Networks Models

There are some variations to the basic RNN which are detailed below.

1. Recurrent networks that produce an output at each time step and have recurrent connections between hidden units. In this model, there are multiple connections. At time step t, there is a connection between the input layer at time ***t***and the hidden layer at time ***t***, another connections between the hidden layer of time step ***t*** to time steps ***t-1*** and ***t+1***, and a connection between the hidden layer ***t*** to the output layer ***t***.

2. Recurrent networks that produce an output at each time step and have recurrent connections only from the output layer at one time step to the hidden units layer at the next time step. At each time step ***t***, there is a connection between the input layer at time ***t*** and the hidden layer at time ***t*** and a connection between the hidden layers at time ***t*** to the output layer at time ***t***.

3. Recurrent networks with recurrent connections between hidden units that read an entire sequence and then produce a single output. At each time step ***t***, there is a connection between the input layer ***t*** and the hidden layer ***t***, another connections between the hidden layers of time step ***t*** to time steps ***t-1*** and ***t+1***, and only one connection between the last hidden layer ***T*** to the output layer ***T***.

4. Bidirectional recurrent neural networks which is used when the output at time step ***t*** may depend on time step ***t-1*** and ***t+1***.

5. Deep bidirectional recurrent neural networks which are bidirectional recurrent neural networks with multiple layers per time step ***t***.

Table 2.2 shows the advantages and disadvantages of each RNN model. This is can be referred to as a comparison between the different models being investigated.

Model	Pros	Cons
Model 1	Any f(x) that is computable by a turing machine TM can be computed by this RNN.	Gradients Loss function computational complexity. Training cannot be parallelized. Expensive totrain. Runtime complexity: O (t).
Model 2	Training can be parallelized with the gradient of each step in isolation. No need to calculate the output for (t-1) since the training set produces the ideal value of that output.	Less powerful (lack of hidden to hidden connections). Cannot simulate universal turin machine TM.
Model 3	Less Computational cost for the output.	Has to read the entire input sequence at each time step.
Model 4 / 5	Fit for situations where output at time step (t) depends on previous and future elements. Higher learning capacity.	Hard to train. Needs a lot of training data.

Table 2.2: Comparison between different RNN Models

2.2.3 RNN Mathematical Foundation

This section presents the mathematical details of the loss function of recurrent neural networks and its gradient calculation [23].

Loss Function

The loss function of an RNN is defined as the sum of error at each time step $\boldsymbol{t}$. The total loss for a given sequence of x values paired with a sequence of y would be the sum of losses over all time steps. It is defined as the negative likelihood of $y^{(t)}$ given a sequence of x^1, ..., $x^{(t)}$. The below equation defines the loss of RNN.

$$
\begin{aligned}
E_t &= L\left(\left[x^1, x^2, ..., x^{(t)}\right], \left[y^1, y^2, ..., y^{(t)}\right]\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log P_{model}\left(y^{(t)} | \left[x^1, x^2, ..., x^{(t)}\right]\right)
\end{aligned}
$$

In the remainder of this subsection, the total loss function is represented as in Equation (2.6).

$$E_t = -y^{(t)} \log \hat{y}_t \tag{2.6}$$

where the loss function is dot products between the vectors $y^{(t)}$ and element-wise logarithm of $\hat{y}^{(t)}$.

Gradient Calculation

In order to be able to calculate the gradient of a recurrent neural network, we must calculate the gradient of its parameters. As previously discussed, the parameters of RNN are U, V, and W. The calculation of each parameter is detailed next.

Gradient of Parameter V

The parameter V is only present in $\hat{y}_t$, then let $q_t = V s_t$, then we have the following:

$$\frac{\partial E_t}{\partial V_{ij}} = \frac{\partial E_t}{\partial \hat{y}_{tk}} \frac{\partial \hat{y}_{tk}}{\partial q_{tl}} \frac{\partial \hat{q}_{tl}}{\partial V_{ij}} \tag{2.7}$$

From Equation. (2.6), we know that:

$$\frac{\partial E_t}{\partial \hat{y}_{tk}} = -\frac{y_{tk}}{\hat{y}_{tk}} \tag{2.8}$$

The function $\hat{y}$ is the *softmax* function, so it will have the same gradient which is given in Equations (2.9) and (2.10).

$$\frac{\partial \hat{y}_{tk}}{\partial q_{tl}} = -\hat{y}_{tk}\hat{y}_{tl} \quad \textbf{for k} \neq \textbf{l} \tag{2.9}$$

$$\frac{\partial \hat{y}_{tk}}{\partial q_{tl}} = \hat{y}_{tk}(1 - \hat{y}_{tk}) \quad \textbf{for k= l} \tag{2.10}$$

Putting equations (2.8), (2.9), and (2.10) together, we obtain the below:

$$\begin{aligned} -\frac{y_{tl}}{\hat{y}_{tl}}\hat{y}_{tl} + \sum_{k \neq l}\left(-\frac{y_{tl}}{\hat{y}_{tl}}\right) - \hat{y}_{tk}\hat{y}_{tl} &= -y_{tl} + y_{tl}\hat{y}_{tl} + \sum_{k \neq l} y_{tk}\hat{y}_{tl} \\ &= -y_{tl} + \hat{y}_{tl}\sum_{k} y_{tk} \end{aligned} \tag{2.11}$$

knowing that y_t are all one hot vectors that sum up to 1, we can define the following:

$$\frac{\partial E_t}{\partial q_{tl}} = \hat{y}_{tl} - y_{tl} \tag{2.12}$$

Since $q_t = V s_t$, then: $q_{tl} = V_{lm} s_{tm}$. Therefore, we have the following:

$$\begin{aligned} \frac{\partial q_{tl}}{\partial V_{ij}} &= \frac{\partial}{\partial V_{ij}}\left(V_{lm} s_{tm}\right) \\ &= \delta_{il}\delta_{jm} s_{tm} \\ &= \delta_{il} s_{tj} \end{aligned} \tag{2.13}$$

Combining equations (2.12) & (2.13), we get the following:

$$\frac{\partial E_t}{\partial V_{ij}} = (\hat{y}_{ti} - y_{ti})\, s_{tj}; \tag{2.14}$$

which is known as the outer product. Therefor, the gradient of V is given in equation (2.21) below.

$$\frac{\partial E_t}{\partial V} = (\hat{y}_t - y_t) \bigotimes s_t; \tag{2.15}$$

where $\bigotimes$ is the outer product.

Gradient of Parameter *W*

The calculation of the gradient of the parameter W depends on s_t and $\hat{y}_t$. Also, $\hat{y}_t$ depends on W directly and indirectly through s_t and s_{t-1} respectively.

Let $z_t = Ux_t + Ws_{t-1}$, then if we ignore the bias vector b, we get: $s_t = \tanh(z_t)$.

Applying the chain rule, we get the following:

$$\frac{\partial E_t}{\partial W_{ij}} = \frac{\partial E_t}{\partial \hat{y}_{tk}} \frac{\partial \hat{y}_{tk}}{\partial q_{tl}} \frac{\partial q_{tl}}{\partial s_{tm}} \frac{\partial s_{tm}}{\partial W_{ij}} \tag{2.16}$$

We already calculated the first two terms of Equation (2.19), so we need to elaborate on the third and fourth terms. The third term is pretty simple and is illustrated in Equation (2.22).

$$
\begin{aligned}
\frac{\partial q_{tl}}{\partial s_{tm}} &= \frac{\partial}{\partial s_{tm}} \left(V_{lb} s_{tb}\right) \\
&= V_{lb} s_{bm} \\
&= V_{lm}
\end{aligned} \tag{2.17}
$$

The gradient of the fourth term requires us to notice that there is a direct dependence of s_t on W_{ij} and implicit dependence of s_t on W_{ij} through s_{t-1}. Therefore, we have:

$$
\frac{\partial s_{tm}}{\partial W_{ij}} \rightarrow \frac{\partial s_{tm}}{\partial W_{ij}} + \frac{\partial s_{tm}}{\partial s_{(t-1)n}} \frac{\partial s_{(t-1)n}}{\partial W_{ij}} \tag{2.18}
$$

where n is a dummy index variable. Note that we can just apply this to yield:

$$
\frac{\partial s_{tm}}{\partial W_{ij}} \rightarrow \frac{\partial s_{tm}}{\partial W_{ij}} + \frac{\partial s_{tm}}{\partial s_{(t-1)n}} \frac{\partial s_{(t-1)n}}{\partial W_{ij}} + \frac{\partial s_{(t-1)n}}{\partial s_{(t-2)p}} \frac{\partial s_{(t-2)p}}{\partial W_{ij}} \tag{2.19}
$$

where n and p are dummy index variables. The above will continue until we reach s_{t-1} which is initialized to a vector of zeros.

Note that we can collapse the last term to $\left(\frac{\partial s_{tm}}{\partial s_{(t-2)n}} \frac{\partial s_{(t-2)n}}{\partial W_{ij}}\right)$ and the first term to $\left(\frac{\partial s_{tm}}{\partial s_{tn}} \frac{\partial s_{tn}}{\partial W_{ij}}\right)$, then we get the collapsed form in Equation (2.25) below.

$$
\frac{\partial s_{tm}}{\partial W_{ij}} = \frac{\partial s_{tm}}{\partial s_{rn}} \frac{\partial s_{rn}}{\partial W_{ij}} \tag{2.20}
$$

where n is a dummy index variable. We sum over all the values of r less than t, we get the following:

$$
\frac{\partial s_{tm}}{\partial W_{ij}} = \sum_{r=0}^{t} \frac{\partial s_{tm}}{\partial s_{rn}} \frac{\partial s_{rn}}{\partial W_{ij}} \tag{2.21}
$$

Combining the above, we get the final equation of the gradient of the parameter W as shown below:

$$\frac{\partial E_t}{\partial W_{ij}} = (\hat{y}_{tl} - y_{tl})\, V_{lm} \sum_{r=0}^{t} \frac{\partial s_{tm}}{\partial s_{rn}} \frac{\partial s_{rn}}{\partial W_{ij}} \tag{2.22}$$

Gradient of Parameter *U*

The gradient of the parameter U is similar to that of W since both have direct and indirect dependencies on s_t and s_{t-1} respectively. Therefore, applying the same derivations on U as in W, we get Equation 2.23

$$\frac{\partial E_t}{\partial U_{ij}} = (\hat{y}_{tl} - y_{tl})\, V_{lm} \sum_{r=0}^{t} \frac{\partial s_{tm}}{\partial s_{rn}} \frac{\partial s_{rn}}{\partial U_{ij}} \tag{2.23}$$

Total Gradient

The total gradient is given as just the summation of the gradient of V, W and U over all the time steps for a given back propagation algorithm. This can be referred to as follows:

$$\begin{aligned} E_{total} = \sum_{t=0}^{t=T} \left[(\hat{y}_{tl} - y_{tl})\, V_{lm} \sum_{r=0}^{t} \frac{\partial s_{tm}}{\partial s_{rn}} \frac{\partial s_{rn}}{\partial U_{ij}} \right] + \\ \sum_{t=0}^{t=T} \left[(\hat{y}_{tl} - y_{tl})\, V_{lm} \sum_{r=0}^{t} \frac{\partial s_{tm}}{\partial s_{rn}} \frac{\partial s_{rn}}{\partial W_{ij}} \right] + \\ (\hat{y}_t - y_t) \bigotimes s_t \end{aligned} \tag{2.24}$$

2.2.4 Back Propagation Through Time Algorithm

Back Propagation (BP) is the key and main algorithm that makes training deep model computationally traceable. The idea behind this algorithm is to calculate the derivatives starting from the error using the chain rule of differentiation aiming to learn the most appropriate and optimized parameters for the learning model.

BPTT is a derived algorithm of the basic back propagation algorithm. The idea is that the derivative is summed up at each time step. As discussed in section 2.2.3, in recurrent neural networks we need to calculate the gradient of the error E_{total} with respect to U, W and V and learn optimized parameters for the model. The gradients of the error are summed up with respect to each parameter as the error values are summed up at each time step $\boldsymbol{t}$.

As shown in section 2.2.3, the gradient of the parameter V does not depend on previous time steps which tends to make it easy to be calculated (simple matrix multiplication). On the other hand, the gradient of U and W depends directly on the current time step and indirectly on all previous time steps less than $\boldsymbol{t}$. Then, it is required to keep track and sum up the gradient of each parameter (U and W) at each time step. This is not considered a simple and straight forward task. This fact will lead to the vanishing / exploding gradient limitation as described in section 2.2.5. This limitation is also known as long term dependencies challenge.

2.2.5 Recurrent Neural Networks Limitation

Long term dependencies challenge is defined in terms of the computation of the gradient of the recurrent neural network at each time step $\boldsymbol{t}$. This limitation is also known as the vanishing / exploding gradient problem. The idea is that the gradient calculated at each time step will either vanish or explode after at time step t_j such that t_j is greater than t_i. Recurrent neural networks with stable parameters (that are the same over the whole network) will not solve the issue.

For the gradient vanishing problem, we can notice from the gradient calculation of U and W parameters that we are applying the chain rule multiple times. Also, knowing that the gradient is the derivative of a vector function with respect to a vector, then the result will be a matrix called the Jacobian matrix whose elements are all point wise derivatives.

According to [19], the L2 Norm of the Jacobian matrix has an upper bound value of 1. This makes sense since the hyperbolic tangent tanh function maps all the values into a range of values between -1 and 1, and its derivate is bounded by 1 as shown in Figure 2.4 [24].

Thus, the derivate of the hyperbolic tangent *tanh* function is zero at both ends since it approaches a flat line. In this sense, the gradients in previous layers will be derived to zero. Subsequently, this will lead to the gradient vanishing problem after few small time steps since multiple matrix multiplications with small values are performed. Therefore, the gradients at time step t_i such that t_i less than t will not contribute to the learning rate of the model where t is the current time

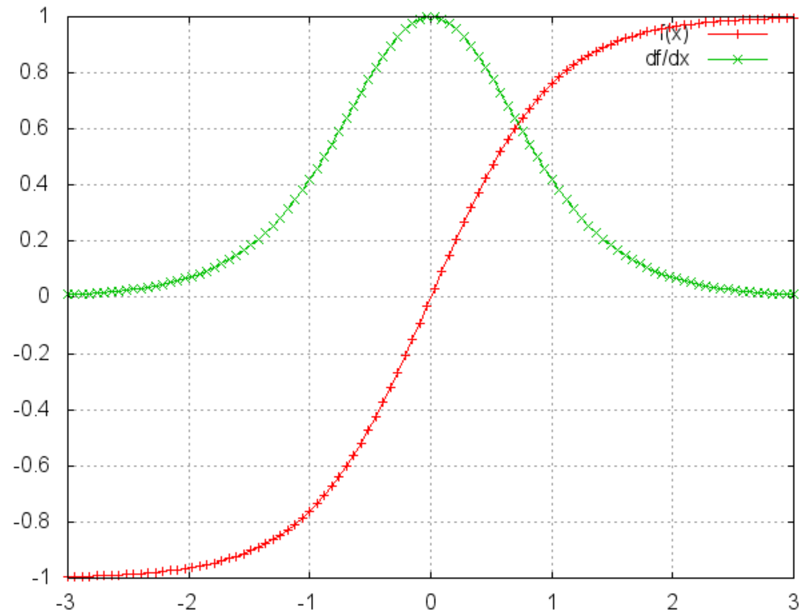

Figure 2.4: Derivative of Hyperbolic Tangent Function

step.

Similarly, the exploding gradient problem is explained based on the same key ideas and concepts but with one small difference, that is the values in the Jacobian matrix will be large enough.

Vanishing gradients are more problematic than exploding gradients since it is not obvious when they occur and how to deal with them. Also, the exploding gradient problem can be noticed during the implementation since the values of the gradients will be **Not a Number (NaN)** and the code will crash.

In order to overcome these limitations multiple solutions can be applied which are listed below [16].

1. The exploding gradient problem can be solved by clipping the gradients at

a predefined threshold.

2. Regularization techniques can solve the vanishing gradient issue by applying proper initialization of the matrices U and W.

3. The use of ReLU instead of hyperbolic tangent *tanh* function. This is helpful due to the fact that the derivative of ReLU is either zero or one which will get rid of the vanishing/exploding gradient limitation.

4. Use of a special type of recurrent neural networks known as Long-Short Term Memory (LSTM) which is detailed in section 2.2.6.

2.2.6 Long Short Term Memory (LSTM)

Long-Short Term Memory (LSTM) can be defined as recursive neural networks with memory that are capable of holding the gradient for long time without being vanished or exploded. This is accomplished due to the fact that LSTMs allow for self loops which generate paths that can hold the gradient over time $\boldsymbol{t}$ with $\boldsymbol{t}$ being large enough. In general, LSTMs have the following four major decision that should be decided at each time step $\boldsymbol{t}$.

1. Decide which information are irrelevant and should be thrown away at the current cell state.

2. Decide which information are useful and should be stored at the current cell state.

3. Update the current cell state at time step ***t*** into the next cell state at time step ***t+1***.

4. Decide which information should be included in the output of the current cell state.

2.2.7 Long Short Term Memory Mathematical Foundations

As discussed in section 2.2.6, at each time step ***(t)***, each LSTM cell should perform four actions. The mathematical background of each decision is explained as following.

Forget Gate

Forget gate is the gate where the LSTM decides which information are irrelevant and should be thrown away from the network. The forget gate is calculated as shown in Equation (2.25).

$$F_i^{(t)} = \sigma\left(b_i^{(F)} + \sum_j U_{i,j}^{(F)} x_j^{(t)} + \sum_j W_{i,j}^{(F)} h_j^{t-1}\right) \tag{2.25}$$

where

- $x^{(t)}$ is the current input vector.

- $h^{(t)}$ is the current hidden layer in which it contains the output of all previous LSTM cells.

- $b^{(F)}$ is the bias vector of the forget gate.

- $U^{(F)}$ is the input weights for the forget gate.
- $W^{(F)}$ is the recurrent weights for the forget gate.

External Input Gate

External input gate is the gate where the LSTM receives new input x_t at time t. The external input gate is calculated as shown in Equation (2.26). The calculation of the external input gate is the same as the forget gate with its own parameters. We will refer to it as $G_i^{(t)}$ at each time step t.

$$G_i^{(t)} = \sigma\left(b_i^{(G)} + \sum_j U_{i,j}^{(G)} x_j^{(t)} + \sum_j W_{i,j}^{(G)} h_j^{t-1}\right) \tag{2.26}$$

where

- $x^{(t)}$ is the current input vector.
- $h^{(t)}$ is the current hidden layer in which it contains the output of all previous LSTM cells.
- $b^{(G)}$ is the bias vector for the external input gate.
- $U^{(G)}$ is the input weights for the external input gate.
- $W^{(G)}$ is the recurrent weights for the external input gate.

LSTM Internal State

Internal state is the state where the LSTM calculates its internal state before passing it to the output gate. The internal state is calculated as shown in Equation (2.27).

$$S_i^{(t)} = F_i^{(t)} S_i^{(t-1} + G_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{t-1} \right) \tag{2.27}$$

where

- b is the bias vector for the LSTM cell.
- U is the input weights for the LSTM cell.
- W is the recurrent weights for the LSTM cell.

Output Gate

Output gate is the gate where the LSTM calculates its output at time t. The output gate is calculated as shown in Equation (2.28).

$$Q_i^{(t)} = \sigma \left(b_i^{(Q)} + \sum_j U_{i,j}^{(Q)} x_j^{(t)} + \sum_j W_{i,j}^{(Q)} h_j^{t-1} \right) \tag{2.28}$$

where

- $x^{(t)}$ is the current input vector.
- $h^{(t)}$ is the current hidden layer in which it contains the output of all previous LSTM cells.
- $b^{(Q)}$ is the bias vector for the output gate.
- $U^{(Q)}$ is the input weights for the output gate.
- $W^{(Q)}$ is the recurrent weights for the output gate.

LSTM Cell Output

Cell output is the output of the LSTM at each cell of the network. The LSTM cell output is calculated as shown in Equation (2.29).

$$h_i^{(t)} = \tanh\left(S_i^{(t)}\right) Q_i^{(t)} \tag{2.29}$$

where

- $S_i^{(t)}$ is the internal state output.
- $Q_i^{(t)}$ is the output gate.
- *softmax* is an activation function.

Chapter 3

Literature Review

This chapter discusses different approaches founded in the literature related to malware detection and classification problem. This chapter is divided into three sections. The first section discusses static analysis techniques while the second is dedicated for dynamic ones. In the third section, we detail several solutions that are based on deep learning. Also, we state the results obtained in different approaches in order to compare our results to them in Chapter 5.

3.1 Static Analysis Approaches

The architecture of malware detection and classification solutions based on data mining, machine learning, and deep learning is mainly composed of two components: feature extraction component and a classification component. In this section, we introduce the literature review of feature extraction component such as static techniques, data mining, and machine learning classification components.

In static approaches, features of malware executables are extracted and analyzed before their execution begins. In [1], a model was designed that computes

frequencies from extracted string in executable binaries. These features are then fed to a Naive Bayes classifier for training. During the training phase, the classifier is trained on those features in order to build the necessary knowledge that is needed for the testing phase. In the testing process, the classifier predicts the correct class of a new malware. Another twisted approach tends to extract the system resources that an executable binary is trying to access such as dynamically linked libraries (DLLs) and their corresponding library functions based on the portable execution representation of an incoming binary [1].

In [6], a malware executable is represented as a gray scale image vector. The approach is based on the assumption which states that the images that have the same layout and texture will most likely belong to the same family of images. On the classifier level, the Euclidean distance between vector representations of malware was used in order to predict the correct class of a new incoming instance. In another attempt [3], each malware instance is represented as a disassembled file and function calls graph were extracted as features. Each feature can be described as follows: each node in the graph is a system call with some number of feature vectors that represent the function attributes. This method tends to adopt an ensemble of classifiers for automated malware classification after learning the confidence level associated with the classification capability of each attribute type.

Many malware detection and classification proposed solutions represent malware instances as text and aim to extract their byte n-grams and opcode n-grams

during the feature extraction phase. These extracted n-grams are then treated as feature vectors of words. In [1], n-grams were collected from binary executables instances as features and were used to train a Multinomial Naive Bayes classifier to predict the correct class of a new malware instance. Different classification models such as Decision Tree, SVM, Naive Bayes, and their boosted versions were tested using the same solution.

Another approach that extracted n-grams from executable binaries was proposed in [2] with a little twist. Feature selection techniques were used to obtain the most significant n-gram terms with variation of their length. Each executable binary was represented as a text document and calculated its frequency term. These frequency terms were inputted to the classification model such as Artificial Neural Network, Naive Bayes, Decision Tree, and SVM. Another similar approach was applied in [7] with only one difference. The extraction and collection of n-grams was facilitated by transforming executable binaries into disassembled files. opcodes were extracted but their parameters were disregarded in order to bypass evasive techniques that obfuscate memory addresses and values.

There exists some limitations for static analysis approaches such as code obfuscation, data obfuscation, and disruption of malware syntax by using opaque constants and predicates. Code obfuscation is the process of modifying an executable so that it is no longer readable but remains fully functional. Data obfuscation is a form of data masking where data is purposely scrambled to prevent unauthorized access to sensitive materials. Disruption of malware syntax by

using opaque constants and predicates is the process of adding opaque constants and predicates to the syntax of the malware instances to overcome the developed model for malware detection. Thus, the malware will be useless for the model and it will not be able to extract the correct features for the classification component. Dynamic analysis approaches were designed and implemented to overcome the aforementioned limitations of static analysis approaches.

3.2 Dynamic Analysis Approaches

Malware detection and classification solution based on dynamic analysis rely on setting executable binaries in some safe and virtual environment and monitoring the behavior of at runtime aiming to extract useful execution patterns as features to the classification model.

An approach was proposed in [28] that extracts behavioral features from a running malware instance in a virtual environment which is passed then for human analysis. Another similar approach was designed and proposed where similarities in the changes that a malware instance tends to perform to the system were clustered and used as feature vectors. This clustering was done based on a normalized compression distance as a measure to perform hierarchical clustering on these vectors. In a similar approach [4],the interaction behavior of malware executable binaries with system resources were extracted and used as features to train a k-medoid clustering algorithm.

Analysis of system calls is known as a common dynamic analysis technique

that is heavily used to facilitate malware detection and classification. In [5], feature vectors were built by collecting system calls from each incoming executable binary located in a safe and virtual environment which are passed to train an SVM classifier. Another similar slightly different approach was designed and proposed in [29]. The difference is that they represented malwares system calls as malware instructions (MIST) in each incoming malware binary executable. MIST q-grams and their corresponding frequencies were collected and used as feature vectors which in their turn were passed to the classification model in order to predict and output the correct class of a new malware instance.

In [30], features were extracted using recurrent neural networks (RNN) from API call log sequences. They considered only the first 5 minutes of the log file in order to reduce the dimension of the input features and the complexity of the classification model. These features were then fed into a convolutional neural network (CNN) to obtain an accuracy value of 96% using a dataset of 170 samples. A comperative study between machine learning alogirthms used for malware detection was investigated in [31]. They compared different classification algorithms that were trained on API calls. Correlation based feature selection and decision tree reported the best accuracy result of value 96.8%.

Another approach that uses API call streams from 100 benign samples being WindowsXP 32-bit system files along with their associated metadata was proposed in [31]. These features were fed into a Naive Bayes classifier where an accuracy of 92.8% was reported. In [32] the same set of features were used with

a Random Forest classifier to achieve 94% and 95% for accuracy and F-measure values respectively. Also, the highest accuracy achieved that uses the same set of features was 97.4% using a shallow feed-forward neural network [33]. Table 3.1 lists the differences between static and dynamic approaches.

Even though dynamic analysis overcome some limitations of static analysis, it still suffers from some limitations when exposed to more powerful techniques. First, malware that recognizes the environment in which it runs can recognize its presence inside a sandbox or a virtual environment and can alter its execution accordingly in order to evade detection. Second, malware that run in kernel space need not use system calls to perform its tasks, and hence it can neither be detected nor classified through system call analysis. Deep learning approaches were introduced to deal with more powerful class of malware that are capable of hacking dynamic analysis approaches.

3.3 Deep Learning Approaches

Neural Network and deep learning approaches were used for malware detection and classification in system call sequences as well. In [34] a solution was designed for malware detection and classification in system calls using deep learning that consists of two layers: convolutional and recurrent neural networks. The convolutional part consists of convolution neural networks and pooling layers. It is used to capture the correlation between neighboring vectors and produce new features. Max pooling is applied on the output of the convolutional part in order

Paper Reference	Static	Dynamic	Classification Algorithms
[1]	X		NB & MNB
[6]	X		Euclidean Distance
[3]	X		Ensemble of Classifiers
[2]	X		ANN, NB, DT, and SVM
[7]	X		ANN, NB, DT, and SVM
[25]	X		-
[26]	X		NN
[27]	X		MNB & LR
[28]		X	-
[8]		X	Clustering
[4]		X	k-medoid Clustering
[5]		X	SVM
[29]		X	SVM
[31]		X	NB & J48 DT
[32]		X	RF

Table 3.1: Static vs Dynamic Malware Detection and Classification Approaches

to reduce its dimensionality. The output is then fed into the recurrent neural networks part for processing using LSTM cells. Mean pooling was applied on the output of the LSTMs in order to further reduce the complexity of processing the data. Dropout [35] was used in order to prevent overfitting.

Another application domain that is tackled using a static analysis deep learning approach for malware detection and classification in images and OpCode 3grams. In [36], the proposed solution used to capture malware and classify them is based on two parts: feature extraction and deep learning model. The feature extraction part combines a gray scale image and OpCode 3-gram which describes the malware from different angles. The dataset was divided into training and testing data where the deep learning model is trained on the training data set, and tested on the testing data. This method was proved to be effective by achieving promising results. Moreover, [37] proposed another approach based on deep learning and recurrent neural networks that is capable of detecting and classifying malware of dynamic data. This approach reached an accuracy that is more than 93% and 98% in 4 and 20 seconds of execution time respectively. Another work was introduced on malware detection and classification in [38] where a solution was formalized based on deep learning for malware detection and classification in software binaries. They reached an accuracy up to 95% and a false positive rate of 0.1% over an experimental dataset of more than 400,000 software binaries.

In [39], Windows APIs were extracted from real and large file sample collection along with SEAs to design an approach for malware detection and classification using deep learning. Their architecture of two phases: unsupervised pre-training and supervised back propagation. SEA model perform pre-training by putting all the parameters of all layers in one parameter space in a bottom-up manner and

supervised BP is applied to tune the models parameters in a top-down direction. Their approach reached an accuracy of 95%. A recent attempt for malware detection and classification was proposed in [40]. In this approach, malware instances were executed using a light weight file emulator developed by Microsoft in order to extract system calls. LSTM was used in order to extract features from system calls and then a linear regression classifier was used to detect the correct class for a new malware instance.

RNNs and Echo State Networks (ESNs) were used as classifiers for malware detection and classification in [41]. ESNs were found to report higher accuracy value of around 95% with an error of 5% than RNNs. In another attempt [9], an ensemble of RNNs was used for early malware detection as well. An accuracy value of 94% was reported by predicting whether an executable is malicious or benign within the first 5 seconds of execution only. Table 3.2 lists the different deep learning based approaches for malware detection and classification problem.

In [25], a comperative study was conducted between static, dynamic, and hybrid approaches solutions for malware detection. Dynamic approaches based on Hidden Markov Model (HMM) were proved to be the most efficient ones with accuracy of 98% using dataset of 785 instances. In another attempt [26], a malware was detected using a deep forward neural network using features derived from code. A true positive value of 92.5% was achieved on validation dataset, but this value decreased on testing data to 67.7%. Moreover, according to work done in [27] accuracy dropped from 97% to 20% on validation to testing dataset

on Android software.

Although deep learning approaches overcome the limitations of static and dynamic analysis, they still have some drawbacks. First, the deep learning models are very complex and requires a lot of processing time to finish its execution. They need a lot of feature engineering methods to extract the needed features for the classification models. Another limitation is that deep learning models requires a large dataset for feature extraction. Also, a huge dataset is needed to build and train the model.

Paper Reference	CNN	Classification Algorithms
[34]	X	RNN
[35]	X	RNN
[36]		RNN
[37]		RNN
[38]		DNN
[39]		DNN
[40]		LSTM & LR
[41]		ESN & RNN
[30]	X	CNN
[9]		RNN
[33]		NN

Table 3.2: Deep Learning Malware Detection and Classification Approaches

3.4 Word Embedding

Word embeddings are known as vector representations of words that will be used as features for our deep learning model. In word embeddings, words that appear frequently in similar context tend to be neighbors in the embedding space. In natural language processing (NLP) domain, we can say that these words tend to have similar features. This major property makes it different from n-grams representations of words. Figure 3.1 depicts a high level example of word embed-

ding. The clusters are decomposed based on colors. Each color represents a set of words that appear frequently in a similar context, so they have similar features. For example, lets consider the two sentences ''***Deep learning is the trend nowadays***" and "***Machine Learning is the trend today***" the two words "***Machine Learning***" and "***Deep Learning***" share the same contextual information, which means that these two new domains are close to each other in the embedding space. Thus, they should belong to the same cluster group in Figure 3.1.

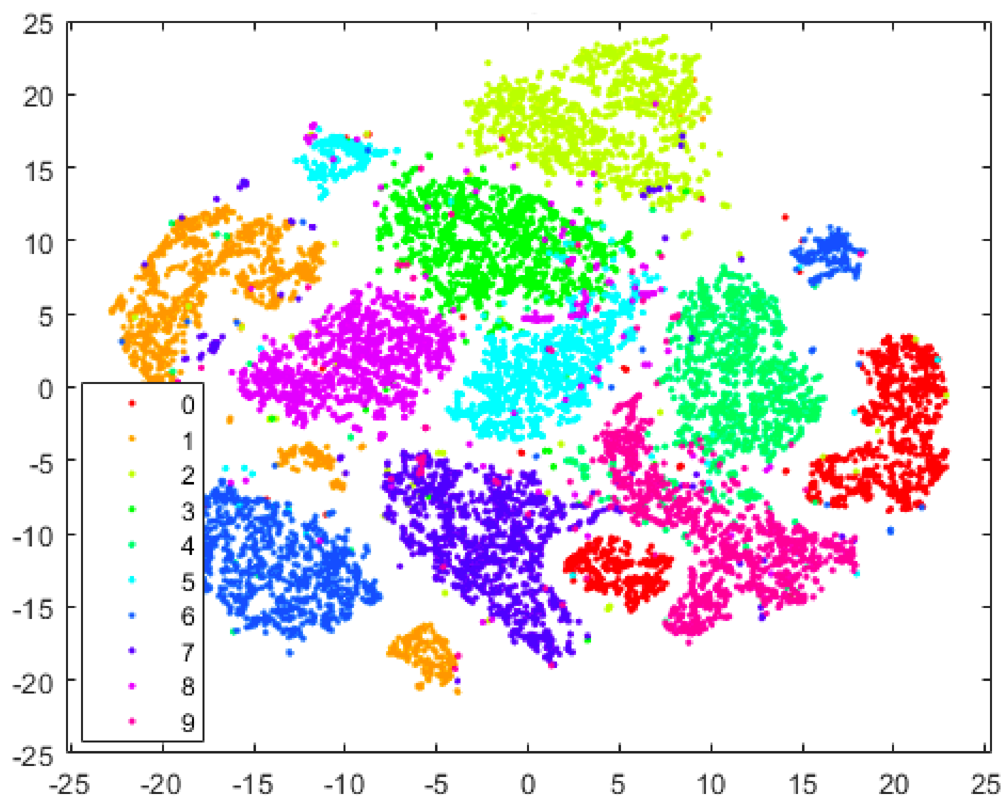

Figure 3.1: Word Embedding Example

Chapter 4

Proposed Approach

This chapter presents the proposed approach which is divided into two main components: feature extraction component and a classification model. We start with an overview of the architecture of the proposed model, then discuss word embedding and detail our feature extraction and CNN-LSTM based classification components.

4.1 High Level Architecture

The methodology of our approach is described as follows. It is composed of two main components: a feature extraction component and a classification component based on CNN and LSTM as shown in figure 4.1. The latter is known as special types of RNN. In this figure, the first component is the feature extraction component that takes as input a set of malware instances represented as assembly text files. This component extracts all the actions that a malware intends to perform on a computing system using a developed parser and a predefined set of assembly instructions. These actions will serve as features for the classification

component. The second component is a deep learning classification component which takes these features as input to train the model. The model predicts the correct class of a new malware instance during the validation and testing phases.

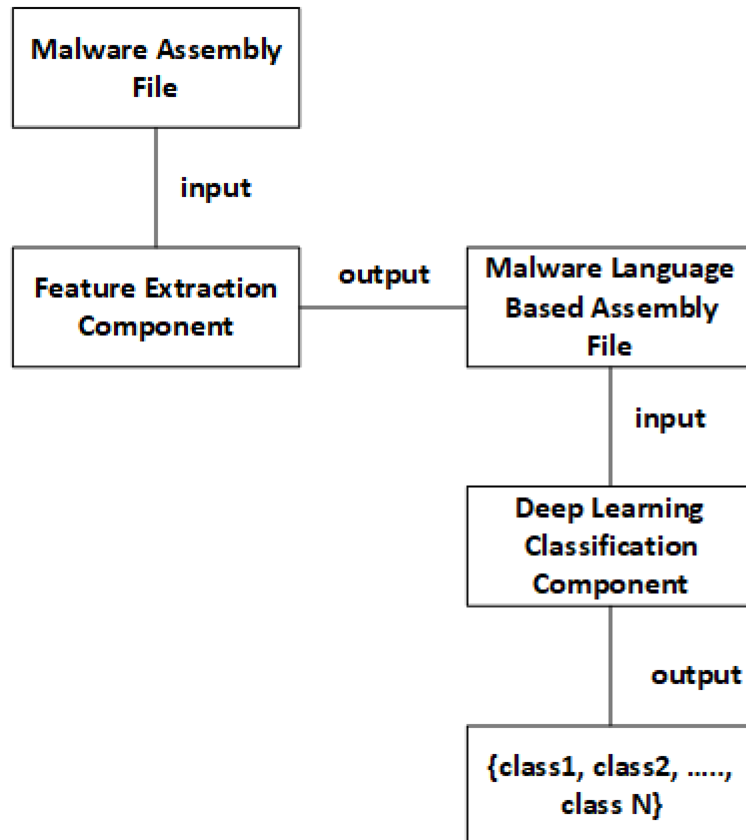

Figure 4.1: Model High Level Architecture

4.2 Word Embedding

There is a set of word embedding algorithms that were applied in our experiments such as Word2vec [42], FastText [43], and GloVe [44]. These word embedding algorithms were not useful to our approach since they were designed and implemented based on meaningful set of English words. Our features are based on assembly instructions such as *jmp, mov, add, sub, etc.* which are meaningless to

those algorithms. The word embedding algorithm used in the proposed approach is built during the feature extraction component. Initially, the word embedding matrix is initialized to random values and then it is adjusted while the features are extracted. The output is a matrix representation that includes vector representation for each extracted feature. The word embeddings matrix $Q_x \in R^{d \times |V|}$, is defined as the stack of all vector representations of all malware instances where d and $|V|$ are the dimension of the word vector and the vocabulary size respectively. These features are then fed into the classification component for further analysis and computations.

4.3 Feature Extraction Component

This section describes the details of the feature extraction component of our approach. We will start by defining our malware language design and then discuss how these features are useful in our solution to malware detection and classification problem [45].

Our malware is designed with the concept of a vocabulary, documents, and words. To that end, we define a malware vocabulary, malware document, and a malware word. Each malware assembly instance is a document, and each assembly action in the malware document is a word. Consequently, a malware vocabulary is defined as a set of malware documents. Figure 4.2 depicts the design of our malware language. In this figure, the parser takes as input a set of malware instances represented in assembly text files and extracts all the actions

that a malware intends to perform.

Assembly instructions executed by a malware instance are considered as very important and useful features in order to build and train a model that is capable of detecting the correct class of a new malware input. In our context, each malware instance is a document and our objective is to extract the words of each document. These words are the set of actions performed by a malware in our language design. In order to achieve our goal, we developed a parser that parses each malware document and extracts all the existing words based on a predefined set of *431* assembly instruction. These features will be fed as input to our classification model as explained in the next section.

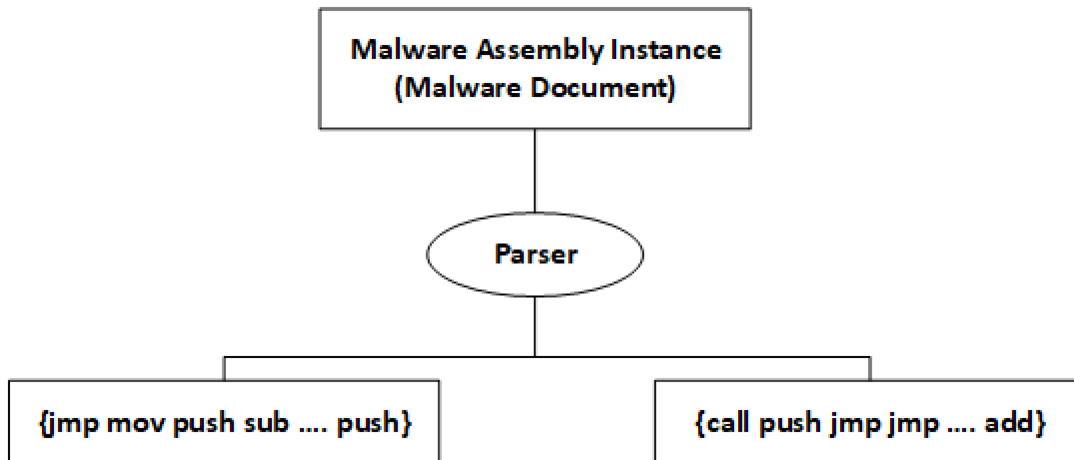

Figure 4.2: Malware Language Design

Algorithm 1 is a pseudo code of the parser. The parser is divided into two main functions. The first one changes the names of all the input files into standard format in order to read the files in an automated manner. The second function reads all the input assembly files and extract their actions. For each

input file, there exists a corresponding output file that contains its list of actions. Then, each output file is stored in a separate list of actions. Also, the correct class of each malware instance is stored in a list.

Data: Labeled Assembly Files As Input AF_I, Output Assembly Files OF_I
Start Parsing
Function Rename
repeat
  Take an assembly file $AF_I^{1'}$ from AF_I
  $OF_{I'}$ = Rename $(AF_I^{1'})$
  OF_I = Append $OF_{I'}$
until *there are no instances in* AF_I;
Function Parse Actions
repeat
  Take an assembly file $OF_I^{1'}$ from OF_I
  while (!EOF):
  words = toWords(line):
  if (words exists in Assembly List):
  Actions $OF_I^{1'}$ = append(words)
until *there are no instances in* OF_I;
Function Read Input Label
repeat
  Input_Label = Label(OF_I)
until *there are no instances in* OF_I;
End Parsing

Algorithm 1: Parser Pseudo Code of Feature Extraction Component

4.4 Classification Component

This section describes the details of the classification component of our proposed approach. The ultimate objective of this component is to classify a new malware instance into its correct class. To that end, we built a deep neural network model which is illustrated in Figure 4.3. Our model makes use of vector representations which are referred to as word embeddings that are extracted in the feature extraction component.

The design of our approach is highly dependent on the the architecture of CNN and LSTM. We designed and implemented four different architectures for our deep learning model. The first architecture depends on long short term memory, the second one depends solely on gated recurrent networks, the third depends on convolutional neural network, and the fourth is a hybrid approach that depends on one dimensional convolutioal neural network (1D-CNN) and an long short term memory (LSTM) with a predefined number of fully connected layers. All architecutres of our classification component make use of an activation function. The activation function can be either ***softmax*** (Equation 2.5) or ***hyperbolic tangent*** (Equation 2.4).

As described in Chapter 2, Hochreiter and Schmidhuber defined and developed LSTM as a special kind of RNN [46]. One major contribution and advantage of LSTM is its capability of mapping variable length word vectors to a fixed length ones by transforming the current input word vectors x_t with the output of the hidden layer of the previous step h_{t-1}. In Figure 4.3, U, W, and V_i, where

$1 \leq j \leq N$, are weight matrices that refer to input-to-hidden, hidden-to-hidden, and hidden-to-output connections, respectively. An LSTM cell is then computed as described in Chapter 2 Section 2.2.7 (Equation 2.29).

In Figure 4.3, the classification model is based on one dimensional convolutional neural network and long short term memory. First, the input X_t is inputted to the CNN layer where each square is a network. second, the output of the convolutional neural network serves as input to the recurrent neural network based component. This component consists of multiple long short term memory (LSTM) cells where each cell contains multiple neural networks. At each time step t, x_t is given as input to each LSTM cell where $0 \leq t \leq n$. $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{V}$ are the input to hidden, hidden to hidden, and hidden to output matrices respectively. These matrices are shared among all LSTM cells. Each LSTM cell processes the output of the previous LSTM cell with its current input x_t. For example, let $t = 5$, then $LSTM_{CELL5}$ processes the output of $LSTM_{CELL4}$ and input x_5. The output of $LSTM_{CELL5}$ serves as input to $LSTM_{CELL6}$ and so on until reaching $t = N$. At $t = N$, the output is given as input to the deep forward neural network for further processing. An activation function is applied on the final output produced after the data is processed by the deep forward neural network. The learning capability of our hybrid model is the combination of both components which are the convolutional neural networks and long short term memory.The description of the LSTM GRU, and CNN are covered within the aforementioned discussion of the 1D-CNN LSTM architecture. Consequently,

the LSTM based architecture makes use of the second component of Figure 4.3. As for the GRU architecture, it uses the second component with only one difference which is the presence of GRU layers instead of LSTM layers. On the other hand, the CNN architecture makes use of only the first component of the model depicted in Figure 4.3.

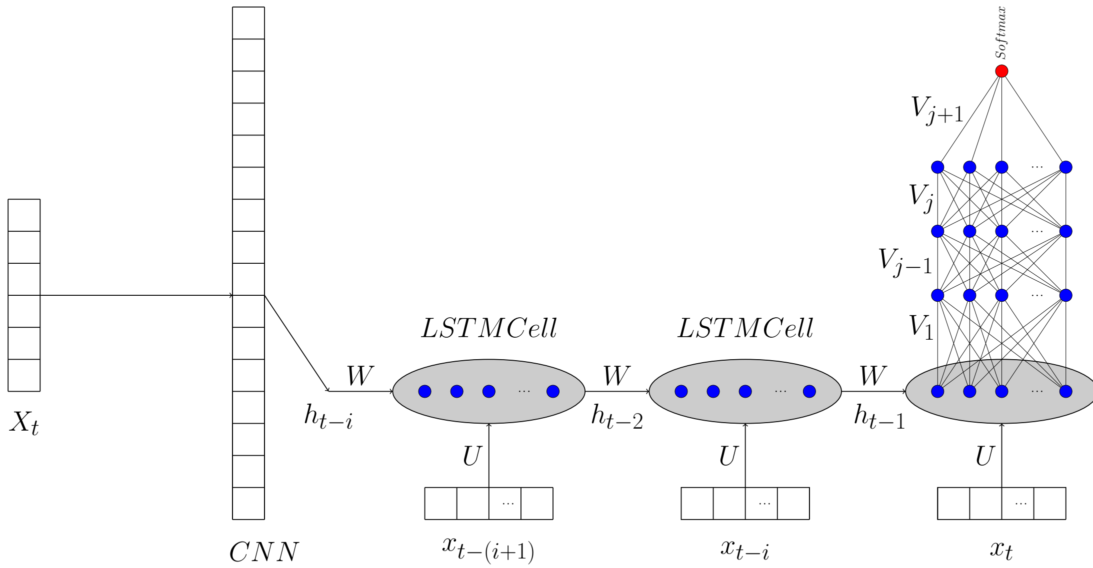

Figure 4.3: Representation of CNN-LSTM Architecture for Malware

Algorithm 2 describes the overall procedure applied in order to read the original assembly input file, extract the needed features which are the actions a malware instance intends to execute, train a CNN-LSTM based classification model, and classify a new malware instance into its correct class of malwares. The implementation and experiments described in Chapter 5 are based on algorithm 2. This algorithm is divided into data pre-processing step and model creation, training, and validation step. In data pre-processing the model takes as input all malware instances as assembly files and parses them as described in Algorithm

1. For all malware instances, the list of actions are extracted and stored inside a list of actions. The vector representation is then computed for each malware inside the list. In the second part, the classification model is built as follows. The embedding which is mainly the vector representation computed in the first part is added to the model. Then, the deep learning component such as convolutional neural network and long short term memory are added to the model. After that, the dropout is added to the model in order to avoid overfitting. The drouput is applied on the level of the input layers, hidden layers, and output layer. To note that dense function is applied on the output in order to transform the vector representation from 128 to 9 since we have 9 different malware families. The activation function is added to the model before compiling it. At the end, the model is trained and validated on the training and validation datasets respectively.

Data: Labeled Assembly Files As Input AF_I, Output Assembly Files OF_I
repeat

Take an assembly file $AF_I^{1'}$ from AF_I

$OF_{I'}$ = Parse $AF_I^{1'}$

OF_I = Append $OF_{I'}$

until *there are no instances in* AF_I;
repeat

Take an assembly file $OF_I^{1'}$ from OF_I

Actions $OF_I^{1'}$ = ParseActions $OF_I^{1'}$

Actions_Vector $OF_I^{1'}$ = Vector_Representation(Actions $OF_I^{1'}$)

until *there are no instances in* OF_I;
Model.add(Embedding (OF_I))

Model.add(CNN (OF_I))

Model.add(LSTM (OF_I))

Model.add((Dropout (OF_I)))

Model.add((Dense (OF_I)))

Model.add((Activation (OF_I)))

Model.compile()

Model.fit(training data, training data labels, validation data, validation data labels)

Model.predict(testing data)

Algorithm 2: Pseudo Code of Hybrid Classification Component

Chapter 5

Performance Evaluation

This chapter presents a description of the dataset used in our performance evaluation and the results of all the investigated models. We start by the results achieved of the machine learning algorithms which are logistic regression and support vector machines. Then, the results of LSTM, CNN, GRU and 1D-CNN LSTM architecture are presented respectively. At the end of this chapter, a comparison of different architecture is discussed.

5.1 Dataset Description

The dataset is provided by Kaggle competition [47]. The data a set of known malware files representing a mix of 9 different families. Each malware file has an Id, a 20 character hash value uniquely identifying the file, and a Class, an integer representing one of 9 family names to which the malware may belong to ***Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, and Gatak***.

The dataset is divided into training and testing data. The training dataset

and testing dataset consists of **10867** and **10872** malware instances respectively. The training dataset is decomposed into 80% as training set to train the model and 20% as validation dataset to validate the learning curve of our model. The training and validation datasets are labeled and our model is tested against the provided testing data. As discussed in Chapter 4, we extracted the actions that a malware intends to perform in assembly format that serves as features to our LSTM based classification model.

5.2 Experimental Results

The performance of our deep learning based model is evaluated by conducting a set of experiments. We started by testing two machine learning based approaches which are logistic regression and support vector machine in order to set a baseline for our analysis. The aim is to check how deep learning approaches added value to the results obtained.

5.2.1 Parameters and Metrics

The evaluation metrics used to asses our experiments are loss and accuracy. In deep learning, the loss is defined as negative log-likelihood which is defined as the cross entropy between two probability distributions p and q over a given set of samples defined in Equation 5.1.

$$L(p, q) = -\sum_{x} p(x) \log q(x) \tag{5.1}$$

The main objective is to minimize the loss value with respect to the model's

parameters by changing the weight vector values through different optimization methods such as back propagation and back propagation with time in recurrent neural networks. The loss value implies how well a certain model is behaving after each iteration of optimization. Ideally, the loss should be reduced after each or several iterations.

The accuracy of a model is determined after the model parameters are learned and fixed and no more learning is taking place. The test samples are fed into the model and wrong classifications are recorded after comparing with the true target. The percentage of mis-classification is calculated. In other words, the accuracy value is calculated as defined in Equation 5.2.

$$Accuracy = 1 - L(p, q) \tag{5.2}$$

We will represent the probability of taking an infected sequence of data from an infected file. This can be summarized as what is the probability that the taken sequence length from the infected file represents a signature of malware? We assume the following assumptions:

- There exist at most one instance of malware at each file.
- Let X be the probability of having an infected sequence of data from an infected file.
- Let M be the position of malware instance.

- Let SL be the sequence length taken into consideration of the complete input file length.
- Let IFL be the complete input file length.
- Let NS be the number of samples.

Then, the probability of X is given in Equation (5.3).

$$\begin{aligned} P(X) &= \frac{SL}{NS} \\ &= \frac{M+1}{IFL-(M+1)} \end{aligned} \tag{5.3}$$

5.2.2 Machine Learning Models

The aim of designing machine learning algorithms is to set a baseline for our evaluations and interpretations of our variations of deep learning based architectures. We implemented and conducted two different machine learning based solutions using the same design of our feature extraction component. Theoretically, a neural network is a set of stacked multiple logistic regression layers and support vector machine is capable of providing good results. Thus, we have chosen our machine learning models to be based on logistic regression and support vector machine classifiers.

The word embedding algorithm used for the machine learning algorithm is known as term frequency-inverse document frequency (TFIDF). TFIDF is a numerical statistical value that reflects the importance of word for a document in a set of vocabulary document [48].

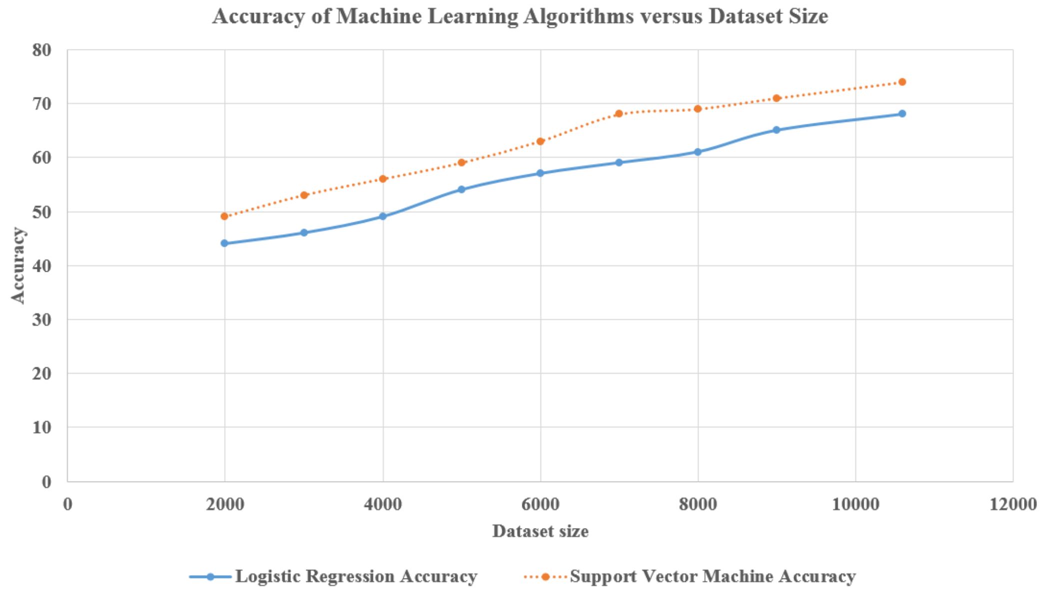

Figure 5.1: Accuracy of Different Machine Learning Classifiers

We conducted several experiments using logistic regression algorithm and support vector machine classifiers. Figure 5.1 shows different accuracy values for logistic regression and support vector machines based classifiers versus the dataset size. The accuracy value of logistic regression starts with a value of 44% with a dataset of 2000 samples. This value increases to 68% as the dataset increases from 2000 to 10600 samples. As for the support vector machines classifier, the accuracy value increased from 49% to 74% as the dataset size increased from 2000 to 10600 malware instances. The best accuracy values obtained by the logistic regression and support vector machine is 68% and 74% respectively. These results will serve as a baseline for our comparison and analysis with our deep learning based classifiers.

5.2.3 Benchmark of Classification Models

We benchmark the parameters versus all the classification models such as LSTM, GRU, CNN, and LSTM CNN. We study how the hyper parameters are affecting the accuracy value of the aforementioned models. The hyper parameters are number of layers, dataset size, dropout percentage, sequence length, number of filters, and kernel size.

Figure 5.2 shows the variation of accuracy value as the number of layers for LSTM, GRU, and CNN LSTM classification models. Generally, the accuracy value is increasing (loss is decreasing) as the number of layers is increasing. For LSTM, the accuracy value increased from 0.5421 to 0.9863 as the number of LSTM layers is increasing from 2 to 8. For GRU, the accuracy value increased from 0.4129 to 0.7991 as the number of GRU layers is increasing from 2 to 8. For LSTM CNN, the accuracy value increased from 0.7954 to 0.9931 as the number of LSTM layers is increasing from 2 to 6. The accuracy value remains 0.9931 as the number of LSTM layers increased from 6 to 8.

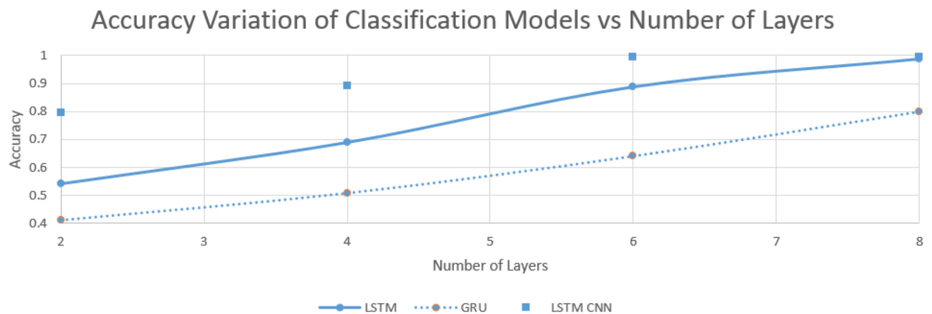

Figure 5.2: Variation of Accuracy of Classification Models versus Number of Layers

Figure 5.3 shows the variation of accuracy value as the dataset size varies for LSTM, GRU, CNN and CNN LSTM classification models. Generally, the accuracy value is increasing (loss is decreasing) as the dataset size is increasing. For LSTM, the accuracy value increased from 0.63433 to 0.98544 as the dataset size increased from 2000 to 8000. The accuracy value remains the same as the dataset size increased from 8000 to 10600. For GRU, the accuracy value increased from 0.43433 to 0.90044 as the dataset size increased from 2000 to 10600. For CNN, the accuracy value increased from 0.50001 to 0.96888 as the dataset size increased from 2000 to 7000. The accuracy value remains the same as the dataset size increased from 7000 to 10600. For CNN LSTM, the accuracy value increased from 0.65433 to 0.99311 as the dataset size increased from 2000 to 7000. The accuracy value remains the same as the dataset size increased from 7000 to 10600.

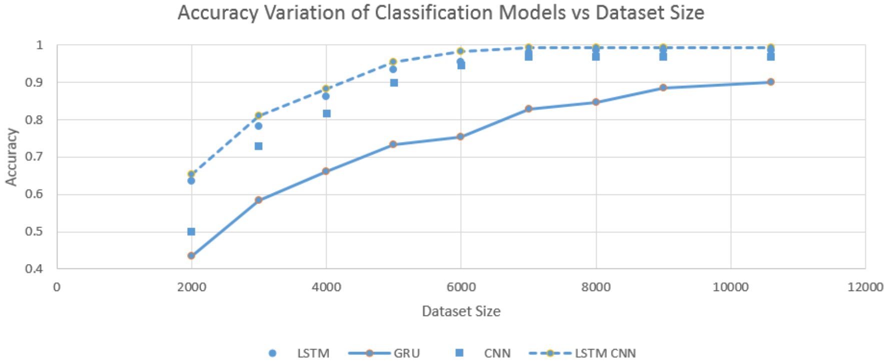

Figure 5.3: Variation of Accuracy of Classification Models versus Dataset Size

Figure 5.4 shows the variation of accuracy value as the dropout percentage varies for LSTM, GRU, CNN and CNN LSTM classification models. Generally, the accuracy value is decreasing (loss is increasing) as the dropout percentage is

increasing. For LSTM, the accuracy value decreased from 0.9321 to 0.6102 as the dropout percentage increased from 20 to 50. For GRU, the accuracy value decreased from 0.8403 to 0.5705 as the dropout percentage increased from 20 to 50. For CNN, the accuracy value decreased from 0.9342 to 0.5409 as the dropout percentage increased from 20 to 50. For LSTM CNN, the accuracy value decreased from 0.9542 to 0.7109 as the dropout percentage increased from 20 to 50.

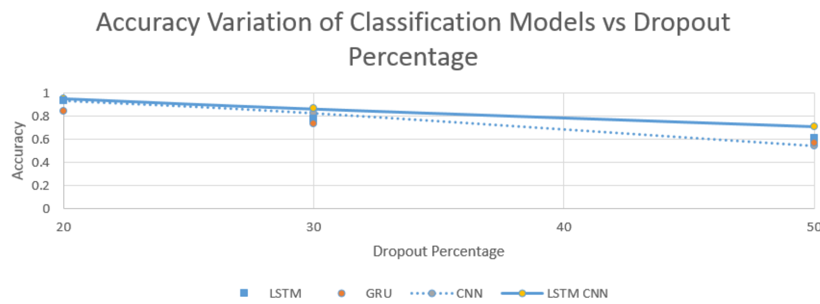

Figure 5.4: Variation of Accuracy of Classification Models versus Dropout Percentage

Figure 5.5 shows the variation of accuracy value as the dropout percentage varies for LSTM, GRU, and CNN LSTM classification models. Generally, the accuracy value is increasing (loss is decreasing) as the sequence length is increasing. For LSTM, the accuracy value increased from 0.6241 to 0.9512 as the sequence length increased from 500 to 2000. For GRU, the accuracy value increased from 0.5343 to 0.8901 as the sequence length increased from 500 to 2000. For CNN LSTM, he accuracy value increased from 0.7541 to 0.9931 as the sequence length increased from 50 to 1500. For the latter model, the accuracy value did not change as the

sequence length increased from 1500 to 2000.

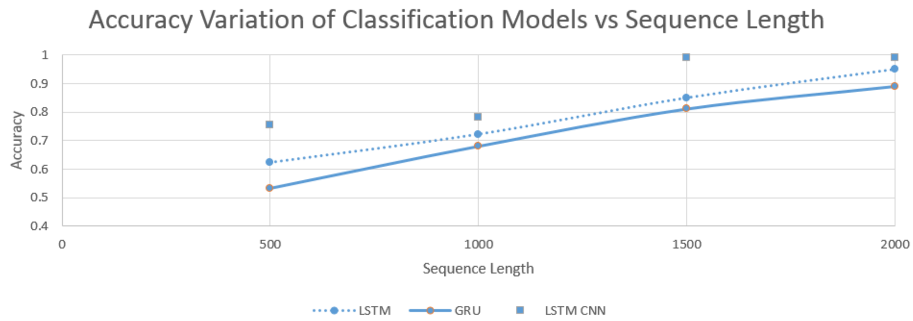

Figure 5.5: Variation of Accuracy of Classification Models versus Sequence Length

Figure 5.6 shows the variation of accuracy value as the number of filters varies for CNN and CNN LSTM classification models. Generally, the accuracy value is increasing (loss is decreasing) as the number of filters is increasing. For CNN, the accuracy value increased from 0.5243 to 0.9453 as the number of filters increased from 3 to 12. For LSTM CNN, the accuracy value increased from 0.6243 to 0.9931 as the number of filters increased from 3 to 9. To note that increasing the number of filters from 9 to 12 did not affect the accuracy value for the LSTM CNN hybrid model.

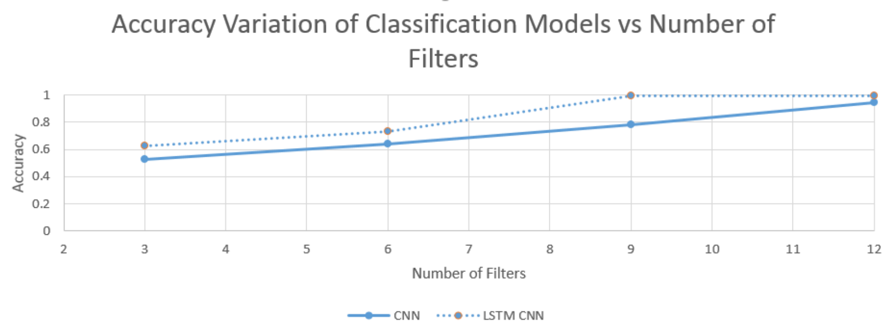

Figure 5.6: Variation of Accuracy of Classification Models versus Number of Filters

Figure 5.7 shows the variation of accuracy value as the kernel size varies for CNN and CNN LSTM classification models. Generally, the accuracy value is increasing (loss is decreasing) as the kernel size is increasing. For CNN, the accuracy value increased from 0.6459 to 0.9614 as the kernel size increased from 5 to 25. For LSTM CNN, the accuracy value increased from 0.7608 to 0.9931 as the kernel size increased from 5 to 20.. To note that increasing the kernel size from 20 to 25 did not affect the accuracy value for the LSTM CNN hybrid model.

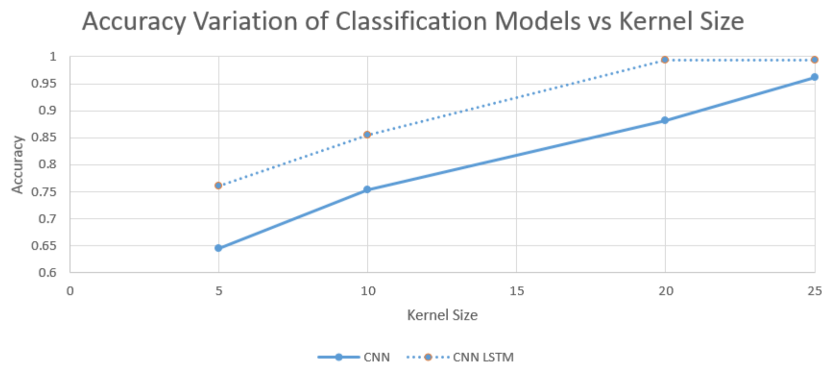

Figure 5.7: Variation of Accuracy of Classification Models versus Kernel Size

5.2.4 Parameters Validation

We study the loss value of the aforementioned hyper-parameters. We start by varying the number of LSTM layers versus the dataset size and sequence length taken into consideration from each assembly input file (number of actions). As shown in figure 5.8, generally the loss value is decreasing as the dataset size increases. For 1 LSTM layer, the loss value decreased from 0.82 approximately 0.4 as the dataset size increases from 2000 to 10600 samples. For 2 LSTM layers, the loss value decreased from 0.63 approximately 0.19 as the dataset size increases

from 2000 to 10600 samples. For 4 LSTM layers, the loss value decreased from 0.44 approximately 0.07 as the dataset size increases from 2000 to 7000 samples. For 4 LSTM layers, the loss value remains the same as the dataset size increased from 7000 to 10600 instances.

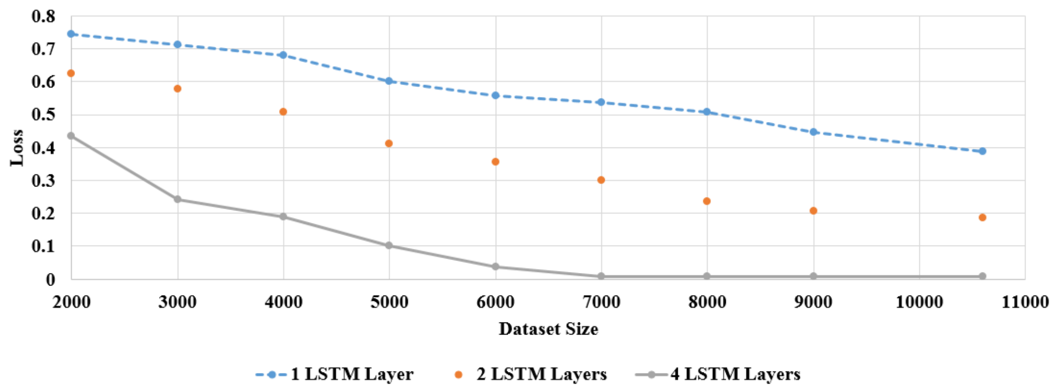

Figure 5.8: Variation of Loss versus dataset size for multiple number of LSTM layers

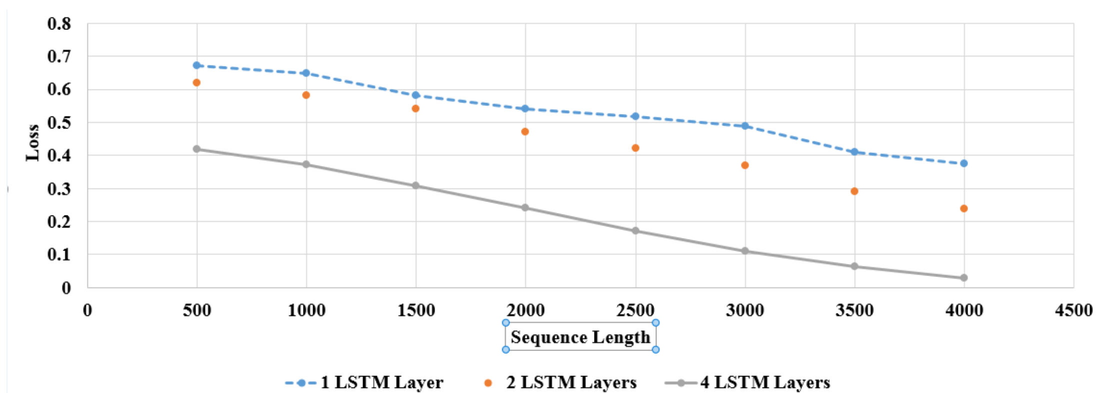

Figure 5.9: Variation of Loss versus dataset size for multiple number of LSTM layers

The embedding algorithm is an important factor of the leaning model as it helps the model to learn complex pattern from unstructured data aiming for better classification. We have tried different embedding algorithms for our deep

learning classification model such as FastText, word2vec, Glove, and a new trainable embedding algorithm that starts from random matrix and adjust its values accordingly.

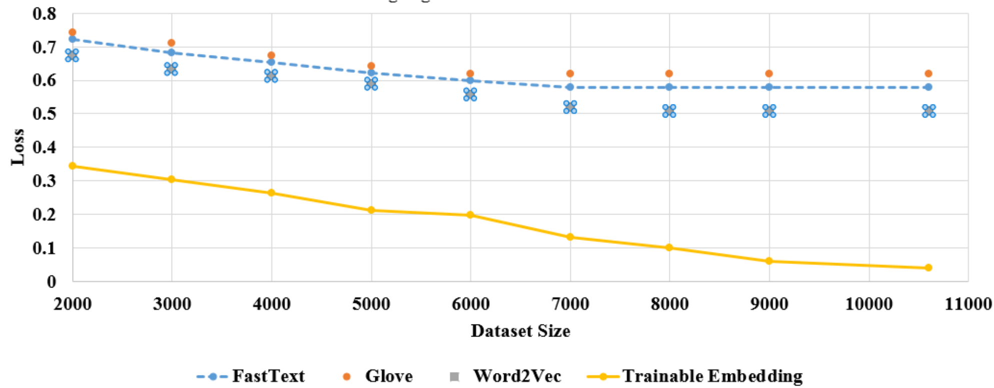

Figure 5.10: Variation of Loss versus dataset size for different word embedding algorithms

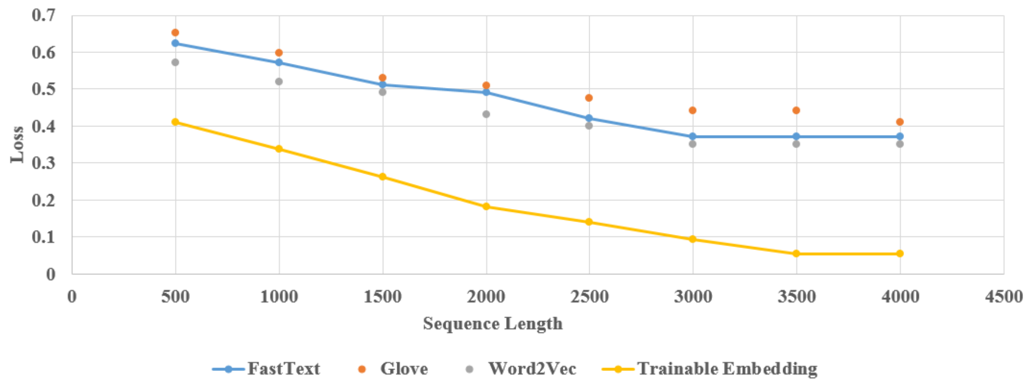

Figure 5.11: Variation of Loss versus dataset size for different word embedding algorithms

It turns out the already implemented embedding algorithms did not work well in our problem since they are trained on English Text rather than assembly actions. The new trainable embedding algorithm returned the best performance among all the embedding algorithms as shwon in Figures 5.10 and 5.11. In those

figures, we show the performance of different word embedding algorithms versus dataset size and sequence length as well. The loss value decreases as the dataset size and sequence length increases.

Dropout is a regularization technique used in deep learning to avoid overfitting. The idea behind dropout technique is to drop a specific number of the nodes in the network to reduce its complexity aiming to avoid overfitting. We tried different dropout percentages that varies from 20% to 50% versus the dataset size and sequence length as shwon in figures 5.12 and 5.13.

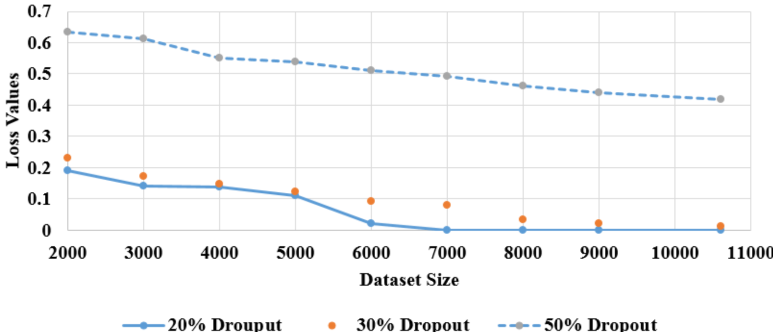

Figure 5.12: Variation of Loss versus dataset size for multiple dropout percentage values

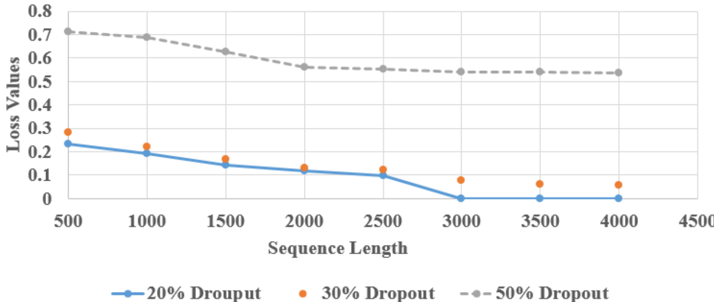

Figure 5.13: Variation of Loss versus dataset size for multiple dropout percentage values

In both experiments, the loss value decreases the dataset size and sequence length increase. Comparing the dropout percentage used, we can see that the loss value remains very high for 50% which led to a very poor classification model since the network is not capable of learning any pattern from the data. As for the dropout percentage of 20%, the loss value reaches ***zero***, but the model did not return good results which implies that the model is overfitting on the training dataset with dropout percentage of 20%. The best value for dropout percentage is 30% which decreases the loss value and did not overfit on the training dataset.

For the convolutional neural networks, we have varied the number of CNN filters of our model and studied their impact on the classification model versus the dataset size and sequence length. The loss value decreases as the dataset size and sequence length increases. The different values of CNN filters used are 3,6,9, and 12. The loss value reaches ***zero*** with 12 filters which turned out to be overfitting on the training dataset on both experiments. The best value for CNN filters is 9 in which it dcreases the loss value and did not overfit on the training dataset. The results obtained for the CNN filters are shown on figures 5.14 and 5.15.

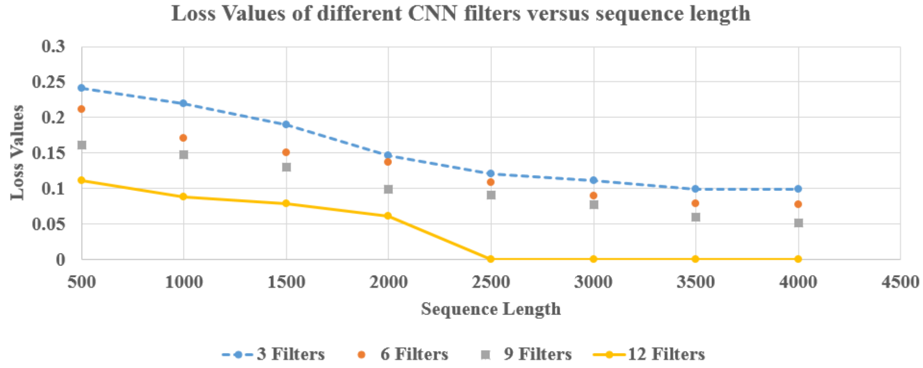

Figure 5.14: Variation of Loss versus dataset size for multiple CNN filters

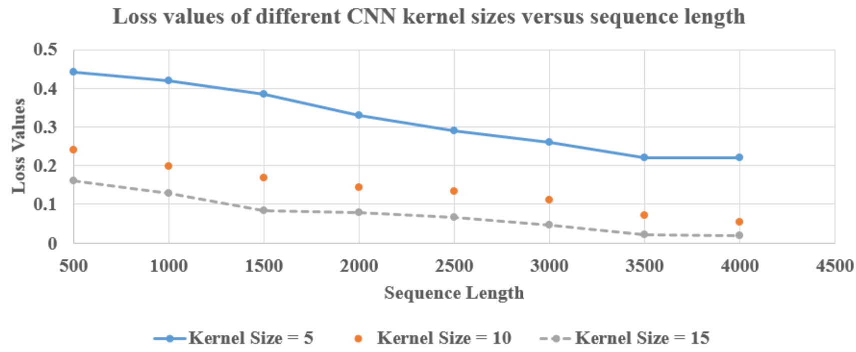

Figure 5.15: Variation of Loss versus dataset size for multiple CNN filters

On the other hand, an important hyper parameter for CNN architecture is the kernel size. Similarly, we varied the kernel size versus the dataset and the sequence length. The loss value of the model decreases as the dataset size and the sequence length increases. The model returned the best results when the kernel size is 15 as shown in figures 5.16 and 5.17.

Table 5.1 summarizes the best values for the different hyper parameters used in our hybrid 1D CNN LSTM model.

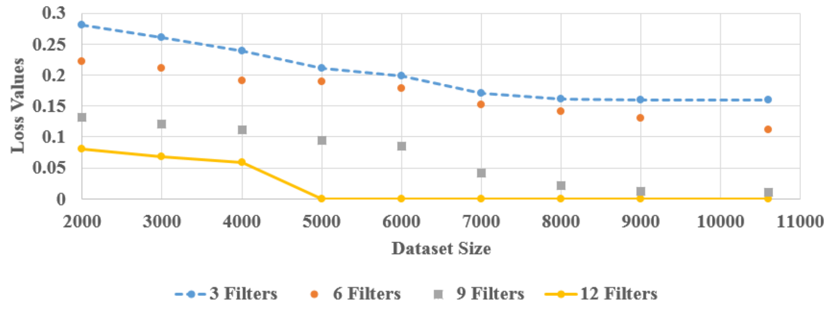

Figure 5.16: Variation of Loss versus dataset size for multiple kernel size values

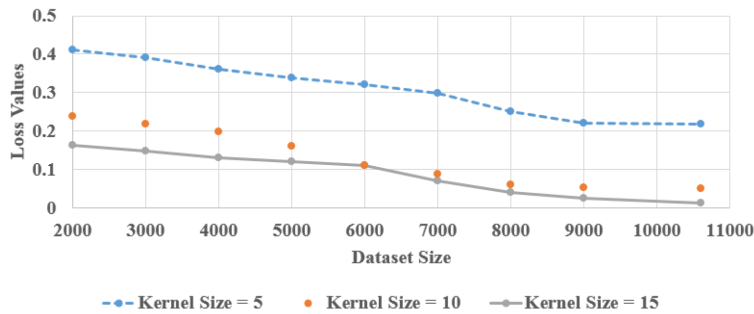

Figure 5.17: Variation of Loss versus sequence length for multiple kernel size values

Hyper Parameter	**Value**
Number of LSTM layers	4
Embedding Algorithm	Trainable Embedding
Dropout	30%
CNN Filters	9
Kernel Size	15

Table 5.1: Summary of best values for different hyper-parameters of our classification model

5.2.5 Analysis

In this section, we will analyze the results obtained using different four architectures of our deep learning based classifiers. Initially, we will start by comparing our architectures to our developed machine learning classifier. Then, we will compare our different models to each other and to previous proposed solutions discussed in Chapter 3.

Table 5.2 summarizes the accuracy values reached by our different models. The logistic regression and support vector machines classifiers accuracy values are 68% and 74% respectively. The lowest accuray value reached among the different deep learning based models is 95%. This value was reached by the GRU model. CNN based model recorded a better accuracy value of 96%. Also, our LSTM based architecture performed better than GRU and CNN models by achieving an accuracy value of 98.544%. The best accuracy value was achieved by our hybrid CNN-LSTM based architecture. This architecture outperformed all the aforementioned approaches and recorded an accuracy value equals to 99.31%. This is due to the fact that the CNN layer is enhancing the model capability of learning more complex patters by treating the malware as a sequence. The output of the CNN is serving as a starting point to the LSTM layers for further analysis as described in our proposed approach. We notice that the number of LSTM layers used in the hybrid approach is 2 and not 4 as opposed to other approaches. Thus, using CNN is increasing the learning capability of the model while reducing the performance complexity of the system.

Classifier	Accuracy Value
Logistic Regression	68%
Support Vector Machine	74%
GRU	95%
LSTM	98.544%
CNN	96%
CNN-LSTM	99.31%

Table 5.2: Summary of Accuracy Values of Different Architectures of our Model

We notice that our CNN-LSTM model outperforms LSTM one using a dataset of 7000 instead of 10600 samples. Also, the number of LSTM layers used in the hybrid approach is 2 instead of 4. Thus, we can conclude that the hybrid architecture is better than LSTM one in terms of accuracy and hyper parameters values. Figure 5.18 depicts the achieved accuracy values for all models.

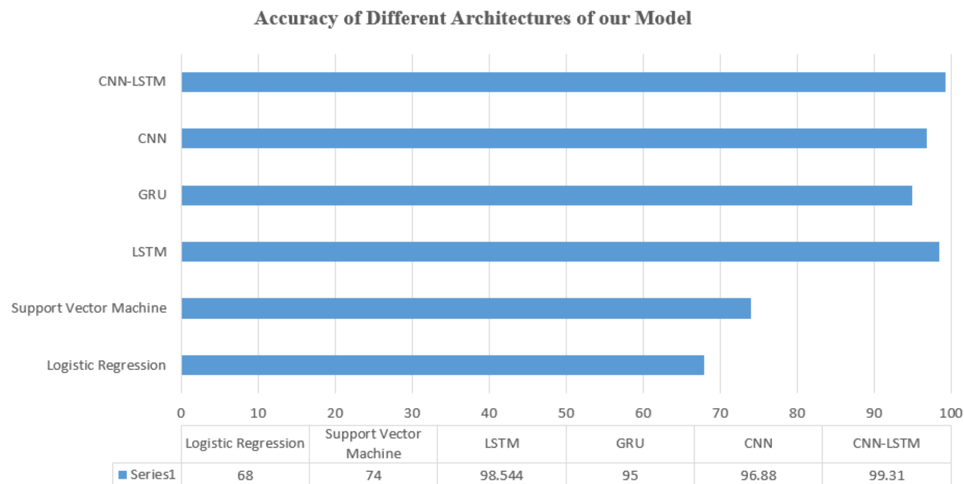

Figure 5.18: Summary of Accuracy Values of Different Architectures of our Model

Comparing our deep learning approaches to our baseline represented by lo-

gistic regression and support vector machine algorithms, we can notice that ;in the worst case; deep learning based classifiers outperforms machine learning algorithms by 22% approximately. In the best case scenario, deep learning technologies is better than machine learning algorithms by almost 25%. The accuracy obtained in [49] is 99.78% which is slightly higher than the accuracy obtained by our hybrid model. This can be due to the fact that in [49], the malware is represented as a gray scale image and the features are extracted from each pixel in the image using convolutional neural network which adds more feature engineering methods to their model. This will lead in a more complex model that needs more data and consequently more time to train, build, and validate the underlying classification model. On the other hand, the hybrid model presented in this thesis uses a predefined set of assembly instructions to extract the actions that a malware intends to execute on a computing system as features. This way, the model will be less complex and the it will needs less time to be trained and validated.

The set of experiments of the different architectures of the deep learning based classification components were conducted on an NVIDIA Quadro P2000 Graphical Processing Unit (GPU) that has a total of 1024 CUDA cores, 5 GB of GPU memory, 140 GB of GPU memory bandwidth, and Theano as a backend for GPU. As for the implementation, the parser is implemented using Python and the classification component using Keras as a deep learning framework. The batch size of the experiments is highly dependent on the specifications of the GPU

being used. Consequently, it affects the execution time of the classification model inversely proportional. As the value of the batch size increases the execution time decreases. On the other hand, the execution time varies proportionally to the dataset size. Thus, the execution time of the classification model increases as the dataset size increases.

The values used for the batch size are {10, 20, and 30} for dataset size of 10600 malware instances. Table 5.3 shows the execution time (in minutes) of the proposed model versus the different aforementioned batch size values. The execution time of the model is 498 minutes when the batch size is 10. As the batch size increases from 10 to 30, the execution time decreases from 498 to 368 minutes. The highest value used for the batch size is 30 due to memory constraints. Memory allocation errors were thrown when trying to use a batch size value greater than 30.

Batch Size	**Execution Time**
10	498
20	407
30	368

Table 5.3: Execution time versus batch size

Bias and variance are two main sources of error in machine and deep learning. An ultimate classifier must not suffer neither from high bias nor high variance. Thus, bias and variance should be optimized to have minimal values. The bias value is considered the same as the error of the model on the training referred to

as training error dataset. Thus, having a high training error indicates high bias. On the other hand, variance is defined as the difference between the training error and the error on the development dataset referred to as development error. In general, there exist a trade off between bias and variance. Techniques that are applied to reduce bias result in high variance and vice versa.

Learning curves are designed to plot the error of the model versus the size of the training dataset. These curves provide good significant if the classification model suffers from high bias or high variance. Figure 5.19 shows that the training error of our 1D CNN LSTM classifier increases from 0 to 21% as the dataset size increases from 500 to 7000 samples. Thus, the classification model has high bias. On the other hand, the training error remains the same as the dataset size increases from 7000 to 10600 instances. The same figure shows that the development error decreases from 60% to 23% as the dataset size increases from 500 to 7000 instances. Also, the development error is not affected as the dataset size increases from 7000 to 10600 samples. This indicates that increasing the datatset size helps in avoiding high variance in the classification component.

To avoid high bias, we increased the complexity of the model size by increasing the number of neurons in our LSTM network to 4 instead of 2. Also, we decreased the dropout percentage from 50% to 10%. This results in decreasing the bias from 21% to 0.5% and thus the classification model does not suffer from high bias. On the other hand, the consequences of these techniques is that the variance value is increased to 22.5% (23-0.5). In order to have an acceptable variance value,

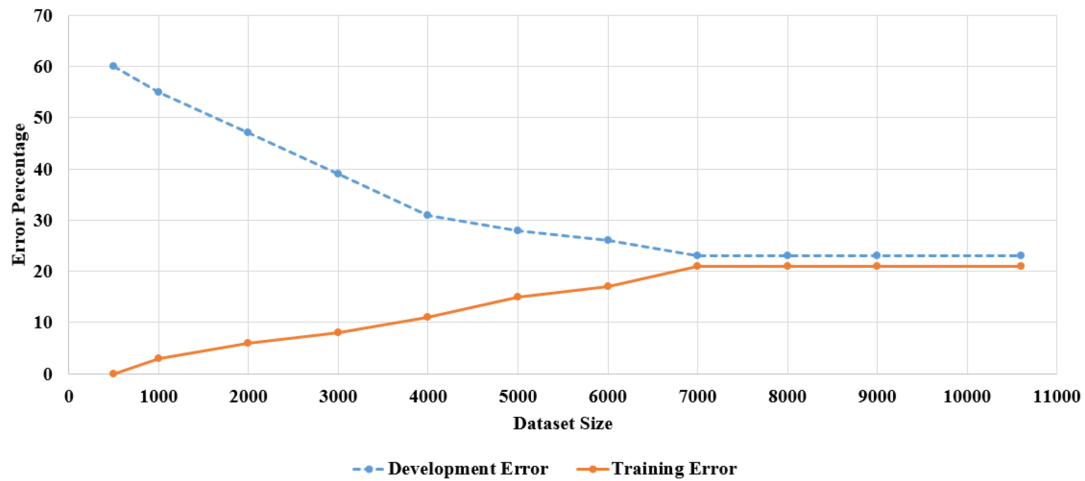

Figure 5.19: Learning Curve of Hybrid CNN LSTM Classifier

we started increasing gradually the dataset size from 500 7000 instances. Also, we increased the dropout value from 30% to 10% which led to a development error of 3%. Thus, the new variance value is 2.5% which is acceptable for our classification model. As a consequent to increasing the complexity of our hybrid classification component and the dataset size, the execution time of our model increased from 2 to 16 hours approximately but without leading to computational or memory issues.

Chapter 6

Conclusions

In this thesis, we presented a new approach for malware detection classification based on deep learning technologies. Our approach is composed of two main components: Feature extraction component and a classification component that has different architectures. We defined a new language for malware based on the concept of word, document, and vocabulary. The feature extraction component is based on the aforementioned malware specific language. As for the classification component, multiple architectures were implemented and tested. We started by developing two machine learning based classification components to set them as a baseline in our performance evaluation study. The first machine learning architecture is based on logistic regression and the second one is based on support vector machines. The SVM based component outperforms the logistic regression one by reaching an accuracy value of 74%. Thus, our baseline accuracy is 74%.

The first and second architectures are fully dependent on LSTM and GRU respectively. We trained and tested our different models using Microsoft malware dataset. By running experiments using these two models, we noticed that LSTM

achieves a higher accuracy from GRU but that the execution time of GRU is less than LSTM. This uis due to the fact that the mathematical equations of LSTM are more complex than GRU and needs more time to be computed. Also, the dataset is smaller, the GRU model outperforms the LSTM one. The third model is based solely on one dimensional convolutional neural network (1D-CNN). This model is considered between both LSTM and GRU model. It reaches a higher accuracy value than GRU but less than LSTM. The fourth and ultimate approach is based on a 1D convolutional neural network (1D-CNN) and Long Short Term Memory (LSTM) followed by a deep feed forward neural network. This architecture outperforms all the aforementioned ones by reaching an accuracy value of 99.31%.

Comparing our deep learning approach to our machine learning baseline, we can conclude that deep learning raised the performance 21% and 25.31% in the worst and best case scenarios respectively. Also, the hybrid 1D-CNN LSTM based approach is the best architecture for the classification component of our automated solution for malware detection and classification.

Appendix A

Abbreviations

RNN	Recurrent Neural Networks
LSTM	Long Short Term Memory
CNN	Convolutional Neural Networks
SVM	Support Vector Machine
API	Application Programming Interface
SEA	Stacked AutoEncoder
PTT	Propagation Through Time
BPTT	Back Propagation Through Time
ANN	Artificial Neural Networks
NB	Naive Bayes
MNB	Multi-Naive Bayes
DT	Decision Tree
DNN	Deep Neural Networks
MLP	Multi-layer Perceptron
NN	Neural Networks

GRU	Gated Recurrent Units
DNN	Dense Neural Network
NLP	Natural Language Processing
HMM	Hidden Markov Model
ESN	Echo State Network
LR	Linear Regression
RF	Random Forest
TFIDF	Term Frequency Inverse Document Frequency
GPU	Graphical Processing Unit

Appendix B

Mathematical Notations

- x_i^t is the input variable of RNN a time step **t**
- $s_i^{(t)}$ is the internal state of RNN at time step **t**
- $o_i^{(t)}$ is the output of RNN at time step **t**
- b is a bias vector
- U is input layer to hidden layer matrix
- V is hidden to hidden layer matrix
- W is hidden layer to output layer matrix
- σ is the Sigmoid function
- $F_i^{(t)}$ is the forget gate of LSTM cell at time step **t**
- $G_i^{(t)}$ is the external input gate of LSTM cell at time step **t**
- $S_i^{(t)}$ is the internal state of LSTM cell at time step **t**
- $S_i^{(t)}$ is the output gate of LSTM cell at time step **t**

- $h_i^{(t)}$ is the output of LSTM cell at time step **t**
- $\hat{y}^{(t)}$ is the sofmax of $o^{(t)}$
- S_j is the actual per element formula of the softmax activation function
- E_t is the partial loss function of RNN at time step **t**
- E_{total} is the total loss function of RNN over all time steps **t**
- $\frac{\partial E_t}{\partial U_{ij}}$ is the partial derivative of the loss function with respect to U
- $\frac{\partial E_t}{\partial V_{ij}}$ is the partial derivative of the loss function with respect to V
- $\frac{\partial E_t}{\partial W_{ij}}$ is the partial derivative of the loss function with respect to W
- $\bigotimes$ is the symbol that represents outer product

Bibliography

[1] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001*, pp. 38–49, 2001.

[2] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici, "Unknown malcode detection via text categorization and the imbalance problem," in *IEEE International Conference on Intelligence and Security Informatics, ISI 2008, Taipei, Taiwan, June 17-20, 2008, Proceedings*, pp. 156–161, 2008.

[3] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pp. 1357–1365, 2013.

[4] T. Lee and J. Mody, "Behavioral classification (2006)."

[5] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*,

vol. 19, no. 4, pp. 639–668, 2011.

[6] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *2011 International Symposium on Visualization for Cyber Security, VizSec '11, Pittsburgh, PA, USA, July 20, 2011*, p. 4, 2011.

[7] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using OPCODE representation," in *Intelligence and Security Informatics, First European Conference, EuroISI 2008, Esbjerg, Denmark, December 3-5, 2008. Proceedings*, pp. 204–215, 2008.

[8] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007, Proceedings*, pp. 178–197, 2007.

[9] M. Rhode, P. Burnap, and K. Jones, "Early stage malware prediction using recurrent neural networks," *arXiv preprint arXiv:1708.03513*, 2017.

[10] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[11] C. Robert, "Machine learning, a probabilistic perspective," 2014.

[12] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[13] S. Marsland, *Machine learning: an algorithmic perspective*. CRC press, 2015.

[14] S. Minton, *Machine learning methods for planning*. Morgan Kaufmann, 2014.

[15] B. K. Natarajan, *Machine learning: a theoretical approach*. Morgan Kaufmann, 2014.

[16] S. Rogers and M. Girolami, *A first course in machine learning*. CRC Press, 2016.

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

[18] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[19] "Application domains of recurrent neural netowrks."

[20] "Recurrent vs unfolded representation of recurrent neural networks."

[21] "Hyperbolic tangent activation function."

[22] "Softmax activation function."

[23] "Gradients for rnn."

[24] "Derivative of hyperbolic tangent activation function."

[25] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017.

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

[27] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[28] M. F. Zolkipli and A. Jantan, "An approach for malware behavior identification and classification," in *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, vol. 1, pp. 191–194, IEEE, 2011.

[29] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings*, pp. 108–125, 2008.

[30] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *Computer*

Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual, vol. 2, pp. 577–582, IEEE, 2016.

[31] I. Firdausi, A. Erwin, A. S. Nugroho, *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, pp. 201–203, IEEE, 2010.

[32] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in api calls with machine learning algorithms for malware detection," in *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, pp. 55–62, ACM, 2009.

[33] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pp. 23–30, IEEE, 2010.

[34] B. Kolosnjaji, A. Zarras, G. D. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *AI 2016: Advances in Artificial Intelligence - 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5-8, 2016, Proceedings*, pp. 137–149, 2016.

[35] J. Liang and R. Liu, "Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network," in *Image and Signal Processing (CISP), 2015 8th International Congress on*, pp. 697–701, IEEE, 2015.

[36] L. Liu and B. Wang, "Automatic malware detection using deep learning based on static analysis," in *Data Science - Third International Conference of Pioneering Computer Scientists, Engineers and Educators, ICPC-SEE 2017, Changsha, China, September 22-24, 2017, Proceedings, Part I*, pp. 500–507, 2017.

[37] R. Boné and H. Cardot, "Advanced methods for time series prediction using recurrent neural networks," in *Recurrent Neural Networks for Temporal Data Processing*, InTech, 2011.

[38] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *10th International Conference on Malicious and Unwanted Software, MALWARE 2015, Fajardo, PR, USA, October 20-22, 2015*, pp. 11–20, 2015.

[39] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "Deepam: a heterogeneous deep learning framework for intelligent malware detection," *Knowledge and Information Systems*, pp. 1–21, 2017.

[40] B. Athiwaratkun and J. W. Stokes, "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*, pp. 2482–2486, IEEE, 2017.

[41] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Acoustics, Speech*

and Signal Processing (ICASSP), 2015 IEEE International Conference on, pp. 1916–1920, IEEE, 2015.

[42] "Word2vec word embedding algorithm."

[43] "Fast text word embedding algorithm."

[44] "Glove word embedding algorithm."

[45] H. S. Yara Awad, Mohamed Nassar, "Modeling malware as a language.," *To appear In : IEEE ICC 2018*, 2018.

[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[47] "Malware detection and classification kaggle competition."

[48] "Term frequency inverse document frequency."

[49] D. Gibert, J. Béjar, C. Mateu, J. Planes, D. Solis, and R. Vicens, "Convolutional neural networks for classification of malware assembly code," in *Recent Advances in Artificial Intelligence Research and Development - Proceedings of the 20th International Conference of the Catalan Association for Artificial Intelligence, Deltebre, Terres de l'Ebre, Spain, October 25-27, 2017*, pp. 221–226, 2017.