PAYLOAD-BASED NETWORK INTRUSION DETECTION USING LSTM AUTOENCODERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Selin Coşan
December 2020

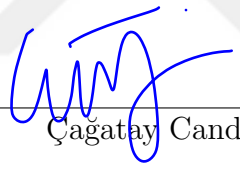
Payload-Based Network Intrusion Detection Using LSTM Autoencoders

By Selin Coşan

December 2020

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)



Çağatay Candan

Aykut Koç

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

PAYLOAD-BASED NETWORK INTRUSION DETECTION USING LSTM AUTOENCODERS

Selin Coşan
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
December 2020

The increase in the use of computer networks by vast numbers of different devices have allowed malicious entities to develop a plethora of diverse attacks, targeting individuals and businesses. The defence systems need to be kept up to date constantly since new attacks emerge daily, in addition to having a wide range of characteristics. Intrusion detection is a branch of cyber-security that aims to prevent these attacks. Machine learning and deep learning approaches gained popularity in this discipline, as they did in many others such as fraud detection and medicine. Given that network traffic usually displays normal behavior, anomaly detection methods can pinpoint threats by identifying connections with abnormal properties. This task can be accomplished in a supervised or an unsupervised manner. Regardless of the path, constructing meaningful representations of network data is essential. In this thesis, we employ different types of feature extraction methods for computer network data and anomaly detection strategies that can detect malicious behaviour. For the feature extraction task, we aim to obtain vector representations of network payloads such that the core information is more reachable and irrelevant information is discarded. In our setting, the input size can vary due to the nature of the computer network data. Considering this, we use feature extraction methods that can map inputs of varying sizes into feature spaces with fixed dimensionality so that some machine learning approaches, that are otherwise unusable in these settings, can be employed. For the anomaly detection task, we utilize both supervised and unsupervised approaches. The supervised methods make use of the aforementioned feature extraction strategies and use the reduced and fixed dimensional representations of the computer network data. For the unsupervised case, we employ autoencoders that can extract information from sequential data. Recurrent neural networks(RNNs) can process sequential data with varying length. We specifically use autoencoders with long short-term memory(LSTM), which is a special form of RNNs with a more complex

structure that allows them to handle long-term dependencies in sequential data. Then, anomaly detection is performed using reconstruction error. We conduct experiments using dynamic and realistic data sets, which consist of various types of attacks. Then, we evaluate the validity of our proposed approaches based on AUC and F1 measures.

Keywords: Intrusion detection, anomaly detection, long short-term memory, deep autoencoders, feature extraction.

ÖZET

LSTM ÖZKODLAYICILAR İLE AĞ YÜKÜ TABANLI İHLAL TESPİTİ

Selin Coşan
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Aralık 2020

Bilgisayar ağlarının çok sayıda farklı cihaz tarafından kullanımındaki artış, kötü niyetli oluşumların, bireyleri ve işletmeleri hedef alan çok sayıda farklı saldırılar geliştirmesine olanak sağlamaktadır. Saldırılar çok çeşitli karakteristik yapılara sahip olmanın yanı sıra her gün yeni formlarda ortaya çıktığı için savunma sistemlerinin sürekli güncel tutulması gerekmektedir. Saldırı tespiti, bu saldırıları önlemeyi amaçlayan bir siber güvenlik dalıdır. Makine öğrenimi ve derin öğrenme yaklaşımları, dolandırıcılık tespiti ve tıp gibi diğer pek çok alanda olduğu gibi bu disiplinde de popülerlik kazanmıştır. Ağ trafiğinin genellikle normal davranış sergilediği göz önüne alındığında, anomali tespit yöntemleri, anormal özelliklere sahip bağlantıları belirleyerek tehditleri tanıyabilmektedir. Bu görev, denetimli veya denetimsiz bir şekilde gerçekleştirilebilir. Anomali tespit yönteminin denetimli veya denetimsiz olması, ağ verilerinin anlamlı temsillerini oluşturmanın çok önemli olduğu gerçeğini değiştirmemektedir. Bu tezde, bilgisayar ağ verileri için farklı özelliklerde öznitelik çıkarma yöntemleri ve kötü niyetli davranışları tespit edebilen anomali tespit stratejileri kullanılmıştır. Öznitelik çıkarma görevi için, temel bilgileri daha erişilebilir kılacak ve alakasız bilgileri ayıklayacak şekilde ağ yüklerinin vektör temsillerinin elde edilmesi amaçlanmıştır. Bizim problem tanımımızda, girdi boyutu bilgisayar ağı verilerinin doğası gereği değişken olabilir. Bunu göz önünde bulundurarak, farklı boyutlardaki girdileri sabit boyutluluğa sahip öznitelik uzaylarına eşleştirebilen öznitelik çıkarma yöntemleri kullanılmıştır. Böylece bu ortamlarda varsayılan halleri ile kullanılması mümkün olmayan bazı makine öğrenimi yaklaşımları kullanılabilir hale getirilmiştir. Anomali tespiti görevi için hem denetimli hem de denetimsiz yaklaşımlardan yararlanılmıştır. Denetimli yöntemler, yukarıda bahsedilen öznitelik çıkarma stratejilerinden faydalanır ve bilgisayar ağı verilerinin indirgenmiş ve sabit boyutlu temsillerini kullanır. Denetimsiz durum için, sıralı

verilerden bilgi çıkarabilen özkodlayıcılar kullanılmıştır. Tekrarlayan sinir ağları (RNN'ler), değişen uzunluktaki sıralı verileri işleyebilir. Özellikle, sıralı verilerdeki uzun vadeli bağımlılıkları işlemelerine olanak tanıyan daha karmaşık bir yapıya sahip özel bir RNN biçimi olan uzun kısa süreli belleğe (LSTM) sahip özkodlayıcılar kullanılmıştır. Ardından, yeniden yapılandırma hatası kullanılarak anomali tespiti gerçekleştirilir. Çeşitli saldırı türlerinden oluşan dinamik ve gerçekçi veri kümeleri kullanılarak deneyler yapılmıştır. Sonrasında, AUC ve F1 ölçümlerine dayalı olarak önerilen yaklaşımlarımızın geçerliliği değerlendirilmiştir.

Anahtar sözcükler: İhlal tespiti, anomali tespiti, uzun kısa-süreli bellek, öznitelik çıkarımı, derin özkodlayıcılar.

Acknowledgement

First, I would like to thank my advisor, Prof. Dr. Süleyman Serdar Kozat, for his guidance throughout my M.S. study.

In addition, I would like to thank Prof. Dr. Çağatay Candan and Dr. Aykut Koç for being in my thesis committee and their time.

I am extremely grateful to Cem Bulucu for his endless support and always being ready to help with every obstacle I encountered during this journey. I am also grateful to my friends Emir Ceyani and Nuri Mert Vural for their great teamwork and valued conversations.

Finally, I would like to thank my family, especially my brother, for their love and support throughout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid increase in the number of devices that utilize the Internet and other public or private computer networks, the frameworks that support these connections are becoming more vulnerable to attacks. The main reason for vulnerability is that the behavior of network traffic is constantly changing. In other words, new network attacks emerge daily, so the definition of normal or anomalous behavior for network traffic varies. Due to this highly variable nature of the network data, it is extremely difficult to precisely evaluate a particular algorithm for detecting attacks.

The discipline that focuses on effective and efficient detection and prevention of threats is called intrusion detection [1]. As the intrusion problem is very critical to ensure the digital safety of individuals and businesses, many counter measures are proposed to overcome this challenge such as statistical models or machine learning based approaches for modeling anomalous behavior. Statistical models work well when the attackers' behaviour is predictable, i.e, the attacks are known. However, as mentioned before, the attacks come in many different types and they do not fit a pattern due to new attack types. Therefore, it may not be practical or even possible to construct well defined models for all types of attacks. Even if it was possible to model every attack that exists as of now, unknown attacks may not be represented by previously designed models.

As a result, similar to many other fields such as signal processing, data driven methods gained popularity lately in this domain as well. The interest in data driven methods is justified since in presence of enough data, these methods can model attacks without the need of detailed expert input and under very mild mathematical assumptions. In fact, rather than trying to model every attack type, aiming to model network behavior that is known to be safe and rejecting abnormal instances is a technique that is used both in the literature and in practice [2]. This framework is called anomaly detection. Assuming non-malicious behaviour to be the common type of behaviour in a computer network, any malicious behaviour can be thought of as an anomaly. Anomaly detection methods are often utilized to filter out instances with undesirable or uncommon characteristics. There is a plethora of research on machine learning based anomaly detection methods such as isolation forests [3] and one-class support vector machines(OC-SVM) [4]. However, the main drawback of these methods, especially in domains like intrusion detection, is that they require experts to extract features from statistical quantities. Moreover, they are not suitable for tasks that require consideration of time dependencies. Deep learning models, on the other hand, can directly work on raw or minimally preprocessed data and they are compatible with feature extraction architectures such as convolutional and recurrent layers which allows systems to make use of spatial and temporal properties of data. In addition to being versatile in terms of data formatting, deep learning models also achieve state of the art performance in many tasks, as they can discover highly complex relations which are not intuitive to humans through neither experience nor statistical examinations. In this thesis, the aim is to detect malicious behavior, specifically intrusions, in computer networks using anomaly detection frameworks powered by deep neural networks.

Anomaly detection is used in various disciplines such as cyber-security, fraud detection and medical diagnosis[5, 6, 7]. As with any other approach that requires precision, extraction of truly useful features is of utmost importance in anomaly detection. Effective representations can be extracted from sequential data by using methodologies like term frequency-inverse document frequency(tf-idf) or n-grams [8, 9]. Similarly, sequences with semantic relationships can be efficiently

represented using Word2Vec models [10]. These representations can be utilized efficiently with architectures that consider time dependencies like recurrent neural networks(RNNs) [11]. For example, tf-idf weighting scheme can be very useful in scenarios where there are a lot of common items that are not interesting and very few items that are much more helpful for learning. Note that, in this thesis we aim to employ these strategies, which we will provide details about later.

A common approach that is used in the anomaly detection literature is to learn a decision function that outputs an anomaly indicator score[12]. This decision function can be achieved through determination of a model and optimization of model parameters according to a loss function, such as hinge loss for support vector machines or mean squared error for autoencoders. One-class support vector machines is an example of this process [13]. However, these kinds of approaches such as one-class support vector machines fall short in presence of sequential data with varying length, since they require vector representations of data. Therefore, for payload-based intrusion detection, they lack the ability of incorporating time dependencies and rapidly changing network traffic[14]. These limitations pave the way to the utilization of deep learning methods in intrusion detection.

In its simplest form, deep learning can be performed by multi-layer perceptrons with many hidden layers. Unfortunately, using a fully connected deep neural network cannot fully exploit the sequential nature of time series data[15]. RNNs are increasingly used in applications where sequential data is present [16]. Once again unfortunately, simple RNN structures are not sufficiently equipped to perform this task the way its meant to be due to vanishing gradient problem. Although simple RNNs can capture time relations in shorter sequences, they fail to do so in longer ones. Long short term memory(LSTM) architecture is a variation of the RNNs that allow better feature extraction when there are long pauses between important events in data[17].

Unlike classification and regression tasks, there is no straightforward way to program neural networks to detect anomalies. At this point, autoencoders can directly be used for anomaly detection by applying a threshold to the reconstruction error. In addition, they can be used to extract features of reduced dimensionality,

which can be used for classification by other approaches like SVMs or random forest(RF). The main use of autoencoders is to produce a representation of the data that has lower dimensionality than the original data, but its direct use as an anomaly detector in our case is as follows.

Autoencoders are made up of two main components: encoder and decoder. They take a time series data as input, process it through LSTM or convolutional layers and encode it such that its dimensionality is reduced. The main idea behind this process is to eliminate irrelevant information and extract the core, meaningful representation of the data. After the encoding step, the encoded features are then fed into a decoder, which tries to reproduce the input data at its output. In other words, it tries to undo what the encoder does and aims to use the lower dimensional representation of the data to construct the original input. In our framework, when a test input comes, the autoencoder processes the test input through the encoder and the decoder, respectively. At the output, a reconstructed version of the input data is created. The reconstruction error is then used as an indicator of anomaly. Note that, this anomaly indication score is meaningful because the autoencoder is trained with data having mostly normal instances and thus learns to model normal behavior. Also note that, actually the training on only normal data is preferred, since the aim is to model normal behavior alone. However, in a realistic framework, training data may include anomalous data instances with very low ratio. All that is needed for a good training session is the insurance of low ratio of anomalies in the training set and enough number of samples that nicely represents the distribution of the normal samples. The assumption is that, in presence of an anomalous input, the autoencoder will fail to reconstruct properly and yield a high reconstruction error. After the reconstruction error, i.e., the anomaly score is calculated, anomaly score is compared against a threshold value. If the anomaly score of an instance is larger than the threshold value, than that instance is labeled as an anomaly. This threshold value can be selected according to the requirements of the system. For example, if the system needs to have a low false alarm rate, then a higher threshold should be selected.

In addition to deep autoencoders, other feature extractors such as n-grams

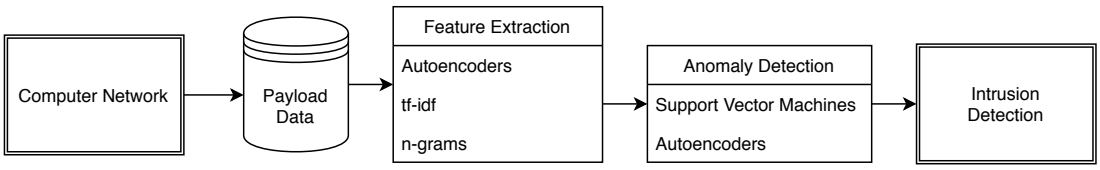


Figure 1.1: Main components of the intrusion detection system

with tf-idf and Word2Vec are going to be used in our setting. Then, we take features obtained from different models and use them as inputs to other algorithms such as SVM and RF. Besides this supervised framework, we also perform anomaly detection in an unsupervised manner. In our unsupervised framework, the system observes payload data that is captured from computer networks and decides whether the behaviour is malicious or not. For this purpose, we employ deep autoencoders as anomaly detectors. In Fig. 1.1, the architecture of the system can be seen.

In this study, LSTM autoencoder structure is employed as a sequential autoencoder. The reasons behind this choice are that the failure of RNNs when lags between time events occur and the known problem of RNN being vulnerable to exploding and vanishing gradients[18].

The data are allowed to contain time series of varying lengths in our setting, but this does not pose a problem as LSTMs are capable of handling inputs with varying length by compressing them into fixed length representations[19]. Even though processing varying sized inputs are possible with regular autoencoders by simply padding them with zeros[20, 21], it is not ideal. Due to its capability of modeling time dependencies, the use of LSTMs is a more suitable choice. In addition, the automatic compression of varying lengths is a pleasant convenience. Another useful property of RNNs, and hence LSTMs, is the use of shared weights between time steps, which in return allows for generalization of the extracted features among different positions in a given sequence.

Besides the LSTM based sequential autoencoder model, we also employ CNN autoencoders in our unsupervised framework. They are mostly employed in image processing applications[22, 23]. They usually yields good performance in varying

applications and have fast training time with the recent development in GPU technology. In addition, they also perform well in NLP tasks. In [24], one dimensional(1D) CNN is designed with residual connections. It captures temporal dependencies in sequential data with variable length. Although their structure resembles RNNs, they are not designed to be used as autoencoders. In another perspective, a 2D CNNs architecture is introduced in [25] to be used in sequence modeling for classification task. In our setting, we also aim to capture semantics from payloads, therefore, CNN is an appropriate alternative to LSTM in this context.

The use of neural networks in intrusion detection systems is investigated in [16]. In [26], LSTMs were utilized for the first time in intrusion detection setting. Also in [27, 28, 29], the possible applications of Support Vector Machines(SVMs), Self-Organized Maps(SOMs) and RF to the network intrusion detection problem is discussed, respectively. One main drawback of these methods is that they require a fixed input length. More importantly, random forests and SVMs are supervised methods that require labeled data. Supervised methods require at least two classes to be trained because the decision boundary is constructed in a manner to distinguish between classes. One could argue that along with normal data, one could feed anomalous data extracted prior to the training of the system, possibly by experts. However, this would again pose two further problems. First problem is that since the intrusion data is considered to be rare, only small amounts of samples can be found which then turns into an imbalanced classification problem. Even though there are methods that deal with imbalanced classification such as undersampling or oversampling, they may not be applicable in our setting. For example, the use of undersampling may not be practical as there would be very few examples left from normal class. In order to match the size of the anomalous class, most samples will be ignored, resulting in a very small training that will lead to easy overfitting. In the case of oversampling, having enough anomalous instances in training set allows efficient representation of both normal and anomalous data. However, the detection of unknown attacks may not be possible, assuming that evolving attacks fits a different pattern.

The second and more important problem about supervised training scheme is

that it may not be practical or possible to gather anomalous data that represent all anomalous characteristics. The supervised learner is limited to distinguishing anomalous data that is drawn from the same distributions in the training set, but it will fail to recognize if anomalous data changes characteristics and yield unstable results. However, with an unsupervised approach as long as the distribution of normal instances stay the same, or drift negligibly, then the changes in anomalous data does not affect the system. Apart from the aforementioned studies, also semi-supervised approaches are considered, such as the one in [30]. Similar to our study, in [31], an LSTM autoencoder is utilized to reconstruct video frames. Also in [32], LSTM and CRF autoencoders are investigated. [33] and [12] also contain unsupervised strategies for the anomaly detection problem. The authors of [34], provide a detailed analysis of the NSL-KDD data set [35] and their work consists of both supervised and semi-supervised frameworks.

A similar architecture to autoencoders is the Restricted Boltzmann machines [36], that can be similarly utilized both in terms of reconstruction and feature extraction. A structure that does not take sequential information into account is given in [37]. In that study, network data is reformatted into images and a convolutional autoencoder is used for the anomaly detection task. In a different hierarchical approach, at first, the raw data is fed into a convolutional neural network, then the output is fed into a recurrent neural network[38]. In [39], the convolutional autoencoder is used while preserving the sequential meaning, however since the loss is computed using the labels, this method can not be properly compared to others we mention that operate in an unsupervised manner.

For the task of intrusion detection, another crucial part is selecting an appropriate data set. Not many data sets are available due to security reasons and the data sets that do exist on networking data generally suffer from heavy anonymization. In this work, we use the ISCX IDS 2012 [40] data set, which provides dynamically generated realistic data. In the process of constructing this data set, real traces are analyzed to create profiles for agents that generate real traffic for a variety of protocols such as HTTP, SMTP and FTP. Similarly, they generate attacks for different scenarios reflecting a real network.

After selecting an appropriate data set and designing intrusion detection systems, evaluation of anomaly detection methods can be tricky. However, investigating the area under curve(AUC) score obtained from the plot of receiver operating characteristic(ROC) is not only a reliable way to both evaluate the algorithms on the test set but also a way to determine hyperparameters such as the threshold for reconstruction error when examined on validation sets. In this setting accuracy is not a good metric because data set is highly imbalanced. Instead, examining the ROC curve, that includes true positive rates and false positive rates, and evaluating based on AUC score is a more robust and meaningful way to test the system. However, to have a finalized system we need to fix the hyperparameters. We determine these hyperparameters using AUC score. Then, the system performance is given by F1 score which is the harmonic mean of the precision and the recall scores.

1.1 Thesis Contribution

- Although payload-based intrusion detection scenarios exist in the literature, we approach the problem from a different perspective by employing algorithms generally used in NLP tasks.
- We perform efficient feature extraction from network payloads using deep autoencoders. While doing so, we consider network payloads as sequential data including semantic information.
- We introduce both a supervised and an unsupervised framework for intrusion detection. Source and destination payloads are utilized together in both settings.
- For the unsupervised setting, the payloads are directly used by CNN and LSTM autoencoders. Then, we perform anomaly detection.
- We evaluate the performance of the proposed algorithms through an extensive set of experiments and provide confidence intervals for our results.

1.2 Thesis Organization

The organization of the thesis is as follows. In Chapter 2, we explain several feature extraction algorithms in detail. We consider tf-idf, Word2Vec and deep autoencoders. In addition, we demonstrate how to incorporate network payloads as inputs in these settings. In Chapter 3, we present our supervised and unsupervised frameworks for intrusion detection. In Chapter 4, we illustrate the performances of the introduced methods by conducting an extensive set of experiments. Finally, we conclude our thesis in Chapter 5.

Chapter 2

Vector Representation of Network Payloads

In this chapter, we study the problem of feature extraction from network payload data. Specifically, we will investigate how n-gram, term frequency-inverse document frequency(tf-idf), Word2Vec and autoencoder approaches can be utilized in the search for meaningful features.

2.1 Vector Space Model

In this section, we will introduce how a vector space model can be used to extract features from network payload data. In particular, we will use n-grams and tf-idf in conjunction. In our study, we consider the size of our dictionary to be the number of n-grams that can be represented by byte sequences. We allow the network payload data to have different lengths, while we obtain fixed length representations for each payload. We also consider that a payload consists of two parts: one that corresponds to the source payload and another that corresponds to the destination payload. Let $\mathcal{X}$ denote the set of payloads and let each payload

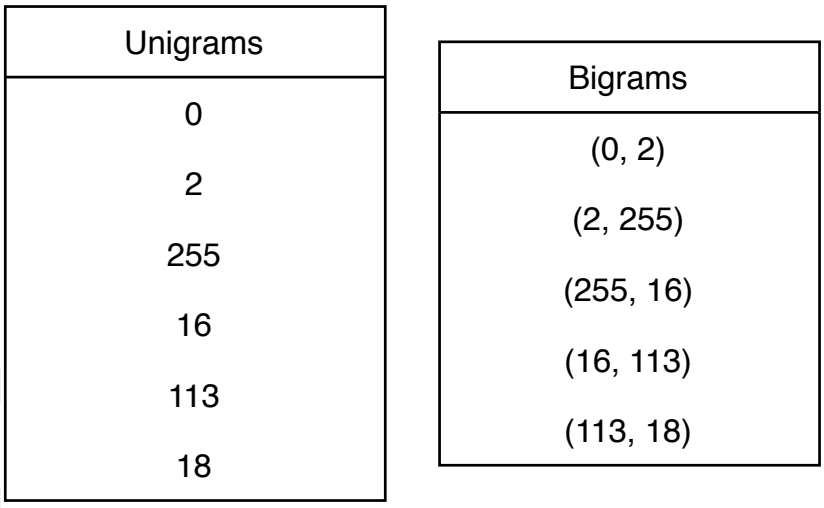


Figure 2.1: An example of unigrams and bigrams for a byte sequence

to be defined as

$$\boldsymbol{x}_t^p \coloneqq [x_{t,1}^p, x_{t,2}^p, \ldots, x_{t,l_t^p}^p]$$

where $p \in \{s, d\}$. s and d are used to identify whether a byte belongs to the source payload or the destination payload, respectively. l_t^p denotes the length of the payload. In addition, $x_{t,i}^p$ denotes the i^{th} byte of the t^{th} data as the payload, for $i = 1, \ldots, l_t^p$. When the superscript p is not present, $x_{t,i}$ can be used in place of both source and destination payloads, meaning any implications on $x_{t,i}$ applies to both the source and the destination bytes. Note that each $x_{t,i} \in \{0, 1, \ldots, 255\}$, since it is a byte. Finally, let $|\cdot|$ denote the cardinality of a set.

An n-gram is a non-interrupted sequence of elements from a given sample. First step is simple, we generate n-grams to populate our dictionary of distinct elements using all the sequences available to us. For example, consider a set of payloads where the elements of a payload are only allowed to have 3 different values: 0, 1 or 2. Then a bigram, where $n = 2$, yields a dictionary that consists of tokens 00, 01, 02, 10, 11, 12, 20, 21 and 22, which makes its total size to be 3^2. In our setting, since each element is a byte, a dictionary of size 256^n will be constructed depending on n. An example containing unigrams and bigrams for a payload vector length of 6 is given in Fig. 2.1.

Let the set $\mathcal{D}_n$ denote a dictionary that is constructed using n-grams of bytes and let $\boldsymbol{d}_{n,i}$ denote the i^{th} element in $\mathcal{D}_n$. Let $a_{t,i}$ denote the number of times the i^{th} element in $\mathcal{D}_n$ occurs in payload $\boldsymbol{x}_t$ and let

$$\boldsymbol{A} \coloneqq \{a_{t,i}\}_{t\in\{1,\ldots,|\mathcal{X}|\},i\in\{1,\ldots,|\mathcal{D}_n|\}},$$

denote the count matrix. Instead of directly using this matrix, the counts are weighted according to their tf-idf measures. Consider a token $\boldsymbol{d} \in \mathcal{D}_n$, to calculate its tf-idf against a payload $\boldsymbol{x}_t$:

$$\text{tf-idf}(\boldsymbol{x}_t, \mathcal{X}, \boldsymbol{d}) \coloneqq \text{tf}(\boldsymbol{x}_t, \boldsymbol{d}) \times \text{idf}(\mathcal{X}, \boldsymbol{d})$$

where $tf(\boldsymbol{x}_t, \boldsymbol{d})$ is the number of times token $\boldsymbol{d}$ is encountered in payload $\boldsymbol{x}_t$. In addition, the inverse document frequency is given by:

$$\text{idf}(\mathcal{X}, \boldsymbol{d}) \coloneqq \log \frac{|\mathcal{X}|}{\text{df}(\mathcal{X}, \boldsymbol{d})}$$

where $\text{df}(\boldsymbol{X}, \boldsymbol{d})$ is the number of payloads that contain at least one instance of $\boldsymbol{d}$. After tf-idf values are calculated, the normalized feature space is given by the following matrix:

$$\bar{\boldsymbol{A}} \coloneqq \{\text{tf-idf}(\boldsymbol{x}_t, \mathcal{X}, \boldsymbol{d}_{n,i})\}_{t\in\{1,\ldots,|\mathcal{X}|\},i\in\{1,\ldots,|\mathcal{D}_n|\}}.$$

This scheme is performed separately on the source and the destination subsets of the payload. After this step, the two representations are merged using mean pooling and then fed into a classifier of choice for supervised classification. The top level schematic of the process is given in Fig. 2.2

Remark 1 *After constructing the tf-idf matrix, each row, and thus the weight vector for each payload, is normalized so that it has unit Euclidean norm.*

Remark 2 *As the lengths of the source and destination payloads for all instances are equalized after the tf-idf step, mean pooling simply takes the average of features that correspond to the same dimensions for the source and destination payloads.*

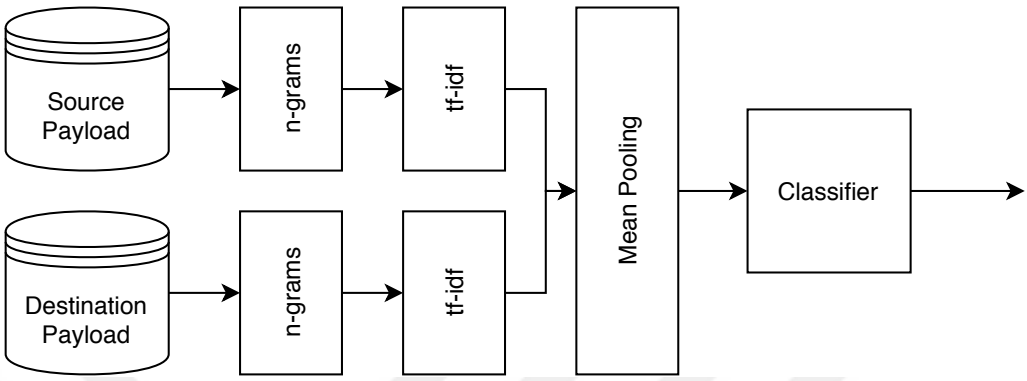


Figure 2.2: Supervised classification scheme that uses vector space model to extract features

2.2 Word2Vec

Word2Vec is a model using shallow neural networks to learn distributed representations of words [41]. Since it is a successful algorithm for capturing semantic relationships in sentences, it is widely used in natural language processing (NLP) tasks. Similar to sentences, network payloads, consisting of bytes instead of words, carry semantic relationships. Therefore, we employ Word2Vec as another feature extraction algorithm.

In order to obtain the vector representations for bytes using Word2Vec model, there are two training algorithms available, namely continuous bag of words (CBOW) and skip gram (SG). CBOW model predicts a word from its context, which is its surrounding words defined by a context window. On the contrary, SG uses a word to predict its context. In terms of the quality of representation, each model has its own properties. SG is useful for predicting rare words where there is limited training data available. On the other hand, CBOW achieves better performance for more frequent words and have faster training time than SG. Since the amount of data is large enough in our case so that it can benefit from faster training, we employ CBOW model which is explained in detail for the remaining of this section. More specifically, we give mathematical equations for the network structure and the update rules introduced in [42]. Note that our main goal is to obtain the vector representations for payloads, not bytes. Therefore,

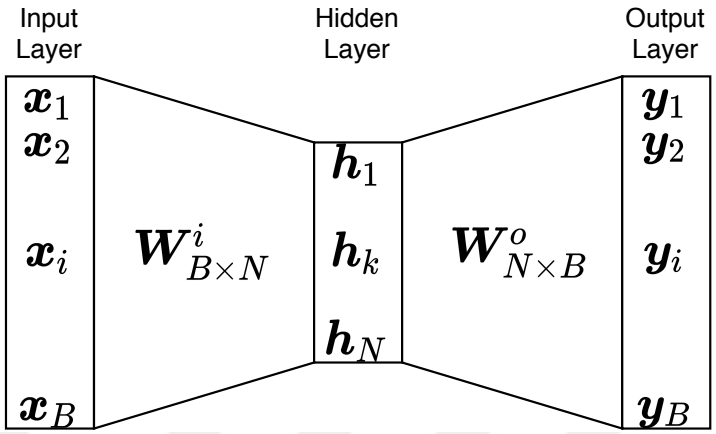


Figure 2.3: Neural network architecture for CBOW

we use byte representations learned by Word2Vec model to form feature vector for each payload. To do so, the mean of the byte representations for each payload is calculated and regarded as the vector representation of that payload.

2.2.1 Continuous Bag of Words

In this section, we explain the CBOW model used for obtaining distributed representations for bytes. First, we start with the simplest form of CBOW predicting a single byte given another byte as the context. To this end, neural networks with fully connected layers are used. The network structure is demonstrated in Figure 2.3.

We define B as the size of our dictionary, which corresponds to the vocabulary size in [41]. In our case, B is 256, since a byte can take 256 different values. Input and output bytes are encoded as B-dimensional one-hot vectors. Therefore, resulting vectors have the value of 1 for the dimension corresponding to the byte values, and 0 for the remaining dimensions. In addition, vector representations resulting from CBOW model are N dimensional.

The weight matrix $\mathbf{W}^i$ between the input layer and the hidden layer is the embedding matrix for bytes. It is a $B \times N$ matrix whose rows are associated with

byte representations. More specifically, each row of $\mathbf{W}^i$ is the vector representation $\mathbf{v}_b$ of input byte b. Given a context as byte b_I, we have

$$\mathbf{h} = {\mathbf{W}^i}^\top \mathbf{x} = \mathbf{v}_{b_I}^\top.$$

This operation implies that the row of $\mathbf{W}^i$ corresponding to the context byte is copied to $\mathbf{h}$. It is inferred from this equation that the activation function of each neuron in the hidden layer is linear. As a result, embedding matrix $\mathbf{W}^i$ is used for passing weighted sum of inputs to the hidden layer.

The weight matrix $\mathbf{W}^o$ between the hidden layer and the output layer is an $N \times B$ matrix that uses context byte data to produce output byte data. For each byte in the dictionary, a score u_j is calculated using $\mathbf{W}^o$. u_j is given by

$$u_j = (\mathbf{W}^o_{[:,j]})^\top \mathbf{h},$$

where the j^{th} column of the matrix $\mathbf{W}^o$ is denoted by $\mathbf{W}^o_{[:,j]}$. Then, the posterior distribution of bytes is obtained using softmax function. This distribution is given as

$$p(b_j|b_I) = y_j = \frac{\exp(u_j)}{\sum_{j' \neq j}^{B} \exp(u_{j'})},$$

where y_j is the output of the j^{th} unit in the output layer. Substituting u_j into the equation given above we obtain

$$p(b_j|b_I) = y_j = \frac{\exp\left((\mathbf{W}^o_{[:,j]})^\top \mathbf{h}\right)}{\sum_{j' \neq j}^{B} \exp\left((\mathbf{W}^o_{[:,j']})^\top \mathbf{h}\right)}.$$

Our training objective is to maximize the conditional probability of observing the actual output byte b_O given the input context byte data b_I with regard to the weights. The index of b_O in the output layer is denoted by j^*. Then, we have

$$\begin{aligned} \max p(b_O|b_I) &= \max y_{j^*} \\ &= \max \log y_{j^*} \\ &= u_{j^*} - \log \sum_{j'=1}^{B} \exp(u_{j'}) := -E, \end{aligned}$$

where $E = -\log p(b_O|b_I)$ is our loss function to be minimized. Let L define the cross entropy measure between two distributions, where L is given by

$$L = \sum_{j=1}^{B} -t_j \log(y_{j^*}).$$

In our case, $t_j = \mathbb{1}(j = j^*)$ is the indicator function so that t_j is 1 if the j^{th} unit is the actual output byte, otherwise $t_j = 0$. Therefore, our loss function is actually a special case of cross entropy measure.

Update rule requires the derivative of the loss function with respect to each element of the output matrix. The derivatives are computed as

$$\frac{\partial E}{\partial w_{ij}^o} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial w_{ij}^o} = e_j h_i.$$

Then, the update equation for the output weight matrix using stochastic gradient descent is

$$\mathbf{W}_{[:,j]}^o = \mathbf{W}_{[:,j]}^o - \eta e_j \mathbf{h} \quad \text{for } j = 1, 2, \ldots, B,$$

where $\eta > 0$ is the learning rate.

Since the update equation for output weight matrix $\mathbf{W}^o$ is derived, the next step is to obtain the update equation for $\mathbf{W}^i$. Taking the derivative of E with respect to $\mathbf{W}^i$, we have

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i}\frac{\partial h_i}{\partial w_{ki}},$$

where $h_i = \sum_{k=1} B x_k w_{ki}$. Then, we need to take the derivative of E with respect to h_i. The derivative is

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{B} \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{B} e_j w_{ij}^o = \mathbf{e}\mathbf{w}_i^o.$$

Using $\partial h_i / \partial w_{ki} = x_k$, we have

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x}\mathbf{e}\mathbf{w}^\top.$$

Since $\mathbf{x}$ is one-hot vector representation of the input, all elements of $\mathbf{x}$ is zero except for the dimension that corresponds to the value of the byte b_I.Therefore,

only the row of $\partial E/\partial \mathbf{W}$ corresponding to the value of b_I is nonzero. Then, we can write the update equation for $\mathbf{W}^i$ as

$$\mathbf{v}_{b_I}^{new} = \mathbf{v}_{b_I}^{old} - \eta \mathbf{e}\mathbf{w}^\top$$

where $\mathbf{v}_{b_I}$ is the row of $\mathbf{W}^i$ corresponding to the context byte data. Since the derivatives of all other rows are zero, they are not updated.

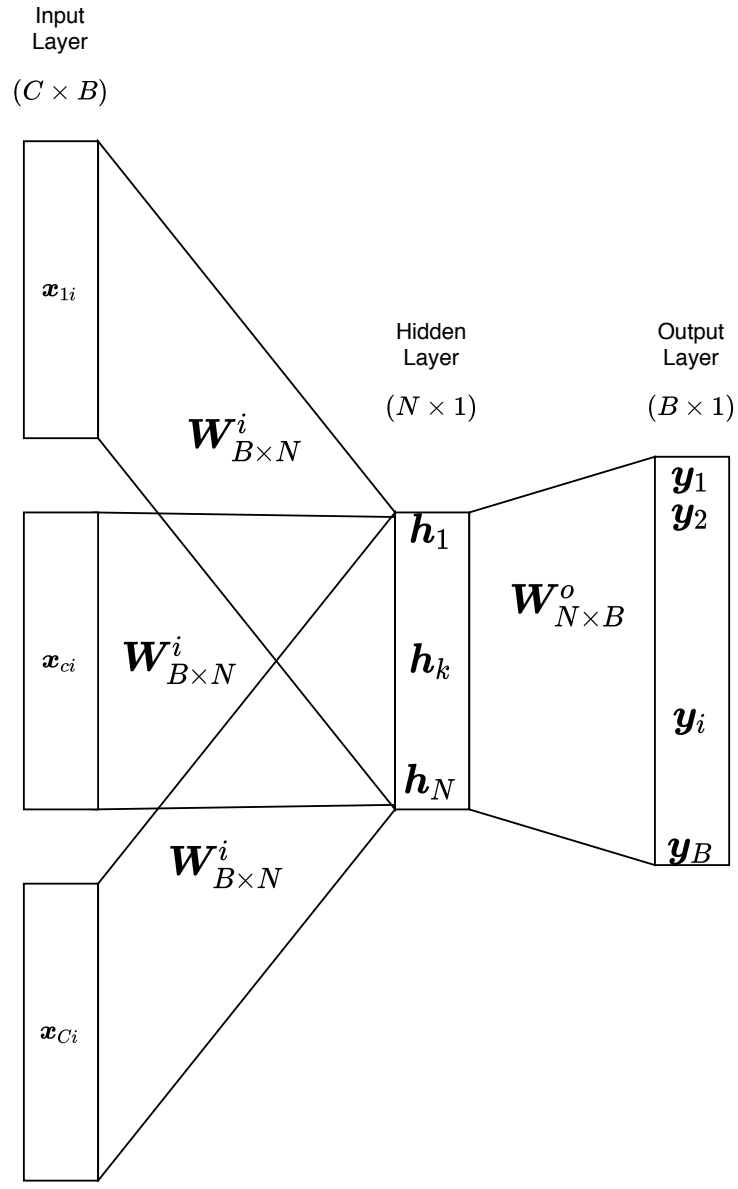


Figure 2.4: NN architecture for multiple-input single-output CBOW

Since we fully explained CBOW model for one context byte data, we can move

on to the case where more than one byte is used to predict the target byte. The schematic of this case is given in Figure 2.4. CBOW model with extended context takes the average of one-hot byte representations forming the context data to obtain the hidden layer vector

$$\begin{aligned}\mathbf{h} &= \frac{1}{C}{\mathbf{W}^{i}}^{\top}\left(\mathbf{x}_1 + \mathbf{x}_2 + \ldots + \mathbf{x}_C\right) \\ &= \frac{1}{C}\left(\mathbf{v}_{b_1} + \mathbf{v}_{b_2} + \ldots + \mathbf{v}_{b_C}\right),\end{aligned}$$

where C is the number of bytes in the context and $b_1, \ldots, b_C$ are the context data bytes. Similar to the case where one context data is available, the byte b is represented by the vector $\mathbf{v}_b$. The loss function is

$$\begin{aligned}E &= -\log p(b_O|b_{I,1}, \ldots, b_{I,C}) \\ &= -u_{j^*} + \log \sum_{j'=1}^{B} \exp(u_{j'}) \\ &= -{\mathbf{v}_{b_O}}^{\top}\mathbf{h} + \log \sum_{j'=1}^{B} \mathbf{v}_{b_{j^*}}^{\top}\mathbf{h}.\end{aligned}$$

Since the derivatives for the output weight matrix is independent of the context, the update equation remains the same for the output layer. Then the update equation is

$$\mathbf{W}^{o}_{[:,j]} = \mathbf{W}^{o}_{[:,j]} - \eta e_j \mathbf{h} \quad \text{for } j = 1, 2, \ldots, B.$$

The update equation for the input weight matrix is also the same except that it is applied for each context byte. In addition, since $\mathbf{h}$ is now the average of the context bytes, the update term is now divided by C. Then, the equation is

$$\mathbf{v}^{new}_{b_{I_c}} = \mathbf{v}^{old}_{b_{I_c}} - \frac{1}{C}\eta \mathbf{e}\mathbf{w}^{\top} \quad \text{for } c = 1, 2, \ldots, C,$$

where $\mathbf{v}_{b_{I_c}}$ is the vector representation of the c^{th} byte in the input context.

2.3 Feature extraction via CNN Autoencoders

Convolutional structures are very popular in tasks such as image classification where spatial relations are important. Under the assumption that the bytes of

payloads relate to other nearby bytes, convolutional neural network(CNN) autoencoders can be employed in the task of feature extraction in our setting. We aim to extract features that include semantic meanings of the payloads by considering and learning the hierarchical structures in them. As with other autoencoders, the purpose here is to reconstruct the input payload at the output. To this end, we use mean squared error at the output. Formally, let $\boldsymbol{X}_t$ be an input payload and let $\hat{\boldsymbol{X}}_t$ be the output of the CNN autoencoder. The mean squared error is given by

$$\sum_{i=1}^{l_t} ||\boldsymbol{x}_{t,i} - \hat{\boldsymbol{x}}_{t,i}||^2.$$

where $\boldsymbol{x}_{t,i}$ is the i^{th} byte of the input payload $\boldsymbol{X}_t$, $\hat{\boldsymbol{x}}_{t,i}$ is the i^{th} byte of the output $\hat{\boldsymbol{X}}_t$ and l_t is the length of the payload sequence. The optimization strategy is the same as in other neural network structures, which is to backpropagate gradients for each payload. After the training of the system, encoder part of the autoencoder is used to map input payloads to the new feature space. The structure of the system is given in Figure 2.5. Once all data is mapped into the new feature space, these features can be used to train classifiers such as random forests or support vector machines.

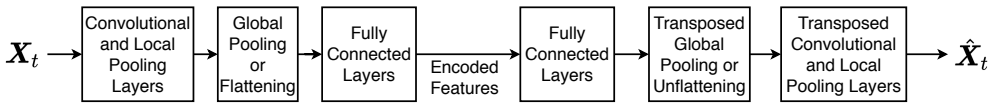


Figure 2.5: CNN autoencoder architecture

2.4 Feature Extraction via LSTM Autoencoders

In this section, we will explain how LSTM autoencoders can be used in order to both achieve a meaningful and a reduced dimensionality while paying attention to temporal dependencies in network payloads. In addition, their ability to process varying length input data will be presented.

For this section, we will slightly change the notation regarding the payloads from how they are defined in Section 2.1. Instead of considering them as a vector

of payloads, we will consider them as a matrix of one-hot vectors where each vector represents a byte. Thus, again we have our set of payloads $\mathcal{X}$ and we have our payload as

$$\boldsymbol{X}_i \coloneqq [\boldsymbol{x}_{1,i}, \ldots, \boldsymbol{x}_{T_i,i}]$$

where $\boldsymbol{x}_{t,i}$ denotes the t^{th} column of the $\boldsymbol{X}_i$ which is a $256 \times T_i$ matrix. Note that we do not consider source and destination payloads separately. This notation is valid for both cases. As in Section 2.1, l_t denotes the length of the payload $\boldsymbol{X}_i$.

We will utilize LSTM autoencoders in this section. However, in order to understand its properties better, we will first mention RNNs. As mentioned in Chapter 1, RNNs can make use of the sequential dependencies in data. Processing data through RNN yields the state vector[11]:

$$\boldsymbol{h}_{t,i} = \kappa\left(\boldsymbol{V}\boldsymbol{x}_{t,i} + \boldsymbol{K}\boldsymbol{h}_{(t-1),i}\right), \tag{2.1}$$

where $\boldsymbol{h}_{t,i} \in \mathbb{R}^m$ with m being the number of units of the RNN. $\boldsymbol{K}$ and $\boldsymbol{V}$ are the weight matrices that are multiplied with the previous state vector and the current input, respectively. We set the element-wise function $\kappa(\cdot) = tanh(\cdot)$, as it is commonly used in the literature and in practice.

As we explained how RNN works mathematically, we will now show how it can be used to create autoencoders. Starting with the encoder, we know that the output of the RNN encoder will be as follows:

$$\boldsymbol{h}_{t,i} = \kappa_\theta^{enc}\left(\boldsymbol{x}_{t,i}, \boldsymbol{h}_{(t-1),i}\right) \tag{2.2}$$

where, similar to the above notation, $\boldsymbol{h}_{t,i}$ is the output of the i^{th} neuron in the encoder and θ denotes the set of parameters of the RNN layer. After this step, we apply last pooling to the set of outputs, $\{\boldsymbol{h}_{t,i}\}_{t=1}^{T_i}$, in this study. Hence we have,

$$\boldsymbol{h}_t = \boldsymbol{h}_{T_i,i}. \tag{2.3}$$

After pooling, we have the encoded version of the input. The next step is to process it in the decoder part. The decoder, which again uses RNN structure,

outputs the following using $\boldsymbol{h}_t$:

$$\hat{\boldsymbol{h}}_{t,i} = \kappa_{\alpha}^{dec}\left(\boldsymbol{h}_t, \hat{\boldsymbol{h}}_{(t-1),i}\right) \tag{2.4}$$

$$\hat{\boldsymbol{x}}_{t,i} = \rho(\hat{\boldsymbol{h}}_{t,i}) \tag{2.5}$$

where, in this notation, each $\hat{\boldsymbol{x}}_{t,i}$ together represent the reconstructed payload. Similar to the above case, α denotes the set of parameters of the RNN layer. Here we set, $\rho(\cdot) = tanh(\cdot)$, which is an element-wise function.

In order to train the system, we need to specify a loss function that will produce gradients which will be used to update the weights through the use of backpropagation. In our case, we used the mean squared loss which is given by

$$\sum_{i=1}^{T_i} ||\boldsymbol{x}_{t,i} - \hat{\boldsymbol{x}}_{t,i}||^2.$$

As mentioned, we actually want to use an LSTM autoencoder instead of an RNN one. The functioning of LSTM is fairly complex compared to RNN. The one layer LSTM structure that we use is defined as follows[40]:

$$\tilde{\boldsymbol{c}}_{t,i} = g\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_{t,i} + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t-1,i} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{2.6}$$

$$\boldsymbol{i}_{t,i} = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_{t,i} + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1,i} + \boldsymbol{b}^{(i)}\right) \tag{2.7}$$

$$\boldsymbol{f}_{t,i} = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_{t,i} + \boldsymbol{R}^{(f)}\boldsymbol{h}_{t-1,i} + \boldsymbol{b}^{(f)}\right) \tag{2.8}$$

$$\boldsymbol{c}_{t,i} = \boldsymbol{D}_t^{(i)}\tilde{\boldsymbol{c}}_{t,i} + \boldsymbol{D}_t^{(f)}\boldsymbol{c}_{t-1,i} \tag{2.9}$$

$$\boldsymbol{o}_{t,i} = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_{t,i} + \boldsymbol{R}^{(o)}\boldsymbol{h}_{t-1,i} + \boldsymbol{b}^{(o)}\right) \tag{2.10}$$

$$\boldsymbol{h}_{t,i} = \boldsymbol{D}_t^{(o)}\beta(\boldsymbol{c}_{t,i}) \tag{2.11}$$

where $\boldsymbol{c}_{t,i}$ is the state vector, $\boldsymbol{x}_{t,i} \in \mathbb{R}^{256}$ is the input vector and $\boldsymbol{h}_{t,i} \in \mathbb{R}^m$ is the output vector. Also, $\boldsymbol{i}_{t,i}$, $\boldsymbol{f}_{t,i}$ and $\boldsymbol{o}_{t,i}$ denote the input, forget and output gates, respectively. In the above equations, $\boldsymbol{D}_t^{(i)}$=$diag(\boldsymbol{i}_t)$, $\boldsymbol{D}_t^{(f)}$=$diag(\boldsymbol{f}_t)$ and $\boldsymbol{D}_t^{(o)}$=$diag(\boldsymbol{o}_t)$. $diag(\cdot)$ function is the diagonalization operation. The $g(\cdot)$ and $\beta(\cdot)$ activations are both selected to be $tanh(\cdot)$ and $\sigma(\cdot)$ is the sigmoid activation. The LSTM autoencoder architecture is given in Fig. 2.6.

Remark 3 *Since LSTM is also an RNN structure, the properties of the RNNs also apply to the LSTM architecture. This means that (2.2)-(2.5) also applies on*

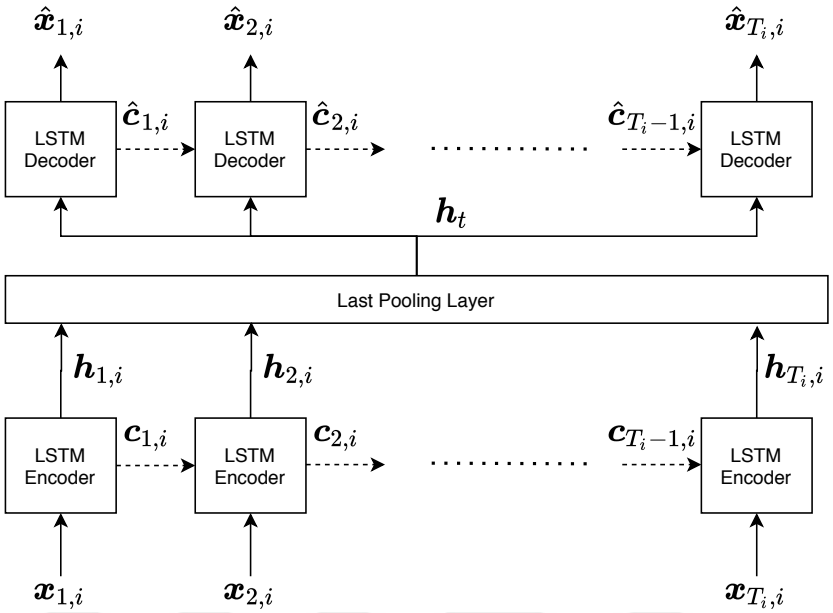


Figure 2.6: Sequential LSTM autoencoder

the LSTM neural network defined in (2.6)-(2.11). However, the equations for the LSTM encoder and decoder needs to be updated as follows:

$$\boldsymbol{h}_{t,i} = \kappa_{\theta}^{enc}\left(\boldsymbol{x}_{t,i}, \boldsymbol{c}_{t-1,i}\right) \tag{2.12}$$

$$\hat{\boldsymbol{h}}_{t,i} = \kappa_{\alpha}^{dec}\left(\boldsymbol{h}_{T_i}, \hat{\boldsymbol{h}}_{t-1,i}, \hat{\boldsymbol{c}}_{t-1,i}\right) \tag{2.13}$$

$$\hat{\boldsymbol{x}}_{t,i} = \rho(\hat{\boldsymbol{h}}_{t,i}) \tag{2.14}$$

where $\boldsymbol{h}_{T_i}$ is the LSTM state vector obtained after pooling operation over $\boldsymbol{h}_{t,i}$ as mentioned in Eqn. 2.3.

Remark 4 *Once the training of the LSTM autoencoder is done, in order to use it as a feature extractor, we actually only use its encoder part. Given an input $\boldsymbol{X}_i$, simply feeding it through the encoder yields the encoded version of the input, $\boldsymbol{h}_t$.*

Chapter 3

Supervised and Unsupervised Network Intrusion Detection

In this chapter, we explain intrusion detection process using different learning architectures. Our approaches to this problem originate from the fact that network attacks mostly target the application layer. Therefore, we can differentiate attacks from normal traffic by capturing meaningful information from network payloads. Since our algorithms are based on processing payload information directly from raw data throughout this thesis work, our approaches to examining the effect of using payloads as input to the intrusion detection systems are two fold.

First, we employ several feature extraction methods that are capable of acquiring vector representations of sequential data with variable length. Each of the algorithms accomplish this goal by exploiting different characteristics of payloads. Then, we use support vector machines (SVM) and random forests (RF) to investigate which type of vector representation yields more meaningful information about the payloads. Then, we investigate how different payload characteristics affect the resulting scores of anomaly detection.

Secondly, unsupervised learning algorithms, specifically autoencoders, are used

for anomaly detection. Instead of using payload features, autoencoders directly utilize one-hot representations of the networks payloads. Although their encoder parts are used as feature extractors in our supervised framework, the feature vector they produce stays hidden and only the calculated reconstruction error for each payload is required for anomaly detection in the unsupervised setting. Since labeled data is not available in most real life applications, our main approach is unsupervised identification of network attacks using autoencoders. We emphasize the choice of LSTM autoencoders due to their recurrent and complex gated structure. These properties of LSTMs make them powerful learning algorithm for sequential data. As a competitor algorithm, we choose CNN autoencoders.

3.1 Supervised Framework Using Vector Representation of Payloads

We approach the intrusion detection problem from a payload-based learning perspective. However, instead of analyzing packets in terms of its properties, we directly use payloads as input to feature extractors that are generally preferred in NLP tasks. In addition, these methods also reduce the need for feature selection since features are automatically extracted using statistical methods and machine learning algorithms. We aim to gather all meaningful information required for intrusion detection. Therefore, in the supervised setting, we employ different algorithms, each having a different approach for obtaining vector representations of sequential data.

For example, n-grams with tf-idf scores can extract information about the importance of each element in a sequential data. However, they fail to capture semantics, since their produced score solely depends on element count in given sample. On the other hand, Word2Vec models are able to capture semantic relationships between the elements by taking into account the context.

As a more sophisticated and dynamic approach, LSTM networks can learn

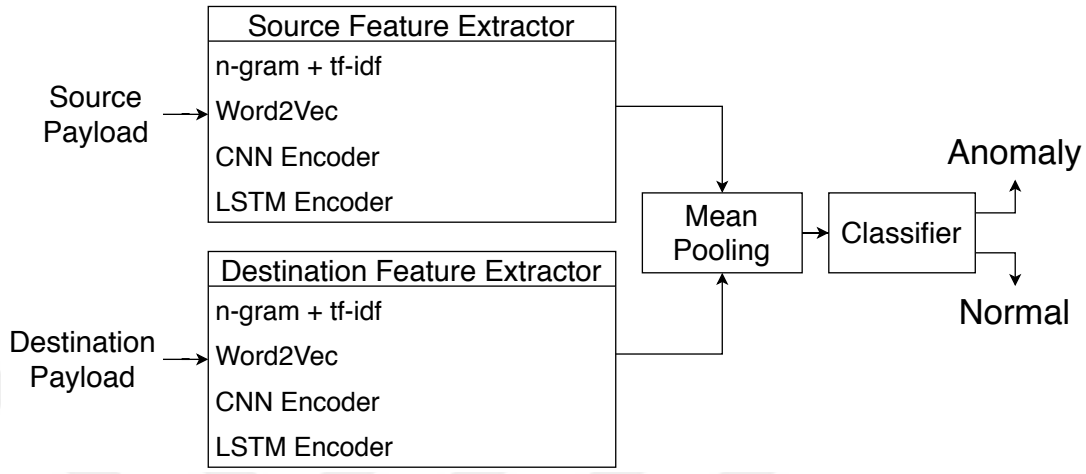


Figure 3.1: Supervised Anomaly Detection Scheme

semantic information from sequences while maintaining a long term memory of sequential information. However, its ability to accomplish these goals are shown mostly on time series data and NLP tasks. Therefore, our main goal is to investigate if the payloads can be processed efficiently using these algorithms so that we can obtain vector representations of payloads to be used for intrusion detection. For comparison, we employ CNNs, which are also used in NLP tasks as a faster alternative to RNNs. Although they are shown to be useful capturing semantics, their ability to efficiently process sequential data is limited.

After obtaining feature vectors, we employ SVM and RF. They perform well for many real life problems differing in context. Then, we perform intrusion detection by means of binary classification. The schematic of the framework is demonstrated in Fig. 3.1. Note that we use both source and destination payloads as our inputs, aiming to extract features for detecting different types of attacks. Therefore, we provide a dynamic approach to the intrusion detection problem.

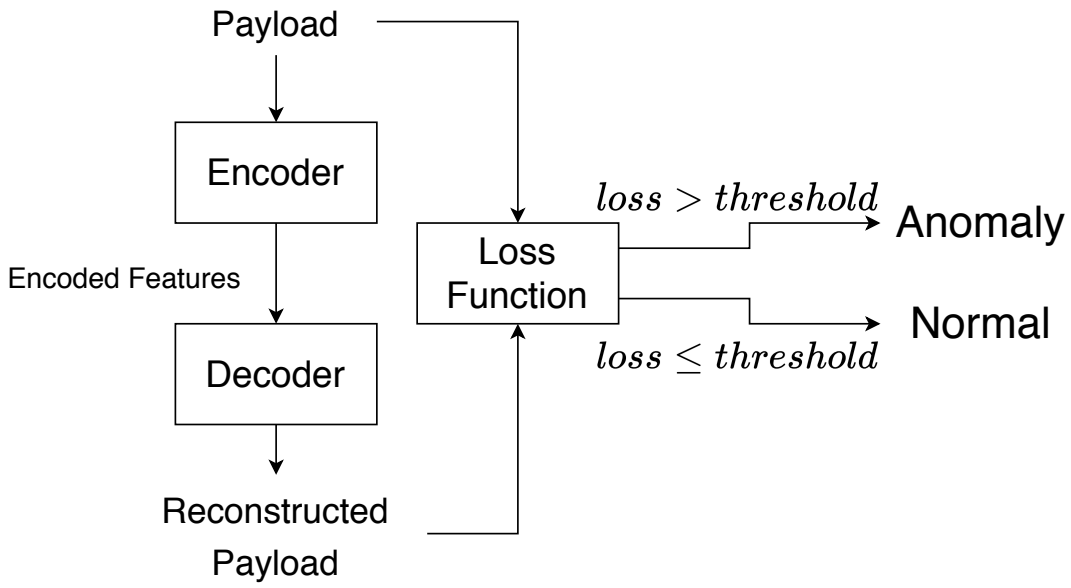


Figure 3.2: Unsupervised Anomaly Detection Scheme

3.2 Unsupervised Framework Using LSTM Autoencoders

Although payload features can carry important information about network traffic, they require training of feature extraction and classifier algorithms separately. In addition, unsupervised learning algorithms are more suitable for real life network data since the labels are not available most of the time.

To this end, we use autoencoders to design an unsupervised framework in which vector representation is learned and used automatically, since we directly use reconstruction error for anomaly detection by comparing it to a chosen threshold. We employ CNN and LSTM autoencoders for comparison. The schematic of the setting is demonstrated in Fig. 3.2

Details of how unsupervised methods are used in the intrusion detection system are as follows. Recall that in previous chapter we derived the equations for the LSTM autoencoders. We do not need the technical details in this section, hence we will reiterate briefly while generalizing the structure. Let $\kappa_\theta^{enc}(\cdot)$ be

the encoder part of the autoencoder with parameter set θ, then for a payload $\boldsymbol{X} \coloneqq [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]$, we have the following as its encoded version

$$\boldsymbol{h} = \kappa_\theta^{enc}(\boldsymbol{X}). \tag{3.1}$$

This value was used in the previous chapter for unsupervised preprocessing to provide fixed length vectors for supervised methods. When we process this vector through $\kappa_\alpha^{dec}(\cdot)$, the decoder with parameter set α , we get

$$\hat{\boldsymbol{X}} = \kappa_\alpha^{dec}(\boldsymbol{h}). \tag{3.2}$$

where $\hat{\boldsymbol{X}}$ is the reconstructed payload whose representation is given as $\hat{\boldsymbol{X}} \coloneqq [\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_T]$. The reconstruction error, which we use as anomaly indication score, is given by:

$$Err(\boldsymbol{X}) = ||\boldsymbol{X} - \hat{\boldsymbol{X}}||_{2,1} = \sum_{t=1}^{T} ||\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t||^2. \tag{3.3}$$

where $||\cdot||_{2,1}$ denotes the $L_{2,1}$ norm which is the sum of the Euclidean norms of the columns for a matrix as seen above.

Lastly, we need a threshold τ to make our final decision. During the experiments, we select τ in both a supervised and an unsupervised manner using the validation set. Only after τ is fixed, the autoencoder is evaluated on the test set. The choice of τ is vital, as setting it too low may cause high false alarm rate and setting it too high may allow intrusions to go undetected.

Chapter 4

Experiments for Supervised and Unsupervised Learning

This chapter contains experiments regarding all the approaches discussed in the previous two chapters. To give a brief introduction to the chapter, let us mention that feature extraction methods discussed in Chapter 2 are employed in the supervised framework from Chapter 3. Also, using autoencoders from Chapter 3, unsupervised anomaly detection will be performed. Before giving details about the experiment setup and results, we will give information regarding the data set we used and data preprocessing steps for each algorithm.

4.1 Intrusion Detection Evaluation Data set

For the experiments, we used the Intrusion detection evaluation data set(ISCX IDS 2012)[40]. There are multiple reasons behind this choice. First and foremost, although many data sets are available for similar objectives, most of the public data sets are heavily anonymized or unrealistic. Another reason is that this data set is dynamically generated.

In the data set, there are approximately 2 million connections from different protocols such as HTTP, FTP and SMTP. HTTP and FTP together make up a large portion of these connections. In the original data set, only about 5% of the entries are anomalies. This data set is also very useful for the purpose of testing the system against unknown and different types of attacks, as it contains a wide range of anomaly instances.

4.2 Data Preprocessing

For n-gram and tf-idf algorithms no preprocessing is necessary since they construct vector representations based on the number of each n consecutive bytes contained in payloads. For Word2Vec models, CNN autoencoders and LSTM autoencoders, each byte is represented as one-hot vectors. We use gensim *word2vec* [43] package in Python to implement Word2Vec models. LSTM and CNN autoencoders are implemented using *Keras* [44] package. As a special case, the length of each payload is regarded as the same for CNN autoencoders due to their inability to process variable length data. Therefore, each sequence is padded with zero vectors achieving fixed size.

For both of the LSTM and CNN autoencoders, the payloads are truncated to reduce computational load. Since payload lengths can reach several thousands, it is not feasible to perform experiments using the maximum payload length. Therefore, we use the following approach to truncate the payloads. In our approach, the length of payloads is determined by the number of payloads for each payload length. We choose the fixed length such that the truncated portion of the data corresponds to less than only 5% of all payloads. In addition, we ensure that this amount is valid for both anomalous and normal instances. As a result, we obtain a realistic representation of the original data. However, we obtain a fixed length of 1400 by this approach. Although the computational load is reduced by this choice, we encountered two problems. First, we obtain very sparse input data, which comes with its own challenges for the training of deep networks. More importantly, we observed that our algorithms fail to construct an output of this

length using the encoded input vector. As a result, we reduced the fixed length to 500 at the expense of losing some information.

4.3 Model Training

Data set has missing payloads for some of the connections. We handle instances with missing packet information at either source or destination by simply omitting them, since our algorithms need both source and destination payloads. As mentioned in data set description, we have varying attack scenarios producing insider and outsider attacks. Therefore, we choose to utilize both source and destination payloads. We can afford omitting data instances without payloads as the size of the data set is sufficiently large. The portion of data obtained using this approach has a similar ratio of anomalous data to normal data. Therefore, the modified data set reflects the original data set in terms of attack ratio.

Since the size of the data set is very large, training of the deep learning methods with the original data set is not feasible. Therefore, we subsample the data set in an efficient manner. For the experiments, we gather 20k samples: 12800(64%) for training, 3200(16%) for validation and 4000(20%) for test sets. All separations are done in a stratified manner, which means that class distributions in all sets are preserved. During training and hyperparameter selection, we work on only training and validation data sets. We perform 5-fold cross-validation for hyperparameter tuning. The test set is not visible to the training algorithm. It is only used to obtain the final scores after all parameters are set.

During training, we use Adam [45] optimizer for LSTM and CNN autoencoders. As the hyperparameter tuning method, we use grid search for different numbers of batch sizes, epochs, dimensions of latent representation for autoencoders. For Word2Vec, context window size and dimensionality of the vector representation are regarded as hyperparameters. Similarly, C and Gamma values for SVM training and depth and number of estimators for RF training are also tuned using cross validation and grid search.

Finally, the threshold values for unsupervised algorithms are chosen with different approaches. Since we aim to perform completely unsupervised learning, we fix the threshold based on the assumption about attack ratio in data set. For comparison, we also use precision-recall(PR) curve and ROC curve of validation data set for threshold selection.

4.4 Evaluation

As mentioned, we gather 20k samples from the data set, but this value is only for one repetition of the experiment. In order to validate the results, we run the experiments 5 times on disjoint random subsets drawn from the original data set. The results we report on precision, recall, F1 and AUC scores are averages over 5 repetitions on test sets. Training deep networks is a time consuming task. Since we also perform grid search on multiple hyperparameters as well, we were not able to increase the number of repetitions. Instead to reinforce the validity of our results, we constructed 95% confidence intervals for our scores using Student's t-distribution. As a rule of thumb, instead of confidence intervals based on normal distribution, Student's t-distribution is used for small sample sizes where true standard deviation is unknown[46]. The results are presented with the confidence radius.

4.5 Discussion on Results

In Table 4.1, we demonstrate AUC and F1 scores for SVM and RF classifiers used with features obtained from word2vec model. For different vector dimension and window size selections, SVM performs significantly better in terms of both AUC and F1 scores. Clear separation between the performances of RF and SVM can be observed through their confidence intervals. In all settings, upper confidence bound of RF is lower than the lower confidence bound of SVM. In addition, it can be seen that both RF and SVM perform better with vector dimension of 128

and window size of 15 in general. However, the change in word2vec parameters does not cause any significant performance change. Therefore, one can adjust its parameters according to the system needs in this setting.

In Table 4.2, several conclusions can be drawn from the performance evaluation of different algorithms in supervised setting. First of all, all methods perform better on HTTP subset. This result is expected since 95% of network traffic is composed of HTTP connections. Therefore, training algorithms are better at learning HTTP characteristics. However, there is an exception for word2vec and tf-idf features used with SVM classifier. AUC scores from these algorithms are similar for both the whole data set and the HTTP subset. However, since our data set is highly imbalanced, this difference is not reflected on the F1 scores. More specifically, other feature extraction algorithms still provide good representations of normal and anomaly instances. On the other hand, we can infer that SVM algorithm can learn more robust decision boundaries with tf-idf and word2vec features compared to CNN and LSTM features.

In terms of F1 scores, n-gram and tf-idf features with RF performs significantly worse than all other combinations of feature extractors and classifiers for both the whole data set and the HTTP subset. Although RF uses an ensemble of decision trees, it still cannot perform as well as SVM using these features. On the other hand, CNN AE and LSTM AE features used with RF achieve the highest mean F1 scores. Then, we can conclude that feature extraction capability of CNN AE and LSTM AE is better compared to tf-idf and word2vec.

In table 4.3, performance results for the LSTM AE and the CNN AE are given. Fig. 4.1 shows the ROC curves of LSTM AE and CNN AE for both data sets, averaged over 5 repetitions. For the LSTM AE, the latent dimensionality of the final model is 128, whereas it is 800 for CNN AE. A smaller latent dimensionality is advantageous as it offers information in a more compact manner. Keep in mind that we use the autoencoders as feature extractors in supervised learning as well. Large number of dimensions causes problems for supervised learning, as models fail to pinpoint important features which leads to overfitting.

For threshold selection, our approach is two fold. For τ_{95} and τ_{97}, we did not use any labels from validation set. 95^{th} and 97^{th} percentiles of reconstruction error for the validation set are used as thresholds. For the supervised setting, τ_{pr} and τ_{auc} are chosen by analyzing precision-recall(PR) and ROC curves, respectively. This setting is also realistic since we only need a limited amount of labeled data.

As shown in Table 4.3, CNN AE performs better overall in terms of both AUC and F1 scores. In addition, it is clear that CNN AE is a robust learner, while LSTM AE is very sensitive to threshold selection. However, for the supervised setting, they are shown to produce similar results. Therefore, the performance difference for this part can be explained by the difference in reconstruction ability, which is susceptible to the dimensionality of the vector obtained from encoder part. Since these vector dimensions highly differ, this may be the cause for the gap between CNN AE and LSTM AE scores. In addition, with an appropriate threshold selection, LSTM AE gives promising results.

In addition, we reduced the payload length, resulting in incomplete byte sequences. Therefore, LSTM AE is expected to be affected negatively from this case, while we have less sparse input for CNN AE. Considering all conditions are in favor of CNN AE, we conclude that LSTM AE performance is acceptable in our case.

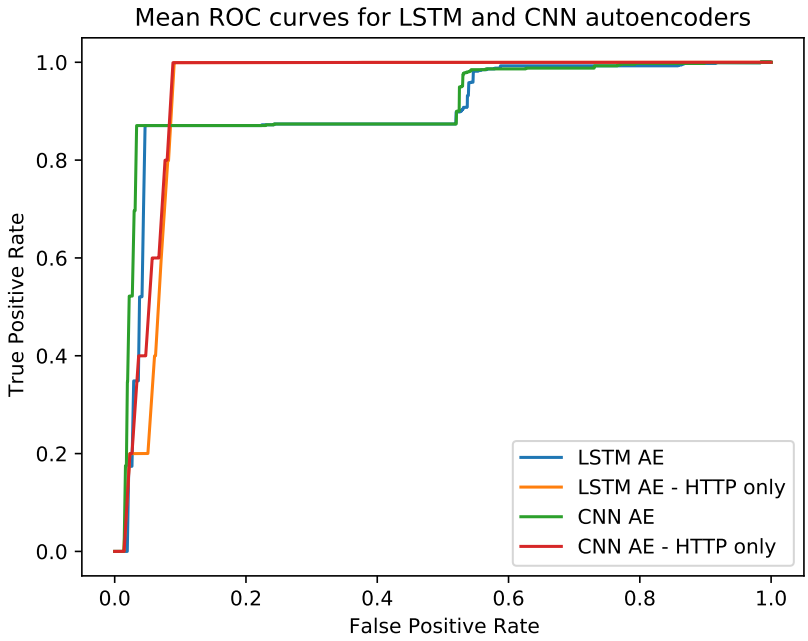


Figure 4.1: ROC Curves for CNN AE and LSTM AE

Table 4.1: Experiment results for Word2Vec with SVM and RF for different vector dimensions and window sizes using ISCX 2012 data set

vec	win	RF - F1	RF - AUC	SVM - F1	SVM - AUC
128	15	0.847 ± 0.010	0.933 ± 0.002	0.869 ± 0.008	0.976 ± 0.008
128	25	0.841 ± 0.012	0.933 ± 0.002	0.869 ± 0.008	0.975 ± 0.011
128	50	0.844 ± 0.014	0.935 ± 0.007	0.868 ± 0.009	0.974 ± 0.009
256	15	0.842 ± 0.014	0.933 ± 0.001	0.869 ± 0.008	0.957 ± 0.022
256	25	0.843 ± 0.012	0.935 ± 0.007	0.869 ± 0.008	0.953 ± 0.021
256	50	0.843 ± 0.013	0.933 ± 0.002	0.869 ± 0.008	0.947 ± 0.027

Table 4.2: Experiment results for supervised methods, for both the whole data set and the HTTP subset

	ISCX 2012 - F1	HTTP - F1	ISCX 2012 - AUC	HTTP - AUC
n-gram + tf-idf + RF	0.839 ± 0.012	0.889 ± 0.015	0.933 ± 0.002	0.978 ± 0.002
W2V + RF	0.847 ± 0.010	0.889 ± 0.012	0.933 ± 0.002	0.978 ± 0.002
n-gram + tf-idf + SVM	0.868 ± 0.007	0.921 ± 0.005	0.980 ± 0.009	0.977 ± 0.002
W2V + SVM	0.869 ± 0.008	0.921 ± 0.005	0.976 ± 0.008	0.977 ± 0.002
CNN AE + RF	0.870 ± 0.009	0.922 ± 0.004	0.932 ± 0.002	0.977 ± 0.001
LSTM AE + RF	0.870 ± 0.008	0.921 ± 0.005	0.932 ± 0.001	0.978 ± 0.001
CNN AE + SVM	0.868 ± 0.007	0.921 ± 0.005	0.930 ± 0.024	0.977 ± 0.002
LSTM AE + SVM	0.858 ± 0.007	0.918 ± 0.006	0.959 ± 0.022	0.977 ± 0.002

Table 4.3: Experiment results for LSTM and CNN autoencoders

		ISCX 2012		HTTP	
		CNN AE	**LSTM AE**	**CNN AE**	**LSTM AE**
	AUC Score	0.909 ± 0.007	0.899 ± 0.006	0.949 ± 0.012	0.939 ± 0.014
τ_{95}	**Precision**	0.677 ± 0.038	0.608 ± 0.037	0.796 ± 0.040	0.770 ± 0.022
	Recall	0.877 ± 0.013	0.870 ± 0.007	0.995 ± 0.006	0.996 ± 0.005
	F1 Score	0.764 ± 0.022	0.715 ± 0.026	0.884 ± 0.024	0.869 ± 0.015
τ_{97}	**Precision**	0.691 ± 0.044	0.377 ± 0.374	0.780 ± 0.041	0.462 ± 0.457
	Recall	0.874 ± 0.010	0.535 ± 0.533	0.997 ± 0.003	0.597 ± 0.601
	F1 Score	0.771 ± 0.026	0.441 ± 0.439	0.875 ± 0.026	0.518 ± 0.523
τ_{pr}	**Precision**	0.678 ± 0.039	0.481 ± 0.302	0.781 ± 0.041	0.619 ± 0.386
	Recall	0.875 ± 0.003	0.702 ± 0.435	0.996 ± 0.004	0.795 ± 0.484
	F1 Score	0.763 ± 0.026	0.570 ± 0.355	0.875 ± 0.025	0.695 ± 0.431
τ_{auc}	**Precision**	0.682 ± 0.046	0.604 ± 0.041	0.783 ± 0.033	0.760 ± 0.032
	Recall	0.878 ± 0.005	0.870 ± 0.007	0.994 ± 0.005	0.996 ± 0.007
	F1 Score	0.767 ± 0.030	0.713 ± 0.028	0.876 ± 0.021	0.861 ± 0.018

Chapter 5

Conclusion

In this thesis, we investigate the problem of intrusion detection and how intelligent systems can be used to counter the possible threats. To this end, we make use of the intrusion detection evaluation data set(ISCX IDS 2012) as a source of dynamic and realistic data. We opted to use machine learning and deep learning approaches, instead of statistical and signature-based ones. The main advantage of learning strategies is their ability to model data without expert input. As a framework, we selected anomaly detection, since the strategy of modeling non-malicious connections, which are much more common than malicious connections, and singling out rare events is suited to our case. We also investigated the problem of feature extraction, which can be used to both learn powerful representations of network payloads and make data compatible with otherwise unusable models. This was also an important task, as computer network data we used is sequential in nature and discovering the temporal properties is a vital part of success in intrusion detection systems. Hence, we can separate our work in this study into two main parts, namely feature extraction and anomaly detection.

In the feature extraction task, we examined three groups of feature extractors. First, we devised a feature extraction scheme combining n-grams with tf-idf weighting. Using tf-idf weighting, we were able to dismiss common and uninteresting features and select more important and useful ones. Secondly, we used

Word2Vec method. The main inspiration behind this choice is that payloads are made up of byte sequences, defining the characteristic of network connections. Therefore, they resemble sentences composed of words in a meaningful order. Lastly, we employed highly popular concept of deep autoencoders. We have experimented with CNN and LSTM architectures. Given the assumption that payloads contain meaningful sequential information, LSTM is expected to work better because it can model time dependencies that exist in the data. The performances of feature extractors can be evaluated by using different classifiers, which brings us to the anomaly detection task. To evaluate the performance of the feature extractors and to build supervised anomaly detection models, we used SVMs and RFs as classifiers after extracting features using aforementioned methods. Using feature extractors and these classifiers, we observed that powerful representations are indeed learned by CNN AEs and LSTM AEs. The SVMs utilized the extracted features very well, yielding high scores for all feature extraction schemes. On the other hand, RFs are more sensitive to the vector representation obtained as expected. Therefore, using the results obtained from RFs, we identify which features are better suited for representing network payloads.

Secondly, for the anomaly detection task, we directly employed CNN and LSTM autoencoders. Once the autoencoders are trained with data sets that contain very little to none threat connections, we used them to reconstruct the test data. The malicious test connections were expected to have high reconstruction errors due to them not being modeled by the autoencoders. The test labels with reconstruction errors higher than a certain threshold were labeled as anomalies. Since this scheme is unsupervised, its performance was lower than the supervised approach. Nevertheless, it proved to be useful when labeled data is not available, which is generally the case for realistic scenarios. Among the autoencoders, CNN structures produce better results. They are also more robust. The analysis of the results were made considering the experiment settings being in favor of CNN structures. As a result, we concluded that LSTM autoencoders give promising results, despite the experiment settings.

Finally, we conclude that deep autoencoders are more useful as feature extractors compared to word2vec and tf-idf models. As unsupervised learners,

CNN structure performs better, which is explainable in our setting. However, LSTM autoencoders also give good performance depending on threshold selection. Therefore, both CNN and LSTM autoencoders can be utilized in an unsupervised framework for intrusion detection.

Bibliography

[1] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[2] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, IEEE, 2008.

[4] Y. Chen, X. S. Zhou, and T. S. Huang, "One-class svm for learning in image retrieval," in *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, vol. 1, pp. 34–37, IEEE, 2001.

[5] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[6] T. Fawcett and F. Provost, "Activity monitoring: Noticing interesting changes in behavior," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 53–62, 1999.

[7] W.-K. Wong, A. W. Moore, G. F. Cooper, and M. M. Wagner, "Bayesian network anomaly pattern detection for disease outbreaks," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 808–815, 2003.

[8] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–480, 1992.

[9] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, 2003.

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

[11] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[13] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[14] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*, pp. 305–316, IEEE, 2010.

[15] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[16] H. Debar, M. Becker, and D. Siboni, "A neural network component for an intrusion detection system," in *null*, p. 240, IEEE, 1992.

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

[18] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, pp. 1310–1318, 2013.

[19] O. Fabius and J. R. van Amersfoort, "Variational recurrent auto-encoders," *arXiv preprint arXiv:1412.6581*, 2014.

[20] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pp. 4–11, 2014.

[21] V. Kustikova and P. Druzhkov, "A survey of deep learning methods and software for image classification and object detection," *OGRW2014*, vol. 5, 2014.

[22] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[25] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[26] R. C. Staudemeyer, "Applying long short-term memory recurrent neural networks to intrusion detection," *South African Computer Journal*, vol. 56, no. 1, pp. 136–154, 2015.

[27] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[28] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[29] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[30] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semi-supervised network traffic classification," in *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 369–370, 2007.

[31] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*, pp. 843–852, 2015.

[32] W. Ammar, C. Dyer, and N. A. Smith, "Conditional random field autoencoders for unsupervised structured prediction," *Advances in Neural Information Processing Systems*, vol. 27, pp. 3311–3319, 2014.

[33] S. Zanero and S. M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," in *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 412–419, 2004.

[34] S. Revathi and A. Malathi, "A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 12, pp. 1848–1853, 2013.

[35] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*, pp. 1–6, IEEE, 2009.

[36] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International joint conference on neural networks (IJCNN)*, pp. 3854–3861, IEEE, 2017.

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

[38] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[39] Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," *Security and Communication Networks*, vol. 2017, 2017.

[40] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[41] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[42] X. Rong, "word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[43] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Citeseer, 2010.

[44] F. Chollet *et al.*, "Keras: The python deep learning library," *ascl*, pp. ascl–1806, 2018.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[46] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005.