AN ATTENTION BASED DEEP NEURAL NETWORK ARCHITECTURE FOR IDENTIFICATION OF PHISHING URLS THROUGH CHARACTER LEVEL N-GRAM EMBEDDINGS

KİMLİK AVCISI URL TESPİTİNDE KARAKTER N-GRAM DÜZEYİNDE ÖZYERLEŞİKLERDEN YARARLANAN DİKKATE DAYALI BİR DERİN SİNİR AĞI MİMARİSİ

FIRAT COŞKUN DALGIÇ

ASSOC. PROF. DR. MURAT AYDOS

Supervisor

ASST. PROF. DR. AHMET SELMAN BOZKIR

Co-Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

June 2022

ABSTRACT

AN ATTENTION BASED DEEP NEURAL NETWORK ARCHITECTURE FOR IDENTIFICATION OF PHISHING URLS THROUGH CHARACTER LEVEL N-GRAM EMBEDDINGS

Fırat Coşkun DALGIÇ

Master of Science , Computer Engineering

Supervisor: Assoc. Prof. Dr. Murat AYDOS

2nd Supervisor: Asst. Prof. Dr. Ahmet Selman BOZKIR

June 2022, 103 pages

Despite the various technological advancements that have been made in the fight against phishing attacks, the problem still remains one of the most common threats in the cybersecurity domain. Due to the increasing number of communication channels and the rise of social media, the need for effective and rapid phishing detection has become more prevalent. In this thesis, we focused on developing an end-to-end deep learning model named Grambeddings that can recognize malicious websites from URL information while introducing the following novelties into the literature; (1) constructing and employing n-gram embeddings seamlessly without requiring any preliminary learning stage, (2) eliminating the necessity of language-knowledge by representing terms from n-grams instead of words or sub-words, (3) providing fast, intelligent and efficient n-gram/feature selection procedure. Besides, we also published an exclusive large-scale novel dataset [1] that contains 800.000 real-world half of which were legitimate and half were phish.

[1] https://web.cs.hacettepe.edu.tr/~selman/grambeddings-dataset/

Grambeddings presents an adjustable and automated n-gram extraction and selection mechanism along with a new deep architecture that enables to merging of four different n-gram level features from its corresponding channel while each channel obtains required deep features through cascading CNN, LSTM, and attention layers. In this way, the model becomes able to capture the multiple discriminative character sequence patterns without requiring any hand-crafted operation. As a result, the proposed approach contributes the following features to the phishing detection domain: (1) real-time inference and protection while providing excellent performance, (2) language-agnostic corpus and embedding construction, and (3) eliminating the necessity of hand-crafted features, or the need of using any third-party service. In addition, we conducted a series of comparative experiments in both dataset-wise and method-wise manner. We verified the superiority of our model in all tests since it outperforms the other models in the literature by achieving 98.27% accuracy. Lastly, we also analyzed the Grambeddings' robustness against adversarial attacks and examined in-depth the characteristics of the model both in the pre-trained and re-trained conditions in terms of seeing any adversarial sample before during the training phase. Our codebase[2] is shared with the community to be used for benchmarking purposes in the future.

Keywords: cybersecurity, phishing, natural language processing, n-gram, embeddings, deep learning,

[2]The code and our supplementary material will be made available at https://github.com/fcdalgic/GramBeddings

ÖZET

KİMLİK AVCISI URL TESPİTİNDE KARAKTER N-GRAM DÜZEYİNDE ÖZYERLEŞİKLERDEN YARARLANAN DİKKATE DAYALI BİR DERİN SİNİR AĞI MİMARİSİ

Fırat Coşkun DALGIÇ

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Prof. Dr. Muray AYDOS

Eş Danışman: Dr. Öğr. Üyesi Ahmet Selman BOZKIR

Mayıs 2022, 103 sayfa

Kimlik avı saldırılarına karşı mücadelede gerçekleştirilen çeşitli teknolojik gelişmelere rağmen, bu sorun hala siber güvenlik alanındaki en yaygın tehditlerden biri olmaya devam etmektedir. Artan iletişim kanalları ve sosyal medyanın yükselişiyle sebebiyle hızlı ve etkili bir oltalama tespiti çok daha önemli bir hale gelmiştir. Bu tezde, kötücül web sitelerini URL bilgilerinden tanıyabilen ve aşağıdaki yenilikleri literatüre kazandıran Grambeddings isimli uçtan uca derin öğrenme modelini geliştirmeye odaklandık; (1) herhangi bir ön öğrenme aşamasına gerek duymayan n-gram gömmelerini sorunsuz bir şekilde oluşturma ve kullanma yöntemi, (2) terimleri kelimeler veya alt kelimeler yerine n-gramlar ile temsil ederek dilden bağımsız bir temsil oluşturma (3) hızlı, akıllı ve verimli n-gram/öznitelik seçim prosedürü. Bunların yanı sıra, yarısı meşru ve yarısı oltalama örneklerine ait gerçek dünyadan toplanan toplam 800.000 URL örneğini içeren büyük ölçekli ve özel bir veri seti [3] yayınladık.

[3] https://web.cs.hacettepe.edu.tr/~selman/grambeddings-dataset/

Grambeddings, her bir kanalı peşpeşe dizili Gömme, Uygulamalı Evrişimsel Sinir Ağları, Uzun Kısa Süreli Bellek ve Dikkat Mekanizması katmanlarından oluşan, otomatik ve ayarlabilir bir n-gram çıkarma ve seçme mekanizması sayesinde her biri farklı n-gram seviyesine ait olmak üzere toplamda dört farklı kanalın birleştirilmesini sağlayan yeni bir derin sinir ağı mimarisi sunar. Böylelikle, öne sürülen modelimiz herhangi bir el yordamı bir işlem gerektirmeden çoklu ayırt edici karakter sekanslarını desenlerini yakalayabilmektedir. Sonuç olarak, Grambeddings kimlik avı algılama alanındaki çalışmalara aşağıdaki katkılarda bulunur: (1) üst düzey bir performans sağlarken gerçek zamanlı çıkarımda bulunma ve koruma sağlama yeteneği, (2) dilden bağımsız korpusun ve gömmenin ilklendirilmesi, (3) herhangi bir el yormadıyla oluşturulmuş özniteliğin kullanılmasını veya üçüncü taraf bir hizmete ihtiyacı ortadan kaldırır. Bunların yanı sıra, tez kapsamında, hem veri kümesi hem de yöntem açısından bir dizi karşılaştırmalaı deney gerçekleştirdik. Bu karşılaştırmalı test sonuçlarına bağlı olarak, modelimizin %98,27'lik bir başarıyla liteatürdeki diğer yaklaşımlardan daha yüksek skor gösterdik. Son olarak, Grambeddings'in çekişmeli saldırılara karşı dayanıklılığını da analiz ettik ve eğitim aşamasında daha önce herhangi bir çekişmeli örneği görüp görmeme açısından hem önceden eğitilmiş hem de yeniden eğitilmiş modelin özelliklerini derinlemesine inceledik. Bu çalışmanın kaynak kodu gelecekte sunulacak diğer çalışmalar tarafından kıyaslama amacıyla kullanması için topluluk ile paylaşılmıştır [4].

Keywords: siber güvenlik, oltalama saldırıları,kimlik avı, doğal dil işleme, n-gram, gömmeler, derin öğrenme

[4]https://github.com/fcdalgic/GramBeddings

ACKNOWLEDGEMENTS

First and foremost, I would like to thank and express my gratitude to my beloved family for their endless love, support, help, and patience.

I would like to thank my thesis supervisor Assoc. Prof. Murat AYDOS for becoming one of my biggest motivations for continuing my master's degree by always being warm and constructive from the first time I entered his room.

And of course, I would like to thank my secondary thesis supervisor Assist. Dr. Ahmet Selman BOZKIR, who always supported me both in my thesis and on many life-related issues, regardless of distance, time, and subject.

Besides I would like to thank my thesis committee members for insightful comments for this thesis.

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

ANN	:	**A**rtifical **N**eural **N**etwork
AUC	:	**A**rea **U**nder **C**urve
BN	:	**B**atch **N**ormalization
CNN	:	**C**onvolutional **N**eural **N**etworks
DL	:	**D**eep **L**earning
DNS	:	**D**omain **N**ame **S**ervice
LSTM	:	**L**ong **S**hort **T**erm **M**emory
ML	:	**M**achine **L**earning
MLP	:	**M**ulti **L**ayer **P**erceptron
NLP	:	**N**atural **L**anguage **P**rocessing
RNN	:	**R**ecurent **N**eural **N**etwork
ReLU	:	**Re**ctified **L**inear **U**nit
SIFT	:	**S**cale **I**nvariant **F**eature **T**ransform
SURF	:	**S**peeded **U**p **R**obust **F**eatures
URL	:	**U**niform **R**esource **L**ocator
UMAP	:	**U**niform **M**anifold **A**pproximation and **P**rojection

1. INTRODUCTION

1.1. Overview and Motivation

Due to the rapid development of the Internet and related information technologies, societies have adapted their habits to access information effortlessly, communicate with other people faster, and manage their financial accounts more robustly via mobile and web-based applications that rely on the Internet. Given the inevitable increase in internet use each year, these technologies not only influence individuals but also significantly affects organizations, businesses, and even governments. On the other hand, although online operations based on personal information make our lives more comfortable in many ways, this situation has also attracted the attention of attackers due to the transfer of information online. Therefore, users become vulnerable to various types of internet-based attacks called cyber-attacks.

A cyber attack is a type of attack carried out by cybercriminals against a person or a company to gain unauthorized access to their sensitive information while compromising the integrity and confidentiality of that information. Over time, the types of such attacks increased, and their scope expanded. Even though there is no consensus on these types, a recent survey published by Dasgupta et al.[9], attacks are mainly categorized as follows; misuse of resources, User Access Compromise, Root Access Compromise, Web Access Compromise, Malware, and Denial of Service. Cybercriminals generally carry out their attacks based on social engineering techniques that are malicious activities relies on human interactions. Phishing is a popular type of social engineering attack that mainly intends to expose the personal, sensitive, and private data of a user. Cyber-fraudsters execute phishing attacks by sending spoofed electronic mails, instant messages over social media, or text messages while mimicking a trusted source. This textual content generally contains a link redirecting the user to the malicious website. Whenever the recipient clicks that hyperlink, it becomes a potential victim since threats are focused on tricking them by predominating their limbic system. Once the attacker obtains the victim's credentials, it exploits sensitive information about the user, such as passwords, financial accounts, and addresses. Hence,

massive illegal financial losses occur every year due to these phishing attacks [10]. After AOHell announced the first phishing attack in 1996, phishing awareness increased, and many approaches have been suggested in this field. However, despite the attempts to prevent phishing attacks, the number of phishing attacks increases year after year. For instance, in the last quarter report of Anti-Phishing Working Group Q4-2021 [11], they stated that the total number of attacks recorded almost tripled. At the same time, they observed the highest monthly report count in December 2021 by registering more than 300,000 attacks. Moreover, according to the FBI's internet crime report [12] published in 2019, the estimated cost of financial losses caused by these attacks was more than $57 million. Since phishing attacks focus on manipulating fundamental human characteristics, especially under some emotional situations such as being under pressure, fear, or urgency, companies started to conduct in-house training to increase phishing awareness [13]. However, as Bhardwaj et al. [14] stated, even though people get training about phishing attacks, they will still continue to fall into these tricks, and this situation will become a never-ending war. In addition, authors like Ferreira, Coventry, and Lenzin [15] stated that the fight against these malicious attacks must be supported by intelligent systems along with user training. Lastly, Oest et al.[16] conducted a phishing campaign that targeted a specific brand named PayPal Inc. According to their experiments, they concluded the same statement about zero-hour protection by measuring the performance of even one of the most popular list-based anti-phishing services named Google Safe Browsing API could only detect less than %93 of zero-hour attacks.

From that point, as phishing attacks become more sophisticated, various studies have been published in the literature to provide defense mechanisms against these attacks. These approaches can be grouped under different categories [17, 18] based on their source of information, as depicted in Fig. 1.1 (e.g., a snapshot of the website, the Uniform Source Locator (URL), source code itself, or tags inside the source code).

The very first and direct solution against malicious websites is the list-based approach which is mainly based on checking the existence of a given URL in a constantly updated database. The database can consist of either restricted malicious URLs or previously validated legitimate URLs named blacklists and whitelists, respectively. Explicitly speaking,

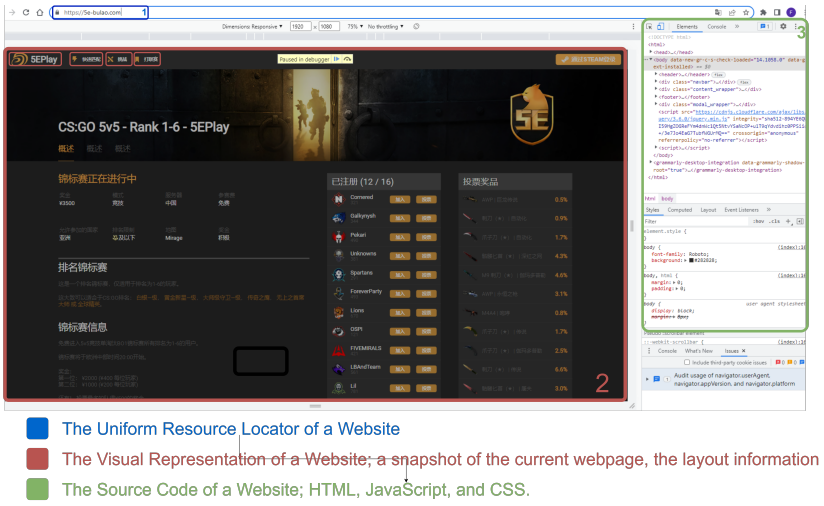


Figure 1.1 An Example of Information Sources of a Website

blacklist-based studies perform the protection by blocking the given website if a given URL occurs in the corresponding malicious dataset. On the other hand, whitelist-based approaches employ a rejection mechanism by default, and the given URL has to validate its legitimacy and must be occurred in that corresponding dataset. Even though list-based methods are powerful, relatively simple, and computationally effective solutions, they are weak against both *zero-hour* attacks and small modifications between phishing instances since they are strictly dependent on the syntactical formation of the URL. For instance, Maneriker et al. [19] observed that the lifespan of a harmful website is decreased drastically from more than 10 hours to minutes. Therefore, updating a list-based database frequently is not feasible, especially when it is a closed system. Moreover, Sheng et al. [20] demonstrated that these methods, unfortunately were highly inefficient against zero-hour attacks by measuring list-based method accuracy of less than %20 in their phishing campaign.

In literature, as previously exemplified, some researchers tend to provide protection on selected domains or brands. Therefore, many of them preferred to use the visual representation of websites such as either a snapshot of or the layout of the corresponding webpage and even the company logo. For instance, Rao and Ali proposed a method [10] that relies on finding visual similarities between the screenshots of benign and phish webpages. In their approach, they extract Speeded up Robust Features (SURF) [21] from screenshots and made classification by using these visual features. However, as they mentioned, their method

inevitably failed to recognize the malicious websites if there are visual differences between visual appearances. In another study different from the previously mentioned approach, Bozkir and Sezer attempted to analyze the visual layout of the web pages through the Histogram of Gradients (HOG) vectors and calculated the visual similarity via Histogram Intersection Kernel [22]. Their method operates on two different sliding windows having sizes of 64 and 128 pixels, respectively. Subsequently, they concatenate them to create combined feature vectors and categorize the documents according to the computed similarity score. Both of these methods expect screenshots of benign and malicious web pages have high similarity. Nevertheless, in many cases, the malicious version does not reflect the original one, and sometimes it only contains a similar logo of that brand since the logo is a universal identity of a brand or company and provides confidence to the user. Thus, Wang et al. [23] studied phishing detection using logo-based visual similarities extracted by employing Scale and Rotation Invariant Features (SIFT) [24]. Even though visual similarity-based studies offer promising results to detect harmful webpages that try to mimic the render of the original one, phishing detection is a global problem that includes numerous domains worldwide. Thus, they considerably suffer from providing an extensive and generalized solution to identify not brand-based attacks.

Since a website is displayed to a user by compiling its source code, some researchers considered using these codes as their source of information. Roughly speaking, the source code contains the HTML, CSS, and Javascript code pieces. In the study published by Mohammad et al. [25], they determined the features and corresponding rules based on the page's source code. They extracted the HTML tags from the code and counted the total occurrences of some specific tags such as Meta, Script, and Link that generally define the relation of a web document with an external source. Additionally, they employed the count of the redirections, the existence of a popup code, and status-bar manipulations using some exclusive Javascript functions like *onMouseOver*. Some of these code-based features proposed by Mohammad et al. have already been employed by a well-known study, CANTINA+, published by Xiang et al. [26]. In CANTINA+, the authors utilized seventeen different textual features, of which the first six of them were extracted from

the URL, the next four of them were obtained from the website's HTML content, and the remainings were collected from Web Services. The previous studies show us that source code-based methods are insufficient to offer a solution per se, and many authors, therefore, have to provide solutions enriched with information such as URL and domain. Moreover, many organizations tend to build dynamic web pages to serve their customers on multiple platforms, such as mobile and web platforms. Thanks to the capabilities of JavaScript and easy-to-use libraries such as React.js and Angular.js, the web is changed from static rendering to dynamic rendering [27]. Therefore, using HTML and CSS contents in further solutions may not be feasible as it used to be.

The one stable source of information that belongs to a webpage is the URL which is its unique address. The URL generally consists of 9 main parts as depicted in Fig. 1.2: protocol, subdomain, domain, top-level domain, port, path, query, parameters, and fragment. The parts after the top-level domain do not require to be involved in a URL. In literature, the vast majority of the researchers proposed phishing detection methods based on the URL information.

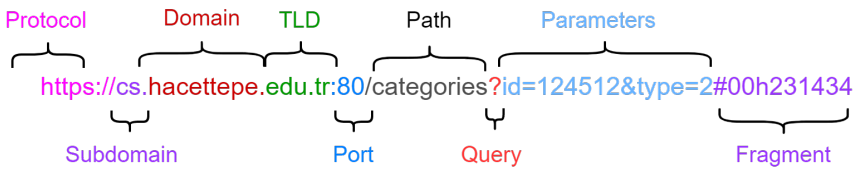


Figure 1.2 The anatomy of an URL

One of the URL-based anti-phishing techniques is building a classifier based on heuristics features. Heuristics could be lexical patterns of a URL or hand-crafted information obtained manually from a URL. The manually declared features could be the presence of protocol information path part of an URL, the number of special characters (e.g., separators, dots, hyphens, etc.), and depth of domain, including subdomains and top-level domain. In many studies, researchers employed a predefined set of lexical features in addition to their existing list-based methods to reduce the weakness against zero-hour attacks [28, 29]. Since the standalone performance of the lexical features-based system is insufficient to provide protection, likewise, Kumar et al. supported this system by employing several domains and

network-based features such as Domain Name System (DNS) Records and Registration Date of the corresponding domain, Web Traffic. Then they implemented a deep neural network architecture that used these hand-crafted features to identify phishing websites. Nevertheless, it is notable that WHOIS and web traffic information are not fully trusted features when considering the increment of newly registered websites every year, according to VeriSign's report. In another approach that enriches the lexical features by combining them with other features, Darling et al. proposed a method that additionally employs character-level n-grams [30]. Then, they trained a Multi-Layer Perceptron to classify a given URL and achieved a great score due to the capability of n-grams in finding syntactical relationships.

Over time, traditional models and heuristic-based approaches became insufficient and started to fail in phishing detection since attackers continuously developed their implementations to bypass a system that involves hand-crafted features. Thus, nowadays, as stated in [19], even modern browsers like Microsoft Edge evolved to run machine learning solutions to seamlessly block phishing threats in the background. At this point, with the advent of Deep Learning (DL), the authors accomplished revolutionary advances in problems in natural language processing (NLP), and inevitably, many others reflected this methodology in their anti-phishing studies. However, as previously exemplified, approaches that employ heuristic-based features in deep architecture could not utilize the full capability of the deep learning model because they were not built in an end-to-end manner and still required a labor force to construct feature sets [28, 31]. On the other hand, some other authors proposed different architectures that preserved the nature of the DL model via composing it in an end-to-end manner [19, 32–37]. In 2018, Le et al. suggested a model named URLNet [32], which fuses both character and word level embeddings and then extracts deep features from these embeddings by using Convolutional Neural Networks (CNN) to detect malicious URLs. Afterward, inspired by URLNet, many studies have adapted similar character-level and word-level embeddings in their approaches while having different architectures [34, 36]. On the other hand, Wang et al. proposed a different model named PDRCNN [33] by noticing that standalone CNN architecture falls short of detecting temporal relationships between embeddings. Thus they encapsulated both CNN and Recurrent Neural Network

(RNN) architectures in their approaches. Apart from the previous methods, Maneriker et al. recently published the URLTran, which employs the state-of-art Transformer models in the phishing detection domain for the first time. Their model utilized a transformer mechanism through Masked Language Modelling, a self-supervised pretraining objective, and is widely used in natural language processing for learning text representations [38]. Even though the URLTran outperformed the previously mentioned studies, it suffers from requiring too high GPU resources when either training the model from scratch or fine-tuning the existing one. Therefore, this situation creates a bottleneck in model inferencing when considering the typical home and office development environments.

As stated above, many studies have been published to identify malicious URLs by training a deep learning model that relies on transforming the textual input into character, sub-word, or word-level representations by adapting a tokenization procedure. Even though these tokenizers were language-dependent at first, they became language-agnostic due to NLP tasks' advances. Due to the syntactical structure, phishing URLs can partially be considered a natural language type. Besides, malicious instances are designated to bypass any defense mechanism by manipulating the parts belonging to the URL structure. Cybercriminals generate exploitations by performing different tricks such as inserting adversarial characters and replacing some letters with their visually similar counterparts known as homoglyphs (e.g., typing paypol instead of paypal). Aside from being able to perform multi-scale analysis by scanning the entire URL with varying-sized text windows, an n-gram itself eliminates the necessity of splitting the URL into sub-parts while requiring a language-specific tokenizer. Thus, in this thesis, our motivation was to provide a deep learning model that employs different n-gram level representations transformed from the URL while hypothesizing that the multi-scale n-gram scanning will extract discriminative enriched representations that provide robust classification.

1.2. Contributions

In this thesis, we propose a novel and scalable deep learning model named *GramBeddings* to recognize malicious websites from a given URL by employing both character and multiple n-gram embeddings in parallel sub-networks. The suggested network model contains CNN, LSTM, and Attention (CLA) architecture to extract discriminative syntactical and temporal features to obtain high performance in terms of accuracy. Even though there are several studies similar to our work [32–37], our approach distinguishes itself from them by adapting a network that performs in a multi-scale fashion, using n-gram based features, and eliminating word or sub-word level knowledge. The main contributions of this thesis can be summarized as follows:

- We suggest a novel and end-to-end trainable neural architecture involving various layers to process character n-grams.
- We present an adjustable pre-processing scheme to select a comprehensive set of best n-grams from raw input data.
- We generate a large-scale collected dataset collected in the long-term, involving 800K phishing and legitimate URLs in total, and make it publicly available for future research.
- We analyze and report the impact of various design choices in terms of embedding matrix and hyper-parameters through an ablation study to improve the model.
- We tested the robustness of the approach against a real-world adversarial attack and explored the ways how to overcome

1.3. Organization

The organization of the thesis is as follows:

Chapter 1 presents our motivation, contributions, and the scope of the thesis.

Chapter 2 describes background information about phishing attacks and their types. In addition, we also provided preliminary knowledge about the anti-phishing techniques used to recognize malicious websites based on the following field of expertise; NLP, ML, and DL.

Chapter 3 provides a comprehensive overview of related studies. At first, we examined the topic under three main categories. We mentioned the information sources used in phishing detection in the first category. Then, we investigated the literature's URL-based datasets in the latter subsection. Finally, we extensively surveyed related works using URLs as their information source

Chapter 4 presents a detailed definition and necessity of the Grambeddings dataset we proposed by explaining why we needed to generate a new dataset and how we constructed it. Moreover, we depicted the importance of this dataset by conducting a series of experiments and providing a comparison chart in this section.

Chapter 5 provides a complete review of the proposed Grambeddings model by analyzing it in three parts. In the first part, we mentioned the URL pre-processing modules responsible for converting the textual data into a dense vector representation by tokenizing it at a specified n-gram level. Then, we examined each component used in deep architecture and their relationships. Lastly, we mentioned the model training phase.

Chapter 6 underlines the experiments that we performed to explore hyper-parameters of the network, choosing the best architecture in terms of multiple n-gram combinations, examining the robustness against adversarial attacks, and comparing it with another state-of-the-art study.

Chapter 7 discusses the proposed algorithm's advantages and limitations, called Grambeddings.

Chapter 8 concludes the thesis, highlights the possible usage in another domain and offers a direction for future works.

2. BACKGROUND OVERVIEW

2.1. Phishing

2.1.1. Definition and Types of Phishing Attacks

Phishing is a form of cyber subterfuge technique used through the Internet to deceive users by posing as a trustworthy entity to acquire their sensitive and private information such as usernames, passwords, and financial accounts. The key factor of why phishing is a successful cybercrime is deception. The attacker tricks users by sending spoofed emails, short messages, and making a convincible voice call. AOHell firstly announced the term phishing in 1996 to describe a program that steals usernames and passwords of America Online users. Since then, the number of Phishing attacks is dramatically increased in recent years; as said by Greg Aaron, "This is the worst period for phishing that the APWG has seen in three years, since the fourth quarter of 2016.". On the other hand, it did not only increase the number of threats but also the number of different phishing techniques to fool users and protection systems.

As the attack techniques become more sophisticated and their targeted environments adapted to the current technologies, many authors divided phishing taxonomy into three main categories to foster better understanding; medium, vector, and technical approach [39, 40].

The medium refers to the channel or communication type preferred by frauds to interact with their victims and deploy their attacks. As illustrated in Fig. 2.1, Frauds mainly used three media to propagate their threats: Voice, Messaging Services, and the Internet.

The Voice based deception techniques are among the oldest and most influential ones that rely on human-to-human interaction through a call. Since it uses personal interactions and hot reactions during the attack, it was easier for attackers to grant users' trust and exploit their personal information.

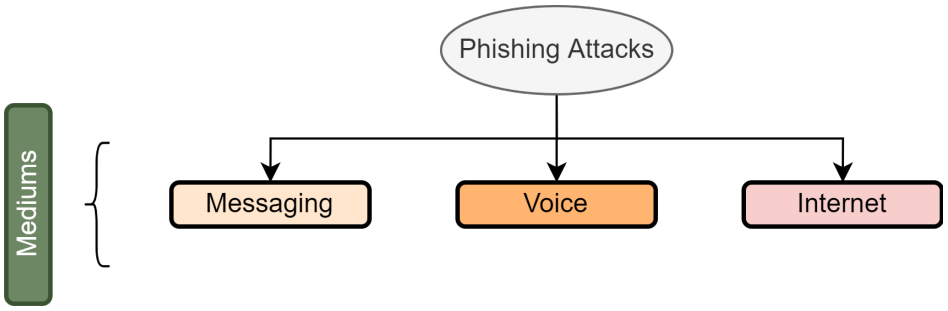


Figure 2.1 Phishing Attack Mediums

The second medium is Messaging, which is employed through mobile phones and called ”texting” in current terminology. In history, Friedhelm Hillebrand and Bernard Ghillebaert introduced the first texting Short Messaging Service in 1984 [41] that was only able to send plain texts. Later, due to technological advances, the Multi-Media Messaging Service is developed that allows messages to contain media such as photographs, videos, and voices in addition to plain text. As ”texting” has become an information-rich communication medium, it turned into a promising opportunity for attackers to inject their threats within a context. These messages generally contain hyperlinks that redirect users to a harmful website to steal their credentials

The last and most popular medium is inarguably the Internet, which substantially impacts our daily lives. As is known, it provides excellent opportunities to humanity in terms of disseminating information, accessing information, and collaborating on the information. However, since people are able to access information effortlessly and quickly on the Internet through their mobile phones, computers, and other smart devices, this convenience of interaction creates an open door for phishers to convey their attacks more solidly by benefiting victims' fast interactions.

Once phishers select their medium to conduct their threats, they will choose the attacking vector, which is how they channel their malicious activities through a selected medium. From this point, vectors are highly related to their mediums and could be categorized as depicted in Fig. 2.2.

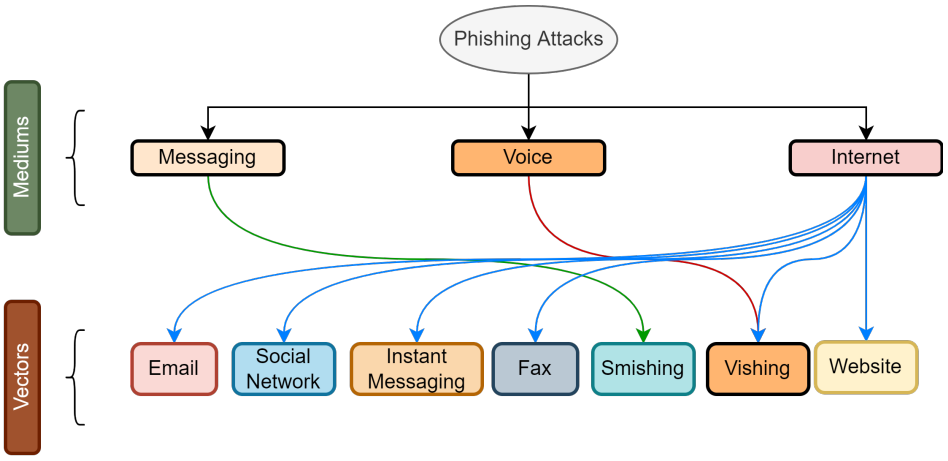


Figure 2.2 Phishing Attack Vectors

The last component of a phishing attack is the technical approach which describes which vulnerability of the selected vector would be employed to deceive users. Some of the most common methods are mentioned below and shown in Fig. 2.3.

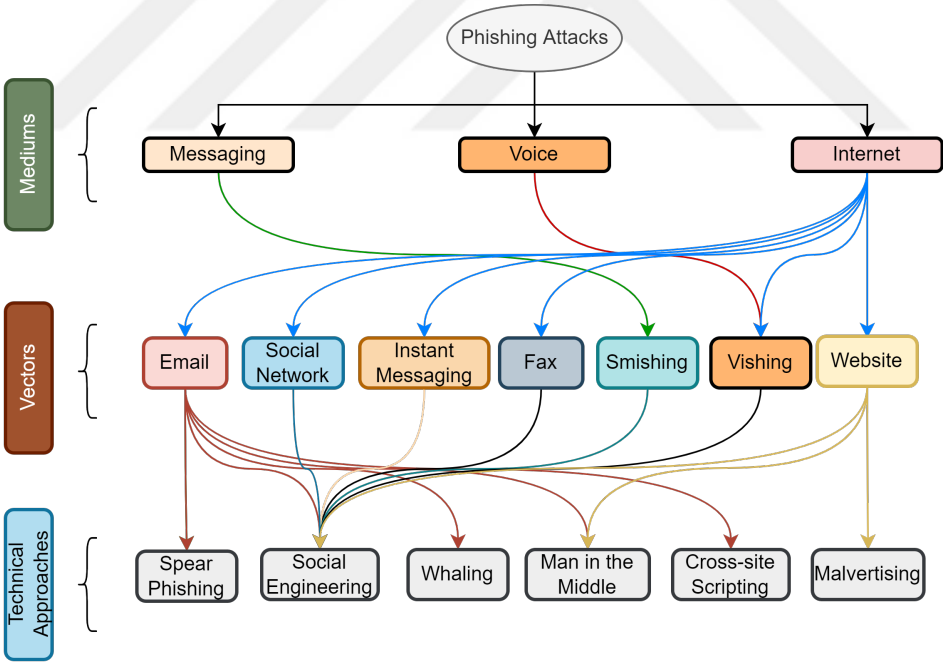


Figure 2.3 Phishing Attack Technical Approaches

Spear Phishing: "is an email or electronic communications scam targeted toward a specific individual, organization, or business. Although often intended to steal data for malicious purposes, cybercriminals may also intend to install malware on a targeted user's computer",

according to the definition made by Kaspersky [42]. Since phishers focus on specific targets, spear-phishing attacks differ from others by requiring a deep knowledge of the targeted person or the company.

Social Engineering: is a type of fraud involving the phishers manipulating the victim's emotions and trust to get them to make irrational decisions. This strategy works by tricking the victim into making emotional decisions instead of rational ones by creating synthetic feelings such as fear, curiosity, sympathy, and anger.

Whaling: is a type of phishing technique that can be considered a specialized form of spear-phishing since it also requires deep knowledge about the targeted individuals or organizations. Whaling attacks are generally performed via electronic mail and fax and are almost indistinguishable from their legitimate versions. However, unlike spear-phishing, whaling attacks select famous or important people as their targets. Because of the victim's highly ranked profile and its privileged access to sensitive or valuable information, these attacks are prepared with the utmost care.

Man in the Middle: is an attack method that monitors the actions of the user by penetrating directly between the user and the related service without permission, instead of waiting for user interactions, and accordingly steals the user's information without consent, unlike conventional phishing attacks.

Cross-site Scripting: (also known as XSS) is an exclusive attack method that injects malicious scripts into trustful benign websites and even manipulates the DOM content of the corresponding webpage instead of creating malicious versions. Hence, it is almost impossible for any browser application to recognize these attacks since requests come from a trustful resource. Moreover, whenever the browser executes the incoming script, the phisher acquires permission to access sensitive information belonging to the victim.

Malvertising: is the abbreviation for malicious advertising, a relatively new cyberattack method where perpetrators inject malicious scripts or codes into benign advertising networks similar to the MIM technique.

2.1.2. Difficulties of Phishing Detection

As mentioned earlier, there are various phishing attacks in the literature, and although many researchers suggested various protection systems to prevent them, phishers continue to find innovative methods to circumvent them. Consequently, phishing detection is an active problem that impacts people's lives in terms of their privacy, financial security, and even social network. Therefore, many authors examine the reason why phishing still works.

Many researchers noted that it is almost impossible to eliminate all the phishing attacks by employing a smart protection system because there will always be a human factor that opens a door for frauds to bypass that system. In a study, M. Alsharnouby et al. conducted a series of experiments to analyze which indicators are generally used by the untrained test group to recognize malicious websites [43]. The results were surprising since only 6% of participants paid attention to fundamental security indicators like the presence of an SSL certificate. In addition, they underlined that most of the users were focusing on the website's appearance, even if it is a replication of the legitimate domain. Then in another study, Krombholz et al.highlighted the importance of user awareness when they concluded their experiments in their survey about advanced social engineering attacks [44]. Recently, Loxdal et al. published their work to contribute to previous methods by introducing the effectiveness of the following four attributes; age, technical proficiency, sex, and smartphone habits [45]. According to their results, they found significant differences between male and female group performances in recognizing malicious websites. However, they did not observe any noteworthy correlation in terms of age, active mobile phone usage, and technical proficiency.

2.2. Deep Learning and Its Building Blocks

2.2.1. What is Deep Learning and Advantages over Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI), and even it has been considered a subset of computer science from many perspectives recently. It can broadly

be defined as a system that is able to learn declared tasks by turning empirical data into domain-specific knowledge through some computational algorithms. As the machine learning methods achieved extraordinary results in solving given problems, they were applied in many different fields such as data analytics, predictive analytics, natural language processing, and computer vision.There are three main learning techniques that enable a machine learning model to improve itself: supervised, unsupervised, and reinforcement learning. In supervised learning, the model requires some labeled data to discriminate samples from each other and acquire expected knowledge. However, in contrast to the supervised learning model, unsupervised learning does not need any label or class information to improve itself. An unsupervised model explores underlying patterns and characteristics from a given context and obtains hidden features and clusters belonging to the input data. On the other hand, reinforcement learning introduces the concept of learning from consequences. The system learns by trial and error methodology, where the system is rewarded if it manages to do the job correctly. Otherwise, it receives negative feedback and tries to regulate itself. Regardless of how it is trained, machine learning-based methods have offered solutions that will make human life easier in many fields from the past to the present. However, traditional machine learning methods have been lacking in effectively solving humanoid problems such as visual recognition. At this point, the concept of deep learning, which is a specialized sub-branch of machine learning, has re-entered our lives, especially after the study named AlexNet [46] by outperforming traditional machine learning-based methods.

The history of deep learning began in 1943 when Warren McCulloch and Walter Pitts introduced the very first artificial neuron model inspired by the human brain's neural network. Later, Frank Rosenblatt introduced the famous model Perceptron, which is accepted as the first artificial neural network in the history. As illustrated in Fig. 2.4, the architecture of ANN consists of three main parts. The leftmost layer is the input layer where our training data enters the architecture. The latter part corresponds to hidden layers whose values are not observable during the training. Finally, the last layer is the output layer, where our model produces the expected results.

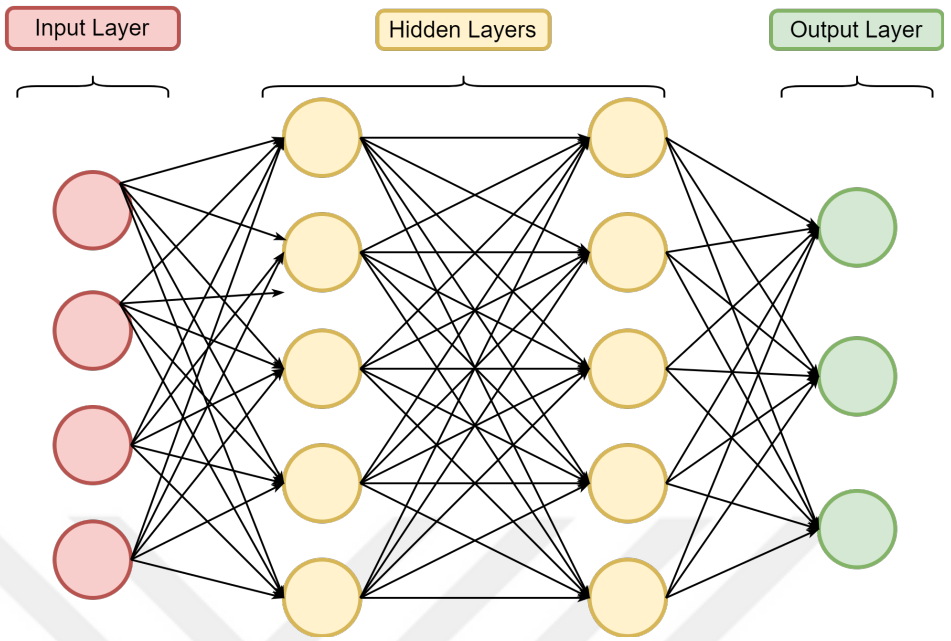


Figure 2.4 Structure of A simple Artificial Neural Network

An ANN comprises interconnected nodes that serve as the inputs to the next node. Since hidden layer values are not observable, ANNs are generally considered as black boxes that transforms the input to the desired output [47]. With the increasing computational power, the design of complex ANN topologies has been made possible. However, many scientific and technological developments had to take place, as shown in Fig. 2.5, for deep learning to reach its current competence.

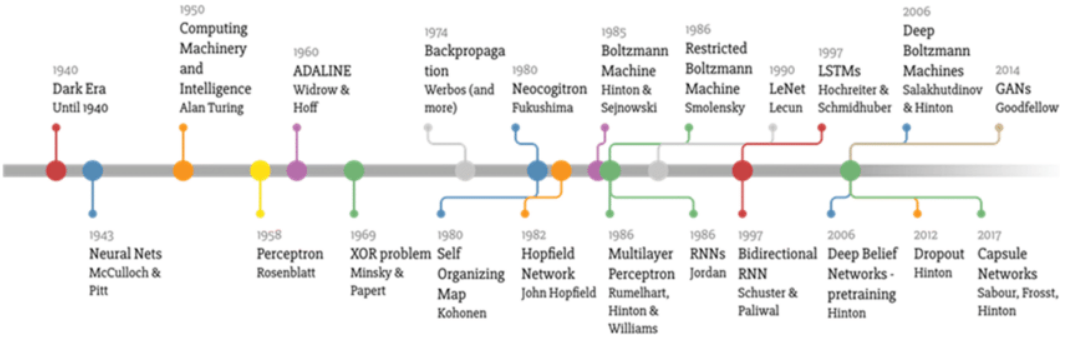


Figure 2.5 Timeline of Deep Learning (adopted from [2])

The following subsections explain some of the most popular Deep Learning layers/architectures used in our proposed network model.

2.2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a specialized class of ANNs, and their architecture is inspired by the structure of the human brain's Visual Cortex. The first modern CNN model was proposed by Yann LeCun et al. in 1998 [48], which demonstrated the power of CNN in extracting complex features from a given handwritten character image. However, this mechanism acquired popularity when AlexNet created a breakthrough in the ImageNet Classification challenge around 2012 [46].

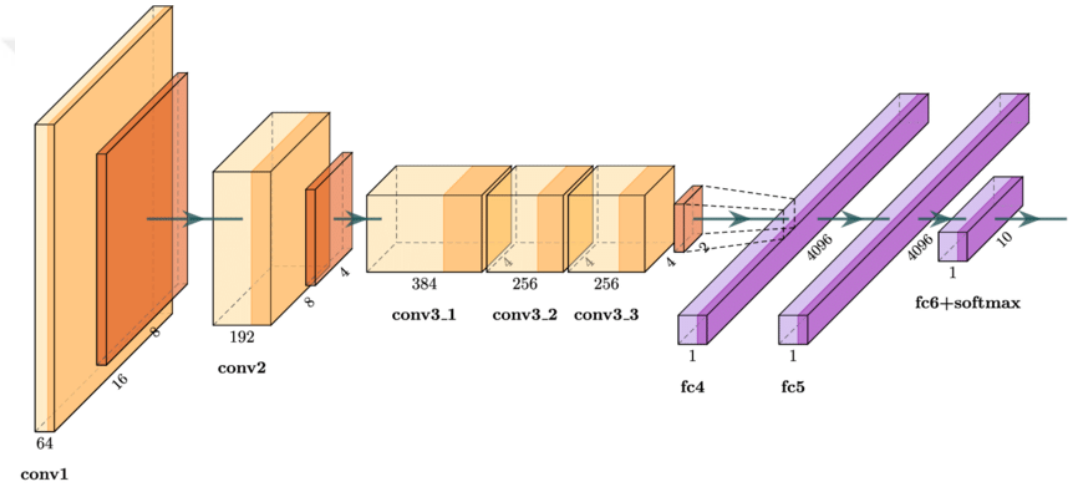


Figure 2.6 AlexNet Architecture (adopted from [3])

The CNN architecture composes convolutional layers and kernel filters to perform convolutional operations as this architecture is illustrated in Fig. 2.6. The kernel is just a unique filter responsible for extracting features from a given input. For example, by assuming the depth of 2D input equals 1, the size of the input matrix is 35x35, and the size of the kernel is 3x3, the size of the resulting output matrix will be 35x35 when applying the following procedure. The kernel slides over the entire input matrix step by step and performs the dot product between the weights of the kernel filters and the value of the corresponding slice in the input matrix, as illustrated in Fig. 2.7.

This dot product results a 2D activation map, and whole process is formulated in Eq. 1, where x and y are the current indices of the slide, and p and q are the related filter sizes.

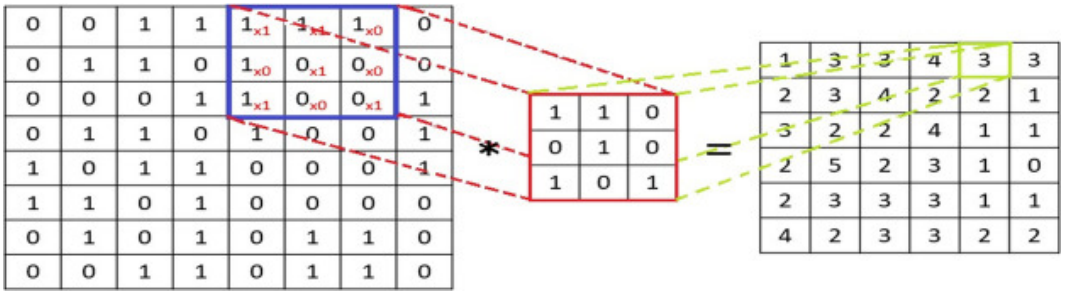


Figure 2.7 An example Convolutional Operation through a CNN (adopted from [4])

$$ActivationMap = Input_slice \cdot Filter = \sum_{y=0}^{columns} (\sum_{x=0}^{row} (Input(x-p, y-q) \cdot Filter)) \quad (1)$$

2.2.3. Embedding Layer

According to the definition given on the official website of one of the popular deep learning frameworks, Keras [49], the Embedding Layer is a class of deep learning that turns positive integers (indexes) into dense vectors of fixed size.

An embedding is a learned representation for text where corresponding subtexts might have the same meaning and have a similar representation. Theoretically, an embedding could be acquired from a text in character, n-gram, sub-word, and word level. The most commonly used embedding type is word embedding.

Before the advent of word-embeddings, many applications employed the traditional bag-of-words approach to solving NLP problems such as document classification and sentiment analysis. However, bag-of-words compose a large sparse vector to represent input textual data, requiring too much computational resource. On the other hand, in an embedding, words are represented by a dense vector that is actually a projection of words into fixed-sized continuous vector space. Besides having dense representation and having low computational cost, embedding layers provide an opportunity to find semantic relationships between embeddings. Word2Vec [50] and GLoVe [51] are two popular

studies in the literature that propose that word-embeddings also offer reusable word vectors containing semantic relationships.From that point, many researchers used those pre-trained word embeddings provided by Word2Vec and GloVe to conduct their phishing detection studies.

Since we use Keras Embedding Layer to construct our embedding matrix from scratch, we explain the working principle of this layer in the following parts. As we mentioned before, the embedding layer only accepts a vector composed of positive integers, which are actually the indexes of each embedding. Thus, any text document needs to be converted into this form of representation at first. This transformation is accomplished by applying tokenization and indexing steps, respectively. The tokenization means splitting a given document into desired substrings that can be performed in character, n-gram, sub-word, and word levels. Once each document in corpora is tokenized, the vocabulary is acquired from a distinct list of extracted substrings. Then each substring is paired with a unique index where the index takes a value between 1 and the vocabulary size. By using this assignment, tokenized documents in the corpus are transformed into an indexed vector form. The whole procedure is visualized in Fig. 2.8, where embeddings are obtained at the n-gram level by setting sequence length to 3.

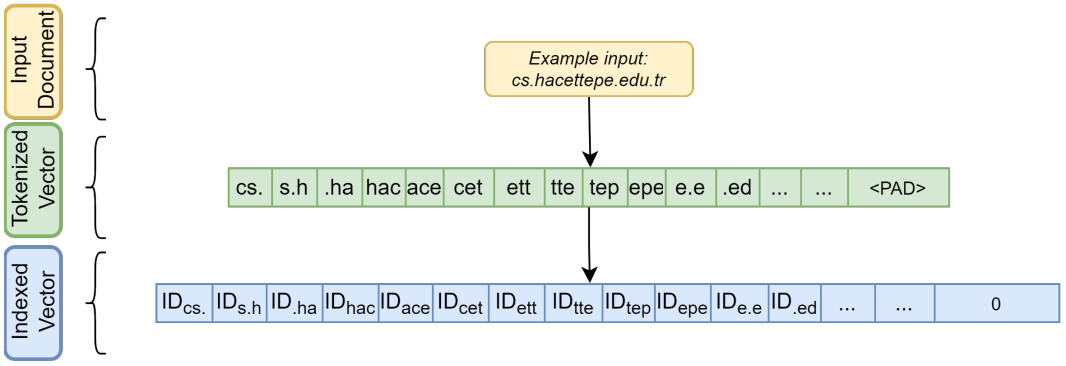


Figure 2.8 An example Document Preprocessing before Embedding Layer

Now, suppose we focus on the mechanism inside the Keras Embedding layer. In that case, we notice two major elements responsible for composing the output matrix in this layer. They are Embedding Matrix and a lookup table, respectively. The embedding matrix is a two-dimensional matrix containing a weight vector for each embedding in the given vocabulary. The weight vector's size is $1 \times d$, where the d refers to the embedding dimension.

Consequently, the Embedding matrix's size becomes $m \times d$, where m is the vocabulary size. On the other hand, the lookup table contains where each index is placed in the embedding matrix. As a result of this mechanism, whenever an input index vector enters the embedding layer, the layer finds the related slices on the embedding matrix using the lookup table. Then, these slices are concatenated together along the y axis and produce the output matrix. This procedure is illustrated in Fig. 2.9.

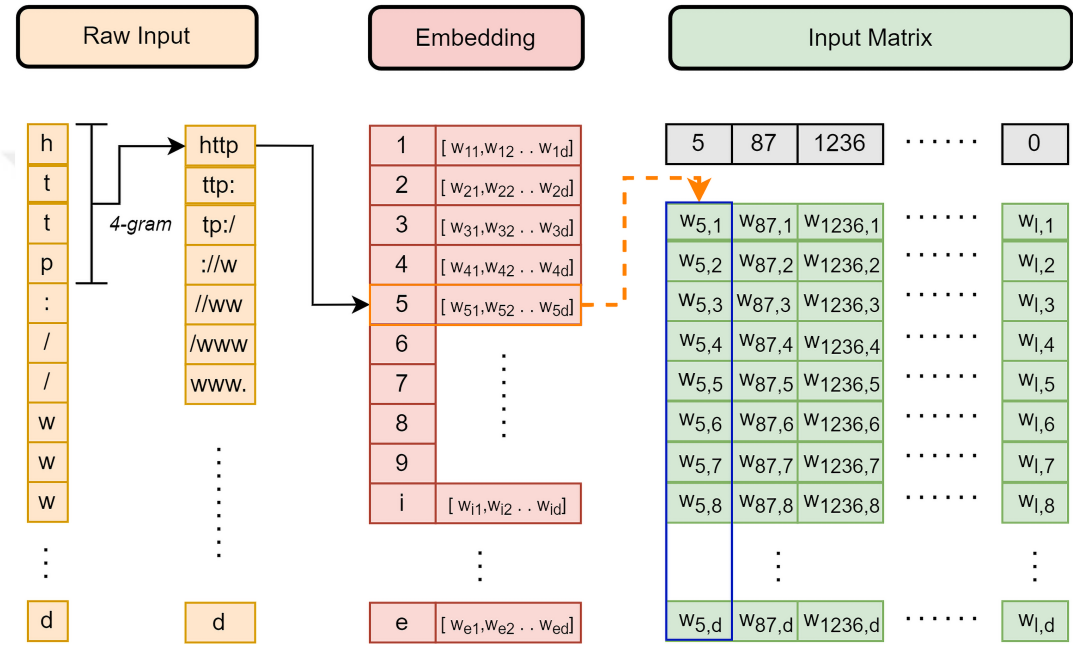


Figure 2.9 An example flow of Embedding Layer

2.2.4. Batch Normalization

The training of multi-layered deep neural networks is a computationally expensive and challenging process due to their complexity in terms of their weights and configuration. During the evolution of deep learning, many researchers tried to find an optimal solution to speed up this process and examined the characteristics of deep neural networks in the training phase. During their observations, they noticed that the network tried to update its weights after each mini-batch, and they thought this could be the reason for slow training and convergence. Moreover, as Ioffe, Sergey, and Szegedy et al. stated [52], most of these researchers believed that it could mitigate the problem of internal *covariate shift*, where

parameter initialization and changes in the distribution of the inputs of each layer affect the learning rate of the network.

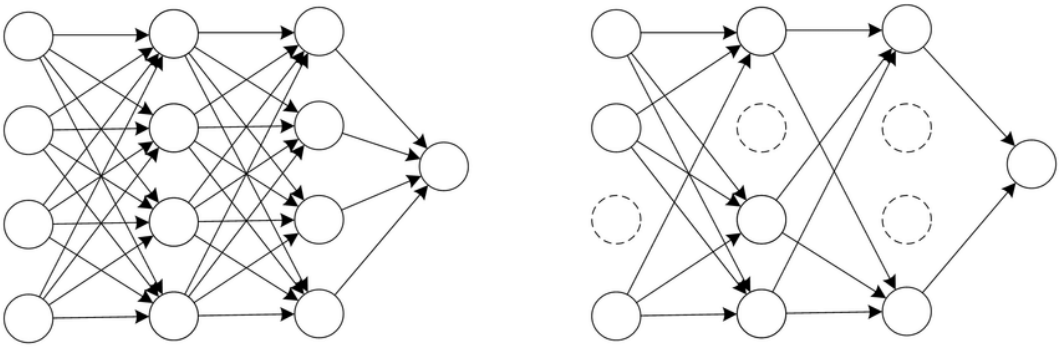
At this point, Ioffe, Sergey, and Szegedy et al. introduced the Batch Normalization (BN) technique to accelerate the deep network training phase. The BN standardizes the inputs before using them in another layer for each mini-batch. In their experiments, when they replaced the Dropout with their proposed BN mechanism, they achieved the same accuracy with 14 times fewer training steps. In conclusion, the advantages of BN can be summarized as follows:

- Reduces the need of preliminary careful parameter initialization
- Provides faster convergence, consequently faster training

2.2.5. Dropout Techniques

It is an inevitable fact that deep learning provided outstanding improvements in many machine learning-related fields. However, one of the drawbacks of deep neural networks is easily over-fitting on training data. Therefore, researchers started to find a technique to overcome this problem and thought that dropping out of both hidden and visible units could do this. Then a popular regulation technique called Dropout (abbreviation of dropping out) was introduced.

As depicted in Fig. 2.10, when dropout is applied to a neural network, randomly selected neurons are blacked out according to the specified proportion or ratio. Thus, remaining interconnections gain more impacts on the network architecture, and the model regulates itself corresponding to these bonds. Thus, remaining interconnections gain more has implications on the network architecture, and the model regulates itself corresponding to these bonds. Since randomly selected neurons are changed at each step, the network avoids over-fitting.



(a) Standard Neural Network (b) Network after Dropout

Figure 2.10 The Structure of Neural Network Before and After Applying Dropout (adopted from [5])

There are several special dropout techniques in the literature, yet in this thesis, we preferred to use Spatial Dropout to regulate our network, which is proposed by Tompson et al. [53]. Unlike the standard dropout, Spatial Dropout drops entire 1D feature maps rather than individual elements to enhance independence between feature maps, as illustrated in Fig. 2.11. Consequently, it minimizes correlation between any adjacent rows in a two-dimensional matrix and offers better regularization and robustness against over-fitting.

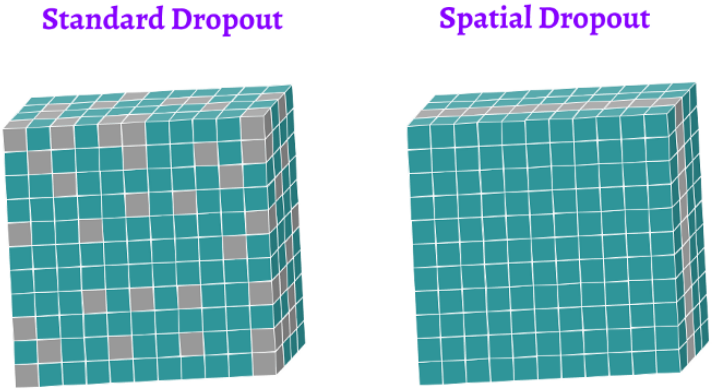


Figure 2.11 Standard Dropout vs. Spatial Dropout (adopted from [6])

2.2.6. Recurrent Neural Networks - LSTMs

At the beginning of the DL era, neural networks were not capable of remembering the temporal or sequential relationships between input data. Therefore, the researchers could not improve the model performance in such fields as NLP. The concept of building a network structure that could remember the temporal relationships between consecutive input samples by implementing a feedback mechanism unlike the traditional feedforward networks was proposed by Elman et al. [54] and Jordan et al. [55]. Their works were accepted as the advent of recurrent neural networks.

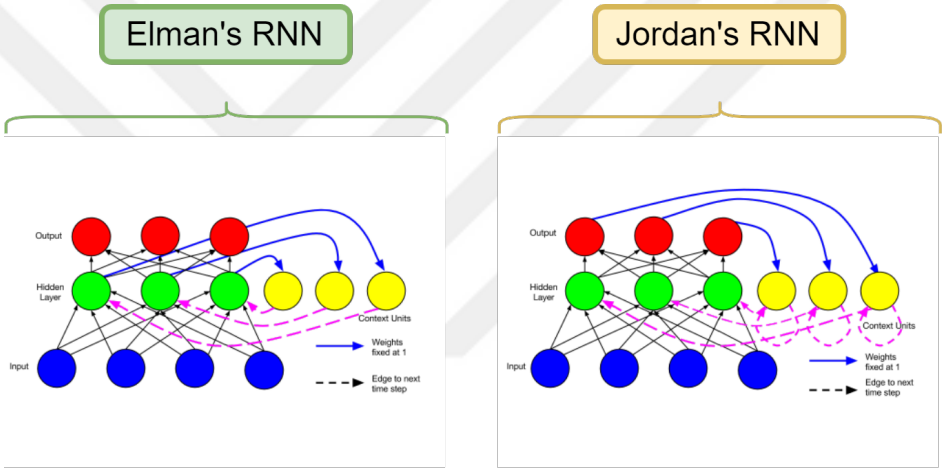


Figure 2.12 Elman's and Jordan's RNN Sturctures (adopted from [7])

These two pioneer approach is illustrated in Fig. 2.12, and as shown in that visualization, Elman's structure was simpler than Jordan's network scheme. The Elman's network is a three-layer network composed of x,y, and z, represented from bottom to top in the illustration and supported by context units colorized with yellow. A one-to-one interconnection exists between each hidden layer cell and the associated context unit. When an input is forwarded to the network, the feedback mechanism (back-connections) holds the previous state of hidden units thanks to the context units. The context units are able to remember the previous state because their bounds enable them to propagate before any learning rule is applied to the model. Thus, the network acquires some sort of memory that allows it to observe sequential relationships, which is beyond a standard ANN's capabilities.

If we recall the timeline of DL depicted in Fig. 2.5, after the advent of simple RNN, Schuster et al. [56] proposed a bi-directional recurrent neural network (BiRNN). The name "bi-directional" stands for preserving state not only from past to future but also from future to past backward. This is accomplished by splitting the neurons in RNN into two directions, one for capturing the information in the forward state and the other for capturing the information in a backward state.

Both RNN and BiRNN have inspired many studies in the literature with their success in finding temporal and sequential relationships. However, as has been revealed in many studies, these network structures have failed to remember a long input temporally because their nature suffered from vanishing gradients and exploding gradients [57, 58]. Thus, many researchers investigated solutions to overcome this problem, and finally, Sepp Hochreiter came up with a new architecture that is able to preserve temporal relationships longer named Long Short-Term Memory (LSTM) [59].

In LSTM, the model proposed a memory cell that takes the responsibility of the standard RNN's context unit, which is responsible for maintaining the temporal information at each timestamp. Unlike the standard context unit, memory cells are able to preserve information over long periods of time. The structure of an LSTM cell and its corresponding equations are given in Fig. 2.13.

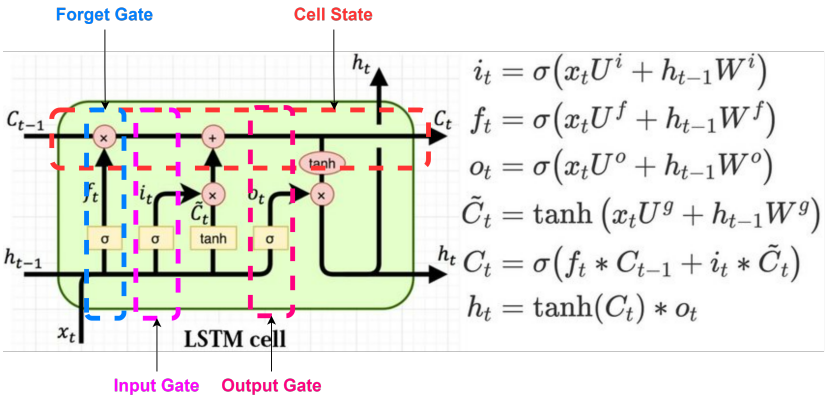


Figure 2.13 The Structure and Formulation of an LSTM Cell (adopted from [8])

In the above graph, *Forget Gate* decides which information should be discarded or kept. *Input Gate* is responsible for updating the cell state. *Output Gate* determines what should

be the next hidden state which preserves the previous state's knowledge. Lastly, *Cell State* refers to an interconnection that carries the information along the sequence chain between Forget Gate and Output Gate.

2.2.7. Attention Mechanism

In the previous section, we mentioned RNNs and LSTM that are proposed to learn sequential relationships from a given context. However, as the input sequence length increases, the recurrent architecture possibly starts to forget that temporal correlations or losses its focus on where to look to extract expected relationships. A similar and even worse scenario also occurred in the Transformer architectures (also known as Encoder-Decoder structure) in the literature [60]. Thus, researchers provide a solution named Attention Mechanism that tries to align the network weights in order to take the benefits from the most relevant parts of the input sequence. In other words, attention provides a cognitive selection from a given input by concatenating relevant parts while ignoring the rest.

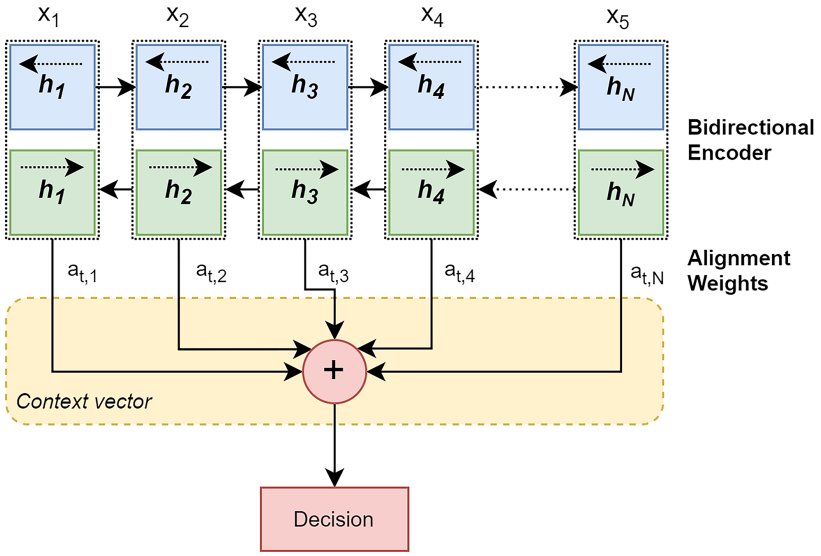


Figure 2.14 Additive attention neural network module

In this thesis, we employed a specialized attention mechanism proposed by Waswani et al. and named *Additive Attention* to align our deep model's LSTM layer [61]. In Additive Attention, the authors employed an alignment model interconnected with each encoded state

of the corresponding recurrent layer. This alignment model is responsible for computing the attention weights used to regulate our model to focus on relevant parts by calculating the correlation scores between inputs around i and outputs j. In the calculation procedure, the context vector is obtained from the weighted summation of hidden states in RNN as described in Eq. (19). Then, each state's weights are computed by applying the formulas (Eq. 20-21). The systematical flow of Additive Attention is depicted in Fig. 5.5

$$c_i = \sum_{j=1}^{T_x} \alpha ij h_j \tag{2}$$

$$w_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x}} \exp(e_{ik}) \tag{3}$$

$$e_{ij} = \alpha(s_{i-1}, h_j) \tag{4}$$

2.3. Evaluation Metrics

In this section, we provide a brief overview of the evaluation metrics used throughout this work. In our study, phishing classification is treated as a binary classification task. Therefore, we did all the evaluations and reported the results in terms of metrics such as accuracy, precision, recall, F1-Score, and Area Under Curve (AUC). As is known, the first four of these metrics rely on True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) counts obtained from the classification results. In addition, the confusion matrix is constructed by use of these four fundamental outcomes. The definitions of all these metrics are briefly outlined as follows.

Since we select phishing samples as the positive class, the count of True Positive (TP) shows the total number of malicious URLs which is correctly classified as phish out of total URLs. In contrast, the True Negative (TN) indicates the total number of malicious URLs incorrectly classified as legitimate out of total URLs. Likewise, False Positive (FP) defines the total number of benign URLs which are falsely classified as phish out of total URLs. In contrast,

False Negative (FN) counts the total number of benign URLs correctly classified as legitimate out of total URLs.

Accuracy: shows the overall ratio of correctly classified samples to total URLs and it is formulated in Eq. 5

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{5}$$

Precision: is defined as the overall ratio of correctly classified malicious URLs to total samples labeled as phish either correctly or falsely and it is formulated in Eq. 6. The higher Precision indicates that model returns more relevant results than irrelevant ones.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Recall: is the ratio of correctly classified malicious URLs to total phish sample in data which is formulated in Eq. 7. Providing high recall rate shows the ability of the model to capture more of the positive samples in the data. However, this cannot guarantee of the model not wrongly identify some negative samples as positive cases.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

F1 Score: equals to harmonic mean of the Precision and the Recall by giving equal weight to these two metrics. It is formulated in Eq. 8.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{8}$$

Receiver Operating Characteristic (ROC) and Area Under Curve (AUC): The ROC is a graph that indicates the performance of a classifier where its horizontal and ordinate axes are False Positive Rate (FPR) and True Positive Rate (TPR), respectively. While the TPR is a synonym for the Recall metric, the FPR refers to the probability of benign URLs being

incorrectly identified as phishing websites. The AUC represents the coverage of the entire two-dimensional area present underneath the ROC curve. It equals the probability that the proposed model will rank a positive sample higher than a negative one as specified by [62] for normalized data. The higher AUC generally points the better performance in distinguishing positive instances from negative ones.

3. RELATED WORK

3.1. Widely Used Information Sources in Phishing Identification

The literature on anti-phishing measures is full of studies that deal with various strategies to catch phishing content from various entities such as web pages, email, and text/HTML content. Even if our proposed model named GramBeddings uses URL as its information source, in this section, we present related works categorized by their strategy sources to give a historical and categorical insight. We group the phishing detection studies under the following subsections; Uniform Resource Locator Based Methods, Vision Based Methods, Content Based Methods, and Multi Modal Studies.

3.1.1. Uniform Resource Locator Based Methods

The Uniform Resource Locator (URL) can be defined as the unique address of any Internet resource. The anatomy of the URL is composed of Protocol, Domain, Subdomain, Top Level Domain, Path, Query, and Parameters depicted in Fig. 3.1. In addition, the domain is directly related to the trademark of the corresponding individual, company, or organization. Thus, phishers focus on fabricating a malicious URL to deceive individuals by manipulating the domain. In a study conducted by M. Alsharnouby et al., they showed that even participants paid attention to the URL of a phishing campaign to recognize malicious webpages [43]. Many researchers used the URL as their source of information in various studies in literature while adopting the same concept. In this section, we grouped those approaches into four categories; List Based Approaches, Heuristic Based Approaches, Machine Learning Based Approaches, and Deep Learning Based Approaches.

List Based Approaches : A list-based approach is the first and most classical way to identify phishing content and is regarded as a fast and light-weighted method to prevent attacks. It focuses on matching the malicious links within a given web page to a list of safe or blocked web pages. In many studies, while safelists are referred white lists containing legitimate

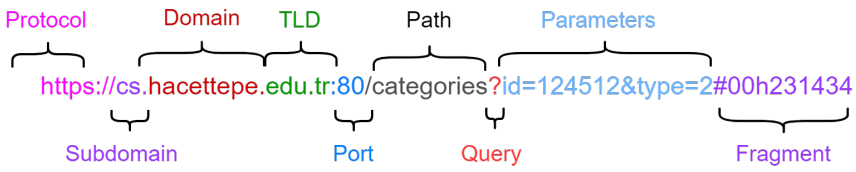


Figure 3.1 The anatomy of an URL

complaint URLs, blocklists are called black lists comprising malicious links that need to be blocked.

Both of these list-based approaches require constantly and frequently updating the control database. For black list-based methods, Dudhe and Ramteke et al. highlighted the importance of the update frequency of blocked lists because of their short lifetime in terms of link activeness in their study. Another supporting work has been published by Sheng et al., which states the average lifespan of a malicious link is approximately 12 hours after its first launch [20]. They additionally stated that list-based methods are weak against zero-hour attacks by showing that they were only able to detect less than 20% of their campaigns within the first hour. Besides, Bell et al. underline the report delay of phishing attacks in popular online blacklists OpenPhish, Phish Tank, and Google Safe Browsing [63]. Also, recently, [64]. developed a Generative Adversarial Network based adversarial attack network that fabricates the phishing version of the targeted legitimate domain. According to their experimental results, their proposed system easily fools the list-based methods.

As a result of the studies mentioned above, we concluded that, unfortunately, due to the high cost of implementing list-based approaches and the weakness of these methods against zero-hour attacks, they can no longer be considered a viable solution.

Heuristic Based Approaches : Heuristic or heuristic techniques can be defined as self-discovery methods that use a practical approach to develop a solution that might not be ideal but can be implemented quickly and effectively. They can be useful when finding an optimal solution for an impossible or impractical problem. They can also help speed up making a decision by providing a mental shortcut.

In the phishing detection domain, heuristic-based approaches are used to reveal the commonly distinguishable characteristics of malicious URLs. One of the very first and most well-known heuristic-based approaches is inarguably the CANTINA+ proposed by Zhang et al. [26]. In this method, the authors employed multiple textual features; six were taken from the URL, four were extracted from the website's content, and the remaining were acquired from web services.

Deep Learning Based End-to-End Approaches : Deep Learning is a sub-field of machine learning inspired by the human brain's structure in design. One of the main differences between DL and ML is that, unlike machine learning, the DL is capable of extracting various features from raw input. This is accomplished by adapting a series of non-linear transformations to derive features from concrete to abstract in an end-to-end manner. In other words, by following a data-driven strategy, deep learning gains an outstanding capability of extracting specific syntactic, semantic, and even temporal relationships from a given context that could not be achieved by conventional machine learning methods before.

In literature, by adopting deep neural networks, many researchers in the NLP domain benefit from acquiring high-level features from textual data to solve related problems such as document classification, sentiment analysis, etc. One of the main milestones in the NLP field is the advent of word embeddings which can be defined as real-valued encoded d-sized vectors able to maintain semantical relationships between words/tokens within a specific corpus [50, 51, 65]. Likewise, in another breakthrough, Zhang, Zhao, and LeCun demonstrated that it is also possible to observe semantical relationships in textual data by using character-level representation instead of word-level ones. In this work, they used a predefined alphabet (See Table 3.1) and encoded a given text by replacing each character with its corresponding order in that alphabet. Then, by employing multiple cascaded CNNs with different kernel sizes, they construct higher-level word-like representations without actually knowing the word.

In the phishing detection domain, many researchers considered the URL as a kind of natural language, and they followed the footsteps of two major deep learning concepts mentioned

Lower Case	abcdefghijklmnopqrstuvwxyz
Upper Case	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Numerical	0123456789
Special Characters	,;.!”?:’/\—_@#$%ˆ&*˜‘+-=¡¿()[]{}

Table 3.1 Characters of English Alphabet that Generally used in Embeddings

above to provide end-to-end URL-based detection approaches with higher accuracies. One of the well-known and state-of-the-art studies named URLNet is suggested by Le et al. in 2018 that outperformed previous traditional machine learning-based methods while eliminating the need for supervision in feature extraction [32]. In this approach, they fused both character and word level embeddings and fed forwarded them into a CNN architecture to recognize malicious URLs. In their preprocessing part, they employed two different strategies for each level of representation while they limited the size of the sample URL to 200 characters to obtain fixed-sized vector representations. In the character level part, they simply followed the strategy stated by Lucun et al. as previously described. On the other hand, in the word-level preprocessing scheme, they firstly collected unique words from their inputs by delimiting each URL via special characters. In addition, since it is not possible to acquire a vocabulary that covers the whole word space, they assumed any word not present in the vocabulary was an unknown word. In other words, an unknown word is encoded by the Out Of Vocabulary (OOV) token’s index while producing a vector representation from textual data. URLNet has inspired many studies that came after it. For instance, Wanda and Jie adopted a dynamic pooling technique while following a similar strategy as URLNet proposed [34].

A more recent CNN-based study was proposed by Tajaddodi et al. in 2020 [36]. Even though their work looks similar to the URLNet architecture at first, it actually differs from it with two main contributions. The first one is employing the adaptive max-pooing technique instead of the standard pooling layer after each convolutional layer. The latter one is using pre-trained word embeddings from FastText. In their work, they showed that even though the URL can be considered a natural language structurally, a pre-trained model targeted to capture contextual similarity is not suitable for URL based phishing detection domain, since the words or sub-words do not need to have semantical relationships between each other.

As mentioned before, one of the advances in deep learning is inarguably the advent of RNNs that allow researchers to obtain sequential or temporal relationships from a given context. From that point, Wang et al. proposed the first study named PDRCNN in the phishing detection field that employs RNN along with CNN, and they achieved promising results in 2019 [33]. In their work, they limit the size of an URL to 255 characters and trimmed or padded the input sequences in that manner. Then, by using pre-trained *Word2Vec* embeddings, they convert input textual data into a vector representation. These encoded vectors are fed forward to two parallel network nodes, which are Bi-LSTM and CNN layers, respectively.

Hybrid Approaches : In the previous part, we exemplified the studies that prefer to use deep learning methods in an end-to-end manner to detect malicious URLs. Even though one of the advantages of deep neural network architecture is obtaining features from the raw data, some researchers consider employing this structure likewise as a traditional classifier to improve their lexical-based solutions in the literature. Since these methods include two different features, which are hand-crafted and deep features, we called them hybrid approaches.

As a first example, Yang et al. suggested a method that packages multiple features from different sources: URL-based deep features, hand-crafted features, webpage's content (code and textual information) features [66]. In order to extract deep features, they build a network structure that comprises CNN and LSTM based on character-level representation. Once they gathered and fused all different features from given websites, they trained the XGBoost classifier to train their proposed model. However, this was not an end-to-end approach and required two-stepped training to build the final model. Hence, their model needed to be fine-tuned with new data, and consequently, their approach was not efficient. From this point of view, recently, Rasymas et al. proposed a network architecture that also employs lexical features with both character and word-level embeddings while preserving being capable of training in an end-to-end manner [67]. They achieved these by building an architecture that has two parallel nodes, where one of them is specialized in learning weights of hand-crafted features, and another one is responsible for learning embedding-based features.

3.1.2. Vision Based Methods

In 350BC, Aristotle noted that *our senses can be trusted but they can be easily fooled*. According to the study written by Richard Gregory claims that only %20 of our visual perception comes through our eyes while the remaining part relies on our inferences [68]. From that point, while some of the phishing makers directly replicate the original web page, the others either slightly change its content or only contain small visual similarities that create an illusion to convince users about they are entering a legitimate website, like having only the company logo on a webpage. Hence, in the anti-phishing domain, many researchers prefer to use the visual representation of targeted websites to recognize phishing attacks. They mainly use snapshots of the corresponding webpage, layout of appearance, and its inner visual markers like a logo as their baselines to detect similarities.

One of the very first studies which employ visual features to detect malicious webpages from their screenshots is proposed by Rao and Ali [10]. In their method, they extracted SURF features from a given image and calculated the similarity between these features and the original screenshot's features for each brand in their dataset. Yet, as also mentioned by them, since their approach strictly depends on the one-on-one match between the visual appearances of each website, it is invulnerable against malicious samples that have even relatively small visual variance. From that point, by noticing the shortcoming of using structural descriptors or features, in a more recent study, Dalgıç et al. employed color-based visual descriptors to observe similarities [69]. Their methods relied on the idea suggested by Chieplinki [70] which pointed out color information is a dominant feature in separating two images. In other words, they believed phishers try to mimic the visual content of benign ones, and they generally tend to preserve the color scheme of the original one. Thus, they utilized the following descriptors to measure visual similarity; Compact Visual Descriptors (CVD) such as SCD (Scalable Color Descriptor), CLD (Color Layout Descriptor), FCTH (Fuzzy Color and Texture Histogram), CEDD (Color and Edge Directivity Descriptor) and JCD (Joint Composite Descriptor)

A logo is a universal identity of a brand/company, and it creates confidence in users subconsciously when they see the logo on a webpage. Therefore, the authors determined to utilize the logo-based visual features as their visual markers in detecting malicious webpages. From this objective, Wang et al. proposed a solution that tries to observe matching logos between a given webpage and legitimate brands via extracting Scale Invariant Feature Transform (SIFT) [24] features on both image [23].

In another study different from the previously mentioned approaches, Bozkır et al. tried to analyze visual layout information from Histogram of Gradients (HOG) features and calculated similarity with Histogram Intersection Kernel equation [22]. In their method, they have used two different sliding window sizes to extract HOG features which are 64 pixels wide and 128 pixel wide cells, respectively. Subsequently, they have concatenated in order to create feature vectors and are finally classified by their similarity scores.

In 2012, Krizhevesky et al. proposed a CNN model named AlexNet [71], which achieved the state-of-the-art results in the visual object recognition challenge ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This breakthrough created a massive impact on Computer Vision studies, and subsequently, many authors took advantage of using Deep Learning in their studies. As an example of applying DL techniques in detecting phishing websites on visual similarities, Abdelabi et al. pointed to triplet CNN so-called WhiteNet [72]. The triplet CNNs (The Siamese Networks) was firstly introduced by Koch et al. [73] on The Siamese networks in the study of FaceNet, and the idea behind this network structure was to eliminate retraining costs when adding a new class or sample in our system while trying to have an in the end. In WhiteNet, the authors trained their system in two phases; in the first one, they took random samples for their triplets to cover all combinations in which all screenshots belonging to the same website have the probability of selection, and they put all together either on anchor image input or negative image input. After they generalized their network for all the samples, they needed to tune the model iteratively to be able to find the hardest examples, in which case those are hardly distinguishable website instances once. The secondary training subset is generated the retrained in their network to have a better

result. Finally, thanks to the final convolutional layer at the end, they have combined feature space that allows them to classify a given webpage.

3.1.3. Content Based Methods

A webpage is visualized to a user when its content is rendered by a browser. Generally, this content is considered the Object Model (DOM) and comprises HTML, JavaScript, and CSS codes. Thus, in the literature, some researchers thought that extracting some textual features from a website's content could be useful for detecting phishing webpages.

Ramesh et al. parsed the hyperlinks from the HTML content of corresponding websites [74]. Then, they compared these links with the DNS lookups containing the user's previously visited URLs. However, since it requires too many resources and creates a lag when finding associated links within a database, it was not a sufficient method.

In another study, Haruta et al. fused multiple webpage features consisting of hyperlinks from HTML, a screenshot of a webpage, and CSS content [75]. They used CSS to maintain the position of *locally different images* such as logos, and the motivation behind the use of CSS is to find visual similarities between visual markers even if the layout information is changed.

3.1.4. Multi Modal Studies

The name Multi-Model refers to studies employing more than one source of information to provide a solution in the anti-phishing domain in the literature. For instance, Rao and Pais et al. proposed a study that comprises visual and textual features in order to detect malicious attacks [76]. This method is initially a search engine-based method that tries to verify the legitimacy of a given URL. However, as they mentioned in their work, they supported this decision system by employing multi-modal features. In the visual similarity part, they used the screenshot of the given webpage. On the other hand, in the textual part, they employed the URL, HTML content, and CCS style content to determine if there was a malicious injection

in the codebase. Consequently, they achieved promising results when it is compared with previous search engine-based methods by improving it with another source of information.

3.2. URL Based Datasets for Phishing Detection

In the anti-phishing literature, the vast majority of researchers, as well as us, prefer to utilize URL-based features to detect malicious websites. Thus, some of them are also published as publicly available datasets to provide benchmark data for future works. In this section, we mentioned datasets that are noteworthy and accessible, according to our best knowledge. They are; Ebbu17 proposed by Sahingoz et al.[77], PDRCNN submitted by Wang et al.[33], PhishStorm published by Marchal et al. [78], and ISCXURL2016 shared by Mamun et al. [79]. Additionally, we also consider some Kaggle datasets, even though these datasets may not initially be submitted in any study. These Kaggle datasets are the followings; KD01 was published by Siddhartha [80], KD02 was shared by Akshaya [81], and KD03 was proposed by Kumar [82].

On the other hand, in this thesis, we also publish a dataset named GramBeddingsDB to provide a comprehensive URL-based benchmark dataset, along with a deep learning model. The reason why we needed the collect our own dataset and its advantages against existing ones are given in Section 4.

4. GRAMBEDDINGS DATASET

4.1. Why We Require A New Dataset?

Nowadays, as we mentioned in the Section 3., most studies employ machine learning and deep learning based methods to recognize malicious websites via their URL. However, as their model becomes deeper or more complex, they face the well-known risk of overfitting. Moreover, another risk occurs when not having a diverse and large-scale dataset because the proposed models may learn in an imbalanced way or would not be able to generalize real-world examples. Nonetheless, we could not find any suitable URL dataset for that purpose even though we conducted an exhaustive search in the literature. Thus, we needed to collect our own dataset to solve the following highlighted problems:

- **Avability:** Even though there are numerous URL-based studies, unfortunately, the number of publicly available UR datasets is limited.
- **Scarcity:** The size of existing and available datasets are generally relatively small to train machine learning and deep learning-based model since proposed methods would suffer from over-fitting.
- **Class imbalance:** Some of the datasets involve not balanced class distributions and the fraction of benign samples is far more than malicious ones in general.
- **Only home pages of legitimate URLs:** As a conclusion to our endless appropriate dataset search, we noticed that in many datasets, legitimate samples are often the collection of index pages of the corresponding domain. However, this situation yields that the length of benign URLs becomes significantly shorter than the malicious ones.
- **Low diversity:** In many studies, malicious samples are collected from well-known services such as OpenPhish [83] and Phishtank [1] in a relatively short period of time. However, according to our observations, the phishing samples in these domains generally replicate the same malicious domain with small differences when

the collection is crawled daily. This produces an intra-class variation problem that misleads the model's learning phase and eases the classification problem.

To this end, to solve the problems stated above and train our model with ideal data, we collected our own dataset named GramBeddings dataset. We shared it with the community to provide a better benchmark dataset for future works. Additionally, in order to provide transparent insight into the crawling strategy, we shared the details in Section 4.2.. Moreover, we showed the advantages of the Grambeddings dataset against them in Section 4.3. while analyzing the characteristic of our dataset and comparing these characteristics with existing ones.

4.2. Dataset Generation

As we mentioned before, we started to build our own dataset due to the aforementioned problems. The crawling process began with collecting phishing samples from popular services such as PhishTank and OpenPhish. However, unlike the many other datasets, instead of crawling from these services daily, we expanded our collection period to almost two years to eliminate duplications, and domain-based low variations within service-provided samples, as illustrated in Table 4.1.

On the other hand, in the legitimate collection aspect, unlike the previously published datasets, we built our own special crawler to collect benign samples from the Internet instead

Table 4.1 Sample Phish URLs from PhishTank [1]

ID	URL
7462768	https://www.login.kddi-jp.com.gffpvjx.cn/
7462767	https://www.login.kddi-jp.com.frhnrbd.cn/
7462766	https://www.login.kddi-jp.com.gejswot.cn/
7462765	https://www.login.kddi-jp.com.flekzgv.cn/
7462764	https://www.login.kddi-jp.com.fremlgj.cn/
7462763	https://www.login.kddi-au.com.eftjiou.cn/
7462762	https://www.login.kddi-jp.com.efpkirk.cn/

The targeted website is *https://www.kddi.com/*

of gathering them from popular website ranking services such as Alexa and Majestic. We did not prefer to use the links provided by these services because they generally consist of the index pages of the targeted domains. Consequently, they would cause the following problems on the dataset in terms of the model training phase.

- **Imbalanced Length Distribution:** A considerable difference between the average lengths of both phishing and legitimate samples could possibly occur. Thus, the vast majority of character, n-gram, sub-word, and word-level features would be extracted from malicious samples, which could mislead any model into learning an imbalanced way.
- **Missing Semantic Relations:** Since legitimate samples would generally not contain parts that come after the TLD part (See Table 3.1), any model could possibly lose meaningful semantic and syntactic relations from a given context. Thus, unintentionally that model would become invulnerable against real-world examples.

From that point, we adopted the following strategy in our crawler. Our crawler starts collecting 400 unique URL lists which belong to the top twenty websites for every twenty different countries and iterates through each candidate sample in the dataset. We select and process the next link to extract hyperlinks from its HTML content at each iteration. Next, we eliminate the extracted links, which are empty, internal navigational links and not actually structured as the valid URL anatomy. The remaining URLs are grouped with respect to their corresponding domain information, and they are also ordered by their lengths within each related domain group. Then we started to filter by domain information. In this step, we checked the presence of each domain in our database and created a new group if it did not exist. Furthermore, we followed our URL Insertion Rule given in Fig. 4.1 to prevent oversampling per domain. Once we perform all eliminations, the remaining URLs are inserted into our dataset as candidates for future crawling steps.

Note that we executed this procedure recursively until the total number of URLs in the database reached 2 million. Then, to provide balanced distribution between phishing and

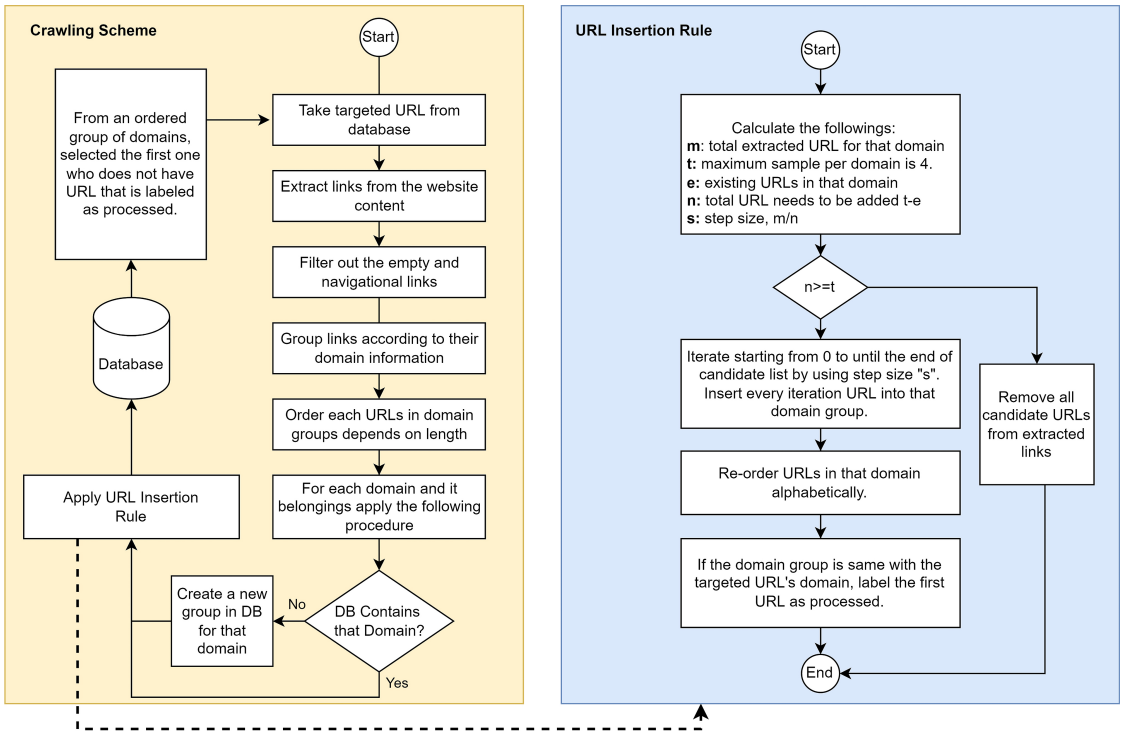


Figure 4.1 Legitimate URL Crawling Scheme

legitimate samples in the final Grambeddings dataset, we randomly selected 400k benign over those 2 million samples.

4.3. Dataset Comparison

Until this section, we mentioned both the problems that occur on the publicly available datasets in literature and our crawling strategy to cope with these problems. In this section, we compare our dataset characteristics with others. In order to conduct a comparison between them, we computed the following properties to measure distinguishable statistics; sample size, average URL length, unique domain count, and unique TLD count. The comparison is done over the following dataset; Ebbu17 [77], PDRCNN [33], PhishStorm [78], KD01 [80], KD02 [81], KD03 [82], and ISCXURL2016 [79].

In the first property, we examined the total sample count and class-wise distributions to point out whether their sizes are sufficient or not to work with deep models, since the lack of data could cause the well-known over-fitting problem as mentioned before. As can clearly be seen

Table 4.2 Comparison of different datasets in terms of various properties

Dataset Name	Sample Sizes			URL Length Properties			Legitimate Domain Statistics		Phish Domain Statistics	
	Phish	Legitimate	Total	Avg. Phish	Avg. Legitimate	Mean	# Unique Domains	# TLDS	# Unique Domains	# TLDS
GramBeddings	400000	400000	800000	86.21	46.43	66.32	277848	1563	128119	1213
Ebbu17 [77]	36394	37173	73567	59.35	78.51	69.03	8733	354	12535	308
PDRCNN [33]	15257	507138	522395	86.05	14.76	16.84	465734	1745	5672	391
PhishStorm [78]	48009	48007	96016	77.97	47.49	62.73	16792	514	13475	457
KD01 [80]	223074	428102	651176	64.91	57.67	60.15	301	48	13612	531
KD02 [81]	900	500	1400	41.59	18.59	33.37	412	47	101	33
KD03 [82]	104438	345738	450176	66.04	58.48	60.23	93005	508	42340	836
ISCXURL2016 [79]	9965	10000	19965	84.41	115.63	100.05	248	42	2917	213

from Table 4.2, the KD-02 dataset potentially suffers from an over-fitting problem since it has a notable lack of samples. On the other hand, besides the total sample count, another concern is having a balanced class distribution over those samples. Otherwise, an imbalanced data distribution could potentially cause a high bias and poor generalization capability. The high bias means that a model would align itself according to the vast majority of the class-wise population since the model incorrectly gains more accuracy by doing that. Consequently, it would inevitably mislabel the incoming sample which belongs to a less frequent class and this situation reduces its generalization capability. From this objective, the PDRCNN dataset suffers from the class-imbalance problem since the vast majority of the examples in it are legitimate URLs.

The second criterion was the class-wise length distributions of URLs, which could possibly mislead the model to mostly extract syntactic and semantic relations from the biased class. As exemplified before in Table 4.1, phishing websites tend to have longer addresses than their targeted legitimate ones. After our observations, we noticed that some of the datasets only contain home page addresses of related legitimate domains. However, it is essential to have an attention to having close inter-class average length distributions when building a dataset. Otherwise, as stated before, some of the NLP metrics (e.g., term frequency (TF), document frequency (DF), term frequency-inverse document frequency(TF-IDF)) would be biased. Consequently, incorrect terms (e.g., n-gram, word, and subword) could emerge due to poor feature selection based on manipulated NLP metrics. Then, any method that utilizes these NLP-based tokens would capture incorrect and inadequate temporal, sequential, semantic, and syntactical features from a given context. From this objective, during our legitimate sample crawling procedure, we focused on collecting not only the index page of the

corresponding domain but also randomly involved three longer unique variations of the same domain in enriching diversity as opposed to KD02 and PDRCNN datasets.

Another factor that motivated us to build our own dataset was alleviating the intra-class diversity problem. Intra-class diversity means the variation of samples within each class which are phish and legitimate in our case. In order to measure this variation, we set two criteria which are unique domain count and unique TLD and calculated them per class. Then we obtained the related ratios by dividing them with each criterion with the total sample count. While the unique domain rate indicates degrees of domain-wise diversity, the unique TLD rate underlines country-wise diversity since country-code TLDs (ccTLDs) are widely used in the domain registration processes. Note that, we used a popular domain parsing javascript library named TLDParser [84] and the ratio is calculated by dividing the obtained TLD count by the maximum TLD count where the total TLD count is retrieved from the Internet Corporation for Assigned Names and Numbers (ICANN) and consists of 8812 different TLDs. When we examine domain-wise diversity, we set unique domain rate threshold values of 0.5 and 0.3 for benign and malicious samples respectively due to the fact that it is harder to obtain unique domains for phishes. As given in Table 4.2, except for our dataset, only the PRDCNN dataset meets the expectation. Moreover, when we compare the TLD rates, we could proudly state that our dataset has the highest number of TLDs in both legitimate and phishing samples and covers more than 10% of existing TLDs.

5. A NEW ARCHITECTURE: GRAMBEDDINGS

Our proposed method Grambeddings is a special ensemble-type DNN that recognizes phishing samples from a given URL and works in an end-to-end manner. The reason for being an ensemble-type deep model is having four separate pipelines to extract and learn features in different levels of character sequences. Each pipeline consists of two main consecutive phases which are embedding creation and deep feature extraction respectively.

The information flow within each pipeline can roughly be described as follows. First, when an URL string is fed into our model, the given text is transformed into four different level representations consisting of one character level and three n-gram levels. This transformation is done by employing a tokenization module(*Character Embedding Module* (Section 5.1.1.) and *N-Gram Embedding Module* (Section 5.1.2.)). Then, these transformed and encoded vectors are fed forward to a deep architecture named Single Embedding Network Model (Section 5.2.). Note that, the structure of each Single Embedding Network Model is identical across different level representations. Once the deep features are obtained from a single network, they are fused with other 3 different levels of deep features to acquire full-fledged features. Finally, these combined features are fed into the fully connected layer to train our model and make predictions.

5.1. Preprocessing Modules

Like many NLP-based applications, our model also requires some preprocessing steps to prepare an appropriate input for our deep neural network architecture. On the other hand, since our model basically depends on both character level and n-gram level features, we have two different preprocessing procedures to convert input textual data into the expected form.

5.1.1. Character Tokenization Module

The character tokenization module is the one specialized in converting given textual input to the character level encoded vector representation. This conversion is done by applying the following procedure, which is also demonstrated in Fig. 5.1. First, each input URL is trimmed if it is longer than the expected maximum sequence length (l_{max}) due to the fact that the length of the URL varies. Then, we split each fixed-length URL into characters and started to encode each character. The encoding is done by replacing each character with their corresponding orders in the alphabet given in Table 3.1. Note that the order yields to their alphabetical order where the indices of the first letter ('a') and the last character ('\') are 1 and 94, respectively. Lastly, in order to maintain fixed-sized vector representation across whole URLs, we checked the final encoded vector sizes. We pad the vector shorter than the expected maximum sequence length (l_{max}) we padded by inserting the adequate number of zeroes at the end of that vector.

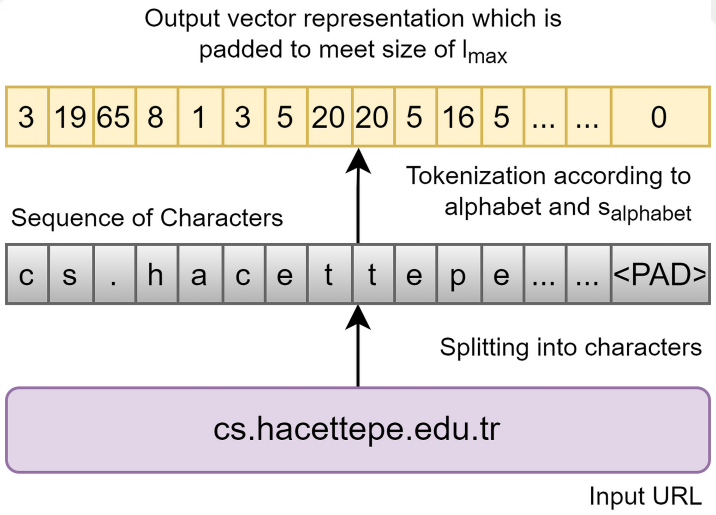


Figure 5.1 An example of character level tokenization

It is noteworthy that our algorithm supports an adjustable maximum sequence length parameter. We achieved the best score by setting (l_{max}) to 128.

5.1.2. N-Gram Tokenization Module

In this thesis, the name of GramBeddings comes from n-gram embeddings, which is one of the contributions of this study. Thus, the N-Gram Tokenization Module is the key factor that not only transforms a given input URL into an encoded vector representation but also provides an n-gram selection algorithm to ensure the deep neural network model accepts fixed and limited embeddings to construct itself.

As stated before, it is critical to limit the n-gram count used to encode given input because the number of possible character combinations N_{n-gram} exponentially increases with respect to the *n*-gram and S_{alph} parameters as it is formulated in Eq. 9. The parameter *n*-gram and S_{alph} stand for the character sequence length (i.e. n value) and the unique character count in alphabet, respectively

$$N_{n-gram} = S_{alph}^{n} \tag{9}$$

It is noteworthy to underline that the massive number of n-gram combinations makes our model unscalable and heavier in terms of required learning parameters. In fact the model possibly becomes prone to the well-known the Curse of Dimensionally problem.

From the point of view stated above, we employed the chi-square analysis method to select features by measuring the independence of two categorical variables.Theoretically, our objective about n-gram selection was that if the computed chi-square score χ^2 is high is highly correlated with its corresponding document class, which is either phish or legitimate URL in our case. Thus, we eliminated the n-grams that have low scores to maintain fixed-sized vocabulary enriched by only using the most important n-grams. The χ^2 scores are calculated by applying the formula given in Eq. 10

$$\chi^2(t, c_i) = \frac{N \times (AD + CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)} \tag{10}$$

where t denotes the *term* (n-gram) to be weighted and c_i represents the i_{th} category where $c_i \in \{0, 1\}$ in our case along with the N referring URLs count in training set. Besides, A

represents the number of samples in category c_i that contains the *term* t whereas B is the sample count in specific category c_i that does not contain the *term* t. Following, C denotes the number of samples not in category c_i that contains the *term* t while D is the number of samples not in category c_i that does not contain the *term* t.

Once we obtained the chi-square scores for each distinct n-gram extracted from a given corpus, we sorted them in descending order according to their calculated scores. Next, we selected the top k of them to limit vocabulary sizes that will be used in both tokenization and deep learning phases, where the parameter k is adjustable by the user. Afterward, we started to apply the following process, illustrated in Fig. 5.2, to convert both training and validation data into the encoded vector representations.

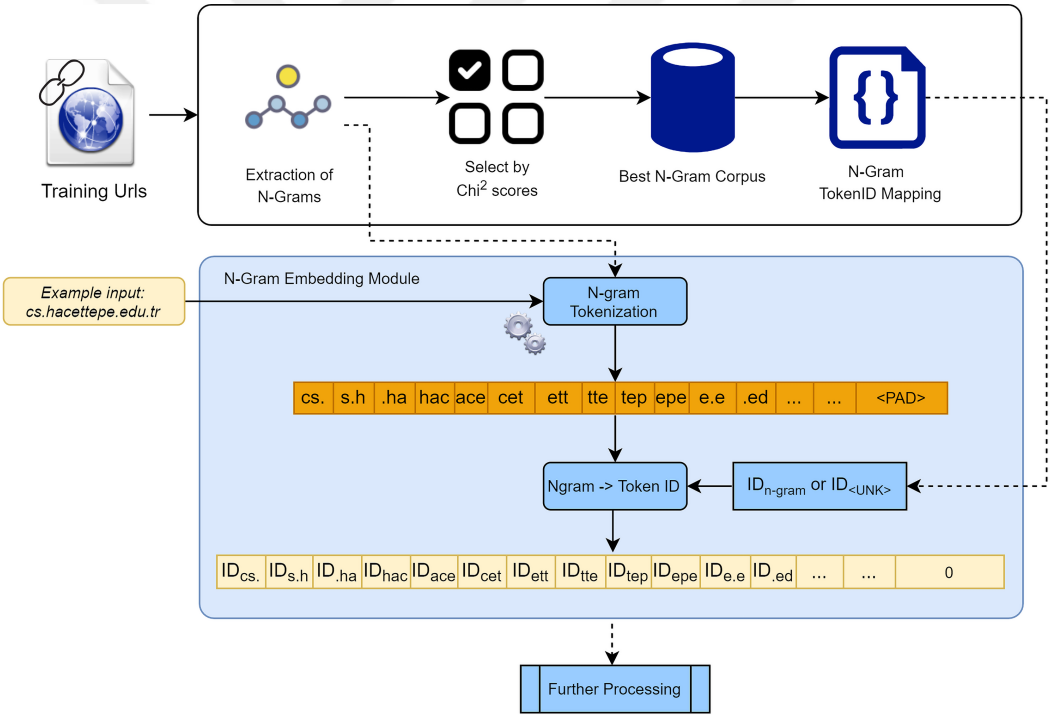


Figure 5.2 N-Gram embedding matrix generation and term-embedding mapping module

As illustrated in Fig. 5.2 above, we needed to provide a selected n-gram corpus before starting the tokenization, unlike the Character Tokenization Module. Hence, we did not manipulate the input URLs by either trimming or padding them, as we did in the Character Tokenization Module. Once we obtained the required selected n-gram corpus, we started tokenization by splitting the given textual input by the specified n-gram size at first as

formulated in Eq. (11). Next, for each extracted n-gram in tokenized vector form $\mathbf{v}_{tokenized}$, we checked their existence in the selected n-gram corpus. If it exists, we map that n-gram with its corresponding index, where the index yields the n-gram's order in the selected n-gram corpus. Otherwise, the n-gram of interest is considered an Out Of Vocabulary (OOV) term, and it is encoded with the $Id_{<UNK>}$ by applying the formula given in Eq. (12). The last encoding step is formulated in Eq. (13). Finally, we aligned the produced vector by either padding or trimming it to maintain a fixed-sized representation concerning the predefined total sequence length (i.e., 128).

$$\mathbf{v}_{tokenized} = f_{tokenize}(u) \tag{11}$$

$$Id_{<UNK>} = f_{count}(corpus) + 1 \tag{12}$$

$$\mathbf{v}_{encoded} = f_{indexing}(\mathbf{v}_{tokenized}) \tag{13}$$

5.2. The Architecture

In this thesis, we proposed a DNN named Grambeddings that specialized in utilizing both one character level and three n-gram level features to detect malicious websites from a given URL. In other words, it is a quadruple network that involves different levels of representation of the same context. Each parallel network within this quadruple architecture is named Single Network, and they are responsible for acquiring exclusive syntactical and sequential relationships between its corresponding features. It is noteworthy to highlight that each *Single Network* has the same architecture where its parameters change with respect to their input sequence characteristics such as n-gram size (n), vocabulary count, etc.

The structure of each *Single Network* consists of four parts. In the first part, we feed forward the output of any Tokenization Module (either character level or n-gram level) to the specialized Embedding Layer to prepare adequate input to the following layers from input encoded vectors. The second part comprises two cascaded convolutional layers responsible for extracting hierarchical and structural features from the embedding layer's output. The

next part is the recurrent layer, Bidirectional-LSTM, which is utilized to acquire deep sequential relationships between the deep features obtained from CNNs. The last part introduces an Attention Mechanism to align recurrent layers to pay attention to the more relevant part of the sequence while improving the model's accuracy. Once we obtained the attention layer's output from each *Single Network* architecture, we fused them to achieve full-fledged vector representation. Then, we fed forward this full vector representation into a two-layered, fully connected network to get the final decision from the Grambeddings model.

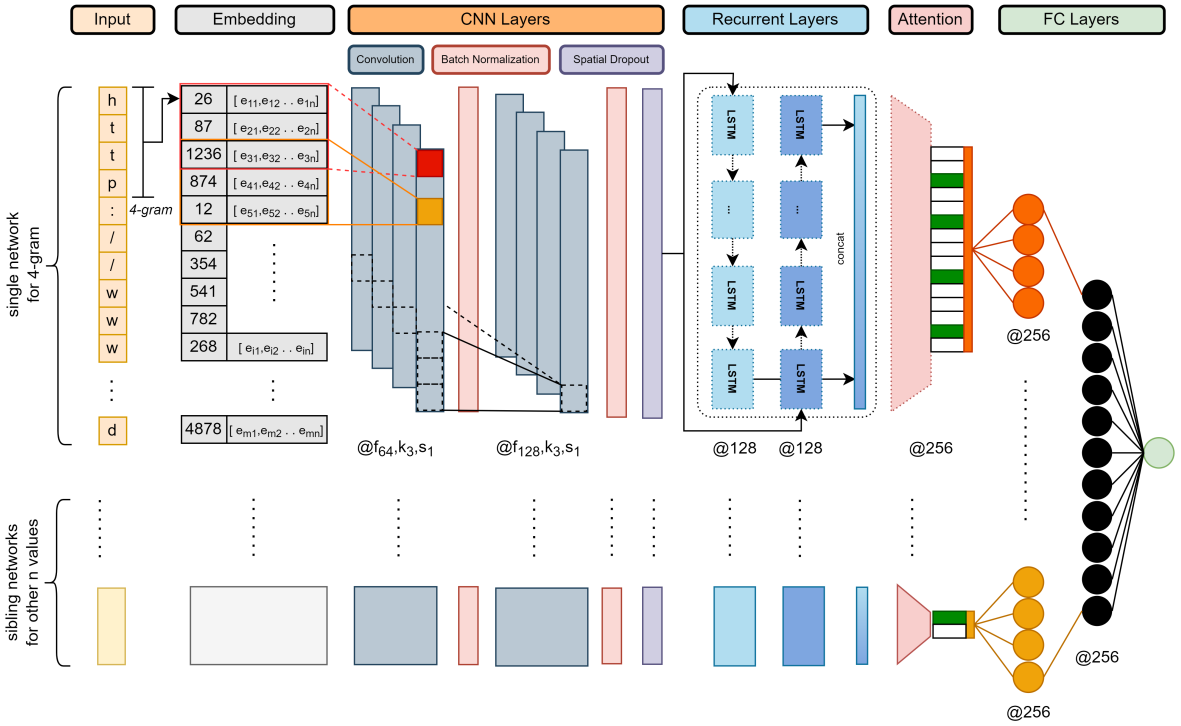


Figure 5.3 Overview of Grambeddings neural network architecture

The overall network architecture of the proposed Grambeddings model is illustrated in Fig. 5.3. We explained the compartments of this architecture in the following subsection in detail. It is noteworthy to highlight that the word *term* refers to either a character in our alphabet or an n-gram in the n-gram corpus

5.2.1. Input Encoding

The Input Encoding part is the one that is responsible for constructing the expected embedding formed output vector from any tokenization module output through the embedding matrix.

The Input Encoding part is the one that is responsible for constructing the expected embedding formed output vector from any tokenization module output through the embedding matrix. This construction process is depicted in Fig. 5.4 and done by proceeding with the following steps. If we consider XU and YU represent an input URL and its corresponding label respectively, the dataset can be formulated as $D = \{(x_u, y_u)|u = 1, 2, ...n\}$ where $y_u \in \{0, 1\}$ denotes the label of the u-th sample data. At first, any given URL is split into terms and tokenized to acquire numerical vector representation. Then, the preprocessing is completed by aligning this numerical vector to keep the vector size constant by either trimming or padding it. The resultant pre-processed integer-valued vector *term-index vector* (i.e. $\mathbf{v}_{encoded}$) is fed and forwarded to the embedding layer.

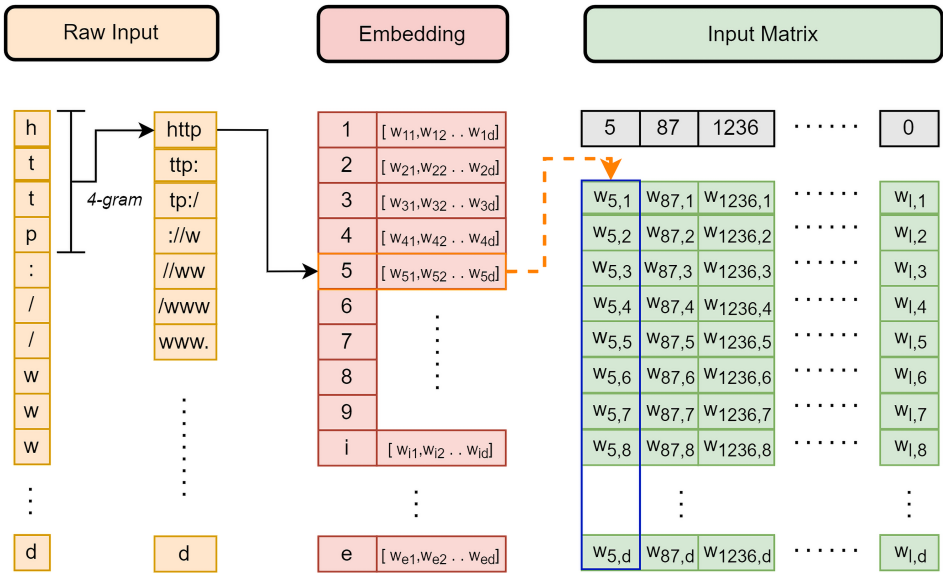


Figure 5.4 Input matrix generation via embedding layer

Before explaining the working principle of the embedding layer, it is noteworthy to mention that this layer is initialized by the following parameters; *embedding dimension* and the

vocabulary size. Moreover, the embedding layer involves two important components: the *Embedding Matrix* and a *lookup table*. The Embedding Matrix is constructed by the two parameters mentioned above and stores the randomly initialized weights for each term within the specified vocabulary. On the other hand, the lookup table holds the row-wise index of each term within the Embedding Matrix. From this point, whenever an encoded vector representation of an URL is fed into the embedding layer, we select related weight vectors (entire row whose length equals to *embedding dimension*) from the Embedding Matrix for each term within the input matrix via the lookup table. Then, we concatenate each selected weight matrix to construct the expected input form for the convolutional layers. The overall URL-input matrix transformation is depicted in Fig. 5.4

5.2.2. Convolutional Layers

The purpose of the convolutional layers for this study is to extract hierarchical features from the given *terms*. Convolutional operation is done by convolutional kernels, which are a set of filters with a specified size. By sliding through the output of the embedding layer **E**, those filters extract features. Even if the CNN architecture we employed is one-dimensional, the sliding operation is actually performed on two dimensions because the selected window size is $k \times d$ to ensure that the filter captures whole weights within the corresponding context where k is the kernel size and d is the embedding dimension. The convolution process can be formulated by Eq. (14).

$$C = f(EW + b) \tag{14}$$

where C is the feature map acquired via convolutional operations; W is the weight matrix; b is the bias term and f is activation function which the Rectified Linear Unit (ReLU) activation function in our case.

After obtaining hierarchical features from each CNN layer, we adopted the Batch Normalization (BN) layer with the purpose of speeding up the training process by having

faster convergence. According to the working principle of BN layer, when the CNN's output C is transmitted , it calculates the mean μ and variance σ^2 by using the Eq. 15 and Eq.16. Then, the resultant normalized C matrix is determined by applying the Eq. 17.

$$\mu = (\frac{1}{n}) \sum_i C^{(i)} \tag{15}$$

$$\sigma^2 = (\frac{1}{n}) \sum_i (C^{(i)} - \mu) \tag{16}$$

$$C^i_{normalized} = (\frac{C^{(i)} - \mu}{\sqrt{\sigma^2 - \mu}}) \tag{17}$$

Note that, since our convolutional network consists of two cascaded CNN-BN mechanisms, another CNN-BN pair where its filter size (total number of filter is used) is doubled and set to 128 is followed by the first CNN-BN pair. The main purpose of using two cascaded CNN-BN pairs is to have higher-level features from input.

The last compartment of our convolutional layers is the Spatial Dropout layer proposed by Tompson et al.[53] whose main purpose is blacking out random cells within the output of the second CNN-BN pair to prevent over-fitting. However, this random cell selection drops the entire 1D feature maps from the matrix instead of choosing individual elements as the standard Dropout technique does. In this way, any possible strong correlations between adjacent n-gram features are minimized, resulting in better regularization and more robustness.

5.2.3. Recurrent Layer

As we mentioned earlier in Section 1, the main purpose of RNNs is processing data that involves sequential and temporal relationships. This operation is proceeded by applying some non-linear functions recursively at each timestamp, and the network tries to predict the next state from its inputs. If we assume the x_t is the current input and h_{t-1} denotes the

previous hidden state, then predicted next hidden state h_t can be computed by applying the formula given in Eq. 18.

$$h_t = \tanh(Wx_t + Uh_{t1} + b) \tag{18}$$

On the other hand, as a quick reminder, we preferred to employ Bidirectional LSTM architecture instead of the standard RNN for the following reasons. The standard RNN faces a well-known vanishing gradient problem because its gradients decay drastically during the training. However, LSTMs are a special type of RNN that resolve this issue by introducing repetitive timing modules. Moreover, the advantage of adopting a bi-directional mechanism is to acquire both forward and backward temporal relationships from a given sequence.

Our motivation for employing a Bi-LSTM architecture is to recognize temporal and sequential relationships between deep hierarchical features obtained through the convolutional layers because we believed that the spatial positions of each term in a URL provides an exclusive information to distinguish malicious samples from benign ones. For instance, a trigram "*www*" can be considered as valid if it comes after the term "*://*", yet it becomes suspicious when it is placed after the term "*com*".

5.2.4. Attention Layer

The last compartment of the Single Network architecture is the Attention Layer, responsible for aligning the recurrent layer to focus on learning the most important term of features within a sequence of deep features. Even though there are several attention methods in the literature, we preferred to use the Additive Attention mechanism initially proposed to improve the performance of an RNN structure by Bahdanau et al. [61].

$$c_i = \sum_{j=1}^{T_x} \alpha ij h_j \tag{19}$$

$$w_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x}} \exp(e_{ik}) \tag{20}$$

$$e_{ij} = \alpha(s_{i-1}, h_j) \tag{21}$$

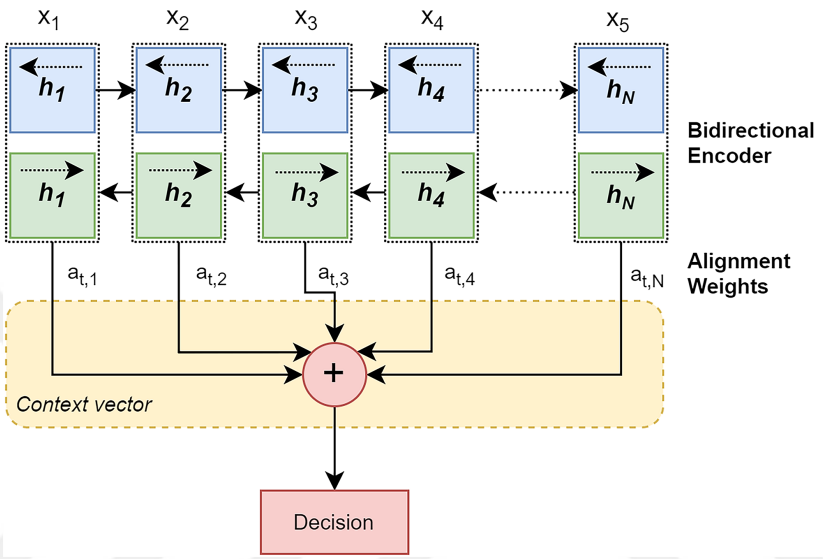


Figure 5.5 Additive attention neural network module

The additive attention realigns the recurrent layer weights by adopting the following procedure. When an attention layer is added on top of an RNN, the feed-forward alignment model α is constructed and paired with each recurrent layer's encoded state. The alignment model measures the correlation scores between $i-th$ input and $j-th$ output. The context vector is computed by applying the formula in Eq. 19 and equals the weighted sum of RNN's hidden states. Next, the new weights of each state are calculated by following the formulas given in Eq. 20-21 and the overall procedure is illustrated in Fig. 5.5.

5.2.5. Fully Connected Layers

The final component in the Grambeddings network architecture is the fully connected layer (FCL), which comprises two dense layers and one standard dropout layer placed between these dense layers. When the model completes the deep feature extraction from each parallel

Single Network structure, these features (one from character level and three from n-gram levels) are concatenated to produce input for the FCL. Since the resultant number of features obtained through a Single Network structure equals 256, the total number of fully-fledged features is 1024, and its computation is given in Eq. 22. Then, the FCL converts these fully-fledged feature sets into a binary class representation to make a prediction as formulated in Eq. 23, where the output could be either phish or legitimate in our case.

$$\alpha_{\mathbf{total}} = (\alpha_{\mathbf{char}} \oplus \alpha_{\mathbf{n-gram_1}} \oplus \alpha_{\mathbf{n-gram_2}} \oplus \alpha_{\mathbf{n-gram_3}}) \tag{22}$$

$$Y_{pred} = f(W \cdot \alpha_{total} + b) \tag{23}$$

where W is the weight matrix and b is the bias terms while f is the activation function (ReLU and Sigmoid used in each Dense Layer respectively)

Table 5.1 The Hardware Configuration is used to train the Grambeddings Model

CPU	Memory	GPU
Intel CoreI7-11800H 2.30 GHz	48 GB DDR4	NVIDIA RTX 3060 GPU (6GB Memory)

5.3. Model Training

In this thesis, we preferred to employ the Adam Optimization Algorithm as the model's optimizer and L2 regularization, also known as the Ridge Regression technique, across whole experiments and the final proposed method parameters. At each iteration over mini-batches, our model tried to achieve the lowest possible binary cross-entropy (BCE) loss as described in Eq. 24. In our approach, we represented phishes and legitimates with 1 and 0, respectively, to construct binary classes.

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \tag{24}$$

Note that, our model training is done by using the hardware configuration given in Table 5.1.

6. EXPERIMENTAL RESULTS

In this section, we mention the series of experiments we used to evaluate our model's validity as well as analyze its performance under different parameter configurations to obtain the best model design. We split these experiments into four main groups to examine Grambeddings' efficiency from different aspects. These groups are Exploring the Hyper-Parameters, Exploring the Best Combination of Different Level Features, Improving the Robustness against Adversarial Attacks, and Comparative Study.

Note that all experiments are performed on the same training and validation data where these data are randomly split from the entire dataset according to their ratios which were %80 and %20, respectively.

6.1. Exploring the Hyper-Parameters

In this subsection, we analyzed each network parameter's impact that affects the particular characteristics of related GramBeddings' Single Network compartment.

It is noteworthy to underline that all experiments are conducted in a case-sensitive manner, and this continued from n-gram extraction to deep feature extraction. Moreover, the overall model is composed of one character level and three n-gram level Single Networks where $n \in \{4, 5, 6\}$ is selected.

6.1.1. Analysis of the Embedding Layer Parameters

The first part of the Grambeddings' Single Network architecture is the Embedding Layer which is the composition of both a lookup table and Embedding Matrix $\mathbf{M}$. The main purpose of this Embedding Layer is to produce an appropriate input tensor $\mathbf{E}$ for the subsequent CNN layer. The Input Tensor is obtained by the following scheme. Whenever an encoded vector representation of an URL is fed into the embedding layer, we select related weight vectors (entire row whose length equals to *embedding dimension*) from the

Embedding Matrix for each term within the input matrix via the lookup table. Then, these selected weight matrices are concatenated to produce the Input Tensor $\mathbf{E}$. From that point, the output size of $\mathbf{E}$ equals $L \times d$ where d is the embedding matrix dimension, and L is the sequence length. Thus, in this section, we examined the impact of those two parameters.

Table 6.1 Impact of n-gram corpus size on model performance

Max. n-gram Corpus Size	Sequence Length	Embedding Dimension	Attention Size	LSTM Cell Size	Val. Accuracy	Val. Precision	Val. Recall	Val. F1	Val. AUC	Training Duration (s)	Total Parameters
20000	128	15	10	128	0.9788	0.9799	0.9777	0.9872	0.9970	2405.4	2624585
50000	128	15	10	128	0.9802	0.9850	0.9754	0.9853	0.9954	2417.2	3974585
90000	128	15	10	128	0.9806	0.9814	0.9799	0.9884	0.9971	2421.6	5774585
160000	128	15	10	128	0.9827	0.9894	0.9759	0.9826	0.9973	2464.9	8924585
300000	128	15	10	128	0.9829	0.9876	0.9780	0.9874	0.9970	2649.0	15224585
400000	128	15	10	128	0.9831	0.9860	0.9801	0.9885	0.9972	2686.8	19724585
500000	128	15	10	128	0.9834	0.9851	0.9816	0.9891	0.9968	3003.2	24224585

According to the experiment results, we examined the impact of corpus size. We measured that as we increased the n-gram corpus size, the model achieved better performance because the number of unknown terms in vector representation decreased. On the other hand, we noticed the trade-off between the corpus size and the required computation time depends on the total number of the learnable model parameters as given in Table 6.1. In other words, when we increase the n-gram corpus size, the network learnable parameter count and computation cost also increase. Thus, we set the corpus size parameter to 160.000 to cover enough n-gram space that we could make generalizations while limiting the computational resource consumption.

Table 6.2 Impact of embedding dimension on model performance

Max. n-gram Corpus Size	Sequence Length	Embedding Dimension	Attention Size	LSTM Cell Size	Training Accuracy	Validation Accuracy
160000	**128**	**15**	**10**	**128**	**0.9884**	**0.9827**
160000	128	20	10	128	0.9938	0.9816
160000	128	25	10	128	0.9944	0.9818

The second parameter we examined is the embedding dimension and we conduct this experiment by varying this parameter with the values of 15,20 and 25. This parameter actually determines the depth of each term-based weight vector in the Embedding Matrix. According to the experimental results depicted in Table 6.2, we acquired the following observations. The first one is that if we set the embedding dimension to insufficiently

small, the model could not be able to construct an Input Tensor $\mathbf{E}$ and therefore, the rest of the network architecture (CNN+BiLSTM+Attention) suffered from to generalize input URLs to make prediction since they tried to extract deep features through squeezed vector representation. The second conclusion we made was that when we increase the embedding dimension more and more, the model becomes prone to over-fit on training data. From these observations, we noticed that our model achieved the best score when we set the embedding dimension parameter to 15. On the other hand, due to the fact that our selected embedding dimension is relatively small, we also reduced the computational requirements such as available memory and estimated time to complete a training iteration. As a final statement, it is necessary to underline that, the parameter embedding dimension selection is highly correlated with the total sample size in the training since if the model does not see enough data on the training phase, it would start to over-fit.

Table 6.3 Impact of sequence Length on model performance

Max. n-gram Corpus Size	**Sequence Length**	**Embedding Dimension**	**Attention Size**	**LSTM Cell Size**	**Training Accuracy**	**Validation Accuracy**
160000	96	20	10	128	0.9886	0.9825
160000	**128**	**20**	**10**	**128**	**0.9939**	**0.9827**
160000	196	20	10	128	0.9732	0.9813
160000	256	20	10	128	0.9937	0.9810

In the analysis of the last Embedding Layer's parameter, we examined the impact of input sequence length which determines the size of the input encoded vector. As a reminder, as illustrated in Table 4.2, in the Grambeddings dataset the average length of phishing samples is higher than the legitimate ones while it equals 86.21. Hence, in this experiment, we set the minimum possible value for the input sequence length to 96. The main reason for this parameter selection was to cover most of the input samples without needing to trim a sample URL in order to prevent any possible feature loss. Hence, in this experiment, the parameter input sequence length value L is varied between the values of $\{96, 128, 196, 256\}$ and achieved the results stated in the Table 6.3. According to these results, as we mentioned before, we observed that our model suffered from the information loss caused by trimming

the important part of an URL when we set the L to a relatively small value. On the other hand, we also noticed that if we expand the vector representation of an URL by increasing the parameter L too much, our model started to perform worse. The reason behind this behavior is that, when we try to describe an URL with an unnecessarily long encoded vector, this encoded vector contains too many padding characters at the end of it. Thus, vector representations of both legitimate and phishing sample URLs start to become similar, and some of the features extracted from these vectors become indiscriminative, unfortunately. We eventually achieved the best score when we set L to 128, which is a really close value to the average phish sample lengths.

6.1.2. Analysis of the Recurrent Layer Parameters

The objective of using the Bi-LSTM layer in this thesis is to acquire temporal and sequential relationships between the deep hierarchical features obtained through convolutional layers. In this section, we analyzed the impact of one of the essential parameters of any recurrent layer named cell size. The *LSTM Cell Size* determines both the dimension of the inner cells within a single LSTM structure and the resultant vector dimension obtained from that LSTM structure. Note that, since we employed a bidirectional LSTM mechanism in this study, the output vector size would be equal to twice the LSTM Cell Size.

Table 6.4 Impact of LSTM cell size on model performance

Max. n-gram Corpus Size	**Sequence Length**	**Embedding Dimension**	**Attention Size**	**LSTM Cell Size**	**Training Accuracy**	**Validation Accuracy**
160000	128	15	10	8	0.9921	0.9814
160000	128	15	10	16	0.9875	0.9815
160000	128	15	10	32	0.9883	0.9821
160000	128	15	10	64	0.9886	0.9816
160000	**128**	**15**	**10**	**128**	**0.9883**	**0.9827**
160000	128	15	10	256	0.9715	0.9812

In this experiment, we varied the parameter LSTM Cell Size between the values of 8 and 256 in multiples of two. According to the results given in Table 6.4, we noticed a gradual increment in the model performance. At the same time, we increased this parameter to 128

because the larger output size allowed our model to capture more complex sequential patterns from a given context. However, we also observed that our model suffered from over-fitting when it was increased to 256. Thus, we set the LSTM Cell Size parameter to 128 as the default and best value.

6.1.3. Analysis of the Attention Layer Parameters

In the last hyper-parameter search of the proposed model architecture, we analyzed the impact of *Attention Width* parameter that affects the last part of parallel sub-network architecture (Single Network) named Attention Layer. As a quick reminder, we employed the Additive Attention mechanism to align the recurrent layer's hidden states to pay selective attention to more relevant parts of sequential deep features and improve the model performance. The used Additive Attention layer accepts only one parameter to initialize itself, and that is the *Attention Width*. This parameter sets the size of Dense Layers that take part in the attention mechanism. In other words, this argument defines the number of the recurrent units employed to compute the context vector formulated in Eq. 19.

Table 6.5 Impact of attention width on model performance

Max. n-gram Corpus Size	**Sequence Length**	**Embedding Dimension**	**Attention Size**	**LSTM Cell Size**	**Training Accuracy**	**Validation Accuracy**
160000	128	20	5	128	0.9890	0.9818
160000	**128**	**20**	**10**	**128**	**0.9888**	**0.9827**
160000	128	20	20	128	0.9888	0.9823

In this section, we performed a series of experiments to analyze the impact of Attention Width on model performance while changing the argument within the values 5,10, and 20. According to the results presented in Table 6.5, we noticed that if we set *Attention Width* to 5, the attention mechanism could not be able to align the recurrent layer since fewer recurrent units are utilized to compute the corresponding context vector. Thus, the model's performance gains were less than expected. On the other hand, we noticed another lack of information gain when we increased *Attention Width* too large (20 in this case) because the attention mechanism tried to align Bi-LSTM with a wider window and therefore missed

some fundamental sequential relationships within that window. We obtained the best score when we set this parameter to 15.

6.2. Design Choice for Network Architecture

In the previous section, we determined the optimal values for hyper-parameters of the proposed Grambeddings' model architecture that yields the best classification performance in recognizing malicious websites from a given URL. The Grambeddings architecture is a quadruplet deep neural network that is the composition of one character level feature processor and three n-gram level ones where each parallel sub-network is responsible for extracting deep features in the associated level of representation. Thus, since each parallel sub-network named Single Network specialized in learning different n-gram levels of structures and patterns through its architecture, their contribution to overall ensemble architecture differs. From this objective, we conduct a series of experiments to measure stand-alone performances of each Single Network architecture as well as the best possible composition of these parallel networks that result in the highest classification result.

6.2.1. Analysis of the Single N-Gram Network

As mentioned before, our first experiment is about discovering the stand-alone performances of each Single Network architecture that captures the textual content with different n-gram windows. If we assume that a character level representation also works in an n-gram perspective when the value of maximum sequence length (n) is set to value one, in this section, we obtained the characteristics and performances of different Single Network architecture by varying the parameter n within the values of $\{1, 3, 4, 5, 6, 7\}$.

According to the results given in Table 6.6, each value of n performed slightly differently but similarly. However, when we increased the maximum sequence length and set it to 7, we also observed a performance loss since, at this point, selected top-k n-grams became insufficient to cover the corresponding possible n-gram space. Hence, model performance is reduced

Table 6.6 Impact of different character sequence lengths (i.e. gram size) in terms of various metrics

n	Max. n-gram Corpus Size	Sequence Length	Embedding Dim.	Attention Size	LSTM Cell Size	Validation Accuracy	Validation Precision	Validation Recall	Validation F1	Validation AUC
1	160000	128	15	10	128	0.9761	0.9840	0.9681	0.9759	0.9963
3	160000	128	15	10	128	0.9770	0.9812	0.9727	0.9769	0.9961
4	**160000**	**128**	**15**	**10**	**128**	**0.9772**	**0.9846**	**0.9695**	**0.9770**	**0.9951**
5	160000	128	15	10	128	0,9768	0,9786	0.9749	0.9768	0.9963
6	160000	128	15	10	128	0.9721	0.9837	0.9600	0.9717	0.9946
7	160000	128	15	10	128	0.9662	0.9801	0.9517	0.9657	0.9926

regarding this lack of informative encoding within the vector representations. Consequently, we stopped increasing the n at this point and obtained the best Single Network performance when this parameter is set to 4.

6.2.2. Analysis of the Combination of Multiple N-Gram Network

In the second model design choice related experiment, we tried to find the best possible quadruplet network architecture by combining different levels of Single Networks. According to the combinations and results are given in Table 6.7, we concluded two outcomes. The first observation was that each different quadruplet network outperformed both the sub-network (*Single Network*) connected to it and the best sub-network architecture obtained. Lastly, we observed the best architectural composition by using $n_1 = 4, n_2 = 5$ and $n_3 = 6$ where $n_1, n_2, n_3 \in n$.

Table 6.7 Impact of n-gram combinations on training and validation sets

n_1	n_2	n_3	Max. n-gram Corpus Size	Sequence Length	Embedding Dimension	Attention Size	LSTM Cell Size	Training Accuracy	Validation Accuracy
3	4	5	160000	128	15	10	128	0.9885	0.9815
3	5	6	160000	128	15	10	128	0.9934	0.9817
3	5	7	160000	128	15	10	128	0.9883	0.9817
3	6	7	160000	128	15	10	128	0.9876	0.9812
4	**5**	**6**	160000	128	15	10	128	**0.9935**	**0.9827**
4	5	7	160000	128	15	10	128	0.9886	0.9822
5	6	7	160000	128	15	10	128	0.9915	0.9811

6.3. Improving The Robustness Against Adversarial Attacks

Thanks to the rapid development of machine learning-based protection systems, users are protected against many conventional cybercrimes techniques. However, as these

systems improved their protection capabilities, cybercriminals also adapted their attack techniques to bypass their defenses by exposing machine learning methods to inaccurate, misrepresentative, or malicious data. These types of attack techniques are often named Adversarial Attacks, and many studies [64, 85] in literature show any machine learning phishing detection method is potentially vulnerable to those kinds of attacks. On the other hand, as we mentioned before, the use of deep learning is escalated in this domain. However, similar to the traditional ML-based approaches, DL-based methods are also weak against these attack types, especially that are gradient-based. From this point, it is an inevitable requirement to measure the robustness of any DL-based method against adversarial attacks to demonstrate its capability to provide wide-range phishing protection. Thus, we conducted a series of experiments to observe the proposed Grambeddings model performance when it is faced with these kinds of attacks.

From this objective, we adopted the *Compound Attack* technique as our Threat Model to simulate almost read malicious attacks derived from their legitimate counterparts by inserting an adversarial character between sub-word tokens extracted from the corresponding domain of the given URL. We choose the hyphen character as our evasion character, as suggested by Maneriker et al. [19]. Unlike the authors of the mentioned study, we employed one of the most recent and advanced multi-lingual tokenizers that are also known as *Transformer* to split the domain part of an URL into sub-words instead of utilizing English based tokenizer. Next, we adopted a similar approach to produce fabricated phishing samples and applied some skipping criteria to avoid URL duplication that could possibly mislead any model to discriminate given two identical inputs with different labels. The entire procedure employed to generate adversarial attacks is given below:

- Locate the domain part from a given URL by using the well-known Python library named TLDParser [84]
- Split the domain into tokens(sub-words) by using the XLM-RoBERTa transformer model proposed by Conneau et al. [86].

- Checked the extracted token count. If it is less than two, then skip to the next URL sample by ignoring the current sample
- Randomly select where to insert the evasion character '-' among these extracted parts, $index \in \{1, n-1\}$ and n is the extracted part count.
- Re-combine whole parts together to produce the malicious version of related legitimate domain while preserving the original order and inserting hyphen at previously selected index.
- Replace the original legitimate domain name with the freshly generated malicious domain name.
- If the final URL representation somehow contains double - character, skip this sample. Otherwise append it to the adversarial list.

In order to provide a fair test environment, we produced one test and two different training datasets while the following procedure is applied when we generate those datasets. At first, we picked a random number between 0 and 1 at each step while iterating through input samples. Then, we checked the randomly generated value, and if it was less than 0.5, we skipped the current sample and continued to iterate. Otherwise, we followed the above procedure to produce new adversarial samples from their legitimate counterparts.

Table 6.8 The summary of newly generated datasets which are used in adversarial attack experiments

Dataset Name	Size	Purpose	Description
AdvTest	~160K	Validation	80K legit. + 40K phish samples of the original validation data + 40K adversarial phish samples
AdvTrain1	~800K	Training	Original training data and ~160K adversarial phish samples
AdvTrain2	~960K	Training	Original training data and ~320K adversarial phish samples

From the adversarial generation objective mentioned above, once we generated all required adversarial samples whose size approximately equals 40k, we started to build the *AdvTest* dataset by merging them with legitimate (80k) and randomly selected malicious URLs (40k) where both of which are gathered from the original test data. Consequently, we produced a significantly challenging test dataset that reflects three different variation characteristics on

binary classes while preserving the balanced class distribution via having the same amount of benign and malicious samples. We applied the same logic to create all three adversarial datasets, and their details are given in Table 6.8

We started to conduct two main experiments to observe model performance under different conditions when we completed the dataset generation; (i) base model performance in phishing detection on the AdvTest dataset, and (ii) re-trained model performance in phishing detection on the AdvTest dataset.

Pre-trained Baseline Model Performance on Advserially Augmented Test Data: In the first experiment, we evaluated our best baseline model against an adversarially augmented test dataset named the AdvTest to measure the model's pure performance (without seeing any adversarial attacks before) in classifying adversarial samples correctly. According to the results in the second row of Table 6.9, the model performance inevitably and expectedly decreased to 68.13%. We believe that since our training phase is conducted in an adversarial-blind manner, the model could not be able to interpret hyphen character and therefore confused when separating two almost identical legitimate and adversarial counterparts.

Table 6.9 The Impact of Adversarial Attacks on GramBeddings Model

Model	Training Data	Validation Data	Val. Accuracy	Val. Precision	Val. Recall	Val. F1	Val. AUC
Our best baseline	Original Training Data	Original Val. Data	**0.9827**	0.9894	**0.9759**	**0.9826**	**0.9973**
Our best baseline	Original Training Data	AdvTest	0.6813	0.3986	0.0274	0.0521	0.5163
Re-trained	AdvTrain1	AdvTest	0.9405	0.9242	0.9569	0.9714	0.9863
Re-trained	AdvTrain2	AdvTest	0.9470	0.9226	0.9733	0.9808	0.9883

In order to analyze the reason for this baseline model performance drop in detail, we decided to project the binary class distribution with three different label information, including adversarial ones in two-dimensional space. For this purpose, we retrieved the logits from our pre-trained model by hooking the output layer to the first dense layer in the architecture. Since the output of the first dense layer equals 128, we employed a well-known Uniform Manifold Approximation and Projection (UMAP) proposed by **(author?)** [87] to reduce the dimensionality. As illustrated in 6.1(a), the distribution of benign and adversarial samples are very intertwined and even over-lapped since their resultant feature vectors obtained from the

pre-trained model are almost identical. This similarity is because their syntactical formations have only one character difference, and most sequential orders are identical for the model that has not learned to pay attention to intrusion character.

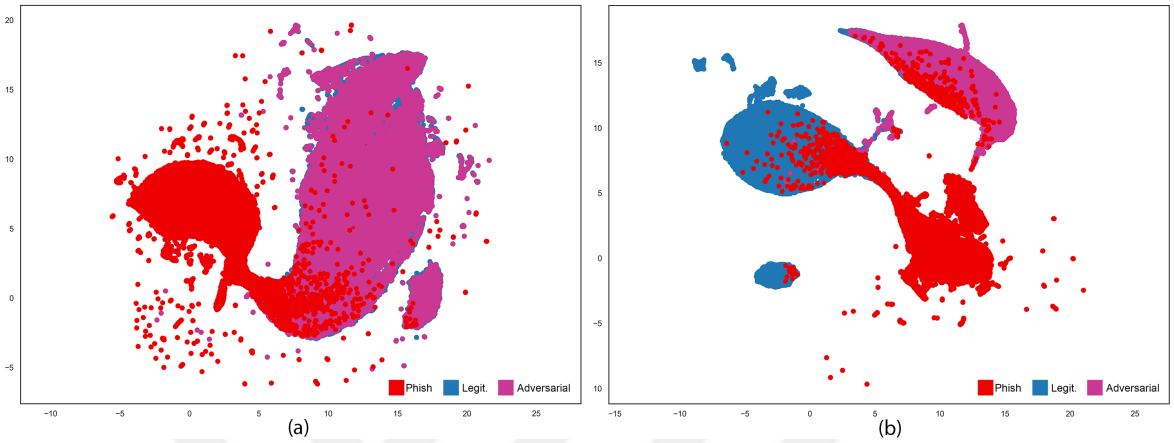


Figure 6.1 The 2-D distributions of the obtained logits from the adversarially enriched validation set AdvTest (a) after trained with our original training dataset which involves no adversarial phishing examples and (b) after re-trained with an adversarially enriched training set AdvTrain1. As can be seen from the leftmost figure, the lack of adversarial training brings the legitimate and adversarial samples closer and overlap. On the other hand, impact of adversarial training depicted in the rightmost figure, helps the GramBeddings network distinguish the newly added adversarial examples from the legitimate ones by also bringing them closer to phishing samples resulting in a much better accuracy score.

We also perform the same experiment on a small subset of *AdvTest* to highlight and support our above statement. The subset comprises 2500 benign, 1250 malicious, and 1250 adversarial URLs, where the entire subset is randomly selected to reduce the possibility of containing the adversarial sample and its legitimate counterparts. According to the focused distribution depicted in Fig. 6.2, it is much clear to observe the interpenetration between legitimate samples and their adversarial counterparts.

Re-trained Baseline Model Performance on Advserially Augmented Test Data: We noticed the dramatic performance loss in the previous experiment when an adversarial attack was fed through our pre-trained model. In this part, we conducted two more experiments to analyze whether adversarial training helps our model regain its performance and mitigates this problem.

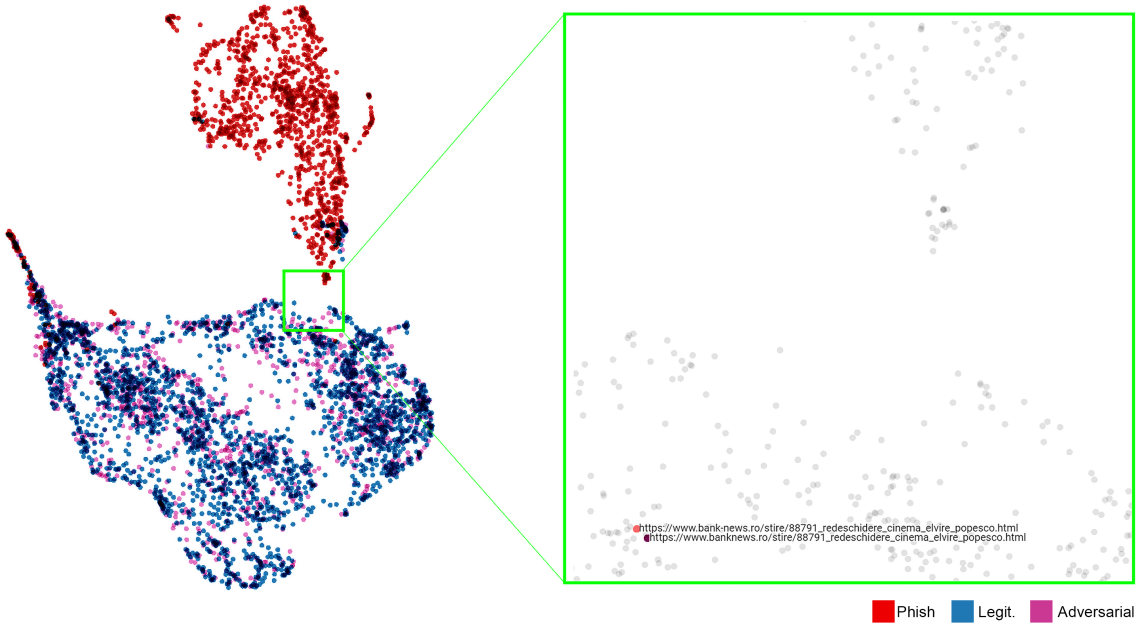


Figure 6.2 The 2-D class distribution of 5000 randomly sampled (i.e. 2500 legitimate + 1250 phishing + 1250 adversarial) validation data whose logits were produced by the model trained without any adversarial samples. On the right side, the similarity of logits belonging to one legitimate and its adversarial counterpart is shown. This illustration clearly shows how a basic hyphen based adversarial attack is capable of deceiving the neural inference.

We noticed the dramatic performance loss in the previous experiment when an adversarial attack was fed through our pre-trained model. In this part, we conducted two more experiments to analyze whether adversarial training helps our model regain its performance and mitigates this problem. Therefore, we generated two training datasets, *AdvTrain1* and *AdvTrain2*, as we did while constructing the AdvTest data. Note that the characteristics of these datasets are given in Table 6.8. Next, we re-trained our model on the adversarial training (*AdvTrain1* and *AdvTrain2*) and validation (*AdvTest*) dataset while preserving our best-performing model's hyper-parameters. According to the results of these two experiments given in Table 6.9, we showed that our model re-gained its performance when it was re-trained on a dataset enriched with adversarial samples by achieving accuracy scores of 94.05% and 94.70% respectively. In other words, our proposed model is capable of learning how to separate adversarial samples from their legitimate counterparts even if their syntactical and character-level difference is small. The distributional difference between

pre-trained and re-trained models is illustrated in Fig. 6.1 wherein the part (b) it could be clearly inferred that all three sample types constructed three different clusters on the two-dimensional vector space. Moreover, especially Fig. 6.3 highlights that re-training with adversarial examples increases model robustness against the compound attack.

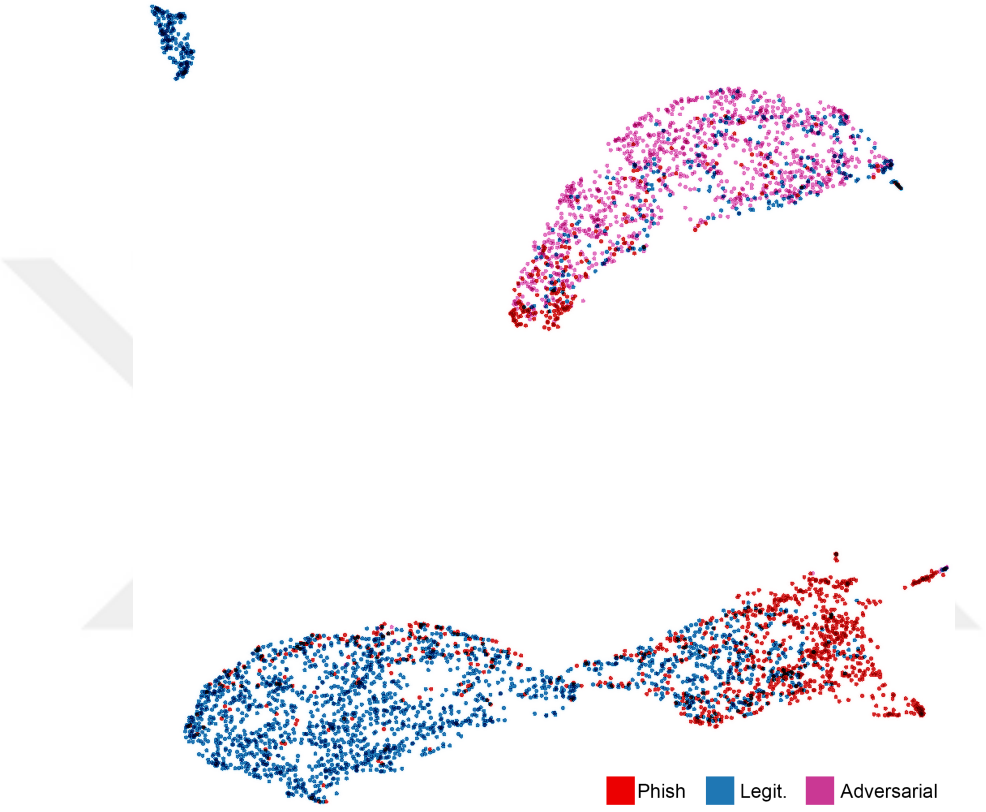


Figure 6.3 The 2-D categorical logit distribution of 5000 randomly sampled (i.e. 2500 legitimate + 1250 phishing + 1250 adversarial) validation data whose logits are produced by the model re-trained by including adversarial samples. This illustration clearly shows how adversarial training helps the model overcome our compound attack. As can be seen from the rendering, the inclusion of adversarial samples in training enables the model to make a clear separation between legitimate and adversarial URLs along with moving adversarial samples closer to phishing data points.

6.4. Comparative Study

In this section, we provide better insight into our model performance-wise advantages against other previous studies in the literature by conducting two different experiments. The first one is dataset-wise benchmarking, where we made the comparison regarding existing

proposed datasets since their implementations were not publicly available according to our best knowledge. The latter is method-wise benchmarking, where we executed different methods on our own Grambeddings dataset.

6.4.1. Dataset-wise Benchmarking

In this section, we have re-trained our model on the datasets that are previously listed in Table 4.2 and made the dataset-wise comparison by taking into account their published scores. As demonstrated in Table 6.10, our proposed model outperformed the other studies on their own datasets. Also, the Grambeddings proved its robustness even if the data is imbalanced and scarce and suffers from imbalanced URL length distribution such as Ebbu17, PDRCNN, PhishStorm, KD01, and KD03. Moreover, even though the required network trainable parameter count is relatively high depending on its deep architecture, our model was still capable of preserving its performance on a minimal dataset like KD02, which contain only 1400 URLs.

Table 6.10 Benchmarking results of our approach in a dataset-wise setting involving validation sets of a variety of studies

Dataset - Work	Validation Accuracy	Validation Precision	Validation Recall	Validation F-1 Score	Validation AUC
EBBU17 - [77]	0.9702	0.9640	0.9770	0.9710	N/A
GramBeddings on EBBU17	**0.9914**	0.9934	0.9893	0.9938	0.9982
PRRCNN - [33]	0.9579	0.9727	0.9424	0.9573	0.9903
GramBeddings on PDRCNN	**1.0000**	1.0000	1.0000	1.0000	1.0000
PHISHSTORM - [78]	0.9491	0.9844	0.9127	0.9472	N/A
GramBeddings on PHISHSTORM	**0.9832**	0.9935	0.9729	0.9850	0.9975
KD01 - [80]	0.9600	0.9600	0.9600	0.9600	N/A
GramBeddings on KD01	**0.9873**	0.9849	0.9779	0.9878	0.9979
KD02 - [81]	0.9590	0.9500	0.9500	0.9500	N/A
GramBeddings on KD02	**1.0000**	1.0000	1.0000	1.0000	1.0000
KD03 - [82]	0.9464	0.9500	0.9500	0.9500	N/A
GramBddings on KD03	**0.9982**	0.9968	0.9951	0.9972	0.9993
ISCXURL2016 - [79]	N/A	0.9260	0.9280	N/A	N/A
ISCXURL2016 - [28]	0.9957	0.9970	0.9946	0.9958	0.9967
GramBeddings on ISCXURL2016	**0.9982**	0.9995	0.9970	0.9985	1.0000

6.4.2. Method-wise Benchmarking

In the second part of the comparative experiments, we trained both URLNet [32] and Character Level CNN[88] models from scratch on the Grambeddings dataset while preserving their proposed hyper-parameters and original implementations. As given in Table

6.11, our proposed model outperformed other methods. Moreover, even our character-level Single Network architecture achieved a more excellent score than the Character Level CNN model concerning the performances given in Table 6.6. Besides, our character level structure also promises better results than URLNet's Embedding Mode version 1, which only works at the character level.

Table 6.11 Performance comparison of our approach against other methods by using the GramBeddings dataset

Method	**Description**	**Val. Accuracy**	**Val. Precision**	**Val. Recall**	**Val. F1**	**Val. AUC**
UrlNet	Embedding mode 1	0.9750	0.9830	0.9667	0.9748	0.9753
UrlNet	Embedding mode 2	0.9696	0.9839	0.9549	0.9692	0.9696
UrlNet	Embedding mode 3	0.9801	0.9936	0.9663	0.9798	0.9802
UrlNet	Embedding mode 4	0.9770	0.9929	0.9608	0.9766	0,9770
UrlNet	Embedding mode 5	0.9818	0.9930	0.9706	0.9816	0.9818
CharLevelCNN	Small model (Sequence length=420)	0.9702	0.9699	0.9706	0.9703	0.9702
CharLevelCNN	Large model (Sequence length=1014)	0.9750	0.9859	0.9638	0.9748	0.9750
GramBeddings (ours)	Our best network configuration	**0.9827**	0.9894	**0.9759**	**0.9826**	**0.9973**

7. DISCUSSION

In this thesis, our prosed solution is to identify phishing websites through the URL information based on a sophisticated deep neural network architecture that employs multiple different character-level n-gram features. Therefore, we first extracted unique character sequences from 640.000 training samples. We selected top-k features from this distinct n-gram list by employing the chi-squared statistical analysis method. Afterward, we initialized the n-gram corpus and the Embedding Matrix concerning the selected n-grams vocabulary. In the experiment where we analyzed the impact of n-gram corpus size (See Table 6.1), we show that our proposed model performance remains satisfactory while the n-gram corpus size is limited with 20k instances thanks to Grambeddings' generalization capacity. From this objective, our proposed approach also could be considered to work on big data by fine-tuning the pre-trained model while taking advantage of preserving previously obtained n-gram embeddings in a similar way as the commonly used pre-trained word embeddings methods such as Word2Vec [50] and GloVe [51]. Nevertheless, it should be underlined that our n-gram embeddings do not construct or compute any semantical relationship between them, unlike these popular methods. On the other hand, this lack of semantic relation is actually not a shortcoming since the URL n-grams per se could occur at any location in that string and do not need to have a semantical meaning as well as any wordy meaning theoretically. In other words, the URL itself is a great example of the difference between human beings and computers in the interpretation of natural languages and this is the particular reason for this situation. However, from a practical perspective, a system that can present semantic relationships between n-grams could be very useful in analyzing the tricks that cyber attackers use. Thus, we consider there is one tradeoff when we preferred to employ n-gram embeddings instead of word embeddings, that is eliminating the necessity of knowing language-dependent words or sub-words while losing any possible semantical relations between the embeddings.

In addition, we conduct a series of experiments to evaluate Grambeddings' performance against adversarial attacks. During this experiment, we highlighted the proposed method

is capable of distinguishing adversarial instances from their legitimate counterparts when it is re-trained with fabricated samples. Nevertheless, there are several different adversarial attack types exist. For instance, while we could give homoglyphs as one of the syntactical manipulation techniques, *Fast Gradient Sign Attack* is one of the sophisticated gradient exploitation attack types. Therefore, any phishing detection method as well as the Grambeddings should be evaluated by considering various types of adversarial attack techniques. We left this analysis as our future work due to space constraints.

The Grambeddings is an advance deep quadrupled neural network that fuses one character level feature with three n-gram level ones to recognize malicious websites from their URL information as long as the input is appropriate to work with. On the other hand, another potential use case of our proposed method is statically analyzing malware since, similar to the URL formation, the taxonomy of malware's byte-level op-code representation can be encoded and fed into our deep architecture. From this point, we believe that the proposed Grambeddings' architecture and n-gram level sequential deep features can be adopted in different fields by performing small modifications on preprocessing step.

8. CONCLUSION AND FUTURE WORK

In this thesis, our proposed end-to-end deep neural network architecture named Grambeddings provides the following contributions to the literature in detecting phishing websites through URL information. The first novelty proposes an effective and efficient n-gram selection and encoding scheme that enables us to construct an n-gram level Embedding Matrix. The latter one is the composition of carefully designed four different parallel deep architectures responsible for acquiring distinctive and particular deep features within different n-gram levels (i.e., 1,4,5,6), where every single network consists of CNN-BiLSTM-Attention blocks. Thanks to this sophisticated architecture, we were able to obtain structural and syntactical features as well as the sequential patterns from a given URL while paying attention to the most important sequence of deep features through the attention mechanism. Thanks to its well-designed structure, our model is highly configurable regarding the number of parallel sub-networks. Besides, thanks to its well-designed structure, our model is highly configurable in terms of the number of parallel sub-networks, and the architecture could be extended by inserting another n-gram level feature if deep n-gram features with different window sizes desired to be captured.

On the other hand, we also examined the model's characteristics when exposed to an adversarial attack. We explored how to eliminate this potential performance loss risk caused by these attacks. Moreover, we compared pre-trained and re-trained network model performance in detail and demonstrated the possible diminishing class problem if action is not taken against these issues.

Lastly, we conducted completely transparent experiments to explore model hyper-parameters yielding the best results and then compared our proposed method with other well-known and influential studies in the literature by using the obtained best network parameters. The Grambeddings outperforms other works by achieving the new state-of-art results according to both method-wise and dataset-wise benchmarking results.

REFERENCES

[1] Phishtank. Phishtank, **2021**.

[2] Favio Vázquez. A “weird” introduction to deep learning, **2018**.

[3] Nicola Strisciuglio, Manuel Lopez Antequera, and Nicolai Petkov. Enhanced robustness of convolutional networks with a push–pull inhibition layer. *Neural Computing and Applications*, 32:1–15, **2020**. doi:10.1007/s00521-020-04751-8.

[4] Sinam Ajitkumar Singh, Takhellambam Gautam Meitei, and Swanirbhar Majumder. 6 - short pcg classification based on deep learning. In Basant Agarwal, Valentina Emilia Balas, Lakhmi C. Jain, Ramesh Chandra Poonia, and Manisha, editors, *Deep Learning Techniques for Biomedical and Health Informatics*, pages 141–164. Academic Press, **2020**. ISBN 978-0-12-819061-6. doi:https://doi.org/10.1016/B978-0-12-819061-6.00006-9.

[5] Amine ben khalifa and Hichem Frigui. Multiple instance fuzzy inference neural networks. **2016**.

[6] Axel Thevenot. 12 main dropout methods: Mathematical and visual explanation for dnns, cnns, and rnns, **2020**.

[7] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, **2015**.

[8] Savvas Varsamopoulos, Koen Bertels, and Carmen Almudever. Designing neural network based decoders for surface codes, **2018**.

[9] Dipankar Dasgupta, Zahid Akhtar, and Sajib Sen. Machine learning in cybersecurity: a comprehensive survey. *The Journal of Defense Modeling and Simulation*, page 1548512920951275, **2020**.

[10] R. S. Rao and S. T. Ali. A computer vision technique to detect phishing attack. In *2015 Fifth International Conference on Communication Systems and Network Technologies*, 1, pages 596–601. **2015**.

[11] APWG Q4-Reports-2021. Apwg 4th. quarterly reports 2021, **2021**.

[12] FBI. Fbi 2019 internet crime report, **2019**.

[13] Daniel Jampen, Gürkan Gür, Thomas Sutter, and Bernhard Tellenbach. Don't click: towards an effective anti-phishing training. a comparative literature review. *Human-centric Computing and Information Sciences*, 10(1):1–41, **2020**.

[14] Akashdeep Bhardwaj, Varun Sapra, Aman Kumar, Naman Kumar, and S Arthi. Why is phishing still successful? *Computer Fraud & Security*, 2020(9):15–19, **2020**.

[15] Ana Ferreira, Lynne Coventry, and Gabriele Lenzini. Principles of persuasion in social engineering and their use in phishing. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 36–47. Springer, **2015**.

[16] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 379–396. **2020**.

[17] Gaurav Varshney, Manoj Misra, and Pradeep K Atrey. A survey and classification of web phishing detection schemes. *Security and Communication Networks*, 9(10):6266–6284, **2016**.

[18] Mohammed Hazim Alkawaz, Stephanie Joanne Steven, Asif Iqbal Hajamydeen, and Rusyaizila Ramli. A comprehensive survey on identification and analysis of phishing website based on machine learning methods. In *2021 IEEE 11th IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, pages 82–87. **2021**.

[19] Pranav Maneriker, Jack W Stokes, Edir Garcia Lazo, Diana Carutasu, Farid Tajaddodianfar, and Arun Gururajan. Urltran: Improving phishing url detection using transformers. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, pages 197–204. IEEE, **2021**.

[20] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Cranor, Jason Hong, and Chengshan Zhang. An empirical analysis of phishing blacklists. Carnegie Mellon University, **2009**.

[21] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, **2006**.

[22] Ahmet Selman Bozkir and Ebru Akcapinar Sezer. Use of hog descriptors in phishing detection. In *2016 4th International Symposium on Digital Forensic and Security (ISDFS)*, pages 148–153. IEEE, **2016**.

[23] Ge Wang, He Liu, Sebastian Becerra, Kai Wang, Serge J Belongie, Hovav Shacham, and Stefan Savage. *Verilogo: Proactive phishing detection via logo recognition*. Department of Computer Science and Engineering, University of California . . . , **2011**.

[24] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, **2004**.

[25] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. Phishing websites features. *School of Computing and Engineering, University of Huddersfield*, **2015**.

[26] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. Cantina+ a feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, **2011**.

[27] Hansi Jiang, Dongsong Zhang, and Zhijun Yan. A classification model for detection of chinese phishing e-business websites. **2013**.

[28] Brij B. Gupta, Krishna Yadav, Imran Razzak, Konstantinos Psannis, Arcangelo Castiglione, and Xiaojun Chang. A novel approach for phishing urls detection using lexical based machine learning in a real-time environment. *Computer Communications*, 175:47–57, **2021**.

[29] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. Lexical feature based phishing url detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, page 54–60. **2010**.

[30] Michael Darling, Greg Heileman, Gilad Gressel, Aravind Ashok, and Prabaharan Poornachandran. A lexical approach for classifying malicious urls. In *2015 International Conference on High Performance Computing Simulation (HPCS)*, pages 195–202. **2015**.

[31] Huan-huan Wang, Long Yu, Sheng-wei Tian, Yong-fang Peng, and Xin-jun Pei. Bidirectional lstm malicious webpages detection algorithm based on convolutional neural network and independent recurrent neural network. *Applied Intelligence*, 49(8):3016–3026, **2019**.

[32] Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, pages 1–13, **2018**.

[33] Weiping Wang, Feng Zhang, Xi Luo, and Shigeng Zhang. Pdrcnn: Precise phishing detection with recurrent convolutional neural networks. *Security and Communication Networks*, 2019, **2019**.

[34] Putra Wanda and Huang Jin Jie. Urldeep: Continuous prediction of malicious url with dynamic deep learning in social networks. *International Journal of Network Security*, 21(6):971–978, **2019**.

[35] Ali Aljofey, Qingshan Jiang, Qiang Qu, Mingqing Huang, and Jean-Pierre Niyigena. An effective phishing detection model based on character level convolutional neural network from url. *Electronics*, 9(9), **2020**.

[36] Farid Tajaddodianfar, Jack W Stokes, and Arun Gururajan. Texception: a character/word-level deep learning model for phishing url detection. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2857–2861. IEEE, **2020**.

[37] Mehmet Korkmaz, Emre Kocyigit, Ozgur Koray Sahingoz, and Banu Diri. Phishing web page detection using n-gram features extracted from urls. In *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–6. IEEE, **2021**.

[38] Atsuki Yamaguchi, George Chrysostomou, Katerina Margatina, and Nikolaos Aletras. Frustratingly simple pretraining alternatives to masked language modeling. *arXiv preprint arXiv:2109.01819*, **2021**.

[39] Rana Alabdan. Phishing attacks survey: Types, vectors, and technical approaches. *Future Internet*, 12(10):168, **2020**.

[40] Kang Leng Chiew, Kelvin Sheng Chek Yong, and Choon Lin Tan. A survey of phishing attacks: Their types, vectors and technical approaches. *Expert Systems with Applications*, 106:1–20, **2018**.

[41] Friedhelm Hillebrand, Finn Trosby, Kevin Holley, and Ian Harris. The creation of the sms concept from mid-1984 to early 1987. **2009**.

[42] Kaspersky. What is spear phishing?, **2021**.

[43] S. Chiasson M. Alsharnouby, F. Alaca. Why phishing still works: User strategies for combating phishing attacks. *International Journal of Human-Computer Studies*, 82(1):69–82, **2015**.

[44] M. Huber E. Weippl K. Krombholz, H. Hobel. Advanced social engineering attacks. *Journal of Information Security and Applications*, 22(1):113–122, **2014**.

[45] Joakim Loxdal, Måns Andersson, Simon Hacks, and Robert Lagerström. Why phishing works on smartphones: A preliminary study. In *HICSS*, pages 1–10. **2021**.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, **2012**.

[47] Philippe De Wilde. *Neural network models: theory and projects*. Springer Science & Business Media, **2013**.

[48] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, **1998**.

[49] Tensorflow. Embedding layer, **2021**.

[50] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, **2013**.

[51] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. **2014**.

[52] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, Lille, France, **2015**.

[53] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 648–656. **2015**.

[54] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, **1990**.

[55] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, **1997**.

[56] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, **1997**. doi:10.1109/78.650093.

[57] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, **2009**. doi:10.1109/TPAMI.2008.137.

[58] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, **2000**. ISSN 0899-7667. doi:10.1162/089976600300015015.

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

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, **2017**.

[61] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, **2014**.

[62] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, **2006**.

[63] Simon Bell and Peter Komisarczuk. An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank. In *Proceedings of the Australasian Computer Science Week Multiconference*. **2020**.

[64] Ahmed AlEroud and George Karabatis. Bypassing detection of url-based phishing attacks using generative adversarial deep neural networks. In *Proceedings of the Sixth International Workshop on Security and Privacy Analytics*, page 53–60. **2020**.

[65] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, **2017**.

[66] Peng Yang, Guangzhen Zhao, and Peng Zeng. Phishing website detection based on multidimensional features driven by deep learning. *IEEE Access*, 7:15196–15209, **2019**.

[67] Tomas Rasymas and Laurynas Dovydaitis. Detection of phishing urls by using deep learning approach and multiple features combinations. *Baltic journal of modern computing*, 8(3):471–483, **2020**.

[68] Richard Gregory. Brainy mind. *British Medical Journal*, 317(7174):1693–1695, **1998**.

[69] Firat Coskun Dalgic, Ahmet Selman Bozkir, and Murat Aydos. Phish-iris: A new approach for vision based brand prediction of phishing web pages via compact visual descriptors. In *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–8. IEEE, **2018**.

[70] Leszek Cieplinski. Mpeg-7 color descriptors and their applications. In *International Conference on Computer Analysis of Images and Patterns*, pages 11–20. Springer, **2001**.

[71] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, **2012**.

[72] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. Whitenet: Phishing website detection by visual whitelists. **2019**.

[73] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823. **2015**.

[74] Gowtham Ramesh, Ilango Krishnamurthi, and K Sampath Sree Kumar. An efficacious method for detecting phishing webpages through target domain identification. *Decision Support Systems*, 61:12–22, **2014**.

[75] Shuichiro Haruta, Hiromu Asahina, and Iwao Sasase. Visual similarity-based phishing detection scheme using image and css with target website finder. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6. **2017**. doi:10.1109/GLOCOM.2017.8254506.

[76] Routhu Srinivasa Rao and Alwyn Roshan Pais. Jail-phish: An improved search engine based phishing detection system. *Computers & Security*, 83:246–267, **2019**.

[77] Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, and Banu Diri. Machine learning based phishing detection from urls. *Expert Systems with Applications*, 117:345–357, **2019**.

[78] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. Phishstorm: Detecting phishing with streaming analytics. *IEEE Transactions on Network and Service Management*, 11(4):458–471, **2014**. doi:10.1109/TNSM.2014.2377295.

[79] Mohammad Saiful Islam Mamun, Mohammad Ahmad Rathore, Arash Habibi Lashkari, Natalia Stakhanova, and Ali A Ghorbani. Detecting malicious urls

using lexical analysis. In *International Conference on Network and System Security*, pages 467–482. Springer, **2016**.

[80] Manu Siddhartha. Malicious urls dataset, **2021**.

[81] J Akshaya. Phishing websites detection, **2020**.

[82] Siddharth Kumar. Malicious and benign urls, **2019**.

[83] Openphish. Openphish, **2021**.

[84] wxiaoguang. Tldparser, **2020**.

[85] Bushra Sabir, M Ali Babar, and Raj Gaire. An evasion attack against ml-based phishing url detectors. *arXiv preprint arXiv:2005.08454*, **2020**.

[86] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, **2019**.

[87] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, **2018**.

[88] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*. **2015**.