A Spam Transformer Model for SMS Spam Detection

by

Xiaoxu Liu

Thesis submitted in partial
fulfillment of the requirements for the
Master of Computer Science degree

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

With the prosperity of the Short Message Service (SMS), the increasing number of spam messages has become a serious problem. The need to block spam messages requires us to develop new SMS spam detection technologies. The Transformer, an attention-based sequence to sequence model, has achieved excellent results in multiple different tasks recently. In this thesis, we propose a modified Transformer model for SMS spam messages detection.

The evaluation of our proposed modified spam Transformer is performed on SMS Spam Collection v.1 dataset and UtkMl's Twitter Spam Detection Competition dataset, with the benchmark of multiple established classifiers such as Logistic Regression, Naïve Bayes, Random Forests, Support Vector Machine, and Long Short-Term Memory. In comparison to all other candidates, our experiments show that the proposed modified spam Transformer achieves the best results in terms of almost all selected performance criteria.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Amiya Nayak, for providing me with guidance, advice, support throughout my master's study. Also, I am grateful to my senior colleague, Haoye Lu, for his valuable advice and support during my thesis work. Finally, I would like to thank my family, who continuously support me both physically and mentally.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objective

The Short Message Service (SMS) has been widely used as a communication tool over the past few decades with the popularity of mobile phones and mobile network growth. According to Statista's report in 2020, the number of mobile users worldwide reaches 6.95 billion with the forecasts indicating an increase to 7.1 billion by 2021, [56]. A survey in 2019 shows that more than 76% of mobile users in UK send mobile messages on a daily basis [70]. Mobile devices and SMS serverces are becoming necessities of people's daily life, The Guardian reports that people pick up their smartphones 58 times a day on average [47] and Finance Online reveals that more than 90% of people prefer to receive messages over calls [19].

With the development of mobile devices and various mobile applications, billions of people are connected and millions of businesses and marketers are benefited from trillions of SMS text messages worldwide. However, SMS users are also suffering from SMS spam. SMS spam, also known as drunk messages, refers to any irrelevant messages delivered using mobile networks [66]. Statista conducted a survey in the United States from March

20 to March 24 last year, and the survey points out that the number of spam messages growing exponentially over the past years with an average number of 14.7 spam text messages per month [72].

The Government of Canada defines spam as unsolicited text messages including malware, spyware, and false or misleading representations [10]. There are a few characteristics of spam text messages summarized by the Federal Trade Commission. In general, spam text messages aim to get your personal information by sending some fake text messages and end up gain financial benefits [75].

There are several reasons that lead to the popularity of spam messages. First and foremost, there is a large number of users who use mobile phones in the world. According to [71], there were 8,304 million mobile subscriptions in 2019. A large amount of mobile phone users makes the potential victims of the spam messages attack also high. Secondly, the cost of sending out spam messages is low compared to the potential gains, which could be taken advantage of by the spam attacker. Last but not least, the capability of the spam classifier on most mobile phones is relatively weak due to the shortage of computational and battery resources on portable devices, which limits them from identifying the spam messages correctly and efficiently.

Machine learning is one of the most popular topics in the last few decades. The applications of machine learning techniques have been greatly changing our lives. Spam detection is a relatively mature research topic in the field of machine learning, and there are a great number of machine learning based approaches proposed. Most of the machine learning based classifiers were dependent on the handcrafted features extracted from the training data [38].

As a class of machine learning techniques, deep learning has been developing rapidly recently thanks to the surprising growth of computational resources in the last few

decades. Nowadays, deep learning based applications play a significant part in our society, making our lives much easier in many aspects. As one of the most effective and widely used deep learning architectures, Recurrent Neural Network (RNN), as well as its variants such as Long Short-Term Memory (LSTM), were applied to spam detection and proved to be extremely effective during the last few years.

Although the current SMS spam detection approaches have been proved to be effective in multiple experiments performed by different researchers, most of them are still merely focus on one single small unbalanced dataset. In reality, a lot of the existing works are based on the SMS Spam Collection v.1 [1] dataset, which is a fabulous dataset with high-quality real messages but is insufficient in terms of the number of spam messages. Therefore, we not only intend to propose a new approach for SMS spam detection that improves the performance compared to the current works. More importantly, we expect to test and evaluate our methodology on a larger dataset with more balanced data.

1.2 Main Contribution

In this thesis, we address the SMS spam detection problem by proposing a modified model based on the Transformer [77], a relatively new attention-based model. Additionally, we analyze and compare the performance of SMS spam detection between multiple traditional machine learning classifiers, an LSTM deep learning solution, a CNN-LSTM [26] approach from existing works, and our proposed spam Transformer model. In order to make the evaluation more persuasive, the experiments are not only performed on the popular SMS Spam Collection v.1 [1] dataset, but also on the UtkMl's Twitter Spam Detection Competition (UtkMl's Twitter) [76] from Kaggle.

1.3 Thesis Organization

The remainder of the thesis is organized as follows.

- Chapter 2 reviews concepts of Neural Network, Logistic Regression, Support Vector Machine, K-Nearest Neighbor, Decision Tree, Recurrent Neural Network, Long Short-Term Memory, Sequence-to-sequence Models, and the attention mechanism.
- Chapter 3 reviews related work on SMS spam detection, including machine learning approaches such as S-Detector, SmiDCA, and Smishing Detector, and deep learning approaches such as SLSTM and CNN-LSTM.
- Chapter 4 discusses the Transformer model in detail and introduces our modified spam Transformer model.
- Chapter 5 describes the experiment setup, and the experimental results are also presented and analyzed in this chapter.
- Chapter 6 concludes this thesis and proposes several potential future works.

Chapter 2

Background

2.1 Neural Network

Neural Network is a machine learning technique that uses artificial neurons to simulate the way how biological neurons in the human brain works. The basic element of each neural network is the artificial neuron which consists of three components: weights and bias, summation function, and activation function.

As shown in Figure 2.1, each neuron is embedded with one summation function and one activation function, the summation function is usually a linear regression function with weight and bias, while the activation function provides non-linearity transformation for the summation output. Suppose we have n attributes labeled for sample i, where each attribute is labeled as x_i, and the coefficient for each x_i is w_i, and for each neuron we would assign a bias value b. The summation function for each neuron is defined as the following:

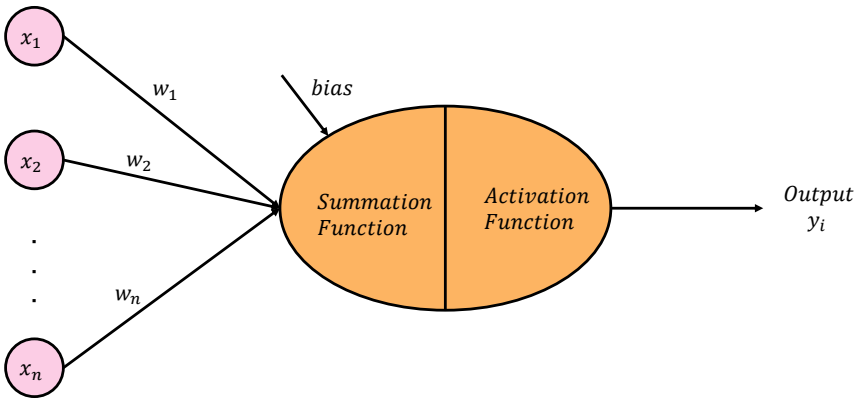

Figure 2.1: Structure of a Basic Neural Network with One Neuron

$$z = \sum_i w_i x_i + b \tag{2.1}$$

The output number from the summation function will be fed into the activation function for non-linear conversion. In the following section, we will look into different choices of activation functions, and introduce the advantage and disadvantages of each candidate function.

2.1.1 Activation Function

Activation Functions are normally used to determine whether a neuron is activated through influencing the output of the neuron. In neural networks, activation functions often introduce non-linearity, which usually contribute to compute or approximate more

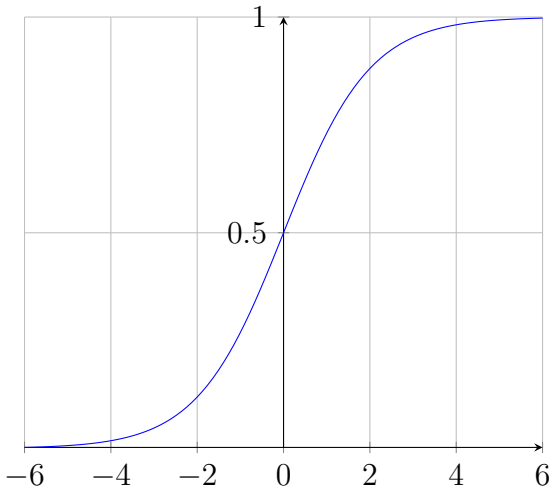

Figure 2.2: Sigmoid Function

complex functions using fewer neurons. There are multiple alternative functions that can be used as an activation function, such as Sigmoid, Tanh, ReLU, LeakReLU, and many others [68]. Among all the candidate activation functions, Sigmoid and ReLU [54] [27] are the most popular today.

The sigmoid function is a logistic function that returns a value between 0 and 1. The sigmoid function is popular for binary classification problems due to its nature of binary mapping. When using the sigmoid function as the activation function for our summation output z, it will map z in range $(-\infty, +\infty)$ to $(0, 1)$, indicating the possibility of being class 1. Suppose we have the summation output z. The sigmoid function can be defined as follows:

$$\sigma(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{2.2}$$

The returning value of the sigmoid function is in the range from 0 to 1, which indicates the extent of activation. When x is large, the sigmoid function returns a value close to 1. On the contrary, it gives 0 when x approaches negative infinity. Besides, the sigmoid

function is also employed in multiple binary classification problems to rescale the output to the range from 0 to 1.

Hyperbolic Tangent Function (tanh) is another commonly used activation, especially suitable for hidden layers. The tanh function can be defined as the ratio between the sine and cosine function for a certain z [17]. Unlike the sigmoid function whose results are in the range of 0 to 1, tanh function maps our summation output in the range of -1 to 1, thus the results will be centered at 0. Suppose we have the summation output z, the tanh function is defined as follows:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.3}$$

The main difference between the sigmoid function and the tanh function is their value range, and the nature of the binary range of sigmoid function makes it exceptionally suitable for output calculation while the nature of the centralized range of tanh function makes it suitable for hidden layer neurons. However, as you can see from Figure 2.2 and Figure 2.3, the downside of both these two functions is that when z is very large or very small, the slope of the function curve would be close to 0 which would slow down the gradient descent process.

Rectified Linear Unit (ReLU) is a widely-used activation function. Nair and Hinston pointed out a side-effect of using binary units or binomial units that makes the training unstable and inefficient [54]; as an alternative solution, they introduced the idea of using rectified linear units to overcome this shortcoming, and since then ReLU has become the most widely-used activation function in deep learning applications [55]. The idea of ReLU is to rectify the summation outputs that are less than zero to be zero and make the positive outputs remain the same; thus, the positive part of the ReLU function is nearly

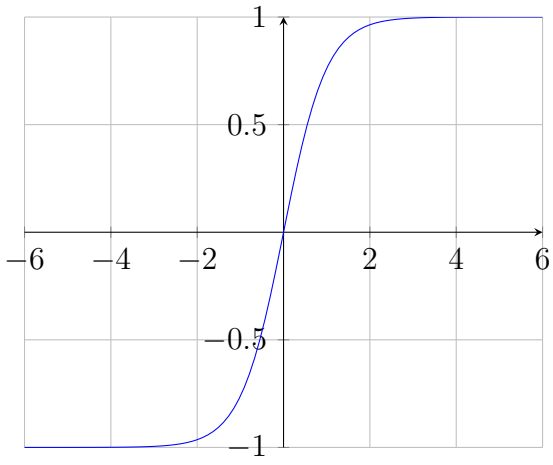

Figure 2.3: Tanh Function

a linear function, which is suitable for gradient descent optimization. The function is defined as follows:

$$RelU(z) = max(0, z) \tag{2.4}$$

As you can see from the function definition above, for any positive x, the output value of the ReLU function is equal to x. However, it returns 0 for negative or zero input. There are several benefits of using ReLU as the activation function. For one thing, the ReLU function is a simple function with comparison computation only, which makes it computationally friendly. For another, ReLU performs better compared to sigmoid in terms of preventing gradient vanishing problem since the gradient of ReLU is a constant number for positive input and zero for negative input.

Although the ReLU function works amazingly fine in practice, there is still a potential drawback of ReLU that it may lead to cases where a unit is never activated because the gradient is 0 when a unit is inactive. In those gradient-based optimization algorithms, the weights of the unit remain unchanged for zero gradients, which makes the unit stuck in

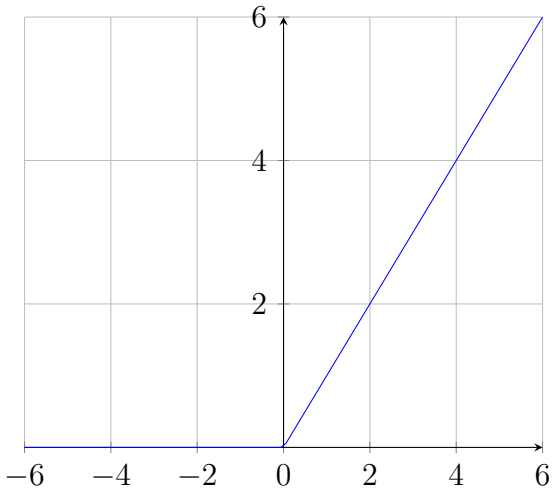

Figure 2.4: ReLU Function

inactive status. To address this, LeakyReLU [46] was introduced. LeakyReLU function is introduced to modify the negative part of the ReLU function so that the optimization performance would not be affected by the gradient equal to zero cases. The LeakyReLU is defined as follows:

$$LeakyReLU(x) = \begin{cases} max(0, x), & x > 0, \\ kx, & otherwise,\ k \in [0, 1] \end{cases} \tag{2.5}$$

As you can from the function definition above, the LeakyReLU only introduces a small slope to the negative internal of the ReLU function [79], and the most common k used by researchers is 0.01. Therefore, the features of the LeakyReLU function are nearly identical to the standard ReLU function, except for a small gradient for negative input, which could potentially contribute to better robustness.

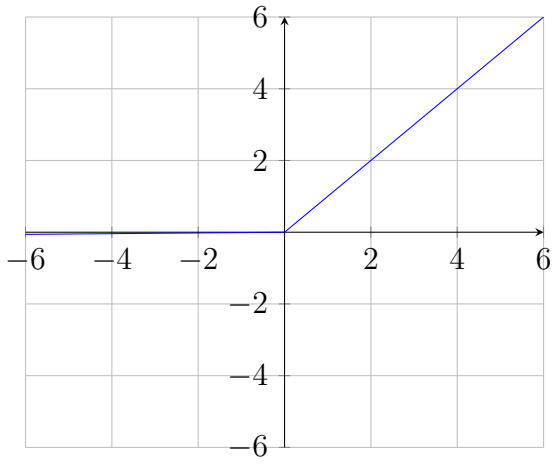

Figure 2.5: LeakyReLU Function

2.2 Logistic Regression

The classic regression model is the standard linear regression model [63] which is defined as follows:

$$y = \beta + \beta_1 x + \epsilon \tag{2.6}$$

As you can see from the model definition above, both the input x and the output y are continuous dimensions in the range from $-\infty$ to $+\infty$. However, when you have classification problems, either the output y should be categorical or in the range of 0 to 1 which indicates the probability of being in certain classes. With such special demands for a more reasonable model for categorical targets, logistic regression was introduced with the nature of a 0 to 1 value range.

Suppose we have multiple independent variables $x_1, x_2, ...x_n$ where n is the total number of features selected, and we have one dependent variable y as our target output whose value range only has two possibilities, either Positive or Negative. Instead of applying linear regression to identify the relationship between multiple features with

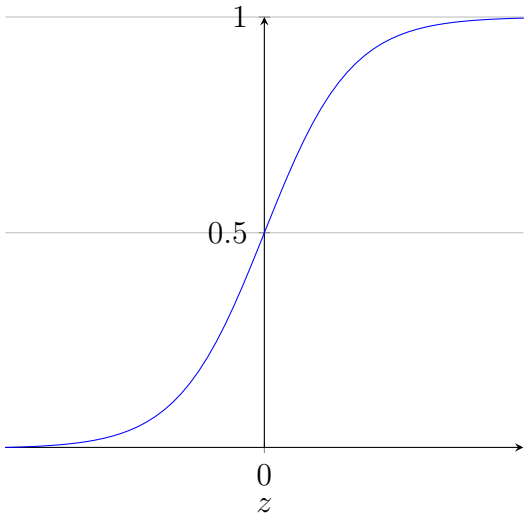

Figure 2.6: Logistic Regression

the depended variable, logistic regression model describes relationship between multiple independent variables to the dichotomous target variable using a S-shape model [18]. The logistic regression mathematical definition can be seen as the following:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{2.7}$$

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}} \tag{2.8}$$

If we plot the logistic regression model based on the function of z and z in the range from $-\infty$ to $+\infty$, we could see that the logistic regression is an S-shape model with two horizontal asymptotes at $y = 0$ and $y = 1$.

On the left side of the function, when z approaches $-\infty$, $f(z)$ equals to 0. On the other hand, when z approaches $+\infty$, $f(z)$ equals to 1. The mathematical proof can be seen as follows:

$$\begin{aligned} f(-\infty) &= \frac{1}{1+\mathrm{e}^{-(-\infty)}} = \frac{1}{1+\mathrm{e}^{(\infty)}} = 0 \\ f(\infty) &= \frac{1}{1+\mathrm{e}^{-(\infty)}} = \frac{1}{1+\mathrm{e}^{(-\infty)}} = 1 \end{aligned} \tag{2.9}$$

In order to interpret the logistic function to be a descriptive model, we assume that the dependent variable has two possible values, 1 if "positive category" and 0 if "negative category". We introduce the probability to describe the information provided by the logistic function as the probability of being $y = 1$ given $x_1, x_2, ..., x_n$, denoted as $P(y = 1|x_1, x_2, ..., x_n) = \frac{1}{1+\mathrm{e}^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n)}}$.

To find the most optimized β coefficient, we would apply gradient descent techniques to find the parameter sets that make the minimum cost function. The cost function is the function we use to describe the difference between our model output and the real output, and our goal is to make the cost function as small as possible so that our model results are close enough to the real results. The cost function is the summation of the loss function of all the training samples; the definition is shown as follows:

$$L(\hat{y}^i, y^i) = -y^i \log(\hat{y}^i) - (1-y^i)\log(1-\hat{y}^i) \tag{2.10}$$

$$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i) \tag{2.11}$$

Since logistic regression model is commonly used for binary classification problems, the possible y values are either 0 or 1 indicating one of the classes. To simplify the loss function, we can say that when y_i equals to 1, the loss function will be just $-\log(\hat{y}^i)$,

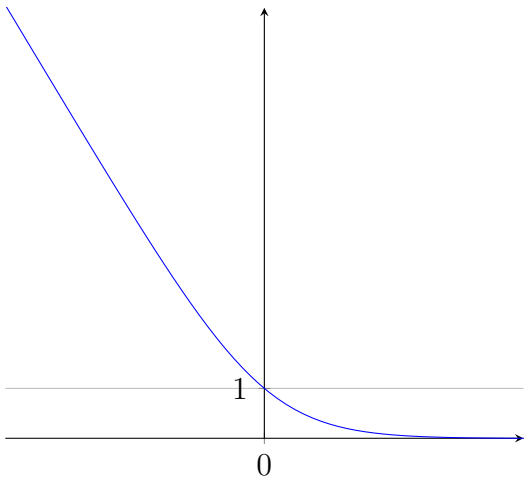

Figure 2.7: Loss Function when $y^i = 1$

and when y_i equals to 0, the loss function will be just $-\log(1-\hat{y}^i)$. The loss function graphs for $y^i = 1$ and $y^i = 0$ are shown in the following two figures.

The logistic regression model is to find the best parameter set that can minimize the cost function J using all the training samples, and once the model is trained, we can feed in new data samples to make predictions based on their probability of being positive class or negative class.

2.3 Support Vector Machine

Support Vector Machine (SVM) is a popular machine learning algorithm for classification problems. One shortcoming of the logistic regression model is that the decision boundary decided by the model is not unique, so the capacity and generalization performance is not guaranteed. Later research has found that a good classifier should automatically tune the capacity and generalization of the classification model by maximizing the margin amounts to achieve a large margin decision boundary [7].

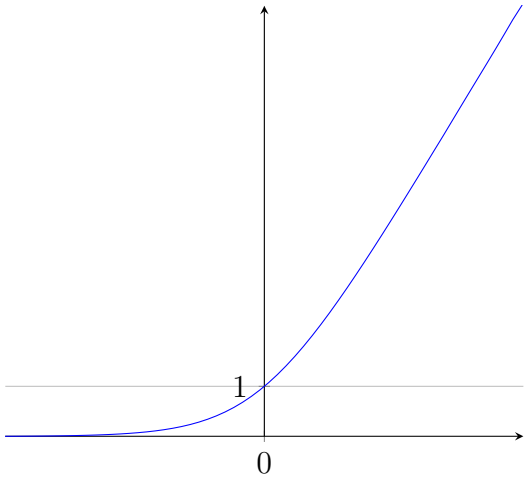

Figure 2.8: Loss Function when $y^i = 0$

Moreover, if you look at Figure 2.9 and Figure 2.10, the logistic loss function has the potential risk of gradient vanishing or gradient exploding problem when the z value is extremely large or extremely small. We would like a loss function that has stable gradient performance and also has high punishment when the error term is too big. To achieve this large margin classification function design, the support vector machine was introduced.

The loss function of the support vector machine algorithm is an optimized function from the cross-entropy function used by the logistic regression; instead of using a smooth curve for the loss function, the support vector machine splits the function into two intervals and makes it a linear function instead of the logistic function.

The loss function for the two y_i cases can be seen as Figure 2.9 for $y^i = 1$ samples set and Figure 2.10 for $y^i = 0$ samples set. We will name the two cases as $Cost_1$ and $Cost_2$ in the remainder of this thesis.

When $y^i = 1$ the loss function will be divided into two intervals based on $z < 1$ or $z > 1$; for the $z < 1$ interval, it will be a linear function with respect to y and z while for the $z > 1$ interval, it will simply be $y = 0$. Similarly, when $y^i = 1$ the loss function

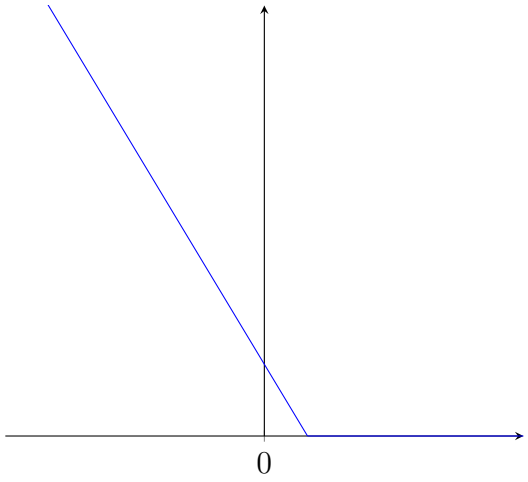

Figure 2.9: Loss Function for SVM when $y^i = 1$

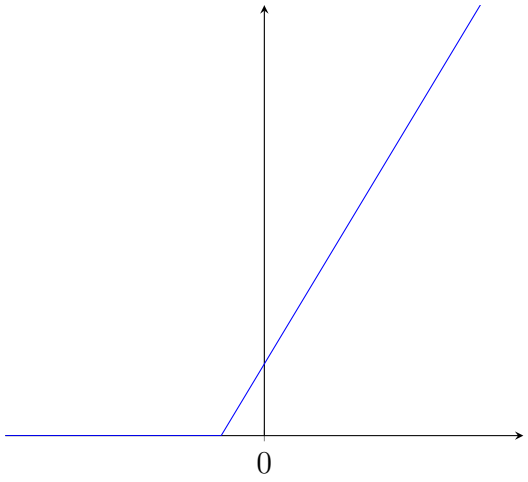

Figure 2.10: Loss Function for SVM when $y^i = 0$

will be divided into two intervals based on $z < -1$ or $z > -1$; for the $z > -1$ interval, it will be a linear function with respect to y and z while for the $z < -1$ interval, it will simply be $y = 0$ as well.

Based on the loss function for the support vector machine, we can now define our cost function using $Cost_1$ and $Cost_2$ as follows:

$$J = C \sum_{i=1}^{m} (\hat{y}^i Cost_1(\hat{y}^i) + (1 - \hat{y}^i) Cost_2(y^i)) + \frac{1}{2} \sum_{j=1}^{n} \beta_j^2 \tag{2.12}$$

So, why the cost function designed for the support vector machine can find the best decision boundary with a large margin? To answer this question, we need to take a look at the optimization process to understand the mechanism supporting the performance.

The horizontal axis of our loss function represents the summation output z which could be written as $\beta_0 + \beta_1 x_1 + ...\beta_n x_n$ for one training sample; if there are multiple training samples, then we can vectorize the calculation process for matrix representation. Assume that we have m training samples and n feature dimension, then the training set can be represented as a matrix X with dimension $m \times n$, and all the parameters β can be represented by the vector B with dimension $n \times 1$ as the follows:

$$X = \begin{bmatrix} ... & (x^1)^T & ... \\ ... & ... & ... \\ ... & (x^m)^T & ... \end{bmatrix} \tag{2.13}$$

$$B = \begin{bmatrix} \beta_0 \\ ... \\ \beta_n \end{bmatrix} \tag{2.14}$$

$$z = XB = \begin{bmatrix} \beta_0 x_0^1 + \beta_1 x_1^1 + ... + \beta_n x_n^1 \\ ... \\ \beta_0 x_0^m + \beta_1 x_1^m + ... + \beta_n x_n^m \end{bmatrix} \tag{2.15}$$

For a certain $\hat{y^i}$, we can use the vector representation of x^i and B to calculate $B^T x^i$. In order to minimize the cost function, we can have $B^T x^i$ to be greater than 1 when $y^i = 1$ and $B^T x^i$ to be smaller than -1 when $y^i = 0$. Equivalently, we can convert $B^T x^i$ to be the inner product of x^i and B which is equal to the projection of x^i on B times the length of B.

$$B^T x^i = p^i \cdot \|B\| \tag{2.16}$$

Since we have a regularization term in the cost function which will punish large β values so that the length of B as $\|B\|$ will be as small as possible, in order to make the inner production larger, we need to make the projection of x^i on B as large as possible, and thus the model will choose the decision boundary which gives large margin as well as large projection.

2.4 K-Nearest Neighbor

K-Nearest Neighbor is a distance-based classification algorithm that is simple but efficient in many cases [82]. For a certain input sample x^i, the algorithm will calculate the distance between x^i with all the remaining data samples available in the data set and choose the k data samples as the nearest neighbors for x^i so that the class prediction of x^i is determined by the class of these nearest neighbors. In most cases, a majority vote is used in order to determine the class of input data sample x^i.

The biggest challenge of using the K-Nearest Neighbor algorithm is how to define the best k. The most straightforward way is to run an iteration loop from a certain range of k and compare the model performance using different k values and pick the best k. There are also other proposals that aim to make the K-Nearest Neighbor algorithm less dependent on the choice of k [30], such as using multiple neighbor sets instead of one neighbor set so the performance is not defined by one single choice of k.

The steps of the K-Nearest Neighbor algorithm can be summarized as the following:

- Choose a distance measurement or similarity measurement to build the similarity matrix.
- Build for loop from $k = k_0$ to $k = k_n$ and validate the model performance, choose the best k.
- feed unlabeled data x^i, calculate the distance or similarity of x^i to all labeled data samples.
- Find the group of data samples as the neighbor set whose d_j is the smallest among all data samples.
- Label x^i as $class_k$ which most data samples in the neighbor set belong to.

With the detailed definition of the K-Nearest Neighbor algorithm, the second challenge of applying this classification algorithm is to choose a suitable distance measurement for similarity calculation. Here we listed a few commonly used similarity measures as follows.

The first commonly used distance measurement is Euclidean distance. Suppose we can two data points x^i and y^i with n dimensions, the distance between x^i and y^i can be defined as their square root of difference among all dimensions.

$$EuclideanDistance(x^i, y^i) = \sqrt{\sum_{k=1}^{n}(x_k^i - y_k^i)^2} \tag{2.17}$$

Instead of using the square root to represent the distance, we can also use the absolute value of the difference between x^i and y^i, and this method is called Manhattan distance.

$$ManhattanDistance(x^i, y^i) = \sum_{k=1}^{n} \mid x_k^i - y_k^i \mid \tag{2.18}$$

Adopting the advantages of both Euclidean distance and Manhattan distance, Minkowski distance was introduced as a generalized metric. When $\lambda = 1$ the form is exactly the same as Manhattan distance; when $\lambda = 2$ the form is exactly the same as Euclidean distance. Other λ values could also be used depending on problem statements.

$$MinkowskiDistance(x^i, y^i) = \sqrt[\lambda]{\sum_{k=1}^{n} \mid x_k^i - y_k^i \mid^\lambda} \tag{2.19}$$

Another popular similarity measurement is Cosine Similarity which calculates the

normalized dot product of two data points. The dot product is applied to x^i and y^i in their vector form as $\begin{pmatrix} x_1^i & x_2^i & ... & x_n^i \end{pmatrix}$ and $\begin{pmatrix} y_1^i & y_2^i & ... & y_n^i \end{pmatrix}$. The cosine similarity measures the angle between these two vectors thus the similarity if decided by their orientation. If the two vectors are at the same orientation with 0°, then the cosine similarity will be 1. If the two vectors are at 90° then the cosine similarity will be 0, all other cases will be a similarity rate between 0 and 1.

$$ConsineSimilarity(x^i, y^i) = cos(\theta) = \frac{x^i \cdot y^i}{\|x^i\|\|y^i\|} \tag{2.20}$$

All the measurements mentioned above are dealing with the distance between two vectors or two data points, but if the two objects are two sets of elements that don't necessarily be numeric elements, then we could use Jaccard similarity. Suppose we have two data sets X and Y, the similarity of X and Y is defined by their intersection set and union set. The intersection set of two data sets is the set of elements that exist in both X and Y while the union set of two data sets is the set of elements that exist in either X or Y.

$$JaccardSimilarity(X, Y) = \frac{|intersection(X, Y)|}{|union(X, Y)|} \tag{2.21}$$

In practice, there is no theory guarantee that a certain distance measurement works best for a certain type of problem, we would tend to try different alternative similarity measures and compare their contribution to the model performance.

2.5 Decision Tree

Decision Tree is one alternative classifier based on tree structures. In general, a decision tree consists of one root node, multiple internal nodes, and multiple leaf nodes. The root node is the original input data set, along the tree path with different internal nodes, each node will split the data set into smaller data sets based on the chosen splitting condition, and each leaf node is the final class result for a certain set of samples [22].

To achieve better performance, the decision tree will choose among all possible features to be the best splitting feature that can give the best splitting results. The criterion used to choose among these features can define different types of decision trees. The three commonly used criteria include Gini Index, Information Gain, and Information Grain Ratio [53].

2.5.1 Feature Selection

Gini index is a popular measurement used for feature selection of decision tree classifiers and this type of decision tree is called Classification and Regression Tree (CART). To understand the Gini index, we have to first understand the definition of impurity. If all elements in the data set belong to one class, then we would say this data set is 100% pure and the Gini index will equal to 0, otherwise, if all elements in the data set belong to various classes and their distribution is totally random, then the Gini index will equal to 1 which means high impurity. Assume that there are n classes and p_i is the probability of a random data sample in the set being a particular $Class_i$, then the definition of Gini index can be seen as the following:

$$GiniIndex = 1 - \sum_{i=1}^{n}(p_i)^2 \tag{2.22}$$

For each internal node on the CART decision tree, the algorithm will calculate the Gini index using each alternative feature as the splitting feature and the algorithm will choose the feature with the smallest Gini index, which is to choose the purest feature for splitting. Suppose D is the data set that need to be split, and C_k is the set of samples in D that belong to $Class_k$, A is a certain feature that can be used for splitting which will result in m smaller data sets denoted by D_i.

$$\begin{aligned} Gini(D) &= \sum_{k=1}^{n} \frac{|C_k|}{|D|}\left(1 - \frac{|C_k|}{|D|}\right) \\ Gini(D|A) &= \sum_{i=1}^{m} \frac{|D_i|}{|D|} Gini(D_i) \end{aligned} \tag{2.23}$$

Another feature selection method is using Information Grain and the decision tree that use information gain as the selection measure is defined as ID3. Unlike the Gini index, information gain makes use of the idea of entropy which also indicates the purity of the data set. The less pure a data set is, the higher the entropy of that certain data set. Thus, we could calculate the entropy of the data set before splitting based on feature A and the entropy of data sets after splitting by featuring A, the gap between these two types of entropy is our information grain, and the ID3 algorithm will choose the feature that gives highest information gain to be the splitting feature.

$$
\begin{aligned}
H(D) &= -\sum_{k=1}^{n} \frac{|C_k|}{|D|} \log_2 \left(\frac{|C_k|}{|D|} \right) \\
H(D|A) &= \sum_{i=1}^{m} \frac{|D_i|}{|D|} H(D_i) \\
InformationGain(D, A) &= H(D) - H(D|A)
\end{aligned}
\tag{2.24}
$$

The nature of information gain determines that the ID3 algorithm will tend to choose the type features that have more possible values, in extreme cases, if one feature is the primary key of the whole data set, then the information gain will definitely be higher since each data sample will be split into individual data sets so that the purity is high. In order to overcome the feature preference of using information gain, the information gain ratio was introduced and the decision tree that applies the information gain ratio is named C4.5.

The idea of the information gain ratio is to introduce a new concept of the intrinsic value of a certain feature A. The intrinsic value is defined as $H_A(D)$ as follows:

$$
\begin{aligned}
Gain_{ratio}(D, A) &= \frac{Gain(D, A)}{H_A(D)} \\
H_A(D) &= -\sum_{i=1}^{m} \frac{|D_i|}{|D|} \log_2 \left(\frac{|D_i|}{|D|} \right)
\end{aligned}
\tag{2.25}
$$

Although implementing the information gain ratio solves the problem of using information grain, it also has its own selection preference toward features with fewer possible values, which is the opposite choice of the information gain. Therefore, the C4.5 algorithm doesn't solely use the information gain ratio but integrates both information gain and information gain ratio to choose the splitting feature.

2.5.2 Random Forest

A decision tree is not a strong classifier in some cases, many researchers proposed to apply ensemble methods by generating multiple classifiers instead of a single classifier to improve model accuracy. Ensemble methods including boosting, bagging, voting, and stacking. Random forest is one ensemble algorithm using bagging with a decision tree as the base model.

The concept of bagging is to generate multiple data sets from the original data sets by random sampling, and each generated data set is independently used to train a classifier [57]. All the classifiers will act like one voter and the majority vote will be used as the final result for a certain input data. Random forest algorithm is one classification algorithm that adopts the idea of bagging and extends its concept to not only sample bagging but also features bagging. To apply a random forest algorithm, different random subsets of features are generated for each splitting step thus the effect of some strong features would be eliminated.

2.6 Recurrent Neural Network

As is known to all, shuffling the order of words in a sentence can severely influence the meaning of the entire sentence, which could potentially turn a legitimate message into spam messages, and vice versa. Therefore, in many Natural Language Process (NLP) problems, the order of words is no less important than the words themselves. To address this problem, we need a new kind of model that is capable to effectively learn from prior knowledge to improve the understanding of the data. Although the classical feed-forward neural network is a powerful deep learning technique that generally works well in many areas, it cannot utilize the information from the past. Derived from the feed-forward

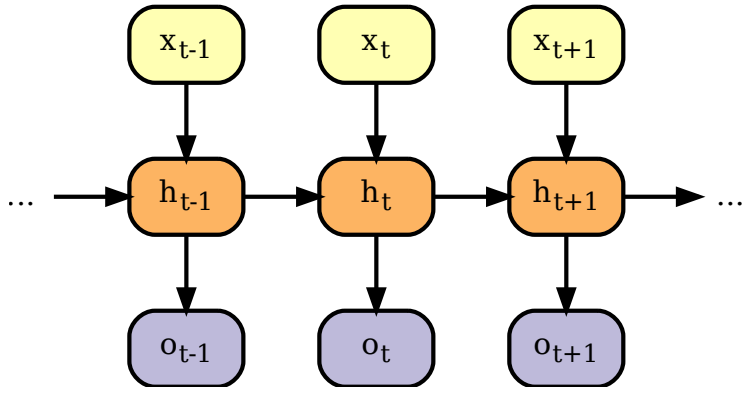

Figure 2.11: Structure of a typical Recurrent Neural Network.

neural network, recurrent neural network (RNN) [67] has the ability to reuse the saving information at the time of processing input values. Additionally, unlike the traditional feed-forward neural network supports only the input sequence with a fixed length, RNN is capable to handle the input sequence with different length.

Figure 2.11 shows the typical structure of RNN models, with the input sequence, output sequence, and the hidden layers at time t are represented by x_t, o_t, and h_t respectively. At time t, the current hidden layers state h_t is calculated based on the current input sequence x_t and the last hidden layers h_{t-1}. After the calculation of h_t is finished, the output at the current time step o_t is generated and the hidden layers state h_t will get involved in the calculation at the next time step t+1. Unlike the normal neural network, where the neurons in the same layer of the hidden layers are independent of each other, RNN models usually allow the data flows within the same layer. In another word, connections between neurons in the same layers or even self-connections are allowed generally allowed in RNN based models.

A major advantage of RNN models is that they are able to utilize the information from previous input and apply it at the current time, which is significantly useful in NLP problems since the context can help us understand the sentence better. However, a major drawback of the vanilla RNN is the vanishing and exploding gradients [5]. In back-propagation training process, the vanishing gradients refers to gradients go exponentially close to 0, while the exploding gradients refers to the gradients go exponentially increase. The vanishing and exploding gradients are usually caused by the multiplication of multiple derivatives in training process. Although there are several approaches [58] existing to address the vanishing and exploding gradients problem, in practice, it is still difficult for vanilla RNN to memorize and learn the features from long distance, which is described as long-term dependencies problem. In order to deal with the long-term dependencies problem, many researchers have proposed multiple variants of RNN, such as the Long Short-Term Memory (LSTM) [35], the Gated Recurrent Unit (GRU) [12], and the Clockwork RNN (CW-RNN) [40].

2.7 Long Short-Term Memory

The Long Short-Term Memory (LSTM) [35] is a famous variant of RNN. The main idea of the LSTM is the introduction of gate units, which are the structures that can determine to keep or discard the current information. A typical LSTM network consists of multiple memory cells, and each memory cell is formed by an input gate, a forget gate, and an output gate. In LSTM, at time t, the state of a memory cell c_t is calculated based on the input x_t and the last hidden state h_{t-1}. The state of input gate, output gate, and forget gate at time t are represented as i_t, o_t, f_t, respectively. Therefore, the LSTM transition functions are defined as follows [83]:

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
q_t &= tanh(W_q \cdot [h_{t-1}, x_t] + b_q) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot q_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{2.26}
$$

The σ denotes the sigmoid function, which is introduced in Section 2.1.1. The operator $\odot$ denotes the element-wise multiplication.

When the output of a gate unit is close to 1, the information is more likely to be memorized. On the contrary, a returning value close to 0 from a gate unit means that the information should not be kept. The input gate i_t is the gate unit that controls how much information should be stored at this time. The forget gate f_t is responsible to determine to what extent the memory from the last time c_{t-1} should be kept at time t. The output gate o_t at time t is designed to be used in the computation of the output (hidden state) based on the memory cell state.

In our LSTM approach for SMS spam detection, the input message embedding is fed into an LSTM network as an input sequence. Meanwhile, the LSTM network saves the important features and outputs a sequence with the same length as the input sequence. The output sequence is then fed into a feed-forward fully connected layer with a single neuron since SMS spam detection is a binary classification problem. Finally, a sigmoid function is applied to the output of the single neuron to produce a final prediction.

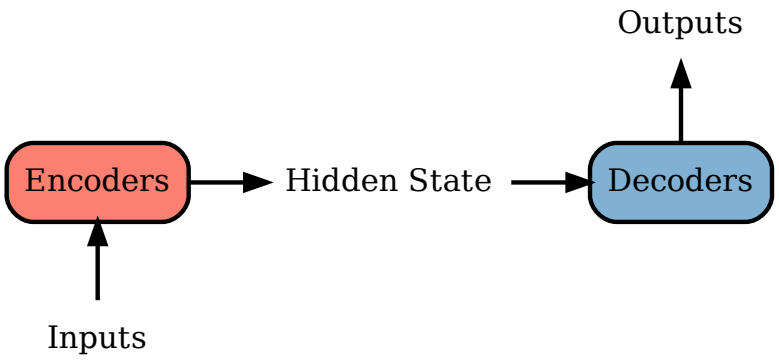

Figure 2.12: Structure of Encoder-Decoder architecture

2.8 Sequence-to-sequence Models

Sequence-to-sequence (Seq2Seq) [73] was introduced in 2014 by Sutskever et al. aiming to find a mapping between two sequences for translation tasks. Seq2Seq models employed the Encoder-Decoder architecture, which consists of an encoder stack, a hidden state, and a decoder stack.

Figure 2.12 presents a typical Encoder-Decoder architecture. The encoders take the input sequence and produce a hidden state with critical information. The hidden state is expected to be a fabulous abstract of the input sequence and contains important features from the input sequence. The hidden state is then consumed by the decodes to generate the output sequences. The vanilla version of the Seq2Seq model proposed in 2014 choose LSTM as both encoder and decoder, because LSTM has the ability to successfully learn on data with long-term dependencies [73].

The Sequence-to-sequence architecture was proved to perform outstandingly in multiple Natural Language Process (NLP) problems, and the methodology of the Encoder-

Decoder architecture has been applied in many different approaches in different fields. For instance, the Neural Machine Translation model proposed by Bahdanau et al. in [3] is a successful English-to-French translation based on the Encoder-Decoder architecture, which will be discussed in detail in Section 2.9.1.

Another crucial advantage of the Encoder-Decoder architecture is that the input sequence and output sequence can be different in terms of size or format, which provides much more flexibility and possibility. In reality, the Seq2Seq models have been proved themselves in language translation [73], Speech Recognition [62], and Video to Text [78]. Undoubtedly, Seq2Seq architecture is designed to fit translation tasks exceptionally well, since it can extract the relationship between the sequences in one language and the sequences in a different language.

A fundamental disadvantage of the Sequence-to-sequence model is related to the fixed-length hidden state. For one thing, a limited size hidden state may cause the incapability of storing enough summarized information of the input sequence. For another, an improper scale of the hidden state could lead to the difficulty for decodes to remembering dependencies within different elements. To address this challenge, the attention mechanism is introduced by researchers. More details of the attention mechanism will be discussed in the next section of this thesis.

2.9 Attention Mechanism

The attention mechanism is motivated by the cognitive attention of human beings. The main purpose of the attention mechanism is to find out the most important part from the input sequence. Concretely, the attention mechanism produces weights that represent the importance of the elements of data based on their correlation with the context. For

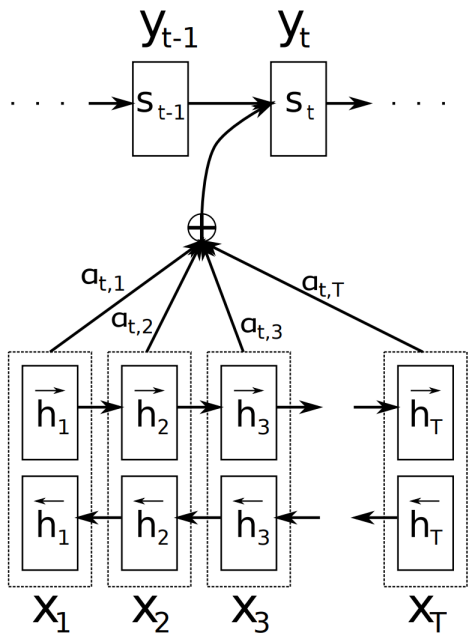

Figure 2.13: The graphical illustration of additive attention from [3]

a specific part of data, the larger the weight of attention it has, the more influential it is to the whole picture of data. Therefore, the attention mechanism makes it possible to focus on the key elements.

2.9.1 Bahdanau Model and Additive Attention

The additive attention mechanism was first introduced by Bahdanau et al. in [3] as an improvement of the hidden state of RNN Encoder-Decoder model [73] [12] in Neural Machine Translation (NMT). The introduction of the attention mechanism is believed to boost performance exceptionally.

The attention model of Bahdanau et al. takes source sequence x with a length of T_x as input, and aims to translate the source sequence into target sequence y with a length

of T_y. The source and target sequence is denoted as:

$$\begin{aligned} x &= (x_1, ..., x_{T_x}) \\ y &= (y_1, ..., y_{T_y}) \end{aligned} \tag{2.27}$$

The encoder of the attention model proposed in [3] is a bidirectional RNN model that generates a series of forward hidden states $(\overrightarrow{h_1}, ..., \overrightarrow{h_{T_x}})$ and backward hidden states $(\overleftarrow{h_1}, ..., \overleftarrow{h_{T_x}})$, and concatenate them to form the annotations $(h_1, ..., h_{T_x})$, which is defined as follows:

$$h_i = \begin{bmatrix} \overrightarrow{h_i} \\ \overleftarrow{h_i} \end{bmatrix} \tag{2.28}$$

The decoder is a Recurrent Neural Network (RNN) model that predicts the target sequence y. At the i-th position of the target sequence, the target word y_i is produced based on conditional probability:

$$p(y_i|y_1, ..., y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i) \tag{2.29}$$

where g is a nonlinear, potentially multi-layered, function that computes the probability of y_i based on the hidden state of RNN at the i-th step s_i, context vectors c, and the previous predictions. The RNN hidden states at the i-th step s_i of the decoders are calculated from.

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{2.30}$$

At each time i, the context vector c_i for output y_i is computed in decoder as a weighted sum of h. Therefore, it is also named as Additive Attention. The calculation of c_i is represented as:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.31}$$

For each annotation h_j, the α_{ij} is the weight that measures to what extent the h_j is going to contribute to the context vector c_i, and influence the hidden state s_i furthermore. The weight α_{ij} is computed as follows:

$$\alpha_{ij} = \frac{\exp(score(s_{i-1}, h_j))}{\sum_{k=1}^{T_x} \exp(score(s_{i-1}, h_k))} \tag{2.32}$$

The $score(s_{i-1}, h_j)$ is an alignment score that expresses how well the h annotation at the j-th position from the encoders h_j and the hidden state at the $i-1$-th position s_{i-1} match. The *score* is defined as:

$$score(s_i, h_j) = v_a^\intercal tanh(W_a[s_i; h_j]) \tag{2.33}$$

where v_a and W_a are trainable weight matrices.

The most important contribution of the additive attention mechanism is to compute the weights based on all the contexts that were generated by the encoder, and the decoder consumes the weighted combination of all the contexts instead of focusing only on a limited number of elements. Thus, with the help of the additive attention mechanism, the model proposed by Bahdanau et al. is not restricted by the length of context vectors in the hidden state of the RNN Encoder-Decoder model, which makes it potentially capable to remember the dependencies with long distance more effectively.

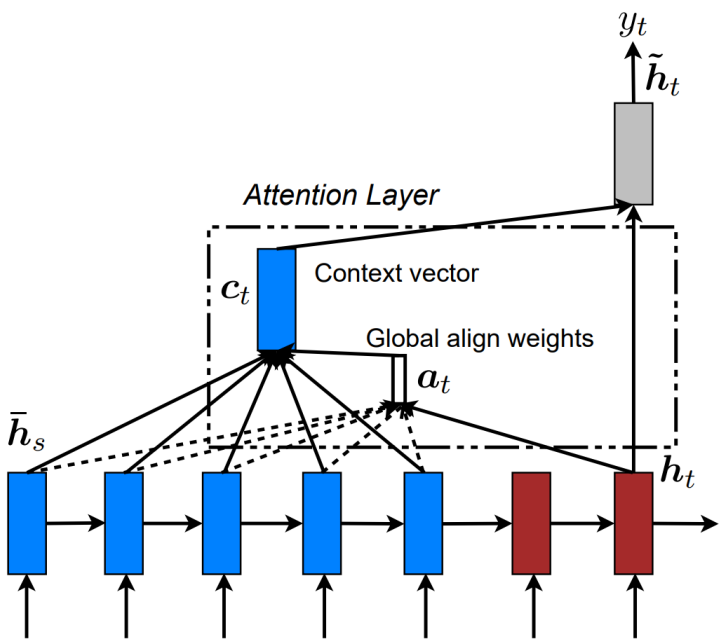

Figure 2.14: The graphical illustration of the global attention from [45]

2.9.2 Global Attention and Alignment Score Function

In [45], Luong et al. proposed several powerful attention techniques and made multiple improvements based on the additive attention translation model proposed by Bahdanau et al. in [3]. More specifically, the methodology of global attention and various different ways of calculating *score* function similar to Equation 2.33 were introduced in the paper of Luong et al. [45].

In the approaches of Luong et al. [45], the attention mechanism is also employed in the calculation of context vectors of the hidden states between the encoders and decoders, which is similar to the Bahdanau translation model. The term global attention refers to the fact that the computation of the context vectors c_t at each step t of the decoding process involves all the hidden states from the encoders. As a matter of fact, the context vectors c_t are generated based on the alignment vectors α_t, whose size is equal to the

length of the input sequence. In [45], Luong et al. proposed several candidate approaches of generating the alignment vectors and the score function including the location-based attention, the general attention, and the dot-product attention.

According to [45], the global attention model employs Long Short-Term Memory (LSTM) as encoders, which is different from the translation model from Bahdanau et al. [3]. For the decoder hidden states h_t and the context vectors c_t that is generated based on all encoder hidden states, the global attention model predicts the target word as:

$$p(y_t, x) = softmax(W_s \tilde{h}_t) \tag{2.34}$$

where h_t, c_t, and y_t are with length of t, and the $\tilde{h}_t$ refers to the attentional hidden states, which is defined as follows:

$$\tilde{h}_t = tanh(W_c[c_t; h_t]) \tag{2.35}$$

Similar to Equation 2.31, the computation of context vectors c_t is based on the alignment vectors α_t. For the encoder hidden state $\bar{h}_s$ at the s-th step, the $\alpha_t(s)$ denotes the weight that $\bar{h}_s$ contributes to the context vector c_t. The alignment vectors α and the score functions of the general attention and the dot-product attention is computed as follows:

$$\alpha_t(s) = \frac{\exp(score(h_t, \bar{h}_s))}{\sum_{s\prime} \exp(score(h_t, \bar{h}_{s\prime}))} \tag{2.36}$$

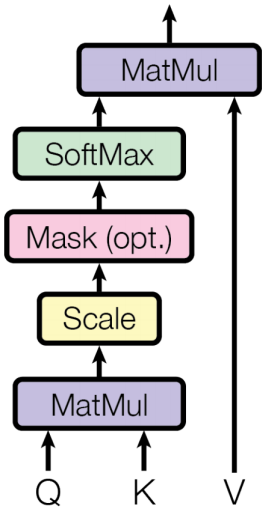

Figure 2.15: The graphical illustration of the scaled dot-product attention from [77]

$$score(h_t, \bar{h_{s\prime}}) = \begin{cases} h_t^{\intercal} W_a \bar{h_s}, & general\ attention \\ h_t^{\intercal} \bar{h_s}, & dot\text{-}product\ attention \end{cases} \tag{2.37}$$

where W_a is a trainable weight matrix.

Apart from the general attention, the dot-product attention, and the additive attention that is defined in Equation 2.33, in [45], Luong et al. also proposed an alternative alignment function named location-based attention for global attention. The location-based attention alignment vectors α_t are only related to the hidden states of decoders h_t. Below is the calculation of α:

$$\alpha_t = softmax(W_a h_t) \tag{2.38}$$

2.9.3 Scaled Dot-Product Attention Function

The scaled dot-product attention is introduced in the Transformer model [77] as the attention function. The scaled dot-product attention is the dot-product attention [45] that is reviewed in Section 2.9.2 with a scaling factor of $\frac{1}{\sqrt{d_k}}$, which aims to counteract the massive growth of dot-product when dimensions of queries and keys d_k is large. The scaled dot-product attention function is defined as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^\intercal}{\sqrt{d_k}}\right)V \tag{2.39}$$

where Q, K, and V refers to query, key, and value, respectively. The attention function takes a query Q and a set of key-value pairs (K, V) as input, and computes the weighted sum of values as output, where the weights are calculated based on the queries and keys.

Compared to the previous attention-based Encoder-Decoder approaches such as [3] and [45], in the decoders of Transformer, keys and values play a role similar to the hidden states generated by the encoders, which is expected to be a compression of the input sequence. On the other hand, queries in Transformer act like the previous hidden states of the decoders that summarize the previous output sequence, and the next output is predicted in decoders by consuming the queries and key-value pairs.

2.9.4 Multi-Head Attention

The multi-head attention mechanism is introduced in the Transformer [77] as a substitution of calculating attention function for a single time in order to enhance the efficiency of attention. The main idea of the multi-head attention mechanism is to compute the attention function multiple times in parallel and concatenate all the results together into

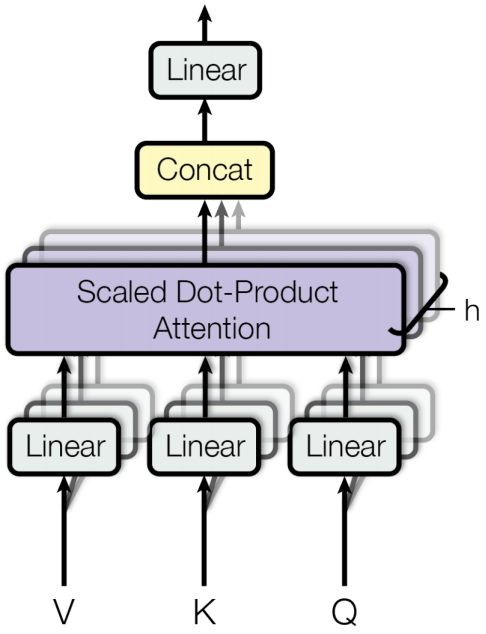

Figure 2.16: The graphical illustration of the multi-head attention mechanism from [77]

a matrix with desired dimensions.

In the previous practice like [3] and [45], the attention is directly performed on the hidden states from encoders and the previous decoder hidden states. In this way, there is only a single attention function calculated at one turn. However, Transformer finds an effective way to apply multiple attention functions at once. Specifically, the d_{model} dimensional queries, keys, and values are sent to some different learned linear layers to be projected h times to the dimension of d_k, d_k, and d_v, respectively. In another word, the projection linear layers are individually learned, and output projections have dimensions of d_k, d_k, and d_v, where $d_k = d_v = d_{model}/h$. After that, a number of h attention functions are performed in parallel on these projected queries, keys, and values, resulting in h different output values. Finally, all these h values are concatenated together and then projected back to a dimension of d_{model}. The entire process of the attention mechanism in the Transformer is defined as follows [77]:

$$
\begin{aligned}
MultiHead(Q, K, V) &= Concat(head_1, ..., head_h)W^O \\
head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V)
\end{aligned}
\tag{2.40}
$$

where W_i^Q, W_i^K, W_i^V, and W^O are trainable parameters matrices. The W_i^Q, W_i^K, and W_i^V are responsible to produce attentional queries, keys, and values in d_k, d_k, and d_v dimension, respectively. The W^O is the weight matrix that projects the concatenated attention result into d_{model} dimensions.

2.9.5 Attention Summary

There are also other forms of attention mechanism proposed. In [80], the attention mechanism is applied to the field of computer vision by Xu et al., and they also proposed two different approaches of attention named "soft attention" and "hard attention". In [45], other than the global attention that is discussed in Section 2.9.2, Luong et al. also proposed local attention, which is a combination of soft and hard attention from Xu et al. in [80].

Additionally, the attention mechanism has been widely employed as an add-on to improve the accuracy and reduce the cost in more and more deep learning based models and solutions, since the attention mechanism can help to extract the most significant parts so that the model can focus on them. In [81], Zhang et al. proposed Self-Attention Generative Adversarial Networks (SAGAN), an image synthesis model that introduces self-attention layers to Generative Adversarial Nets (GAN) [29]. Equipped with the self-attention module, SAGAN is able to handle details better coordinated with other distant details. In [74], Tian et al. proposed Dempster-Shafer and Deep Learning based Insider Threats Detection (DSDLITD), which employs the multi-head scaled dot-product atten-

tion that is discussed in Section 2.9.3 and Section 2.9.4 to generate the annotation of the sequence efficiently. In [65], Rocktäschel et al. proposed several attention-based methods of entailment relations recognition. In [44], Lu et al. employed multiple classifiers for audio classification, and the attention mechanism is used to integrate the results from different classifiers.

Chapter 3

Related Work

There are several different approaches proposed to identify SMS spam messages in the last few decades. A great number of these SMS spam classifiers are based on traditional machine learning techniques, such as Logistic Regression (LR), Random Forests (RF) [34], Support Vector Machine (SVM) [14], Naïve Bayes (NB), and Decision Trees (DT). Recently, with the prosperity of the deep learning techniques, an increasing number of researchers have been working on the SMS spam problem using deep learning based models such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN). Specifically, RNN and some of its variants are widely employed in many Natural Language Processing (NLP) problems. Among all the RNN variants, Long Short-Term Memory (LSTM) [35], an exceptionally successful variant of RNN that focus on addressing gradient vanishing problem and learning from long gap dependencies, has been more and more popular in many NLP tasks including SMS spam detection solutions.

The chapter is organized as follows. Section 3.1 reviews several SMS spam detection solutions based on typical machine learning classification algorithms. Section 3.2 introduces multiple deep learning based approaches that accomplished state-of-the-art performance in the field of SMS spam detection.

3.1 Machine Learning Spam Detection Approaches

3.1.1 SMS Spam Detection on 32 Machine Learning Classifiers

In [48], Mathew et al. compared the performance of 32 different machine learning classification algorithms on SMS spam detection. In order to implement implement and evaluate various algorithms, the authors made good use of the Weka Project [32] [21], an open-source machine learning software that is widely used for research and industrial applications. Additionally, the "StringToWordVector" function from Weka is employed to convert the string SMS messages data into feature vectors that the algorithms can understand.

The experiments performed by the authors were based on the SMS Spam Collection v.1 [1] dataset, which is also one of the datasets used in this thesis. According to the experimental results from [48], the Bayesian classifiers are the best approaches among multiple traditional machine learning classification algorithms in terms of the performance of identifying spam SMS messages. Specifically, Naïve Bayes Binomial accomplished the optimal 98.22% correct classification and also observed by the authors to be fast on the SMS Spam Collection v.1 dataset.

3.1.2 Rule-based Spam Detection

In [37], Jain et al. proposed a rule-based model for the SMS spam detection problems. The architecture of the rule-based model is presented in Figure 3.1. The authors extracted 9 rules and implemented Decision Tree, RIPPER [13], and PRISM [11] based on the rules to classify the SMS spam messages. According to the experimental results from the authors, the RIPPER outperformed the PRISM and Decision Tree on the proposed method and rules, yielding a 99.01% True Negative Rate (TNR) and a 92.82% True

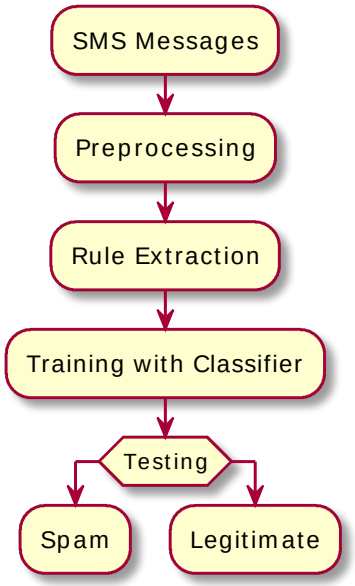

Figure 3.1: The architecture of the rule-based model to SMS spam detection proposed in [37]

Positive Rate (TPR) on the SMS Spam Collection v.1 dataset.

3.1.3 S-Detector

In [39], Joo et al. proposed a technique that combines rule-based URL analysis and the Naïve Bayes classifier. The authors found that a great portion of the SMS smishing messages involves URL. However, the spam URL often changes fast and uses shortcut service, making it difficult to filter harmful URLs using traditional blacklist and whitelist methods. The methodology of S-Detector is presented in Figure 3.2.

To determine whether a message is spam or not, S-Detector first searches for URLs in the message. For the URLs that employ shortcut service, S-Detector converts them back to the original long URLs. Thus, S-Detector is capable to find out the final destination of redirecting URLs. After that, S-Detector attempts to access the URLs in order to check if they are leading to the downloading of executable files on smartphones such as APK

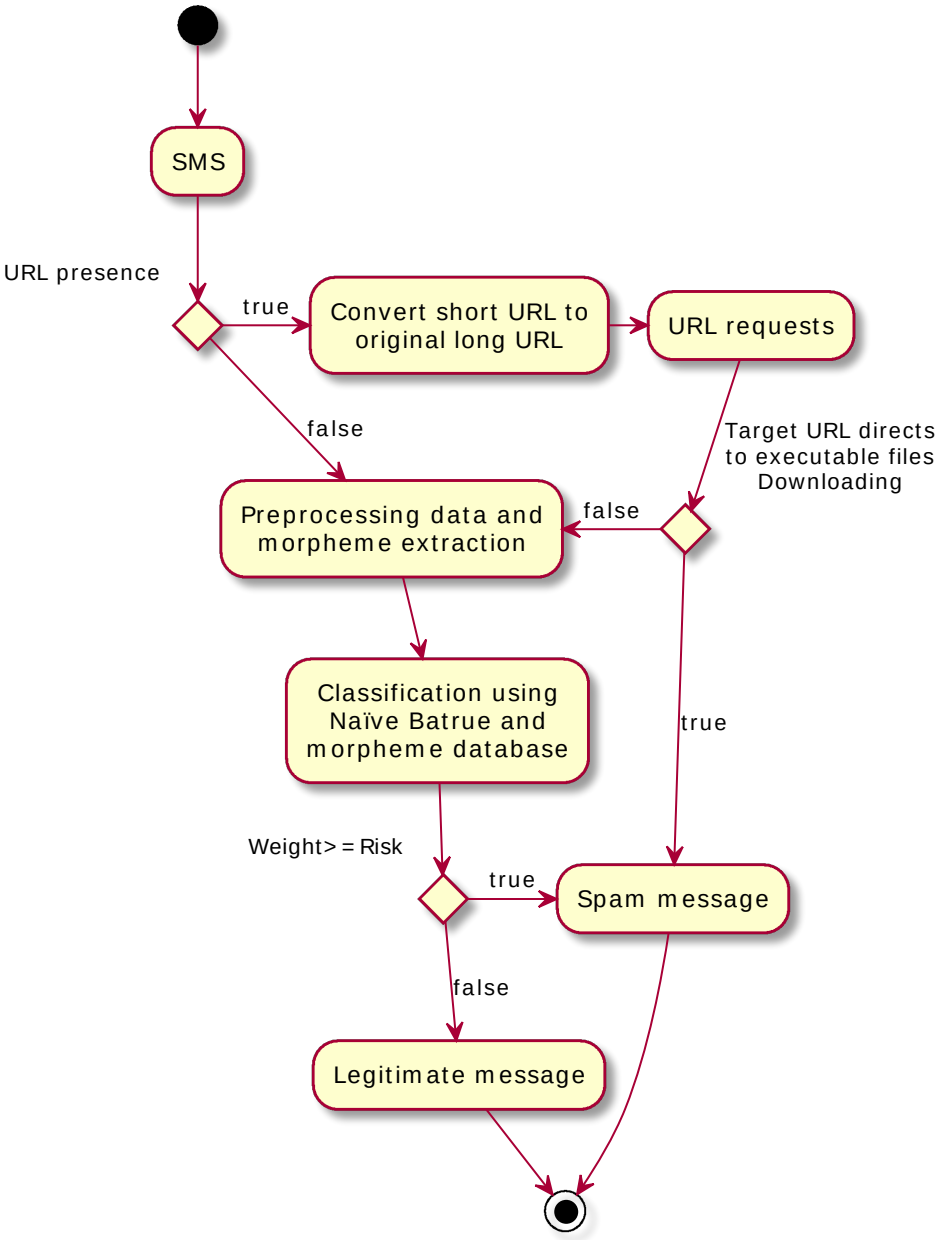

Figure 3.2: Methodology of S-Detector that was proposed in [39]

files. The messages with URLs targeting to downloading of executable files are classified as spam messages.

For the rest of the messages without malicious, S-Detector starts from pre-processing and morpheme extraction. Then, S-Detector uses the Naïve Bayes algorithm to learn from extracted words, and weight is calculated based on the extracted data. When the weight is greater than or equal to risk, the message is classified as spam. Otherwise, it is regarded as a legitimate (ham) message.

3.1.4 SmiDCA

In [69], Sonowal et al. proposed a methodology named "SmiDCA" for SMS smishing messages detection based on machine learning. The main idea of SmiDCA is to find a feature selection that achieves the best performance on a specific machine learning classifier. The methodology of SmiDCA is shown in Figure 3.3.

Firstly, SmiDCA pre-processes data and extracts candidate features from the processed data. Secondly, SmiDCA employs Pearson correlation coefficient (PCC) [4] scores to rank different features from high to low. After that, SmiDCA iterates every candidate features one by one starting from the highest-ranking feature. In each iteration, SmiDCA adds one candidate feature with the highest score currently to the feature selection and trains a machine learning classifier on all selected features. SmiDCA also evaluates the machine learning classifiers for each possible feature selection and stops adding features when the accuracy starts decreasing.

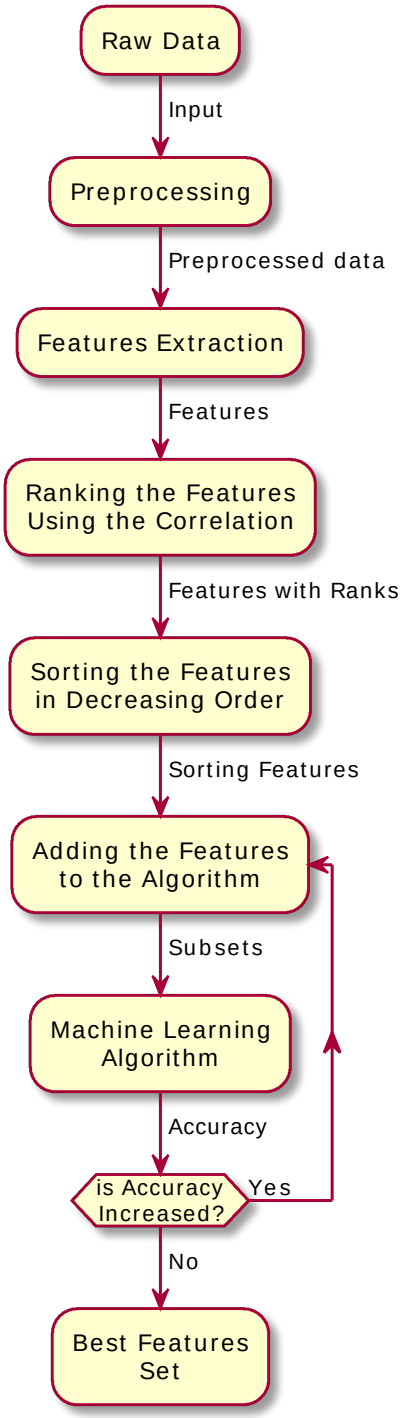

Figure 3.3: Methodology of SmiDCA that was proposed in [69]

For two feature sets $X = x_1, x_2, ..., x_n$ and $Y = y_1, y_2, ..., y_n$, the Pearson correlation coefficient $PCC(X, Y)$ is calculated as follows:

$$PCC(X, Y) = \frac{cov(X, Y)}{\sigma X \sigma Y} \tag{3.1}$$

where $cov(X, Y)$ is actually the covariance of X and Y. The σX and σY denotes the stand deviation of X and Y, respectively. $cov(X, Y)$, σX, and σY are defined by the following euqation:

$$\begin{aligned} cov(X, Y) &= \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \sigma X &= \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} \\ \sigma Y &= \sqrt{\frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n-1}} \end{aligned} \tag{3.2}$$

where $\bar{x}$ and $\bar{y}$ are arithmetic mean of X and Y, respectively, and they are given by the following equation:

$$\begin{aligned} \bar{x} &= \frac{1}{n} \sum_{i=1}^{n} x_i \\ \bar{y} &= \frac{1}{n} \sum_{i=1}^{n} y_i \end{aligned} \tag{3.3}$$

According to [69], SmiDCA extracts 39 different features including word features, message size, E-mail address, URL, phone number, special characters, misspelled words, part of speech, and readability of text. Additionally, Random Forests, Decision Tree, AdaBoost, and Support Vector Machine are used as machine learning classification algorithms. The experiments that were conducted by the author on the SMS Spam Collection v.1 dataset suggested that the feature selection algorithm of SmiDCA reduced 48.72%

of feature dimensions and SmiDCA with Random Forests classifier achieved an accuracy of 96.16% with 20 features.

3.1.5 Smishing Detector

In [52], Mishra et al. proposed the Smishing Detector, an SMS smishing messages detection model based on the message contents and the URLs in the messages. Smishing message, a subset of spam messages, refers to the messages that try to trick people into giving out their private sensitive information. Since Smishing Detector focuses on identifying smishing messages, it is designed based on some assumptions that may not apply to all spam messages. For instance, Smishing Detector only looks at the messages with a URL, a phone number, or an E-mail ID. A message without these items could be spam but should not be classified as smishing messages.

Smishing Detector consists of SMS Content Analyzer, URL Filter, Source Code Analyzer, and APK Download Detector. Similar to S-Detector [39] that was mentioned in Section 3.1.3, Smishing Detector [52] also treats the text content and URLs separately. Specifically, SMS Content Analyzer is responsible to check the textual message contents using a machine learning algorithm to detect smishing messages, while URL Filter, Source Code Analyzer, and APK Download Detector are designed to classify messages by analyzing the URLs in three different aspects.

The flowchart of SMS Content Analyzer is illustrated in Figure 3.4. The major responsibility of SMS Content Analyzer is to analyze the textual content of messages. When a message is passed to SMS Content Analyzer, the analyzer first looks for URLs and sends them to URL Filter. After that, the SMS Content Analyzer checks the existence of phone number and E-mail ID in the message and marks the message containing neither of them as a legitimate message as we discussed previously. Besides, a message with a phone

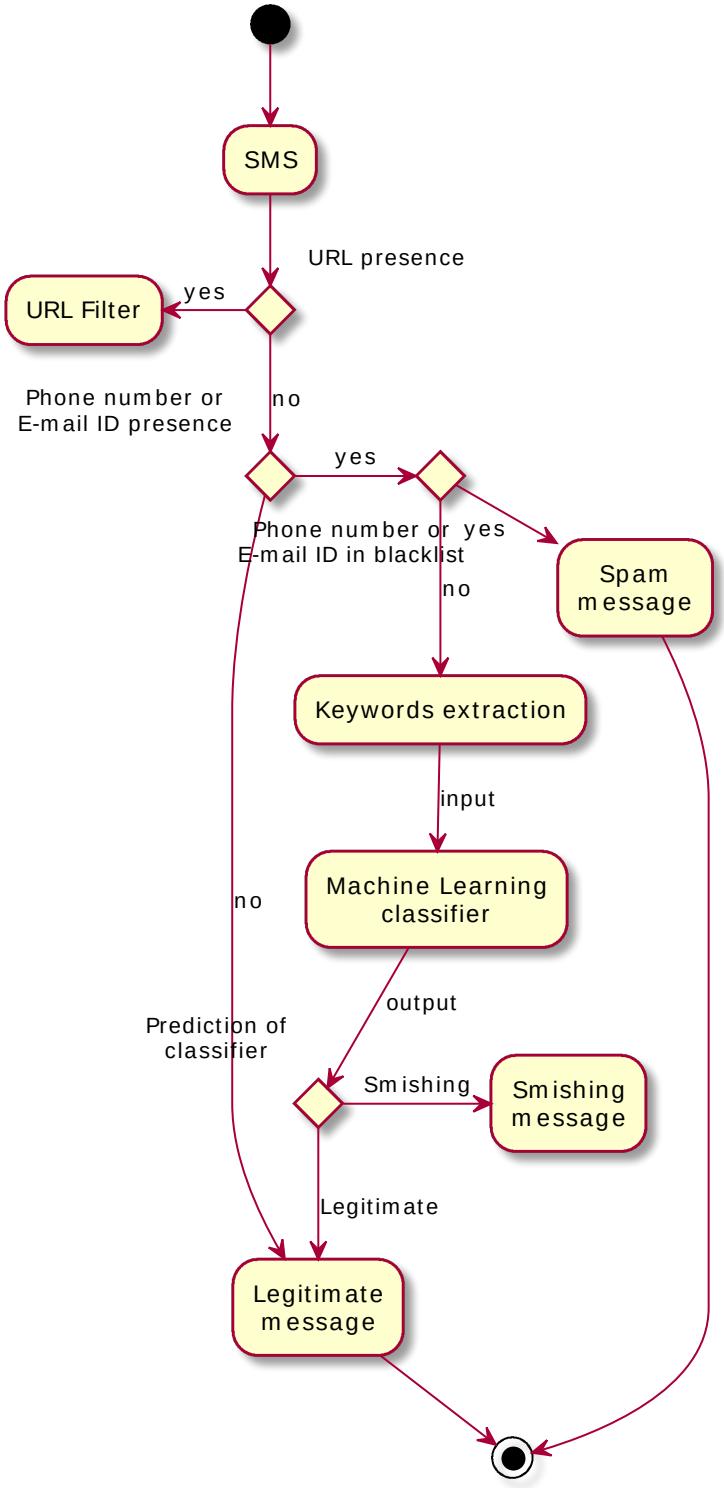

Figure 3.4: Flowchart of SMS Content Analyzer in Smishing Detector proposed in [52]

number or E-mail ID in the blacklist is classified as a smishing message without a doubt. For the cases where there is a phone number or E-mail ID that is not on the blacklist, SMS Content Analyzer extracts the keywords as features and employs a machine learning classifier to decide whether or not the message is legitimate based on extracted features. In [52], Naïve Bayes, Random Forests, and Decision Tree are candidate classification algorithms used in SMS Content Analyzer.

URL Filter analyzes the URLs forwarded from SMS Content Analyzer. When a URL arrives at URL Filter, it first searches the incoming URL in the blacklist. If the URL is on the blacklist, the corresponding message is classified as a smishing message. Otherwise, the following four criteria of the URL are checked:

- Age of Domain
- Presence of @tag
- Presence of hyphen(-)
- Number of the presence of dot(.)

When there are at least three out of four criteria are satisfied or larger than the threshold, the related message is also regarded as a smishing message. Otherwise, the URL is passed to Source Code Analyzer for additional checking.

The major functionality of Source Code Analyzer is to examine the HTML source code of incoming URLs. After retrieving the source code of the incoming URL, the module inspects the source code in the following two aspects:

- Existence of form tags in source code
- Difference between URL and its actual destination URL

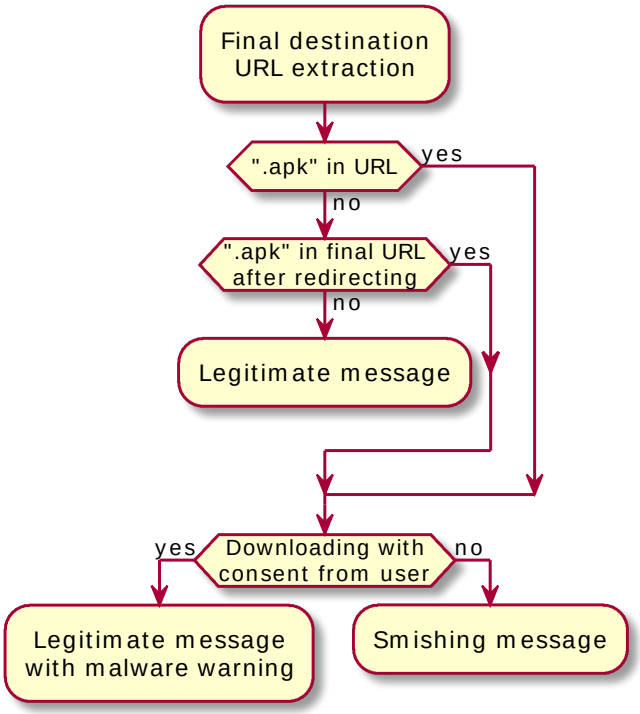

Figure 3.5: Flowchart of APK Download Detector in Smishing Detector proposed in [52]

Source Code Analyzer classifies the associated message as a smishing message when both criteria are true. In another word, if there exist form tags in the source code and the URL is not the same one as its final destination URL after redirecting, the related message is treated as a smishing message immediately. Otherwise, the URL is forwarded to APK Download Detector for more analysis. The author [52] believed that both of these two criteria mentioned before are suspicious and potentially malicious. Therefore, it is reasonable to assert the maliciousness of a message when it contains a URL that satisfies both suspicious aspects.

The task of APK Download Detector is to check whether the given URL contains unauthorized APK downloading attempts. As the flowchart presents in Figure 3.5, APK Download Detector finds whether ".apk" suffix is included in the given URL or in the

final target URL after redirecting. When ".apk" is founded in neither the given URL nor the final destination site of the URL after redirecting, the corresponding message is marked as a legitimate message. Otherwise, the consent of downloading from the user is checked. The message related to the given URL is regarded as a legitimate message with a malware warning if the user's consent is provided. On the other hand, APK Download Detector identifies the corresponding message as a smishing message when the consent of downloading is absent.

For the experiments of Smishing Detector, the authors created a revised smishing SMS dataset based on the SMS Spam Collection v.1 dataset [1] and the SMiShing dataset [16]. Since Smishing Detector focuses only on smishing messages instead of all spam messages, the SMS Spam Collection v.1 dataset not suitablefor the experiments. As is described [52], Mishra et al. manually filtered 254 smishing messages from the SMS Spam Collection v.1 dataset. For the rest of spam messages in the SMS Spam Collection v.1 dataset, although they might be advertising or meaningless messages that were sent with negative intention, they were not the objectives of Smishing Detector, and thus they were regarded as legitimate messages in experiments. Besides, a number of 284 smishing messages from the SMiShing dataset [16] from Pinterest was also included. Thus, the authors had collected a smishing SMS dataset with 538 smishing messages and 5320 legitimate messages. Table 3.1 presents the overview statistics of the dataset used in [52] for Smishing Detector.

According to the experimental results in [52], among Naïve Bayes, Random Forests, and Decision Tree, the best performance is achieved with an accuracy of 96.29% when Naïve Bayes is employed as the machine learning classification algorithm in SMS Content Analyzer. The authors also believe that the involvement of verification for consents from users during APK file downloading reduces the false-positive rate significantly.

Table 3.1: The statistics of the dataset used in [52] for Smishing Detector

Source \ Label in [52]	Smishing messages	Legitimate messages	Total
The spam of SMS Spam Collection v.1 dataset [1]	254	493	747
The ham of SMS Spam Collection v.1 dataset [1]	0	4827	4827
SMiShing dataset [16]	284	0	284
Total	538	5320	5858

3.2 Deep Learning Spam Detection Approaches

While the traditional machine learning techniques do perform well in many fields, they are still much interference or guidance from human specialists required when people try to apply these technologies to address problems. For instance, extracting and representing the features from data is always a challenging but indispensable work for machine learning scientists. In another word, the inadequate capacity of many traditional machine learning classifiers is a major limitation to a more effective and massive application. However, many deep learning techniques are able to not only learn much more amount of features but also extract more higher-level features that are formed by the composition of lower-level features. With an effective training process, the deep learning techniques are more capable to consume and make good use of a large amount of data and thus perform better especially in coping with difficult jobs compared to the traditional machine learning approaches.

3.2.1 CNN Based SMS Spam Detection Approaches

In [31], Gupta et al. compared the performance of 8 different classifiers including Support Vector Machine, Naïve Bayes, Decision Trees, Logistic Regression, Random Forests,

AdaBoost [23], Artificial Neural Network, and Convolutional Neural Network. The experimental tests on the SMS Spam Collection v.1 [1] dataset that was conducted by the authors shows that the CNN and Artificial Neural Network are better compared to other machine learning classifiers, and they achieved an accuracy of 98.25% and 98.00%, respectively.

In [66], Roy et al. aimed to adapt the Convolutional Neural Network and Long Short-Term Memory to the SMS spam messages detection problem. The authors evaluated the performance of CNN and LSTM by comparing them with Naïve Bayes (NB), Random Forests (RF), Gradient Boosting (GB) [24], Logistic Regression (LR), and Stochastic Gradient Descent (SGD) [8] classifier. Additionally, they also tuned the hyper-parameters carefully and tested different multiple model stack structures for LSTM and CNN. The experiments that were conducted by the authors showed that the CNN and LSTM perform significantly better than the traditional machine learning approaches when it comes to SMS spam detection. Concretely, the 3CNN with Dropout and 10-fold cross-validation setting accomplished the best results among all candidate solutions in terms of accuracy, AUC value, precision, recall, and F1-Score.

3.2.2 Semantic Long Short-Term Memory (SLSTM)

In [38], the authors proposed the Semantic Long Short-Term Memory (SLSTM), a variant of LSTM with an additional semantic layer. Figure 3.6 presents the workflow of the semantic layer. The Word2vec [49] [50], the WordNet [51], and the ConceptNet [42] are employed in the semantic layer to create word vectors. The semantic layer first attempts to convert the input word w into word vectors using Word2vec. When there is no mapped vector in Word2vec, the semantic layer tries to search for a similar word $similar(w)$ in WordNet and ConceptNet dictionaries, respectively, and uses the Word2vec vector

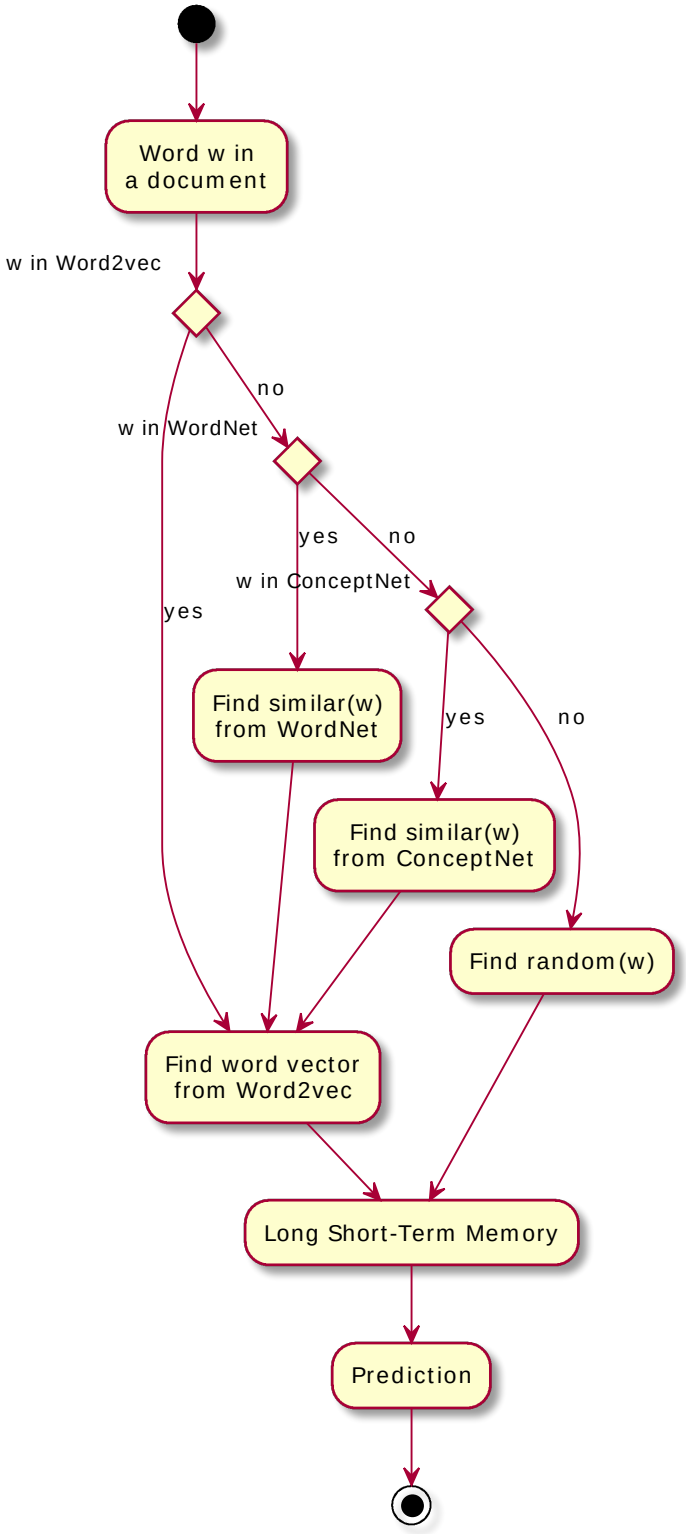

Figure 3.6: Flowchart of the semantic layer of SLSTM [38]

for $similar(w)$ as the word embedding for the input word instead. Lastly, the word embedding from the semantic layer is fed as the input sequence to train a Long Short-Term Memory model for SMS spam detection.

The authors tested the SLSTM with different hyper-parameters values and found an optimal setting on the SMS Spam Collection v.1 dataset. The experimental evaluation that was conducted by the authors claimed that the SLSTM achieved an accuracy of 99% on the SMS Spam Collection v.1 dataset, with the hyper-parameter setting of 100 LSTM units, the input feature size of 6000, 0.1 as Dropout, Sigmoid function as the activation function, and 10 epochs of training. The comparison test on several SMS spam classifiers including SLSTM, k-nearest neighbors algorithm (k-NN) [20], Naïve Bayes (NB), Random Forests (RF), Artificial Neural Network(ANN), and Support Vector Machine (SVM) show that SLSTM outperforms other candidate classifiers regarding precision, recall, accuracy, and F1-Score.

3.2.3 CNN-LSTM

In [26], Ghourabi et al. proposed the CNN-LSTM model that consists of a Convolutional Neural Network (CNN) layer and a Long Short-Term Memory (LSTM) layer in order to identify SMS spam messages in English and Arabic. The methodology of CNN-LSTM is illustrated in Figure 3.7. The main idea of CNN-LSTM is to combine the advantages of CNN and LSTM. Since CNN is very effective in extracting features, CNN-LSTM first feeds the input vectors that are generated from the input sequence into CNN to extract relevant features from the text data. After that, LSTM is employed to learn past information and long-term dependencies. Lastly, a fully connected layer with one single neuron and a sigmoid activation function is applied to the output of LSTM to produce the prediction on whether the message is spam.

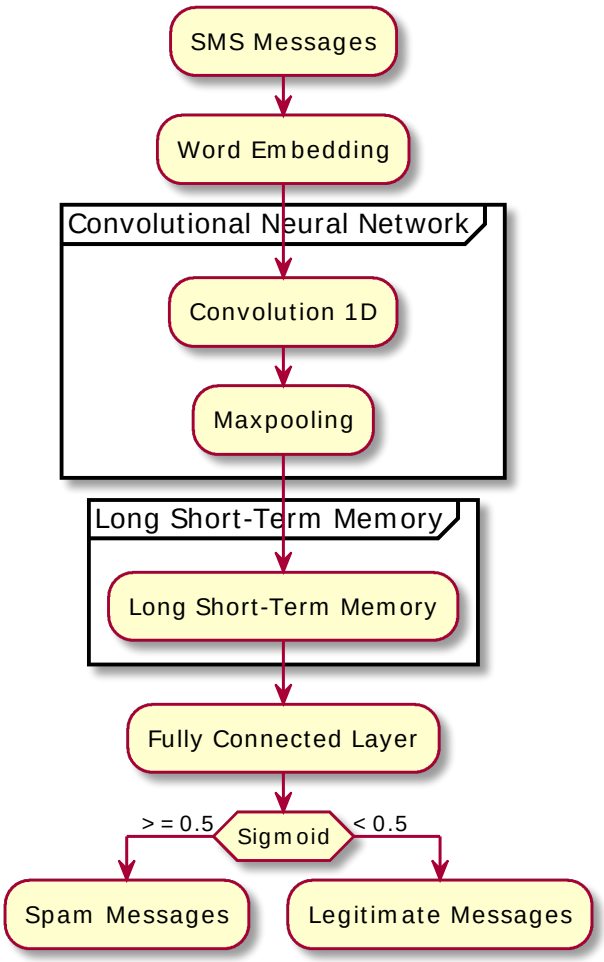

Figure 3.7: Flowchart of CNN-LSTM [26]

The authors evaluated the CNN-LSTM by comparing it with the CNN, LSTM, and 9 traditional machine learning techniques including Support Vector Machine, k-nearest neighbors, Naïve Bayes, Decision Trees, Logistic Regression, Random Forests, AdaBoost, Bagging classifier [9], and Extra Trees (Extremely Randomized Trees) [25]. The experimental tests that were conducted by the authors showed that the CNN-LSTM solution performed better than other approaches in terms of accuracy, recall, F1-Score, and ROC AUC, with an accuracy of 98.3% and an F1-Score of 0.914.

Chapter 4

The Transformer Model and Proposed Modified Transformer for SMS Spam Detection

4.1 The Transformer Model

4.1.1 Introduction

The Transformer [77] model is a sequence-to-sequence (Seq2Seq) model that was proposed in 2017 by Vaswani et al., as an approach to English-German and English-French translation tasks. Compared to those previous Seq2Seq models, the main innovation of Transformer is that it completely relies on the attention mechanism to efficiently learn from the most informative elements [64].

Though LSTM and some other RNN variants were shown to perform well as encoders and decoders in Seq2Seq based models, the high training consumption of recurrent models becomes a significant limitation. At time t, the computation of hidden state h_t relies on

the previous hidden state h_{t-1}, which is the sequential computation nature of recurrent models. This sequential computation nature prevents the computing of RNN variants from parallelization, leading to the limitation on computational efficiency during the training process.

In order to address the computational efficiency limitation of RNN variants, the Transformer uses only multi-head attention mechanism instead of RNN variants as encoders and decoders. This not only greatly reduces the cost of training through parallelization, but also surprisingly improves the performance in translation tasks as is mentioned in [77].

4.1.2 Architecture and Dataflow of Transformer

In Transformer, the source language texts and shifted right target language texts are first sent to embedding layers as input sequence and output sequence, where the texts are converted into numerical embedding vectors of d_{model} dimensions. Secondly, positional information is injected into the input and output sequence in the positional encoding layer, which is discussed thoroughly in Section 4.1.3. After that, the input and output sequence is fed into the encoder stack and the decoder stack, respectively. The specific composition of the encoders and decoders is described in Section 4.1.5 and Section 4.1.6. Similar to other Encoder-Decoder architecture, in Transformer, the output data from the encoders is also passed to the decoders as part of the input. The decoders combine the output (target) sequence and the data from the encoders and perform additional attentional processing. The results of decoders are passed to a linear layer. Finally, the softmax function [6] [28] is performed on the output of the linear layer, producing the translation in the target language.

4.1.3 Positional Encoding

In RNN, the computation of the hidden states is based on the previous states, making it available to learn from the order of words naturally. However, there is no recurrent or convolutional structure in Transformer, making it not possible to understand the relative position of different elements. In another word, up to now, from the perspective of Transformer, there is not much difference between two sequences with the same elements but different order. In reality, the order of words actually matters the meaning of a sentence. For instance, the meaning of English phrases "long before" and "before long" differ from each other. Therefore, in order to enable the Transformer to learn from the positional information of elements, a positional encoding function that injects the positional information into data is introduced.

According to [77], the positional encoding function is based on *sine* and *cosine* functions of different frequencies, which is calculated as:

$$\begin{aligned} PE_{(pos,2i)} &= sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \\ PE_{(pos,2i+1)} &= cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \end{aligned} \tag{4.1}$$

where i is the dimension, pos is the position, and the size of the positional encoding vector is d_{model}. For a given position, the frequencies decrease as the dimension increases. Therefore, the wavelength is a geometric progression from $[2\pi, 4\pi, ..., 10000 \cdot 2\pi]$. The authors of [77] believe that the property of this function allows the model to learn from relative position easily.

4.1.4 Feed-Forward Network

In Transformer, a fully-connected feed-forward network is used in each encoder and decoders as an addition of the multi-head self-attention sub-layer. The fully-connected feed-forward network consists of two separate linear layers with ReLU activation function in the first linear layer, which is reviewed in Section 2.1. For each neuron in the feed-forward network, it is calculated as follows:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{4.2}$$

where W_1, W_2, b_1, and b_2 are weight matrices of two linear layers, and the output dimension of the second layer is d_{model}.

4.1.5 Encoder

Similar to other Encoder-Decoder models, the responsibility of the encoder stack is to generate an attentional sequence that is capable of emphasizing more important elements based on the input sequence. The multi-head self-attention mechanism is employed in encoders to capture the crucial part from the input sequence.

In Transformer, each encoder consists of a multi-head self-attention sub-layer and a position-wise feed-forward sub-layer. Firstly, the multi-head self-attention sub-layer consumes the input sequence and captures the most significant part. After that, the attentional data go through the position-wise feed-forward sub-layer, where the output sequence of the encoder is calculated.

In the multi-head self-attention sub-layer, the input sequence is first sent to a multi-head attention part that has been discussed in Section 2.9.4. Then, the attentional results

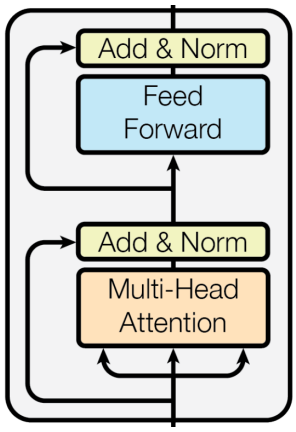

Figure 4.1: The graphical illustration of the Transformer encoder from [77]

are passed to a layer normalization [2], which is defined as follows:

$$output = LayerNorm(x + Sub\text{-}layer(x)) \tag{4.3}$$

where x is the input sequence, and the $Sub\text{-}layer(x)$ is the sub-layer that the layer normalization is plugged into. Specifically, in the context of the multi-head self-attention sub-layer in the encoders, the output is computed as:

$$output = LayerNorm(x + MultiHead(x, x, x)) \tag{4.4}$$

where the $MultiHead()$ denotes the multi-head attention discussed in Section 2.9.4.

The position-wise feed-forward sub-layer consists of a feed-forward network that is explained in Section 4.1.4, and a layer normalization [2] similar to the one described in Equation 4.3. Plugging the Equation 4.2 to the Equation 4.3, the output of position-wise feed-forward sub-layer is generated as:

$$\begin{aligned} output &= LayerNorm(x + FFN(x)) \\ &= LayerNorm(x + max(0, xW_1 + b_1)W_2 + b_2) \end{aligned} \tag{4.5}$$

where x is the input sequence from the previous multi-head self-attention sub-layer, and W_1, W_2, b_1, and b_2 are weight matrices of the feed-forward network.

In Transformer, there could be multiple totally identical encoder layers in the encoder stack to facilitate the attentional results. The outputs of all the sub-layers, as well as the encoders, have dimension d_{model}.

4.1.6 Decoder

The structure of the decoder in the Transformer is illustrated in Figure 4.2. The decoders consume two sequences, the sequence generated by the encoder stack, and the output sequence from the output embedding layer or previous encoders. Each decoder consists of a masked multi-head attention sub-layer, a multi-head attention sub-layer, and a position-wise feed-forward sub-layer. Besides, a layer normalization defined in Equation 4.3 is employed in each of three three sub-layers.

The masked multi-head attention sub-layer of the decoders is responsible for extracting those significant parts from the output sequence that is generated by the output embedding layer or the previous decoder, which is similar to the multi-head self-attention sub-layer of the encoders. However, the process of predicting the i-th element of the output sequence should only rely on the elements before position i. Therefore, compared to the attention sub-layer in the encoders, a masking mechanism is introduced in the attention part of the decoders to prevent the future elements from involving the attention computation of the current element.

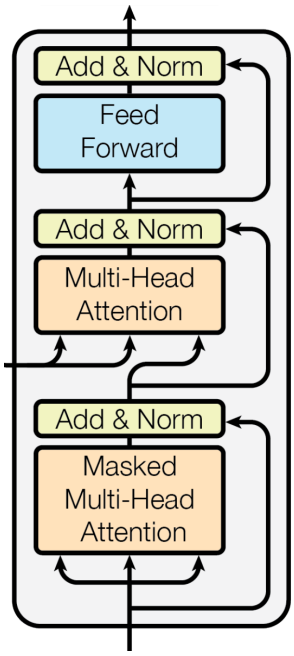

Figure 4.2: The graphical illustration of the Transformer decoder from [77]

Located between the masked attention sub-layer and the position-wise feed-forward sub-layer, the multi-head attention sub-layer combines the results from the masked multi-head attention sub-layer with the output sequence of the encoder stack and computes attention function based on that. Concretely, in the context of the attention function introduced in Equation 2.39 and Equation 2.40, the multi-head attention sub-layer of the decoders regards the encoder output sequence as (K, V) pairs, while the result sequence of masked attention sub-layer is treated as query Q. Therefore, after applying the layer normalization introduced in Equation 4.3, the output of the multi-head attention sub-layer in the decoders is calculated as:

$$output = LayerNorm(x + MultiHead(x, o_e, o_e)) \tag{4.6}$$

where the o_e represents the output sequence of the encoder stack.

Finally, the output sequence of the multi-head attention sub-layer is sent to a feed-forward sub-layer that is identical in structure to the one in the encoders, which is defined in Equation 4.5.

4.2 Proposed Modified Transformer Model for SMS Spam Detection

In Figure 4.3, the main architecture of the modified Transformer model for SMS spam detection is described. In order to apply the Transformer model to the SMS spam detection task, two major modifications are done to the vanilla Transformer model, which are described in Section 4.2.1 and Section 4.2.2, respectively. After that, several implementation details are discussed.

4.2.1 Memory

The first modification for the SMS spam detection task is the introduction of memory. Since there is no output sequence (target sequence) in the SMS spam detection task, we used a list of trainable parameters named "memory" to be the substitute for output sequence embedding. The length of the memory is a configurable hyper-parameter. Each element of the memory is a vector of dimension d_{model} so that it can be adapted to the Transformer model without any extra projection. In other words, the memory is a matrix of dimension $len_{memory} \times d_{model}$. The output embedding layer in the original Transformer model is also removed since there are no target sequence texts anymore to be mapped to numeric vectors. Similar to the output sequence in the vanilla Transformer model, the positional information is injected into the memory at the positional encoding layer before being fed into decoders.

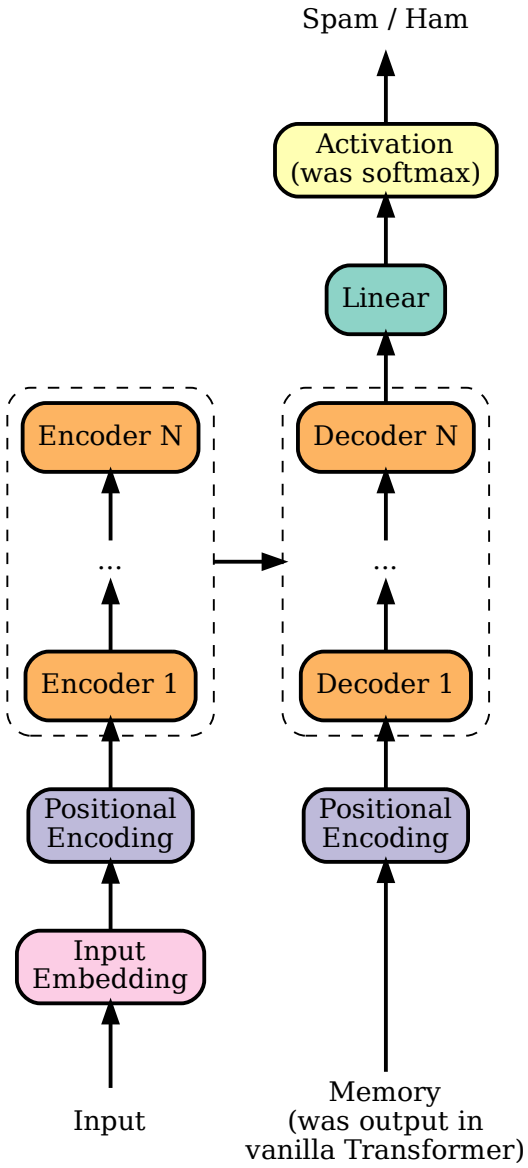

Figure 4.3: Structure of proposed modified Transformer model for SMS spam detection.

During the training process, the parameters of memory are trained, and the memory matrix is expected to contain the important information that can help to predict whether or not a message is spam. Therefore, in the decoders of the modified spam Transformer model, with the help of the attention mechanism, the memory can contribute to locate the significant part of the output sequence of the encoder stack that summarized the message, and eventually help to classify the spam SMS messages.

4.2.2 Linear Layers and Final Activation Function

The second modification is the final activation function. In the vanilla Transformer, the dimension of outputs of decoder layers is $T \times d_{model}$, where T is the target sequence length and d_{model} is the model size (number of features). Therefore, intuitively, it is a promising approach to use the linear layers to map the output to a vector that has the same dimension as the number of words in the dictionary and apply a softmax function [28] [6] on the vector to find the closest candidate word from the dictionary.

However, the SMS spam detection task is a binary classification problem. Therefore, to convert the output from the decoder stacks with dimension d_{model} into a single probability of the message being spam, the linear layers after the decoders are also modified. Instead of mapping the output of the decoder stack to a vector, the linear layer in the modified Transformer model for SMS spam detection has only one single neuron in the last layer. Thus, the outputs of the decoder stack are converted into a single numeric probability value.

Additionally, the final activation function needs to be replaced with a function with a binary outcome. Thus, in the modified Transformer for SMS spam detection, a sigmoid function, which is defined in Equation 2.2, is applied to the output of the linear layers after decoders, generating a binary result of whether or not the message is spam.

4.2.3 Data Pre-Processing

Before being converted to the word embeddings, the textual messages in the dataset are first tokenized. Tokenization refers to the task of splitting textual into meaningful words. Specifically, the SpaCy [36] library is employed for data pre-processing in order to tokenize the data. At the same time, lemmatization is also performed.

4.2.4 Word Embedding

The GloVe [61] is an unsupervised learning algorithm for obtaining vector representations for words. The main methodology is to map words into a meaningful space where the distance between words is related to semantic similarity. The GloVe model produces a vector space with a meaningful substructure, and it can also find the relations like synonyms between words.

We also tested the Word2vec [49] [50] as the word embedding algorithm. The performance is closed to use GloVe. However, the calculation of the GloVe is faster in practice. Therefore, we choose GloVe as our word embedding algorithm.

Today, there are multiple pre-trained GloVe models available. In our spam Transformer model, to construct word embedding from textual input data, we used the "glove.840B.300d", a pre-trained model with 2.2 million words in the dictionary that converts textual data into 300-dimensional vectors.

4.2.5 Dropout

Dropout [33] is a powerful technique published by Hinton et al. in 2012 in order to prevent over-fitting in a large feed-forward neural network. Concretely, the Dropout refers to randomly omit some nodes in those large feed-forward layers on each specific

training case. The modified spam Transformer model that we proposed employs multiple feed-forward layers. Thus, the Dropout technique is also implemented in the feed-forward layers of our spam Transformer model. Besides, the Dropout technique is also used in positional encoding and calculation of attention function.

4.2.6 Batches and Padding

During each epoch of training on our proposed models, the whole training set is divided into multiple batches. As the length of the message with the same batch should be the same, some padding words (empty words) should be added into the shorter message vectors, interfering with the detection to some extent. Therefore, the algorithm of dividing the training set into batches is designed to minimize the padding words. Specifically, the training data is sorted by the message length first, and the batches are created to minimize the padding words based on the sorted messages.

Admittedly, adding padding words may pose a negative influence on the model. However, using batch has been proved to be a good idea for model training as it increases the training speed extraordinarily. In fact, a larger batch size accelerates a ton for the training speed. Additionally, the negative influence of padding words is addressed by minimizing the use of padding words. Besides, the padding masks are also passed into the model along with the training batches so that the Transformer model can ignore the padding words during training.

4.2.7 Optimization and Learning Rate

The gradient descent is employed to optimize our modified spam Transformer model. The main idea of the gradient descent algorithm is to minimize the loss function of the model by updating the parameters along the opposite way of the gradient to the loss function,

where the gradient is the partial derivatives of the loss function of the parameter. There are plenty of variant optimizers of gradient descent. We use the AdamW [43] optimizer for our proposed modified spam Transformer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$. Learning rate is a critical hyper-parameter in machine learning. It is defined as the step size of updating parameters, which basically represents the speed of learning of the model. Having the learning rate set too high will lead to the situation that the model fails to locate the best parameters (weights and biases), while a learning rate that is too small sticks the model around the local optimal point rather than finding a better parameter solution.

For the modified spam Transformer model, the same way of determining the learning as mentioned in [77] is utilized. The calculation of the learning rate proposed by Vaswani et al. in [77] is as:

$$lr = d_{model}^{-0.5} \cdot min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{4.7}$$

where the $step\_num$ is the current training step. The learning rate lr first increases linearly until reaching the $warmup\_steps$ steps and then decreases exponentially.

In our modified spam Transformer model, we used $warmup\_steps = 8000$. Figure 4.4 depicts the learning rate changes. During the experiments, we tested multiple candidate values for the $warmup\_steps$ including 2000, 4000, 8000, and 12000, and found that the 8000 is the optimal $warmup\_steps$ value.

4.2.8 Measures Against Unbalanced Datasets

In the real-life, the data that we faced may not be ideal for our model. For instance, the SMS Spam Collection v.1 dataset [1], which is one of the datasets used in experiments,

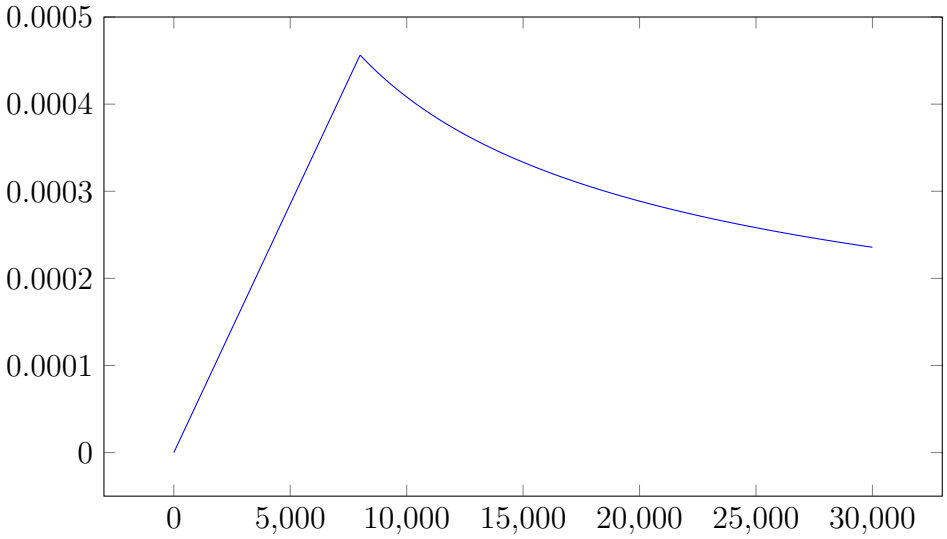

Figure 4.4: Learning rate changes when $warmup\_steps = 8000$ and $d_{model} = 600$

is unbalanced, where the number of ham messages is much larger than the number of spam messages. In order to address the negative influence of the unbalanced dataset, a higher weight is given to the positive data (spam messages).

4.2.9 Early Stopping

To prevent our model from over-fitting, the early stopping technique is applied during training. Firstly, a validation set is created using part of the data. After every epoch of training, a validation loss is computed on the validation set. Since the data in the validation set is different from the training set, and the data are never used for training, the loss should be able to indicate the performance of the model correctly and objectively. When the loss on the validation increases consecutively for 5 epochs, it indicates that the model is threatening by over-fitting. Therefore, the training process is stopped and the best model with the minimum loss so far is selected.

4.2.10 Dataflow of Modified Transformer

As is shown in Figure 4.3, the input messages are first converted into word embeddings using the Glove model. Following this, the memory (trainable parameters) and the embeddings of the input sequence are positionally encoded, respectively. Then, the processed message vectors are passed to encoder layers, where the multi-head self-attention is performed and the important parts of the input sequence are given larger weights. The results of encoder layers are passed to decoder layers. In decoder layers, the multi-head self-attention is computed on the memory. After that, the multi-head attention is executed based on the results of encoder layers and the processed memory. Finally, the decoded vectors are sent to some fully-connected linear layers, followed by a final activation function for classification.

Chapter 5

Experiments and Results

5.1 Datasets

In the experiments, two different datasets are utilized. The first dataset is SMS Spam Collection v.1 [1] dataset, which is labeled SMS messages dataset that has been collected for mobile phone message research. The second one is UtkMl's Twitter Spam Detection Competition (UtkMl's Twitter) [76] from Kaggle. Table 5.1 shows the overview statistics of the two datasets.

Although the Twitter posts are not precisely the same as the SMS messages, they are still in some ways common. For instance, they both have approximately less than 100 words. People tend to use more casual language and abbreviations in both Twitter posts and SMS messages. Therefore, UtkMl's Twitter dataset can also be used to test our model. Besides, we can also analyze the extensibility of our model by comparing the performance of our model on these two datasets.

In comparison with SMS Spam Collection v.1 [1] dataset, UtkMl's Twitter dataset contains more data in both spam and ham classes. Besides, UtkMl's Twitter dataset is

Table 5.1: The statistics of two datasets

	SMS Spam Collection v.1	UtkMl's Twitter
Spam	747	5815
Ham	4827	6153
Total	5574	11968

balanced since the number of spam messages and ham messages are approximately equal. In terms of the language, although they are a lot of casual language and abbreviation used in both datasets, casual language and abbreviation appear more frequently in UtkMl's Twitter dataset. The reason for this observation may be the feature of the Twitter posts. Alternatively, it could also because of the date that the dataset was collected, as SMS Spam Collection v.1 was published in 2011.

5.2 Evaluation Measures

In order to evaluate the performance of the proposed modified spam Transformer model, some metrics such as accuracy, precision, recall, and F1-Score are used in the experiments. All these metrics are calculated based on the confusion matrix. As is mentioned in the previous section, the spam messages in the SMS Spam Collection v.1 dataset are significantly less than the ham messages, which means that the dataset is unbalanced. Therefore, the accuracy is not sufficient as a measurement to evaluate the performance of the proposed model, and the F1-Score is employed in the experiments. The accuracy, precision, recall, and F1-Score are defined as follows:

Table 5.2: The confusion matrix

	Predicted Spam	Predicted Ham
Actual Spam	True Positive (TP)	False Negative (FN)
Actual Ham	False Positive (FP)	True Negative (TN)

$$Accuracy = \frac{TP + TN}{TP + FP + FP + FN} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.4}$$

The precision, also known as the positive predictive value, represents the percentage of the predicted positive cases that are actually positive, meaning the possibility that the classifier is correct given that it predicts positive. The recall, also known as sensitivity, denotes the number of true positives instances divided by the number of actual positive instances, which can also be described as the percentage of the positive cases that are identified successfully. The F1-Score is the harmonic mean of precision and recall, which measures the performance of a classifier in terms of precision and recall in a balanced way.

5.3 Data Splitting

For the traditional machine learning approaches, the data is divided into training set (70%), and test set (30%). For the LSTM and our proposed modified spam Transformer model, the data is split into training set (50%), validation set (20%), and test set (30%),

where the validation set is used after each epoch of training to help us select the best model and perform early stopping to avoid over-fitting. The details of the early stopping has been discussed in Section 4.2.9.

5.4 Data Representations

The TF-IDF representations of the textual messages are calculated and sent to the benchmark machine learning algorithms. For the deep learning approaches such as LSTM and our proposed spam Transformer model, the GloVe [61] model is employed to create representation vectors for them.

The TF-IDF (Term Frequency–Inverse Document Frequency) is a widely-used numerical statistic in NLP. It is designed to reflect the importance of a word to a document in the given text corpus. In the experiments, TF-IDF is used to convert the messages to numeric vectors for traditional machine learning approaches.

The Term Frequency (TF) is defined as the number of times that a term occurs in a document. A larger TF means the term is referred for more times in the given document, showing that the term is more relevant to the document. There are different means to weigh the TF in order to adapt it to different applications. In our experiment, we use the raw count of the term in the document as the TF. The TF of term t_i and document d_j is calculated as follows:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{5.5}$$

where $n_{i,j}$ refers to the number of appearance of term t_i in document d_j, and $\sum_k n_{k,j}$ means the total count of all terms in document d_j.

The Inverse Document Frequency (IDF) is a value to qualify the specificity of a term, which is normally defined as the logarithmically scaled inverse fraction of the number of documents that contain the term. In another word, when a term occurs in a great number of documents, the IDF is numerically low, leading to a low TF-IDF. For instance, the term "the" occurs in almost every English document, leading to a document frequency of almost 1 (100% of the documents in the corpus contain the term "the"). Thus, the IDF of "the" is close to 0, which means that its importance to any documents in the corpus is low. The IDF of term t_i in document d_j is defined as:

$$idf_{i,j} = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|} \tag{5.6}$$

where $|D|$ denotes the total number of documents, and $|\{d_j : t_i \in d_j\}|$ denotes the number of documents that term t_i appears.

For a specific term t_i in document d_j, the TF-IDF measure $tfidf_{i,j}$ is defined as:

$$tfidf_{i,j} = tf_{i,j} * idf_{i.j} \tag{5.7}$$

5.5 Loss Function

The loss function we used for deep learning approaches including LSTM and modified spam Transformer is Binary Cross Entropy function, which is defined as follows:

$$l(x_i, y_i) = -w_i[y_i \cdot logx_i + (1 - y_i) \cdot log(1 - x_i)] \tag{5.8}$$

The weight w_i is the rescaling factor for loss. Since the SMS Spam Collection v.1 is unbalanced, where spam messages are severely less than ham (legitimate) messages, a

larger weight is given to the actual spam messages to counteract the negative effect of the unbalanced dataset. The rescaling weight is calculated based on the ratio between the number of ham messages and spam messages.

5.6 Optimization

In our experiments for deep learning approaches, in order to optimize the models, we employ the gradient descent algorithm, which is reviewed in Section 4.2.7. We choose the Adagrad [15] as the optimizer for the benchmark LSTM approach, and the AdamW [43] for the modified spam Transformer model.

5.7 Model Training

The specifications of the workstation for our model training and experiments is presented in Table 5.3. For the machine learning classifiers, the experiments are performed on the Scikit-learn 0.24.0 [60] environment. For deep learning approaches like LSTM and spam Transformer model, the early stopping technique, which has been discussed in Section 4.2.9, is implemented to fight against the over-fitting. Besides, we also trained and tested the CNN-LSTM SMS spam detection model proposed in [26] on both datasets as a benchmark to evaluate our modified spam Transformer model.

5.8 Hyper-parameters Tuning

In order to tune the models and find the best hyper-parameters set, the Ray Tune [41] library is employed. The Ray Tune is a hyper-parameter tuning extension tool that supports multiple machine learning frameworks. Given a candidate hyper-parameters

Table 5.3: Workstation Specifications

GPU	NVIDIA GeForce RTX 3090
CPU	AMD Ryzen 7 3700X 8-Core 3.60 GHz 4 MB L2 cache
Memory	32 GB DDR4
Storage	2 TB
OS	Ubuntu 20.04 LTS
Libraries	CUDA 11.1 Python 3.8.5 PyTorch 1.7.1 [59] torchtext 0.8.0 spacy 2.3.4 [36] scikit-learn 0.24.0 [60]

Table 5.4: Optimized hyper-parameters for LSTM

Hyper-parameter	SMS Spam Collection v.1	UtkMl's Twitter
LSTM units per layer	100	100
LSTM layers	1	2
LSTM Dropout	0.1	0.1
Linear Dropout	0.4	0.1

set, the Ray Tune can find the optimized hyper-parameters set by training multiple models with different settings and comparing the results automatically.

For the LSTM model, the optimized parameters on both datasets are shown in Table 5.4. For our modified spam Transformer model on SMS Spam Collection v.1, Table 5.5 presents the initial hyper-parameters that we started from and the optimized values when the better result was achieved after tuning. Table 5.6 demonstrates the initial as well as the optimized hyper-parameters of modified spam Transformer on UtkMl's Twitter dataset.

Table 5.5: Initial and optimized hyper-parameters for modified spam Transformer on SMS Spam Collection v.1

Hyper-parameter	Initial	Optimized
Encoder layers	6	6
Decoder layers	6	6
Model size	512	600
Feed-forward size	2048	1200
Attention head size	8	8
Transformer Dropout	0.1	0.01
Linear Dropout	0.1	0.05
Weights for positive samples	1	6.46
Weights for positive samples	1	1

Table 5.6: Initial and optimized hyper-parameters for modified spam Transformer on UtkMl's Twitter

Hyper-parameter	Initial	Optimized
Encoder layers	6	2
Decoder layers	6	4
Model size	600	600
Feed-forward size	1200	1200
Attention head size	8	8
Transformer Dropout	0.01	0.01
Linear Dropout	0.05	0.1
Weights for positive samples	6.46	1
Weights for positive samples	1	1

5.9 Results and Analysis

5.9.1 Evaluation Results

We demonstrate the performance of the modified spam Transformer model by comparing it on two datasets with some other typical spam detection classifiers, including Logistic Regression, K-Nearest Neighbor, Naïve Bayes, Random Forests, Support Vector Machine (classifier), and Long Short-Term Memory. Besides, for the SMS Spam Collection v.1 dataset, we also compare our models with CNN-LSTM approaches in [26], since they aim to solve the same problem on the same dataset with us.

Table 5.7 summarizes the results on SMS Spam Collection v.1 dataset. For accuracy, our modified spam Transformer model achieved the best value of 98.92%. Concerning precision, the best score was from the Random Forests classifier with a value of 1.0, and our proposed spam Transformer got a value of 0.9781. When it comes to recall, the optimal result came from the spam Transformer model with a value of 0.9451, and the same value came from the Naïve Bayes classifier as well. Finally, in terms of F1-Score, our spam Transformer also achieved the best value of 0.9613. The experiment of CNN-LSTM [26] that was conducted by Ghourabi et al. on the same dataset, are also included in Table 5.7. In Table 5.9, we demonstrate the confusion matrix of all the approaches that we tested in the experiments on the SMS Spam Collection v.1 dataset.

Table 5.8 summarizes the results on UtkMl's Twitter dataset. The modified spam Transformer model outperformed all other candidates in all four aspects that we tested with the values of 87.06%, 0.8746, 0.8576, and 0.8660 on accuracy, precision, recall, and F1-Score, respectively. The confusion matrix of the modified spam Transformer model on UtkMl's Twitter is presented in Table 5.10.

Table 5.7: Results obtained on SMS Spam Collection v.1

Classifiers	Accuracy	Precision	Recall	F1-Score
Logistic Regression	98.56%	0.9863	0.9113	0.9473
K-Nearest Neighbor	95.27%	0.8691	0.7848	0.8248
Naïve Bayes	98.38%	0.9411	**0.9451**	0.9431
Random Forests	97.90%	**1.0**	0.8535	0.9209
SVM	98.62%	0.9908	0.9113	0.9494
LSTM	98.56%	0.9570	0.9409	0.9489
CNN-LSTM [26]	97.66%	0.9159	0.9198	0.9178
Spam Transformer	**98.92%**	0.9781	**0.9451**	**0.9613**

Table 5.8: Results obtained on UtkMl's Twitter

Classifiers	Accuracy	Precision	Recall	F1-Score
Logistic Regression	81.51%	0.8441	0.7615	0.8007
K-Nearest Neighbor	63.47%	0.6166	0.6632	0.6391
Naïve Bayes	83.21%	0.8316	0.8221	0.8269
Random Forests	79.28%	0.8449	0.7044	0.7683
SVM	82.68%	0.8681	0.7604	0.8107
LSTM	81.04%	0.8594	0.7307	0.7898
CNN-LSTM [26]	79.45%	0.8182	0.7438	0.7792
Spam Transformer	**87.06%**	**0.8746**	**0.8576**	**0.8660**

5.9.2 Analysis

Although the experimental results show an improved performance of the proposed spam Transformer model compared to other candidates, the false predictions also indicate the drawback of the proposed model. We analyzed the content of the false prediction samples including false positive and false negative samples and found that there were a great number of the UNK marks in the data passed to the model, which is produced because the words are never seen in the training data. In other words, the unknown words obstruct the model from understanding the messages. Besides, the SMS messages are usually short, which increases the influence of every single word and makes the unknown words more influential. Actually, due to the unknown words, the model did not have enough information to detect spams in many false prediction cases.

Though our proposed model performs better than other candidate algorithms on UtkMl's Twitter dataset, the results are still not as good as that in case of SMS Spam Collection v.1 dataset. From our observation, the major cause is also the unknown words. Compared to SMS Spam Collection v.1 dataset, there are more casual language and abbreviations in UtkMl's Twitter dataset, which may be caused by the feature of Twitter posts or the date of collection of the dataset, as is discussed in Section 5.1. Therefore, the negative influence from casual language and abbreviation is more severe on UtkMl's Twitter dataset, and that is the major cause of more unknown words and eventually worse performance from our perspective.

Besides, the Random Forests classifier achieved the best precision with a value of 1.0 on SMS Spam Collection v.1 dataset. We believe that it is because the dataset is small while the depth of the decision trees in the Random Forests is relatively high. In these cases, the Random Forests classifiers would not make false-positive predictions since the decision trees over-fit or have a bias toward the positive cases.

As is presented in Table 5.9, our proposed model works well on the unbalanced SMS Spam Collection v.1 dataset without significant bias with the assistance of the measures against unbalanced data that has been discussed in Section 4.2.8. Moreover, Table 5.9 and Table 5.10 show the excellent robustness of our model to classify both the spams and hams effectively, regardless of the datasets being balanced or unbalanced.

Table 5.9: The confusion matrices on SMS Spam Collection v.1

	Act. / Pred.	**Spam**	**Ham**
Logistic Regression	**Spam**	216	3
	Ham	21	1433
K-Nearest Neighbor	**Spam**	186	51
	Ham	28	1408
Naïve Bayes	**Spam**	224	14
	Ham	13	1422
Random Forests	**Spam**	204	0
	Ham	35	1434
Support Vector Machine	**Spam**	216	2
	Ham	21	1434
Long Short-Term Memory	**Spam**	211	17
	Ham	26	1419
CNN-LSTM	**Spam**	218	20
	Ham	19	1416
Spam Transformer	**Spam**	224	5
	Ham	10	1431

Table 5.10: The confusion matrices on UtkMl's Twitter

	Act. / Pred.	**Spam**	**Ham**
Logistic Regression	**Spam**	1332	246
	Ham	417	1592
K-Nearest Neighbor	**Spam**	1160	589
	Ham	721	1117
Naïve Bayes	**Spam**	1438	291
	Ham	311	1547
Random Forests	**Spam**	1232	226
	Ham	517	1612
Support Vector Machine	**Spam**	1330	202
	Ham	419	1636
Long Short-Term Memory	**Spam**	1278	209
	Ham	471	1629
CNN-LSTM	**Spam**	1301	289
	Ham	448	1549
Spam Transformer	**Spam**	1500	215
	Ham	249	1623

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed a modified Transformer model that aims to identify SMS spam based on the Transformer [77] model. We evaluated our spam Transformer model by comparing it with several other SMS spam detection approaches on the SMS Spam Collection v.1 dataset [1] and UtkMl's Twitter dataset [76]. The experimental results show that, compared to Logistic Regression, K-Nearest Neighbor, Naïve Bayes, Random Forests, Support Vector Machine, Long Short-Term Memory, and CNN-LSTM [26], our proposed spam Transformer model performs better on both datasets.

On the SMS Spam Collection v.1 dataset [1], our spam Transformer has a better performance in terms of accuracy, recall, and F1-Score compared to other classifiers. Specifically, our modified spam Transformer approach accomplished an exceeding result on F1-Score.

Additionally, on the UtkMl's Twitter dataset [76], the experimental results acquired from our modified spam Transformer model demonstrate its improved performance on

all four aspects in comparison to other alternative approaches mentioned in this thesis. Concretely, our spam Transformer does exceptionally well on recall, which contributes to a distinct F1-Score.

6.2 Future Work

Although the experimental results in Chapter 5 of this thesis have shown an improvement of our proposed spam Transformer model in comparison with some previous approaches on SMS spam detection, we still believe that there is great potential in the model we proposed.

Firstly, since our current two datasets contain only thousands of messages, in the future, we plan to extend our spam Transformer model to a larger dataset with more messages or even other types of content, for the purpose of better performance.

Besides, in our proposed model, we flattened the outputs from decoders and applied linear fully-connected layers before applying the final activation function and getting the prediction. We believe that some dedicated designs or implementations instead of simple flattening and linear layers could absolutely boost the performance, which would be one of the most important future works.

Finally, as is discussed in Section 5.9.2, the proposed model is severely influenced by the unknown words in many cases of false prediction. To address this problem, more data pre-processing techniques could be applied. For instance, a larger vocabulary with more words could be a good option, and some semantic operations such as replacing unknown words with their synonyms could also be explored. Besides, there are some other data-preprocessing and feature extraction techniques that could be done, such as the extraction and analysis of the abbreviation, URLs, tags, or emoji in data.

References

[1] Tiago A Almeida, José María Gómez Hidalgo, and Akebo Yamakami. Contributions to the Study of SMS Spam Filtering: New Collection and Results. In *Proceedings of the 11th ACM Symposium on Document Engineering*, pages 259–262, 2011.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. jul 2016. URL http://arxiv.org/abs/1607.06450.

[3] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[4] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson Correlation Coefficient. In *Noise Reduction in Speech Processing*, volume 2, pages 1–4. 2009.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[6] Christopher Bishop. The Exponential Family. In *Pattern Recognition and Machine Learning*, chapter 2.4., pages 113–116. Springer-Verlag New York, 2006.

[7] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. Training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[8] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of COMPSTAT*, pages 177–186. 2010.

[9] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[10] Canada's Anti-Spam Legislation. Recognizing and reporting spam. Technical report, apr 2019. URL https://fightspam.gc.ca/eic/site/030.nsf/eng/00003.html.

[11] Jadzia Cendrowska. PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014.

[13] William W. Cohen. Fast Effective Rule Induction. In *Machine Learning Proceedings*, pages 115–123. 1995.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[15] John Duchi and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization * Elad Hazan. Technical Report 61, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

[16] Sec Edu, Jeremy Lee, and LarryITSec. SMiShing dataset, 2018. URL https://in.pinterest.com/seceduau/smishing-dataset/?lp=true.

[17] Jianli Feng and Shengnan Lu. Performance Analysis of Various Activation Functions in Artificial Neural Networks. In *Journal of Physics: Conference Series*, volume 1237. International Journal of Artificial Intelligence And Expert Systems (IJAE), 2019. doi: 10.1088/1742-6596/1237/2/022030.

[18] Andy Field. Logistic regression Logistic regression Logistic regression. *Discovering Statistics Using SPSS*, pages 731–735, 2012. doi: 10.1007/978-1-4419-1742-3.

[19] Finances Online. 10 Mobile Marketing Trends for 2020 / 2021 : Current Predictions You Should Be Thinking About, 2020. URL https://financesonline.com/mobile-marketing-trends/.

[20] Evelyn Fix and Joseph L. Hodges. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. Technical report, 1951. URL https://apps.dtic.mil/dtic/tr/fulltext/u2/a800276.pdf.

[21] Eibe Frank, Mark A Hall, Ian H Witten, and Morgan Kaufmann. The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques". Technical report, 2016.

[22] Yoav Freund and Mason Llew. The alternating decision tree learning algorithm. *International Conference on Machine Learning*, 99:124–133, 1999.

[23] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[24] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[25] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[26] Abdallah Ghourabi, Mahmood A. Mahmood, and Qusay M. Alzubi. A hybrid CNN-LSTM model for SMS spam detection in arabic and english messages. *Future Internet*, 12(9):156, 2020.

[27] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Softmax Units for Multinoulli Output Distributions. In *Deep Learning*, pages 180–184. 2016. URL http://www.deeplearningbook.org.

[29] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, 2014. URL http://arxiv.org/abs/1406.2661.

[30] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. KNN model-based approach in classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2888: 986–996, 2003.

[31] Mehul Gupta, Aditya Bakliwal, Shubhangi Agarwal, and Pulkit Mehndiratta. A Comparative Study of Spam SMS Detection Using Machine Learning Classifiers. In *11th International Conference on Contemporary Computing*, 2018.

[32] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[33] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. 2012. URL http://arxiv.org/abs/1207.0580.

[34] Tin Kam Ho. Random decision forests. In *Proceedings of the International Conference on Document Analysis and Recognition*, volume 1, pages 278–282, 1995.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[36] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020. URL https://doi.org/10.5281/zenodo.1212303.

[37] Ankit Kumar Jain and B. B. Gupta. Rule-Based Framework for Detection of Smishing Messages in Mobile Environment. In *Procedia Computer Science*, volume 125, pages 617–623, 2018.

[38] Gauri Jain, Manisha Sharma, and Basant Agarwal. Optimizing semantic LSTM for spam detection. *International Journal of Information Technology*, 11(2):239–250, 2019.

[39] Jae Woong Joo, Seo Yeon Moon, Saurabh Singh, and Jong Hyuk Park. S-Detector: an enhanced security model for detecting Smishing attack for mobile computing. *Telecommunication Systems*, 66(1):29–38, 2017.

[40] Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. A Clockwork RNN. *31st International Conference on Machine Learning, ICML 2014*, 5:3881–3889, 2014.

[41] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*, 2018.

[42] Hugo Liu and Push Singh. ConceptNet - a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, 2004.

[43] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv*, 2017. URL http://arxiv.org/abs/1711.05101.

[44] Haoye Lu, Haolong Zhang, and Amit Nayak. A Deep Neural Network for Audio Classification with a Classifier Attention Mechanism. *arXiv*, jun 2020. URL http://arxiv.org/abs/2006.09815.

[45] Minh Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[46] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the 30 th International Conference on Machine Learning*, 2013.

[47] Adrienne Matei. Shock! Horror! Do you know how much time you spend on your phone?, aug 2019. URL https://www.theguardian.com/lifeandstyle/2019/aug/21/cellphone-screen-time-average-habits.

[48] Kuruvilla Mathew and Biju Issac. Intelligent spam classification for mobile text message. In *Proceedings of the International Conference on Computer Science and Network Technology*, volume 1, pages 101–105, 2011.

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. Technical report. URL http://ronan.collobert.com/senna/.

[50] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems*, 2013.

[51] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[52] Sandhya Mishra and Devpriya Soni. Smishing Detector: A security model to detect smishing through SMS content analysis and URL behavior analysis. *Future Generation Computer Systems*, 108:803–815, 2020.

[53] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. An introduction to decision tree modeling. *Journal of Chemometrics*, 18 (6):275–285, 2004.

[54] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[55] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018. URL http://arxiv.org/abs/1811.03378.

[56] S O'Dea. Forecast Number of Mobile Users Worldwide from 2020 to 2024 (in billions), 2020. URL https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/.

[57] M. Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[58] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. *30th International Conference on Machine Learning, ICML 2013*, (PART 3):2347–2355, 2012.

[59] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[60] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[61] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[62] Rohit Prabhavalkar, Kanishka Rao, Tara N Sainath, Bo Li, Leif Johnson, and

Navdeep Jaitly. A Comparison of Sequence-to-Sequence Models for Speech Recognition. In *Proc. Interspeech*, 2017.

[63] Daryl Pregibon. Logistic Regression Diagnostics. *The Annals of Statistics*, 9(4): 705–724, 1981.

[64] Eduardo Souza Dos Reis, Cristiano André Da Costa, Diórgenes Eugênio Da Silveira, Rodrigo Simon Bavaresco, Rodrigo Da Rosa Righi, Jorge Luis Victória Barbosa, Rodolfo Stoffel Antunes, Márcio Miguel Gomes, and Gustavo Federizzi. Transformers aftermath. *Communications of the ACM*, 64(4):154–163, 2021.

[65] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, and Phil Blunsom. Reasoning about Entailment with Neural Attention. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2015.

[66] Pradeep Kumar Roy, Jyoti Prakash Singh, and Snehasish Banerjee. Deep learning to filter SMS Spam. *Future Generation Computer Systems*, 102:524–533, 2020.

[67] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[68] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04(12):310–316, 2020.

[69] Gunikhan Sonowal and K S Kuppusamy. SmiDCA: An Anti-Smishing Model with Machine Learning Approach. *The Computer Journal*, 61(8):1143–1157, 2018.

[70] Statista. Frequency of text messages on a mobile phone 2018 —

UK Survey, 2019. URL https://www.statista.com/statistics/387291/mobile-phone-use-by-frequency-of-texting-in-the-uk/.

[71] Statista. Number of mobile (cellular) subscriptions worldwide from 1993 to 2019, 2020. URL https://www.statista.com/statistics/262950/global-mobile-subscriptions-since-1993/.

[72] Statista. Average monthly phone spam in the United States 2015-2020, jun 2021. URL https://www.statista.com/statistics/1050050/average-monthly-phone-spam-in-the-united-states/.

[73] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems*, 4(January): 3104–3112, 2014.

[74] Zhihong Tian, Wei Shi, Zhiyuan Tan, Jing Qiu, Yanbin Sun, Feng Jiang, and Yan Liu. Deep Learning and Dempster-Shafer Theory Based Insider Threat Detection. *Mobile Networks and Applications*, 2020. URL https://doi.org/10.1007/s11036-020-01656-7.

[75] United State of America Federal Trade Commision. How to Recognize and Report Spam Text Messages, feb 2020. URL https://www.consumer.ftc.gov/articles/how-recognize-and-report-spam-text-messages{#}spamhttps://www.consumer.ftc.gov/articles/how-recognize-and-report-spam-text-messages.

[76] UtkMl. UtkMl's Twitter Spam Detection Competition — Kaggle. URL https://www.kaggle.com/c/utkml-twitter-spam/data.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,

Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5999–6009, 2017.

[78] Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. Sequence to Sequence-Video to Text. Technical report, 2015. URL https://github.com/vsubhashini/caffe/.

[79] Jin Xu, Zishan Li, Bowen Du, Miaomiao Zhang, and Jing Liu. Reluplex made more practical: Leaky ReLU. In *Proceedings - IEEE Symposium on Computers and Communications*, volume 2020-July, 2020.

[80] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 2048–2057. International Machine Learning Society (IMLS), 2015.

[81] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention Generative Adversarial Networks. *36th International Conference on Machine Learning, ICML 2019*, 2018. URL http://arxiv.org/abs/1805.08318.

[82] J Zhang and I Mani. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*, 2003.

[83] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM Neural Network for Text Classification. 2015. URL http://arxiv.org/abs/1511.08630.