Project No: A1111-201

Improving Spam Detection on Twitter using Deep Learning

Submitted by: Ng Yi Rong

Matriculation Number: U1820156H

Supervisor: A/P Ponnuthurai Naga Ratnam Suganthan

Co-Supervisor: Shi Qiushi

School of Electrical & Electronics Engineering

A final year project report presented to the Nanyang Technological University
in partial fulfilment of the requirements of the degree of
Bachelor of Engineering

2020

Table of Contents

Abstract

The advancement of technology in a modern era has allowed Internet users to access social media easily. However, the number of content polluters also known as spammers have increased rapidly over the years. Spammers attract Internet users' attention by broadcasting unsolicited content repetitively on social media platforms. Their actions have caused negative social experience for legitimate Internet users. As a result, spam detection models are required to deter social media spammers. The goal of spam detection is to automatically classify content such as tweets into spam or non-spam. Past studies have shown that the success of spam detection models was built by numerous types of machine learning and deep learning methods.

In this project, deep learning models such as LSTM, CNN, and Transformer were experimented on publicly available Twitter dataset. Strategic text processing techniques were performed on original dataset to create 3 modified datasets for experiment. Word embedding techniques such as Word2Vec model, pre-trained GloVe vectors, and random embedding weight initialisation were evaluated. Lastly, classification performances of LSTM, CNN, and Transformer were compared with related works. Experimental results have showed that LSTM with random embedding weight initialisation achieved the best spam precision and specificity scores of 80% and 87%, respectively. Furthermore, my LSTM experimental results have shown comparable performance to other related works.

Acknowledgement

First and foremost, I would like to express my gratitude to my Final Year Project supervisor, Associate Professor Ponnuthurai Naga Ratnam Suganthan for providing me the opportunity and guidance to work on this project. I am also grateful to his PhD student, Mr Shi Qiushi, for his continuous support and encouragement to keep me motivated and work towards my project goals. I would also like to thank Senior Assistant Manager of EEE Academic Programme Office, Mr Yeo Sung Kheng, for giving me the permission to access the GPU in Machine Learning & Data Analytics lab. Without his authorization, I would not be able to complete neural network training efficiently. Lastly, I would like to express my utmost gratitude to my family and friends who have supported me during my entire course of undergraduate study.

Acronyms

API	Application Programming Interface
BPTT	Backpropagation Through Time
CBOW	Continuous Bag-of-Words
CNN	Convolutional Neural Network
DT	C4.5 Decision Tree
EDA	Exploratory Data Analysis
FN	False Negative
FP	False Positive
GloVe	Global Vectors for Word Representation
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbour
LR	Logistic Regression
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
NB	Naïve Bayes
NLP	Natural Language Processing
NLTK	Natural Language Tool Kit
OSN	Online Social Network
POS	Parts-of-Speech
Regex	Regular Expression
ReLU	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Network
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
URL	Uniform Resource Locator

List of Figures

List of Tables

Chapter 1: Introduction

Online Social Networks (OSN) have become an essential platform for Internet users to communicate and broadcast information. Twitter [1] is an example of popular social media platform. Due to its high traffic volume, there was close to 10 million spam accounts reported in 2018 [2]. Spammers broadcast false information and unsolicited content to multiple Twitter accounts in a short duration. These spam content contain trending words to attract users' attention. Moreover, shortened URLs are included to lead victims to malicious websites which may infect their machines with malware [3]. These negative outcomes can cause a sharp increase in crime rate if not dealt seriously. Thus, a robust spam detection system is required to tackle this issue.

1.1 Motivations

Spam detection on social media is not a new research topic. Twitter is commonly studied by researchers because it is one of the platforms that has high traffic and potential attacks by spammers [4]. Researchers collect data using Twitter API to create a dataset for analysis.

Twitter datasets contain user profiles and tweet content related features. Several studies [4], [5] employed the combination of these features to design spam detection models based on supervised machine learning methods. Both studies showed that machine learning models such as SVM [6] and RF [7] could achieve good classification results using user-based and content-based features. However, these spam detection features are not future-proof. Spammers' techniques are continuously evolving which enables them to mimic legitimate users' social networking behaviour and avoid being detected.

In contrast, past studies [8], [9] showed that tweet content was processed by NLP [10] techniques to create text-based features. The combination of text-based, user-based, and content-based features could enhance the performance of machine learning models to detect spams. However, the classification performance on text-based features alone was still relatively poor compared to user-based and content-based features [8], [9]. This could be due to the unsuitability of text-processing method on text data.

1.2 Objective and Scope

The main objective of this project is to perform spam detection on tweet using deep learning models. Furthermore, different text processing techniques will be explored to enhance the performance of deep learning models in spam detection.

The scope of this project is to evaluate the spam detection performance on Twitter public dataset using deep learning models. Data cleaning process will be carried out to remove non-English tweets. Next, data pre-processing step will be performed to extract word features and convert words into dense vectors for machine computations. Different word embeddings techniques will be studied and experimented. Multiple deep learning models such as CNN [11], RNN [12], LSTM [13], and Transformer [14] will be studied to perform spam classification on tweets. Hyper-parameters tuning is performed on deep learning models to improve classification results. Classification performance of deep learning models will be compared with state-of-the-art machine learning model such as SVM, RF classifier, and other related works. Different text features processing techniques will be explored and tested on deep learning models.

1.3 Organization of Reports

This report is organized into 6 chapters which contain sub-sections for comfortable reading. Snippets of each chapter are briefly described as follows:

Chapter 1: Introduction

This chapter covers the motivations, objective, and scope of this project. This allows the readers to understand the reason for choosing this project and the actions that will be performed to achieve my project goal.

Chapter 2: Literature Review

This chapter covers past works related to Twitter datasets and sophisticated deep learning techniques on text classification tasks.

Chapter 3: Background Research

This chapter covers the background research needed for this project in areas such as Twitter dataset, text processing techniques, deep learning model architecture, and model training.

Chapter 4: Proposed Approach

This chapter covers the resources used in this project and step-by-step procedure from data pre-processing to selection of word embedding techniques and neural networks.

Chapter 5: Experimental Results

This chapter covers neural network model setup, types of deep learning models, and the evaluation metrics used in experiment. Additionally, experimental results are presented in figures and tables format for ease of visualization.

Chapter 6: Conclusion and Future Work

This chapter covers the entire project outcomes and possible future work to improve on Twitter spam detection.

Chapter 2: Literature Review

2.1 Spam Detection on Twitter

In early implementation of spam detection model, statistical data such as user-based and content-based features were extracted from Twitter user profiles and tweets, respectively. Examples of user-based features are *number of followings, number of followers, length of username, number of tweets posted* etc. Examples of content-based features are *number of words, number of URLs, number of mentions (@username)* etc. Traditional machine learning methods such as NB [15], KNN [16], SVM, DT [17], LR [18] and RF were experimented on these features to create Twitter spam detection model.

Benevenuto et al. [5] utilized Twitter API to crawl tweets which contain trending hashtags back in year 2009. Tweets were collected and manually labelled into spam and non-spam to create a labelled dataset for machine learning. SVM, a supervised machine learning model, was used in the experiment to learn user-based and content-based features. Wang et al. [4] employed several machine learning methods to learn user-based, content-based, word n-grams, and sentiment features. The combination of user-based and content-based features achieved best f-score using RF classifier. However, RF classifier did not achieve good f-score on word n-grams (text-based) feature alone. Ashour et al. [8] employed n-grams models to create character n-grams feature. N-gram models use a fixed window size to extract local context of a character based on its surrounding characters. Several machine learning methods such as SVM, LR, and RF classifiers were experimented to learn n-grams features. The best method was achieved by LR classifier, which has a slight improvement in f-score compared to [4].

In recent years, deep learning has gained popularity in spam detection. Deep learning models incorporate word embedding techniques to convert words into numerical vectors based on semantic word similarities [19]. Most commonly used word embedding techniques are Word2Vec [19] model and pre-trained word vectors, GloVe [20]. Both of these word embedding techniques will be further discussed in Chapter 3.

Deep learning method enables models to learn text-based feature in the form of word vectors and raise their effectiveness in spam detection.

Ban et al. [3] employed Word2Vec [19], and Bi-directional LSTM network to extract features. Bi-directional LSTM consists of two LSTMs to learn past and future context of each sequence data (tweets). Word2Vec model was used to convert textual tweets into high dimensional word vectors called Word2Vec features. Likewise, Bi-directional LSTM network was used to learn tokenized textual tweets and create deep-learnt features. Spam detection results based on the two feature sets were compared using state-of-the-art machine learning algorithms such as NB, KNN, SVM, DT, and RF. Similarly, Martinelli et al. [21] employed Word2Vec model by using CBOW [19] approach to learn efficient word embedding representation. On the contrary to [3], word vectors were loaded to MLP [22] classifier to perform text classification. The best result was obtained by MLP with 3 hidden layers. On the contrary, Wei et al. [23] adopted GloVe to map vocabulary words found in the dataset to higher dimensional matrix (up to 300d) using its pre-trained word vectors. Instead of using random weight initialisation, this high dimensional matrix was assigned as weights in the embedding layer of Bi-directional LSTM network.

The combination of user-based, content-based, and text-based features were also experimented in deep learning models. Founta et al. [24] employed two neural networks to learn metadata (user-based and content-based features) and text data. Metadata classifier comprises multiple dense layers, whereas text classifier used RNN with attention layer to focus on specific parts of the text data. GloVe [20] word vectors were used as pre-trained embedding weights in the neural network to reduce model training time. The outputs of these two networks were merged and passed to classification layer.

Similar to [24], Alom et al. [9] implemented a dual network to learn user-based, content-based, and text-based features. Text-based classifier was built by Word2Vec algorithm to transform tweets into higher dimensional vectors. This was followed by CNN model to learn the high dimensional matrix. Meta-data classifier utilized user-based and content-based features as inputs to second network. These features were normalized before passing to a five-layered dense network. The outputs of these two classifiers were merged and passed to classification layer.

2.2 Deep Learning on Related Works

Based on the mentioned related works, it can be inferred that pre-trained word embeddings paired with CNN and LSTM networks has strong effectiveness in spam detection. Apart from Twitter spam detection, these approaches were experimented in other applications such as sentiment analysis, sentence classification, and email spam detection.

Archchitha et al. [25] employed a CNN model with three parallel convolution layers with different filter sizes for opinion spam detection in online reviews. Since online reviews are usually longer than tweets, review text sequences were truncated to a maximum length of 1000 words. Word tokens were mapped to high dimensional vectors using pre-trained GloVe word vectors before passing to CNN model as inputs. Wang et al. [26] adopted a method which uses Word2Vec skip-gram model and LSTM to create a sentiment classification model. In the experiment, LSTM outperformed NB and ELM [27] in short text classification. In addition, classification performance of LSTM improves with the increase in training data size.

Ding et al. [28] employed a Densely Connected Bi-directional LSTM (DC-Bi-LSTM) for sentence classification on movie reviews. The key differences between this approach and [3] are GloVe was used to map word tokens to word vectors and average pooling module was included at the output of last LSTM layer. Zhou et al. [29] employed a Convolution LSTM (C-LSTM) approach for short text classification task. This approach utilized Word2Vec to learn vector representations of the words. In addition, these word vectors were passed to convolution layers to create multiple feature maps of higher-level features. Feature maps were passed to LSTM layers and the final time step was used as input to the classification layer.

Kumar et al. [30] employed LSTM network for spam message detection. Instead of using pre-trained word embedding models, this approach used random weight initialization in the embedding layer to learn the word tokens. Those weights were updated through back propagation during model training. This method was adopted as the baseline model for this FYP. A summary of related works is tabulated in Table 1.

Table 1: Summary of Related Works. TF: Text-based Feature, UF: User-based Feature, CF: Content-based Feature, ML: Machine Learning, DL: Deep Learning.

Related Works	**Features**			**Learning Methods**		**Datasets**	
	TF	**UF**	**CF**	**ML**	**DL**	**Twitter**	**Others**
[4], [5]		x	x	x		x	
[8]	x			x		x	
[3]	x			x	x	x	
[21], [23]	x				x	x	
[9], [24]	x	x	x		x	x	
[25], [26], [28], [29], [30]	x				x		x

Chapter 3: Background Research

3.1 Twitter Dataset

In this FYP, the Twitter dataset that was selected to work on is called "Social Honeypot Dataset" created by Lee et al. [31]. This dataset was created by the deployment of 60 social honeypot accounts on Twitter which posted random messages to engage with Twitter users. Twitter accounts that interacted with these social honeypot accounts were recorded. The distributions of legitimate users and content pollutants collected in the experiment are tabulated in Table 2.

Table 2: Distributions of Social Honeypot Dataset.

Class	**No. of User Profiles**	**No. of Tweets**
Legitimate Users	19,276	3,263,238
Content Polluters	22,223	2,380,059

The original dataset contains six text files as described in Table 3. In my approach, only "*content_polluters_tweets.txt*" and "*legitimate_users_tweets.txt*" were used since the proposed method is to detect spam tweets.

Table 3: List of Text Files and its Corresponding Features.

Text File	**Features**
"content_polluters.txt" "*legitimate_users.txt*"	UserID, CreatedAt, CollectedAt, NumberOfFollowings, NumberOfFollowers, NumberOfTweets, LengthOfScreenName, LengthOfDescriptionInUserProfile
"content_polluters_followings.txt" "*legitimate_users_followings.txt*"	UserID, SeriesOfNumberOfFollowings
"content_polluters_tweets.txt" "*legitimate_users_tweets.txt*"	UserID, TweetID, Tweet, CreatedAt

3.2 Text-Processing Techniques

3.2.1 Stemming & Lemmatization

Words come in many grammatical forms such as noun, verb, singular, plural, different tenses etc. Unlike human beings, machines cannot understand words with different variations. These variations can create ambiguity in machine learning and prediction. Additionally, large computational resource is likely required to handle the huge variation of words in language. As a result, text normalization is an essential step in machine learning tasks to reduce word complexity, computational time, and space. Stemming and lemmatization are two widely used text normalization methods in NLP tasks to convert a word to its root form. Examples of such word conversion are shown in Figure 1. These two methods can be implemented using NLTK [32] package.

Original Word		**Root Form**
Carries	→	Carry
Carried	→	Carry
Carrying	→	Carry

Figure 1: Examples of Word Conversion.

There are many types of stemming algorithm in NLTK package. One example is the Porter Stemmer [33] which automatically removes suffixes of a word. This is a fast algorithm, but the resultant root word may not be an English word. Examples of word conversion using Porter Stemmer is shown in Figure 2.

Original Word		**Root Form**
Carries	→	Carri
Carried	→	Carri
Carrying	→	Carri

Figure 2: Examples of Word Conversion using Porter Stemmer.

Unlike stemming, lemmatization accurately reduces a word to its root form found in English language dictionary. In lemmatization, the root word is called a lemma which is the dictionary form of a set of words. For example, *carries*, *carried*, and *carrying* are variations of the word *carry*. Thus, *carry* is the lemma of those words. Lemmatization achieves high accuracy in word conversion because it provides context to words using Parts-of-Speech (POS) tagging. The word lemmatizer converts a word to its root form based on POS tag assigned. The default POS tag in lemmatization is noun. The effects of POS tag in lemmatization is shown in Figure 3. Despite its high accuracy, lemmatization takes longer time to process because it requires to scan the WordNet [34] corpus to look for the correct root word to convert the word.

1) POS Tag is Noun

Original Word		**Root Form**
Carries	→	Carry
Carried	→	Carried
Carrying	→	Carrying

2) POS Tag is Verb

Original Word		**Root Form**
Carries	→	Carry
Carried	→	Carry
Carrying	→	Carry

Figure 3: Effects of POS Tag in Lemmatization.

3.2.2 Word Embedding

Word embeddings is an important technique in machine learning to represent words into dense vectors of real number. These word vectors were obtained via word embedding model training on text corpus. The model learns the text corpus and maps semantically similar words close together on embedding space using cosine similarity. As a result, words which are closely related in terms of semantics have almost identical vectors.

Both Word2Vec [19] and GloVe [20] are commonly used as word embeddings in machine learning tasks to capture the contextual meaning of words. Furthermore, these word embeddings can reduce machine computational time and space through dimensionality reduction on training data.

Word2Vec employed a two-layered neural network to train on large text corpus output a set of vectors. Word2vec relies on context information of words using skip-gram and CBOW algorithms. These algorithms are suboptimal because the semantics learnt for any given word is based on its surrounding words according to pre-defined window size. The main difference between CBOW and skip-gram is that CBOW uses context to predict a target word, whereas skip-gram uses a word to predict target context. The graphical representation of CBOW and skip-gram is shown in Figure 4.

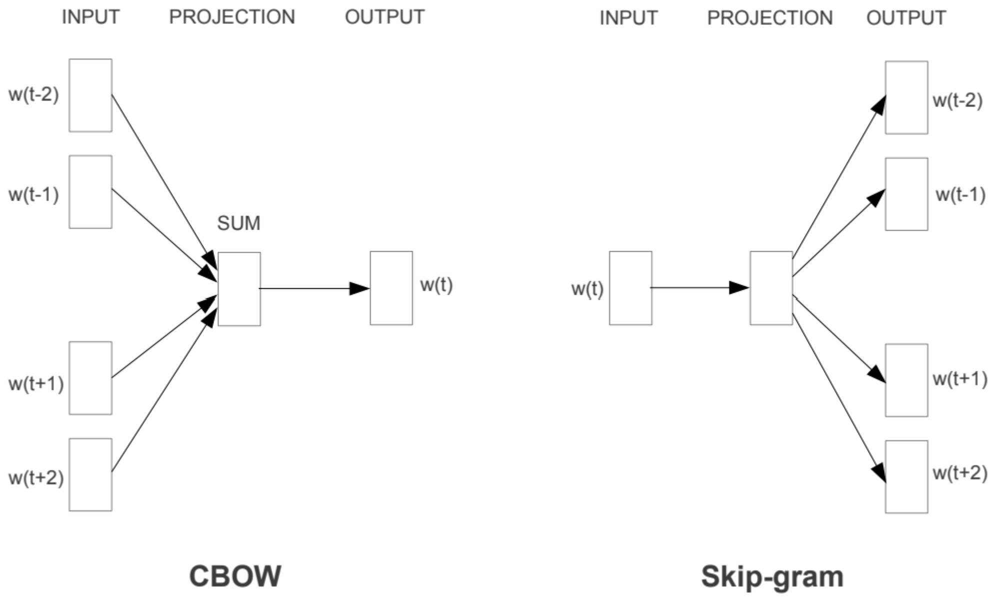


Figure 4: Graphical Representation of CBOW and Skip-gram. Extracted from [35].

GloVe is a pre-trained dense vector trained using Word2Vec model. In this project, GloVe-Twitter vector is specifically used which was trained on 2 billion tweets and 1.2 million vocabulary words. GloVe vector was developed by employing co-occurrence matrix which computes how frequently words co-exists with one another in a corpus. This computation has the potential to encode some form of contextual meaning between words. For example, the co-occurrence probability for *ice* and *solid* is higher than *ice* and *gas*.

Another word embedding method is random embedding weights initialization in embedding layer of a neural network. The main difference between this method and Word2Vec is it learns vector representation of words in the same network instead of a separate network. Embedding weights are randomly initialized based on a uniform distribution and updated during model training via backpropagation algorithm [36]. After training, each index in a sequence has its own vector representation of a fixed dimension.

3.3 Deep Learning Architecture

3.3.1 Convolutional Neural Network

Convolutional Neural Network (CNN) [11] is a deep learning model commonly used in computer vision tasks such as image classification, object detection, and image segmentation. Moreover, CNN can also be used to learn one-dimensional text data and three-dimensional video images. In this section, two-dimensional image data will be used to explain the architecture of CNN.

CNN architectures use multiple feature extraction layers to progressively extract multi-level hierarchical features from the input image. A typical CNN architecture contains convolution layer, pooling layer, fully-connected layer, and classification layer. An example of a CNN architecture is illustrated in Figure 5.

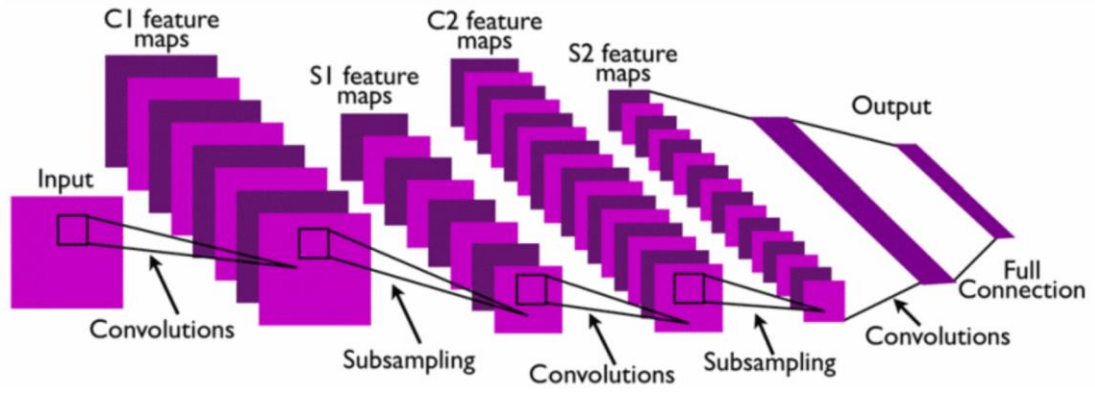
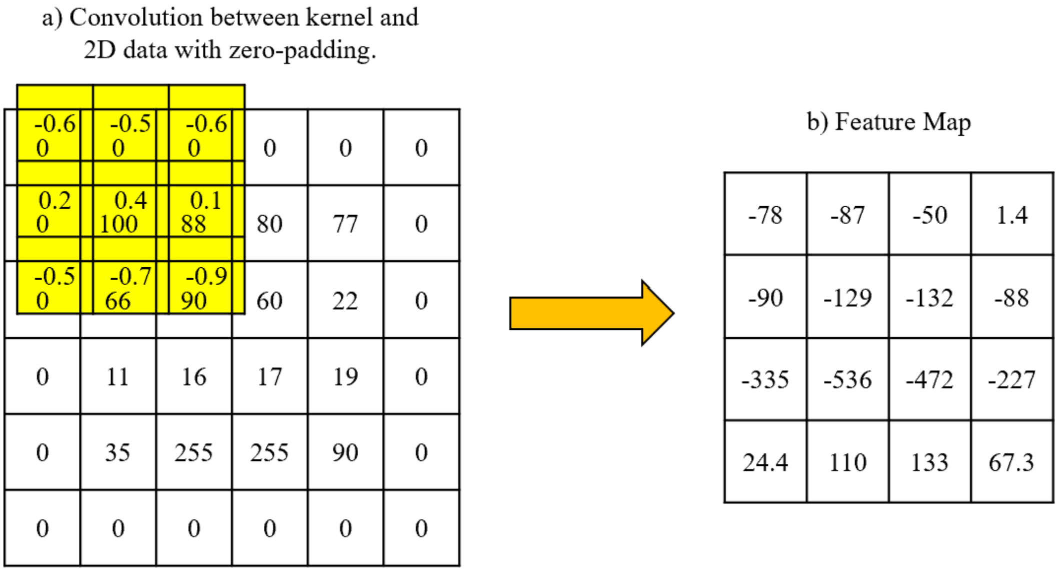


Figure 5: CNN architecture with two feature stages. Extracted from [37].

Convolution layer is the first layer in feature extraction stage to extract features from input image. This layer contains multiple $m \times m$ square kernels to extract features in a hierarchical manner, where m represents the dimension of the kernel. Early convolution layers extract low-level features, whereas deeper convolution layers extract high-level features. Kernel contains weight parameters which is used to perform convolution with image pixel values via dot product. The kernel slides over each input image based on a given stride to produce feature maps. The output of each convolution layer is n feature maps, where n is equal to the number of kernels used in that convolution layer. An example of convolution operation on 2D data is illustrated in Figure 6.



a) Convolution between kernel and 2D data with zero-padding.

-0.6 0	-0.5 0	-0.6 0	0	0	0
0.2 0	0.4 100	0.1 88	80	77	0
-0.5 0	-0.7 66	-0.9 90	60	22	0
0	11	16	17	19	0
0	35	255	255	90	0
0	0	0	0	0	0

b) Feature Map

-78	-87	-50	1.4
-90	-129	-132	-88
-335	-536	-472	-227
24.4	110	133	67.3

Figure 6: An Example of Convolution Operation on 2D Data.

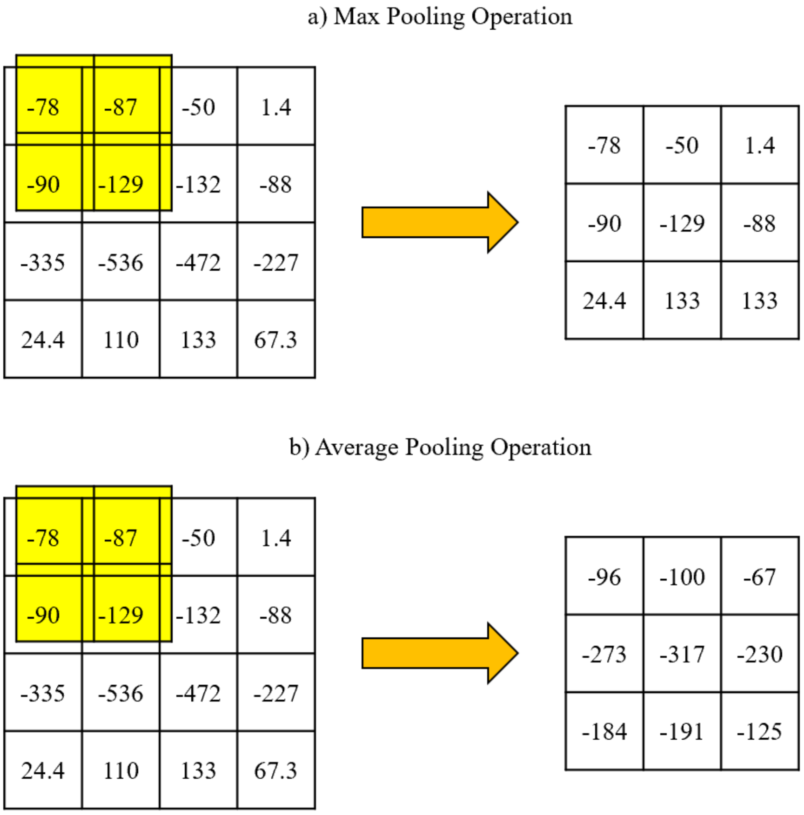
Activation function such as Rectified Linear Unit [38] (ReLU) is often used in convolution layer to provide non-linearity properties. Additionally, padding is sometimes included in the convolution layer parameter to prevent the shrinking of original image size after convolution. The output dimensions of feature maps are computed as follows:

$$W_o = [(W_i - F + 2P)/S] + 1$$
$$H_o = [(H_i - F + 2P)/S] + 1$$

, where W_o, H_o, W_i, and H_i are the output width, height, input width, and height, respectively. F is the kernel size, m. P is the amount of padding. S is the number of strides.

Pooling layer is the second layer in feature extraction to reduce spatial dimensions of feature maps. The number of feature maps increases with feature extraction stages. As a result, pooling layer is required to reduce the computational time and memory storage. In addition, pooling layer provides position invariance characteristics so that training performance of CNN model is not dependent on pixel location.

Pooling layer contains square kernel of size smaller than the feature maps and it does not contain weight parameter. The kernel slides over the feature maps based on a given stride to produce a reduced-size feature map. Two common pooling operations are max pooling and average pooling. In max pooling, kernel overlays a region in feature map and extracts the maximum value in the region. This process is repeated until the whole feature map is covered by the kernel. On the other hand, average pooling overlays a region in feature map and extracts the average value in the region. An example of pooling operations is illustrated in Figure 7.



a) Max Pooling Operation

-78	-87	-50	1.4
-90	-129	-132	-88
-335	-536	-472	-227
24.4	110	133	67.3

-78	-50	1.4
-90	-129	-88
24.4	133	133

b) Average Pooling Operation

-78	-87	-50	1.4
-90	-129	-132	-88
-335	-536	-472	-227
24.4	110	133	67.3

-96	-100	-67
-273	-317	-230
-184	-191	-125

Figure 7: Example of Pooling Operations, a) Max Pooling and b) Average Pooling.

The combination of a convolution layer and pooling layer forms a feature extraction stage. The number of feature extraction stages required typically depends on the types of classification task. Optimal number of stages can be achieved through trial experiments.

Feature extraction stage is followed by fully-connected layer(s). Based on Figure 5, feature maps are flattened into one-dimensional vectors before passing to fully-connected layer. Each fully-connected layer contains neurons with non-linear activation function, weight and bias parameters to compute forward propagation [39]. The classification layer which is typically called softmax [40] layer computes the loss function based on the target output. The weight and bias parameters of CNN are updated via backpropagation algorithm for optimization.

3.3.2 Recurrent Neural Network

Recurrent Neural Network (RNN) [12] is applied in some machine learning applications which involves sequential data. Such applications include speech recognition, time-series prediction, and sentiment classification. RNN is a suitable model for predicting textual data because it has internal memory state to store information from past inputs. Example of RNN architecture is illustrated in Figure 8.

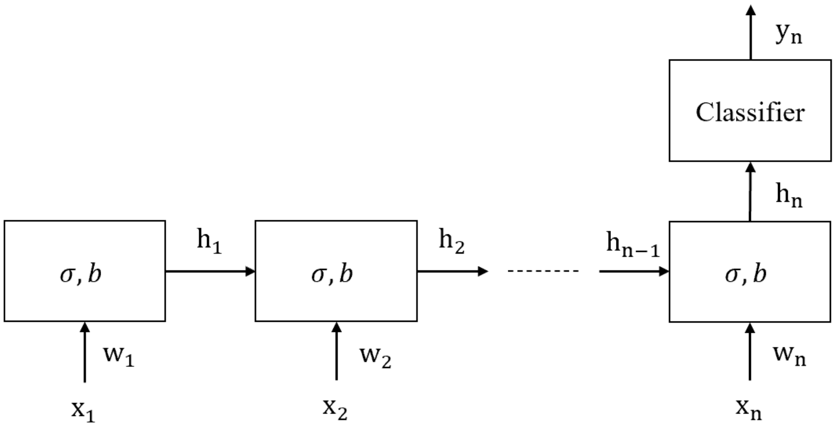


Figure 8: RNN Architecture.

RNN has a chain-like structure with repeating modules. Each module is a copy of RNN which contains a single layer neural network to compute forward propagation. Input textual data $\{x_1, x_2, \ldots, x_n\}$, where x_1 denotes the first word in a sentence. Input at each time step is multiplied with weight parameter denotes as *w* before it is passed to the hidden layer sequentially. Each hidden layer contains hyperbolic tangent activation function and bias value denote as σ and b, respectively. Additionally, every hidden layer has the same weight parameters so that weights are shared over time to reduce computational resources. The shared weights are computed and updated for each copy of RNN via Backpropagation Through Time (BPTT) algorithm [41]. The output of first hidden layer denotes as h_1 together with the second word, x_2, are passed into the second hidden layer and so on. This means that the output of any hidden layer, h_n, is dependent on the previous layer output, h_{n-1}. The output of last hidden layer, h_n, is passed to sigmoid [42] activation function for classification.

Hidden layers can be stacked vertically to learn higher level features, but the weight and bias parameters are not the same between stacks. The output of hidden layer at bottom stack is retained and passed to top stack as input. An illustration of stacked RNN is shown in Figure 9. Stacked RNN is commonly used in machine learning applications such as image captioning and language translation to produce output sequence $\{y_1, y_2, \dots, y_n\}$. On the contrary, only the output of last hidden layer is required to determine the classification outcome, y_n, as shown in Figure 8. Whereas the outputs of previous hidden layers are eliminated. The classification outcome is obtained by the summation of all hidden layer non-linearity output.

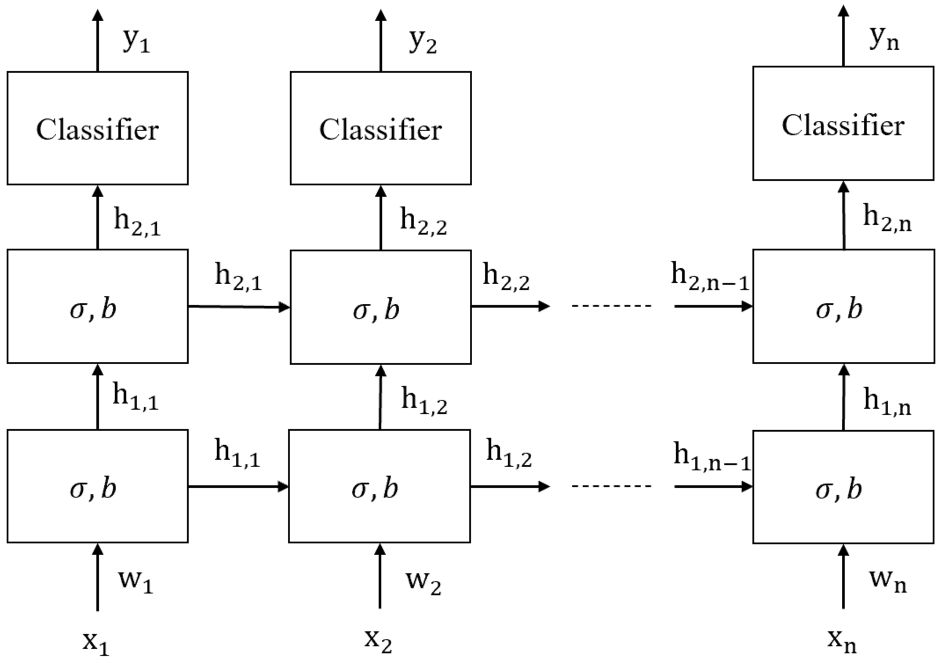


Figure 9: Stacked RNN Architecture.

3.3.3 Long Short-Term Memory

RNN model has shown it capability to learn sequential data by connecting past information to present information. However, RNN cannot learn very long sequential data effectively due to vanishing gradient problem. Vanishing gradient problem arises when weight updates become significantly small after passing through many hidden layers. This will cause the model to stop training because there is almost no change in weights. Therefore, LSTM [13] was designed to solve vanishing gradient problem and retain state information over long period of time.

LSTM has a chain-like architecture with repeating modules similar to RNN. However, LSTM has four activation functions in a module instead of one activation function in RNN. The illustration of LSTM architecture with three repeating modules is shown in Figure 10.

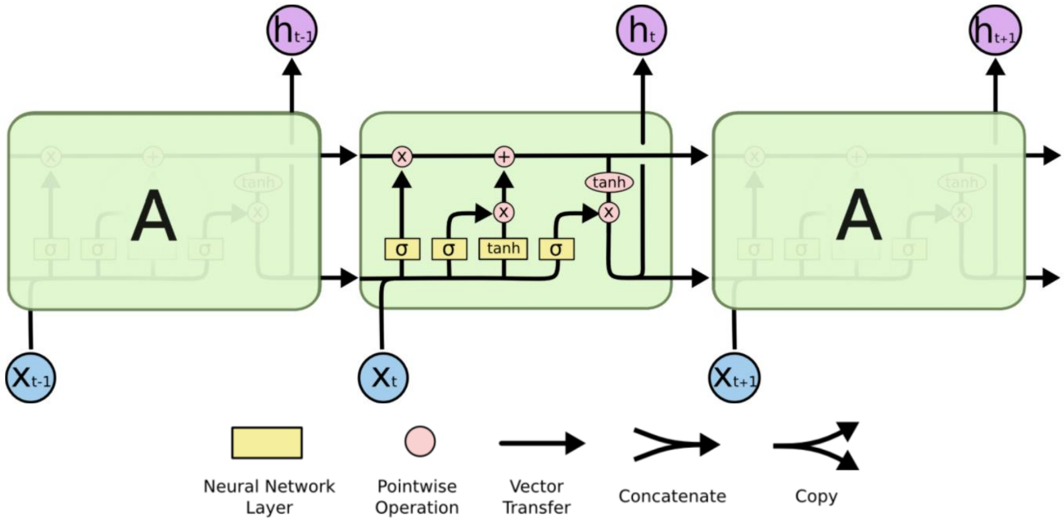


Figure 10: LSTM Architecture. Extracted from [43].

A LSTM module contains cell status, forget gate, input gate, and output gate. These gates are operated by the sigmoid activation functions which control information flow. Cell status denotes as c_t has additional delay which allows the flow of information from previous module to be stored in the current cell. The cell status is modified by the forget gate and input gate denote as f_t and i_t, respectively. The forget gate determines which information from previous cell is important and not important. The decision is made by sigmoid activation function which outputs either a 0 or 1 based on h_{t-1} and x_t. If the output is 0, the information is eliminated. The input gate determines which information to be inserted into the cell state via sigmoid and hyperbolic tangent activation functions. The new cell state is updated by multiplying the old cell state with forget gate and adding the input gate. Finally, the output gate denotes as o_t computes information from previous cell and current cell via sigmoid activation function. The output information is combined with the updated cell state and passes to next LSTM cell. The computation of information is based on element-wise multiplication and addition operations.

3.3.4 Transformer

Transformer [14] is a self-attention model which can attend to different positions of input sequence and compute a representation of that sequence. This deep learning technique is commonly used in NLP tasks such as language translation, language model, and text classification. A typical Transformer architecture of language translation model is made of stacks of encoder and decoder layers as shown in Figure 11. Every encoder is identical to one another and has its own set of weights, likewise for decoders. However, not all NLP applications use this type of architecture. Language model architecture uses decoder layers to decode the input phrase and predict its subsequent words. On the other hand, text classification only requires the encoder layers to predict the class labels of the input text data.

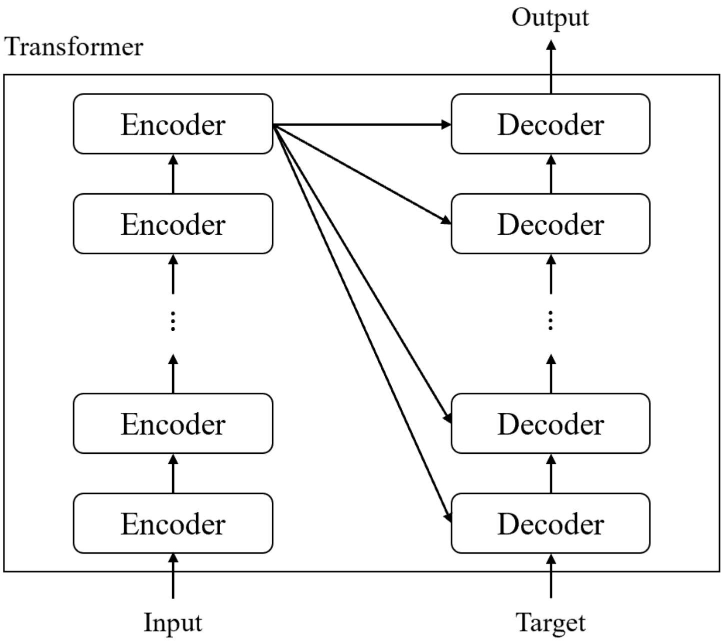
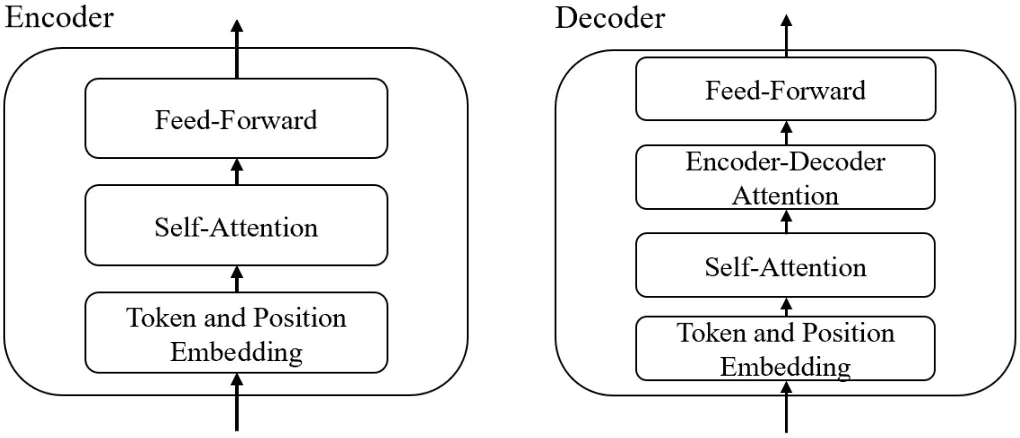


Figure 11: Transformer Architecture of Language Translation Model.

A typical encoder in Transformer model contains embedding layers, self-attention layer, and feed-forward neural network as shown in Figure 12(a). Token embedding layer converts tokenized words in input sentence into embedding vectors. Unlike RNN and LSTM models, Transformer model does not have recurrent properties to remember the positions of words because all the words in a sentence are input in parallel. As a result, positional embedding layer is used to provide the model with information about the word positions in an input sequence. The position embedding is added to token embedding to produce word embeddings in *n* dimensional space based on semantic similarities and word positions in a sentence.


Encoder
Feed-Forward
Self-Attention
Token and Position Embedding
Decoder
Feed-Forward
Encoder-Decoder Attention
Self-Attention
Token and Position Embedding

(a) Encoder

(b) Decoder

Figure 12: Illustration of an Encoder and Decoder Layer in Transformer.

Word embeddings are multiplied with weight matrices in linear layers to create three other vectors, namely query, key, and value vectors for each word. These vectors are used as inputs to self-attention layer to produce a word encoding.

The self-attention layer computes an attention score for each word in a sentence against the target word. Attention score determines the attention level to be put in place on other positions of the input sentence when a target word is encoded at specific position. The attention score is calculated by taking the dot product of target word's query vector and neighbouring word's key vector. All the attention scores are normalized and passed to softmax activation function to compute the probabilities of words appearing at that position. These probabilities are multiplied with value vectors to put less attention on less relevant words. Finally, the weighted value vectors are summed up which produces the output of self-attention layer at every word positions. The output vector is passed to feed-forward neural network to produce encoded representation of the input sequence.

Decoder has similar architecture as the encoder, except an additonal encoder-decoder attention layer between self-attention layer and feed-forward neural network as shown in Figure 12(b). The purpose of encoder-decoder attention layer is to compute an encoded representation of the target sequence. This computation is performed based on the encoded representations of input sequence from encoder output and target sequence from decoder input. The encoded representation of target sequence is passed to the output layer to produce an output sequence. The loss function is computed using the output and target sequences during model training. An illustration of the Transformer model training process is shown in Figure 13.

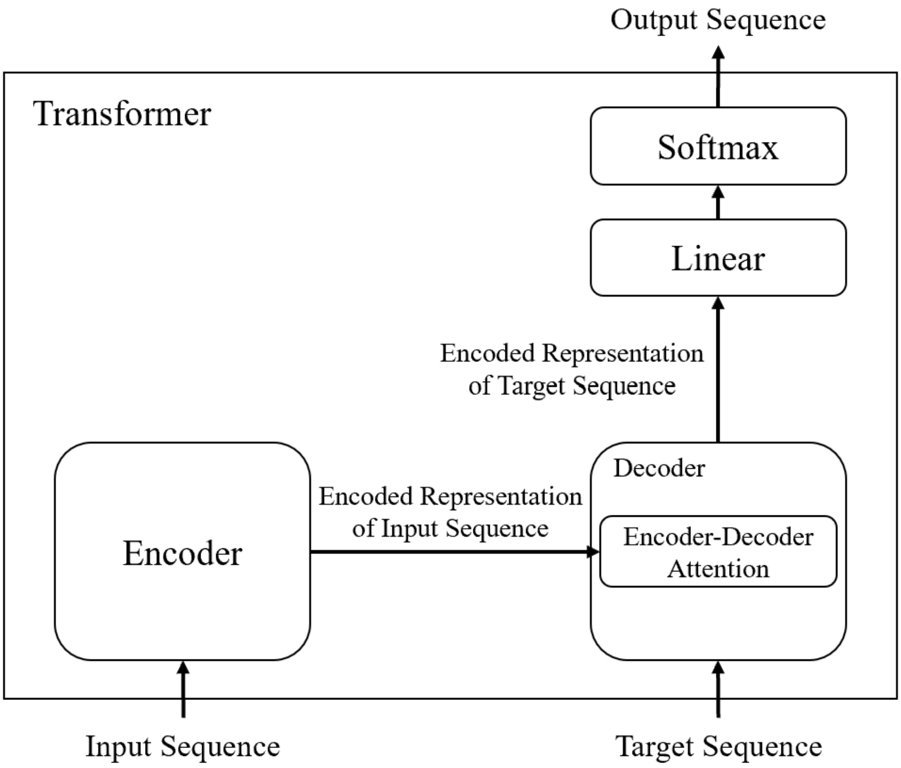

Output Sequence
Transformer
Softmax
Linear
Encoded Representation of Target Sequence
Decoder
Encoder
Encoded Representation of Input Sequence
Encoder-Decoder Attention
Input Sequence
Target Sequence

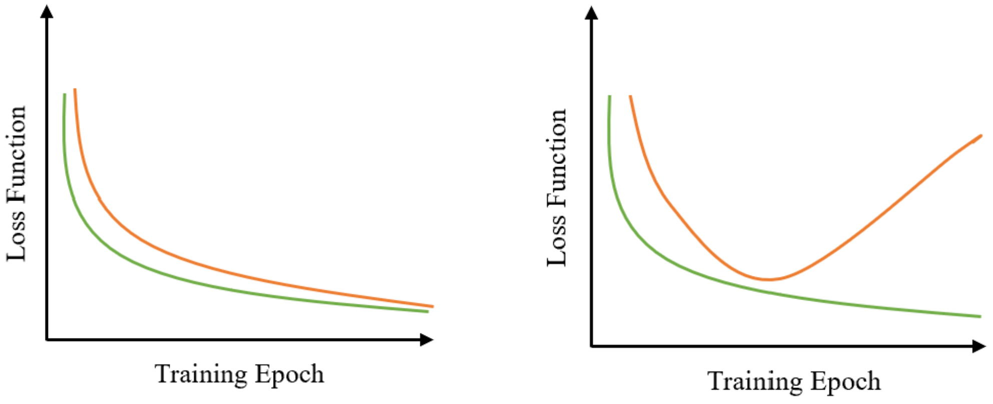
Figure 13: Illustration of Transformer Model Training Process.

3.4 Model Training

A deep learning architecture requires to go through training process so that it can automatically perform its machine learning task. Model performance is affected by training parameters such as optimizer, learning rate, loss function, batch size, and training epoch. In general, there is no ideal set of training parameters that can fit all model architectures for optimal results. Therefore, these parameters are chosen empirically based on multiple training experiments.

Optimizer is an algorithm to update model weights and minimize the loss function through gradient descent. The loss function is used to guide the optimizer towards the direction to achieve global minimum. Learning rate plays a role in optimization as well. A high learning rate can reduce training time, but it can also cause the loss function fails to converge. On the contrary, small learning rate can result in longer training time due to small weight updates and cause the model to be stuck at local minimum. Batch size controls the accuracy of error gradient estimation by dividing the training samples into smaller sample size. Typically, batch sizes are in the power of two to align with the number of physical processors in a GPU. Number of training epoch is chosen to ensure the model achieves optimal performance and terminate training process when overfitting occurs.

Overfitting is a phenomenon when the loss function stops improving after a certain number of training epochs. This phenomenon can be monitored by splitting the training data into two subsets, namely training data and validation data. The validation data does not involve in model training but is used to evaluate the model training progress. Training loss will keep decreasing, but validation loss does not. Therefore, training can be terminated when the validation loss does not improve anymore. An illustration of overfitting phenomenon is shown in figure 14.



(a) No Overfitting (b) Overfitting

Figure 14: An illustration of overfitting phenomenon, where green curve represents training loss and orange curve represents validation loss.

Chapter 4: Proposed Method

Deep learning models such as CNN, LSTM, and Transformer were used in the proposed method to investigate individual model's effectiveness in text classification. Prior to model training, data preprocessing steps such as feature extraction and data cleaning were performed. Features which are relevant and important were extracted from "*content_polluters_tweets.txt*" and "*legitimate_users_tweets.txt*" files. Feature extraction was followed by data cleaning to remove outliers. Text processing was performed to remove redundant content in tweets using regular expressions (regex) method in NLTK packages. Processed tweets were normalized to equal length and mapped to vector representation using pre-trained word vectors from Word2Vec model and GloVe. Finally, normalized training dataset was used as inputs to text classifiers for machine learning. Tweets not used in training, also known as test data, were used to evaluate model performance based on the following metrics: accuracy, precision, recall, and f-score. The pipeline of proposed method is illustrated in Figure 15.

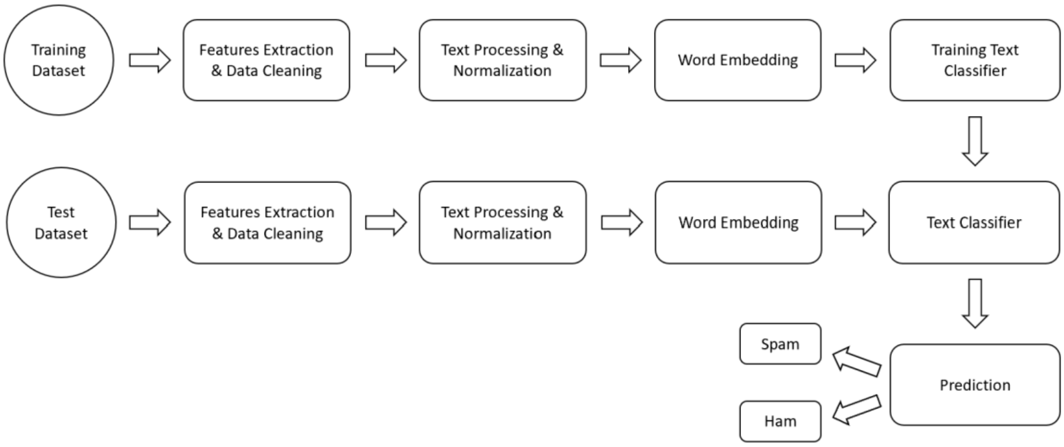

Training Dataset
Features Extraction & Data Cleaning
Text Processing & Normalization
Word Embedding
Training Text Classifier
Test Dataset
Features Extraction & Data Cleaning
Text Processing & Normalization
Word Embedding
Text Classifier
Spam
Prediction
Ham

Figure 15: Pipeline of Proposed Method.

4.1 Resources

The hardware resources used in this project are personal computer and GPU server with NVIDIA GTX 1080 Ti graphics card.

The software resources used in this project are Python language version 3.6.10 [44] with Pandas [45], NLTK [32], Scikit-Learn [46], Tensorflow [47], Keras [48] packages installed. FileZilla application was used for files transfer between local computer and GPU server. PuTTY application was used to remotely access the GPU server. The software applications used in this project are free and open-sourced.

4.2 Data Pre-processing

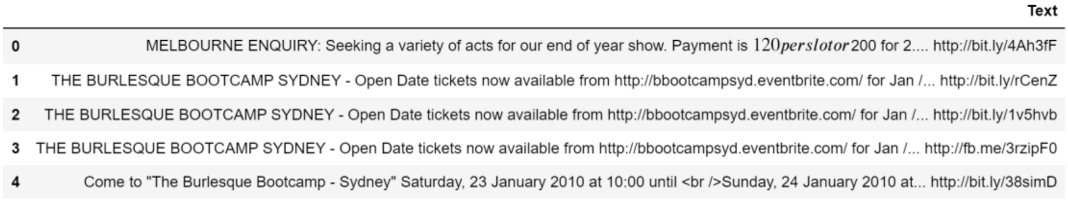
Feature extraction and data cleaning are important steps in data pre-processing to produce a high-quality dataset for machine learning. Therefore, these steps must be planned and performed strategically to achieve optimal classification performance of the model.

Feature extraction is an important procedure to extract relevant features which are useful for spam classification. Table 4 shows a sample data from the "*content_polluters_tweets.txt*" file. UserID symbolizes the identity of a Twitter user. TweetID is the identity of a Tweet. CreatedAt represents the date and time of the tweet created. These features are not beneficial to machine learning because they contain mostly unique data. As a result, these three features were not extracted. Instead, the remaining Tweet feature was extracted for machine learning.

Table 4: A sample data containing four features.

UserID	**TweetID**	**Tweet**	**CreatedAt**
6301	5702173484	I need a sound person tomorrow a t an event in melbourne. Please e mail apply@jacbowie.com	2009-11-13 23:15:12

Social Honeypot dataset contains over 5 million tweets with class labels, namely Spam and Non-Spam (Ham). Spam and Ham datasets were handled separately due to a large volume of tweets. Each of these datasets was represented as dataframes to perform Exploratory Data Analysis (EDA) using Pandas [45] package. Sample tweets in spam dataframe is shown in Figure 16.



	Text
0	MELBOURNE ENQUIRY: Seeking a variety of acts for our end of year show. Payment is 120*perslotor*200 for 2.... http://bit.ly/4Ah3fF
1	THE BURLESQUE BOOTCAMP SYDNEY - Open Date tickets now available from http://bbootcampsyd.eventbrite.com/ for Jan /... http://bit.ly/rCenZ
2	THE BURLESQUE BOOTCAMP SYDNEY - Open Date tickets now available from http://bbootcampsyd.eventbrite.com/ for Jan /... http://bit.ly/1v5hvb
3	THE BURLESQUE BOOTCAMP SYDNEY - Open Date tickets now available from http://bbootcampsyd.eventbrite.com/ for Jan /... http://fb.me/3rzipF0
4	Come to "The Burlesque Bootcamp - Sydney" Saturday, 23 January 2010 at 10:00 until <br />Sunday, 24 January 2010 at... http://bit.ly/38simD

Figure 16: Sample Tweets in Spam Dataframe.

After EDA, it was observed that the dataset contains several non-English tweets. This will hinder the performance of text classifier due to mixture of different languages. Therefore, data cleaning on the original dataset was required. Text processing was performed to standardize tweet format used for machine learning. An overview of data pre-processing for baseline dataset is summarised in Table 5.

Table 5: Overview of Data Pre-processing for Baseline Dataset.

Data Cleaning	**Actions Taken**
Step 1	Removed URLs, mentions (*@username*), hashtags (*#keyword*), and retweets (*RT*).
Step 2	Removed stopwords (E.g. *I, the, you*), and punctuations. Lemmatized texts to group similar words as a single item.
Step 3	Removed non-English words, and words with less than 3 characters.

Firstly, features such as URLs, user mentions, hashtags, and retweets were removed using regex method. These features were removed by calling the regex function *regex.sub()* which replaces a string into any replacement. For example, URLs usually in the form of "https://..." are removed by passing "https\S+" as the first argument and empty string '' as the second argument in the *regex.sub()* function to replace all URLs with empty space. On the other hand, user mentions, hashtags, and retweets were removed by passing "@\S+", "#\S+", and "\bRT\b", respectively as the first argument. \S+ in regex term means that any non-space character. Therefore, this term was appended to annotations such as "@" and "#" to detect user mentions and hashtags because any non-space characters after the annotations are to be replaced with empty string. \b in regex term means boundary of word. This term was added on both end of the word "RT" to ensure only word "RT" is removed instead of removing "RT" in "CART".

Secondly, stopwords were removed because they occurred in abundance which have less significance compared to other vocabulary words. Words appear in *nltk.corpus.stopwords* package were replaced with empty string. Punctuations were removed using *string.punctuation* package. Word lemmatization groups similar words with different formats such as singular and plural terms, into one single item. This helps to reduce the number of words for computation. *WordNetLemmatizer()* function from WordNet [34] package was used to lemmatize words. Although lemmatization takes longer to process than stemming, the former has better accuracy in terms of word simplification.

Thirdly, non-English words (including misspelt and abbreviated words) were removed because pre-trained word embeddings do not contain vectors for these words. Words do not appear in *nltk.corpus.words* package were replaced with empty string. Lastly, words contain less than three characters were removed because they have little meaning in texts.

Table 6: Overview of Data Pre-processing for Improved Dataset.

Data Cleaning	**Actions Taken**
Step 1	Replaced URLs and mentions (*@username*) with "url" and "usermention", respectively.
Step 2	Removed duplicates.
Step 3	Removed "#" in hashtags (*#keyword*) but keep keyword and replaced retweets (*RT*) with "retweet".
Step 4	Removed stopwords (E.g. *I, the, you*), and punctuations. Lemmatized texts to group similar words as a single item.
Step 5	Removed non-English words, and words with less than 3 characters.
Step 6	Removed tweets that contain less than 2 words.

Strategic text processing procedure was carried out to improve on the baseline model performance. An overview of data pre-processing for the improved dataset is summarised in Table 6. Instead of removing URLs, user mentions, hashtags, and retweets, these features were retained. URLs, user mentions, and retweets were replaced by "url", "usermention", and "retweet", respectively. These three words are not found in *nltk.corpus.words* package. Therefore, they were added to English dictionary so that they will not be removed in Step 5. Duplicate tweets were removed using *drop_duplicates* function from Pandas package to prevent overfitting of the model. Removal of duplicates was performed only after URLs and user mentions replacement. The reason is duplicated spam tweets usually contain distinct URLs and user mentions to spread information. Therefore, these two features are required to be simplified and treated as simple terms in all tweets so that duplicates can be removed.

Duplicates removal was followed by stopwords removal, punctuations removal, word lemmatization, non-English words removal, and short-length words. Finally, tweets which contain only single word were removed because they do not have neighbouring words to create semantic relationships in a tweet.

Inspired from Wei et al. [23], data cleaning and text processing steps were carried out on the original dataset to create a new dataset called GloVe dataset. The main purpose of creating a GloVe dataset is to optimize the matching rate of vocabulary words to pre-trained GloVe dense vectors. Significant terms such as *URL*, user *mentions*, *retweets*, and *hashtags* were tagged as *<url>*, *<user>*, *rt*, *<hashtag>*, respectively in GloVe vectors. The overview of text processing steps and actions taken is showed in Table 7. Most of the steps are the same as Table 6 except in step 3, hashtags were replaced by “hashtag” instead. Additionally, “url”, “usermention”, “retweet”, and “hashtag” were replaced by “<url>”, “<user>”, “rt”, and “<hashtag>”, respectively to match the respective terms in GloVe vector in step 7. The reason for performing this conversion at the final step is to prevent the removal of angle brackets at Step 4. The source code for data pre-processing can be found in Appendix A.

Table 7: Overview of Data Pre-processing for GloVe Dataset.

Data Cleaning	**Actions Taken**
Step 1	Replaced URLs and mentions (*@username*) with “url” and “usermention”, respectively.
Step 2	Removed duplicates.
Step 3	Replaced hashtags (*#keyword*) and retweets (*RT*) with “hashtag” and “retweet”, respectively.
Step 4	Removed stopwords (E.g. *I, the, you*), and punctuations. Lemmatized texts to group similar words as a single item.
Step 5	Removed non-English words, and words with less than 3 characters.
Step 6	Removed tweets that contain less than 2 words.
Step 7	Replaced “url”, “usermention”, “retweet”, and “hashtag” with “<url>”, “<user>”, “rt”, and “<hashtag>”.

After data pre-processing stage, tweets were labelled as “ham” and “spam”. Both Spam and Ham dataframes were merged into a single dataframe. The class distributions of original dataset, baseline dataset, improved dataset, and GloVe dataset are illustrated in Figure 17.

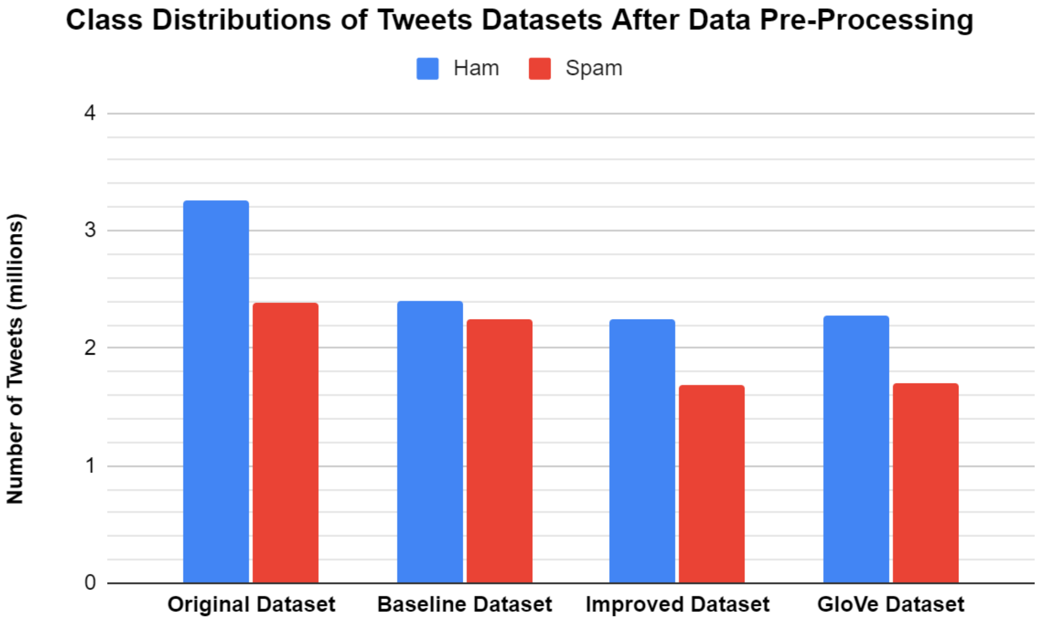


Figure 17: Class Distributions of Tweets Datasets After Data Pre-Processing.

Each dataset was further split into subsets: training (98%), test (1%), and validation (1%) datasets. Training dataset was primarily used for machine learning. Validation dataset was used to monitor signs of model overfitting in training process. Test dataset was used to evaluate the trained model's spam detection performance in prediction step.

Prior to word embedding, text normalization was performed on the processed tweet datasets. The workflow of text normalization is shown in Figure 18. Tweet sentences were tokenized and converted to sequences of numbers. This was achieved by utilizing *Tokenizer* function and its *fit_on_texts*, *texts_to_sequence* attributes from Keras [48] package.

Tokenizer fits the training dataset using *fit_on_texts* and output a dictionary. The dictionary contains all vocabulary words found in the training dataset as keys while its corresponding unique sequence number as value. Sequence number ranges from 1 to the total number of vocabulary words found in the training dataset. The dictionary was arranged in descending order based on frequency of vocabulary words appearing in training dataset. Low sequence number signifies high occurrence of the word in dataset. Sequence number was assigned to unknown words tagged as <OOV> which were not learned by Tokenizer. This action is essential so that unknown words in validation and test datasets are tagged to <OOV> sequence number. Trained tokenizer was used to convert every vocabulary word in training, test, and validation dataset into sequence of numbers using *texts_to_sequence*. Furthermore, trained tokenizer was saved using Pickle [44] package to reduce experimental time.

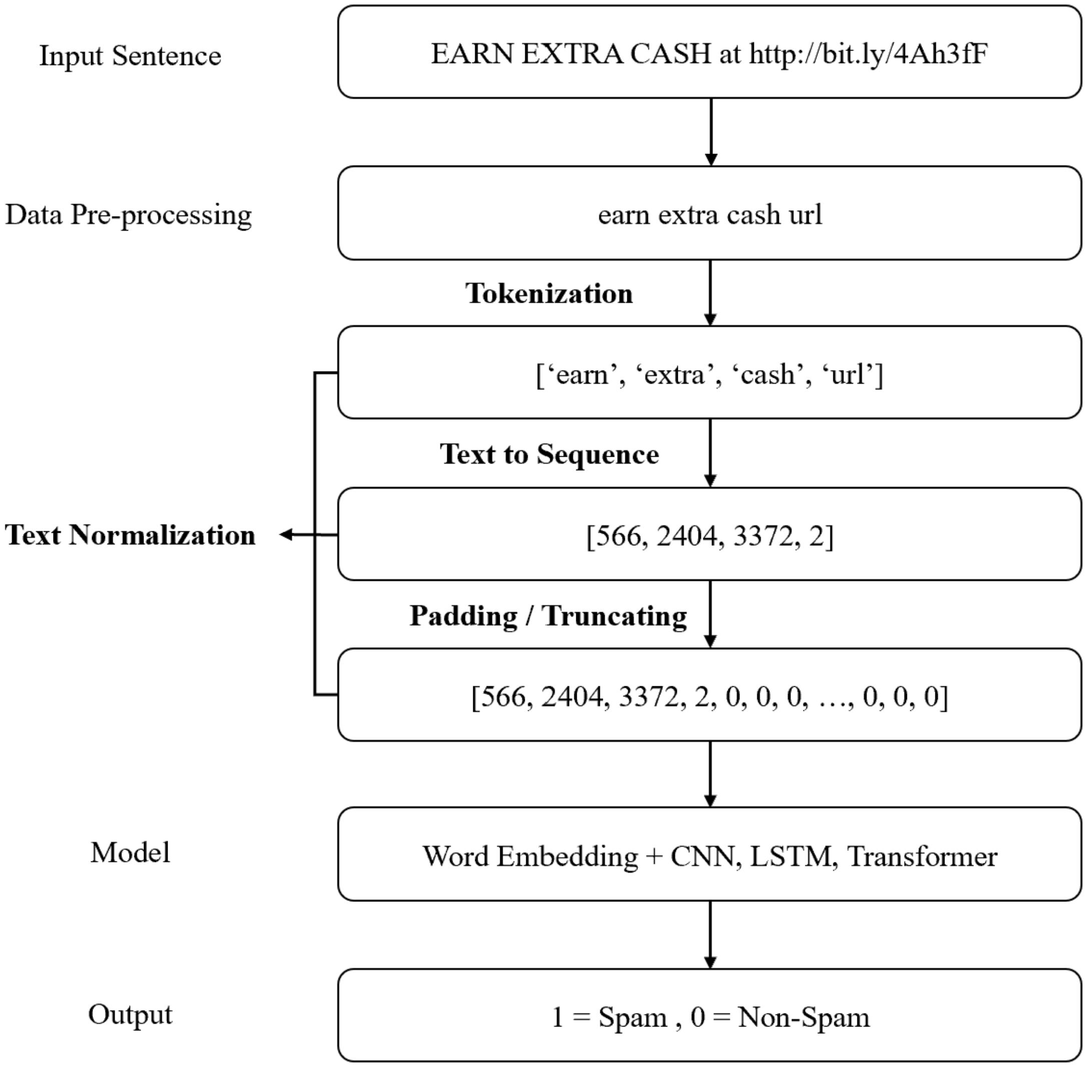


Figure 18: Workflow of Text Normalization.

Finally, tweet lengths were transformed to equal length because machine learning models can only accept fixed input dimension. This was achieved by using the *pad_sequences()* function from Keras package. Sequence data was passed as the first argument in the function. A maximum length of 20 was passed as the second argument to fix the length of sequence data. The reason for choosing this value is tweets which have more than 20 words mainly contain only single vocabulary word in the dataset. This action will not have negative impacts on spam detection because only 100 of them were truncated. First 20 words in a tweet were retained while the remaining words were removed. On the other hand, tweets which have length below maximum length were padded with zeros. Zero-padding was performed at the back of the sentence. The source code for text normalization can be found in Appendix B.

4.3 Word Embedding

4.3.1 Word2Vec Model

Word2Vec model was utilized to train the improved training dataset from scratch and produce word vectors as outputs. Firstly, every tweet in the dataset was tokenized into word tokens. Secondly, Word2Vec model from Gensim [49] package was used to build vocabulary from the training dataset and learn vector representations of words. Vocabulary word and its corresponding vectors were stored in two separate lists. List of vocabulary words can be retrieved by calling *wv.index2word* attributes of the trained Word2Vec model. On the other hand, learned vectors can be retrieved by calling *wv.syn0* attribute of the trained Word2Vec model.

There are several important training parameters such as *sg*, *vector size*, *window*, and *min_count* in Word2Vec model which must be considered. These parameters were finalized after multiple test experiments which produced optimal result. Two separate training algorithms, namely CBOW and Skip-gram were used to evaluate the performance of Word2Vec model on improved training dataset. Training parameter *sg* represents skip-gram. Therefore, if CBOW algorithm is used, *sg* is set to 0, otherwise 1. Output dimension of the vectors, *vector size,* was set to 50. The maximum distance between current word and predicted word in a sentence which is determined by the *window* was set to 5 for CBOW and 10 for Skip-gram. Minimum count, *min_count,* was set to 1 so that all words in the training dataset will have a vector representation.

The vector representations of vocabulary words were saved as pre-trained embedding weights using Pickle package. This helps to reduce experimental time as these weights can be reused on other deep learning models without having to train again. These pre-trained weights will not be updated via backpropagation algorithm during model training to reduce computation time. An overview of the training parameters for CBOW and Skip-gram is summarized in Table 8.

When Word2Vec model is used, training dataset must be re-tokenized without the <OOV> token for unknown words. The reason is Word2Vec model is not able to produce embedding weight for unknown words outside training dataset. Lastly, the trained tokenizer for Word2Vec model was saved using Pickle package. The source code for Word2Vec model can be found in Appendix C.

Table 8: Training Parameters of CBOW and Skip-gram.

	CBOW	**Skip-gram**
Input	Tokenized Training Dataset	
Vector Dimension	50	
Minimum Count	1	
Window Size	5	10
Sg	0	1

4.3.2 Pre-trained Word Vectors: GloVe

Instead of learning word vectors from scratch, training dataset in the form of sequence numbers was mapped to vector representation using pre-trained 50-dimensional GloVe vectors. If a vocabulary word in the trained tokenizer is found in the GloVe embedding dictionary, GloVe vector for the word will be extracted and saved in an embedding weight matrix. On the contrary, zero-word vector is saved in the embedding weight matrix if vocabulary word is not found in the GloVe embedding dictionary. Embedding weight matrix was used as pre-trained weights in the embedding layer of a neural network. Figure 19 shows the results of GloVe embedding on training datasets. Based on the figure, about a quarter of total vocabulary words in trained tokenizer were not found in GloVe embedding dictionary. The source code for GloVe vector can be found in Appendix D.

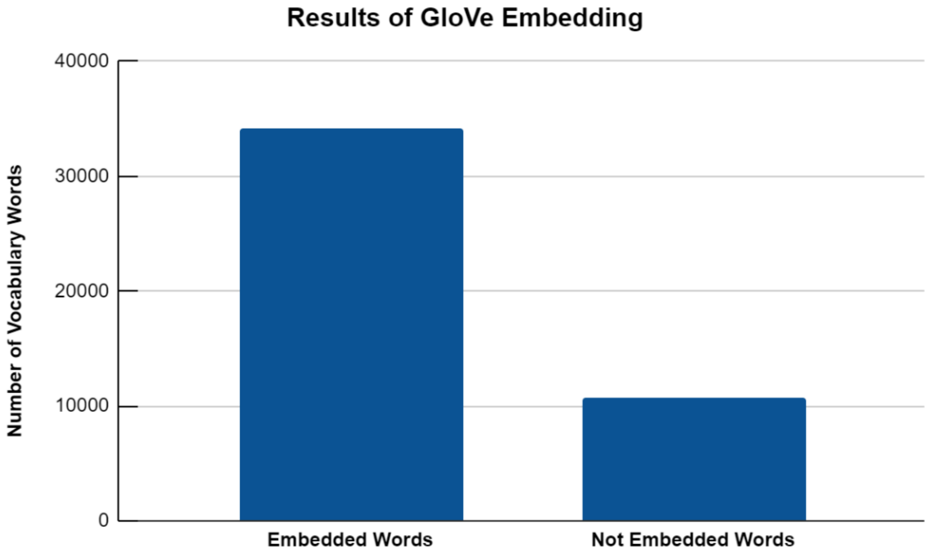


Figure 19: Results of GloVe Embedding.

4.3.3 Random Embedding Weights Initialization

Instead of using pre-trained embedding weights, the embedding weights can be randomly initialized based on a uniform distribution. When pre-trained embedding weights are not used, the weight parameter in an embedding layer is set as *weight* = *None* and the trainable parameter is set as *trainable* = *True*. On the other hand, if pre-trained weights are used, the weight parameter is set as *weight* = *[pre_trained_weights]* and trainable parameter is set as *trainable* = *False* so that pre-trained weights will not be updated via backpropagation algorithm after every training iteration.

The advantage of random embedding weights initialisation ensures every vocabulary word has non-zero embedding weights. The disadvantage is optimal weights may not be achievable because training vocabulary size is significantly lesser than the one used to produce pre-trained GloVe vectors.

Chapter 5: Experimental Results

In this section, neural network model setup and its training parameters will be described. Effects of word embeddings obtained from Word2Vec model, pre-trained GloVe vectors, and random embedding weights initialization used in neural networks will be evaluated. Lastly, classification results of LSTM, CNN, and Transformer will be evaluated and compared with past related works. The source code for LSTM, CNN, and Transformer model implementations and model training can be found in Appendix E and Appendix F, respectively.

5.1 Neural Network Model Setup

5.1.1 LSTM

The network architecture of LSTM is made up of embedding layer, LSTM layer, dense layer, dropout layer, and classification layer. In embedding layer, input dimension was set to the vocabulary size equals to the number of vocabulary words in trained tokenizer plus one. An additional token is included because sequence number 0 was used as padding. Output dimension was set to 50 because word tokens were mapped to 50-dimensional vector representations. Input length was set to 20 because all tweet sequences were normalized to a length of 20. Since normalization was performed by zero-padding tweet sequences which were shorter than 20, a sparse data was produced. Masking was set to *True* for zero sequence number so that the model does not treat zero-padding as inputs. If pre-trained embedding weights were used, the weight parameter will be set to *word2vec_weights* or *glove_weights*, otherwise set to none. When pre-trained weights were used, trainable parameter is set to false so that these weights will not be updated during model training process.

In LSTM layer, output dimension was set to 32 and dropout was set to 0.5 to prevent overfitting. The first dense layer contains 32 neurons with ReLU [38] activation function to provide non-linearity in the output. Dropout layer was used to randomly assign zero value to input units with a frequency of 0.5 at each training epoch. Finally, sigmoid activation function in the last dense layer was used to classify input tweets into spam or ham. All assigned values were chosen after multiple experiments to achieve optimality. The overall LSTM network structure is illustrated in Figure 20(a).

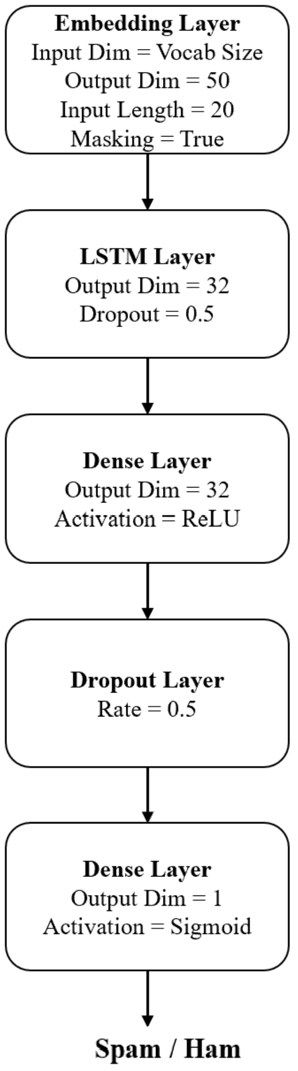


Figure 20(a): LSTM Network Structure.

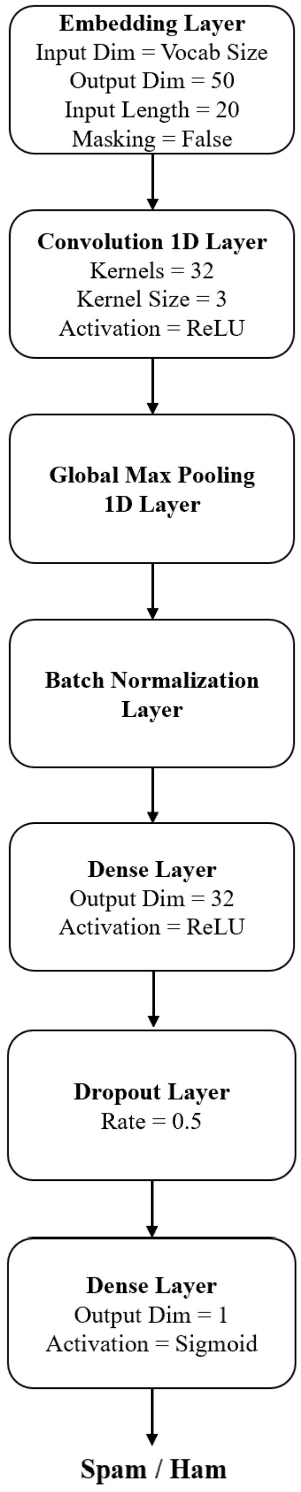


Figure 20(b): CNN Network Structure.

5.1.2 CNN

The network architecture of CNN is made up of embedding layer, convolution layer, global max pooling layer, batch normalization layer, dense layer, dropout layer, and classification layer. Parameters in embedding layer were set the same as those in LSTM embedding layer, except masking was set to *False* because convolution layer does not support masking. In convolution layer, 256 kernels with size 3 were used to convolve with the text data and output 256 feature maps. Activation function used in convolution layer is ReLU. Convolution layer was followed by global max pooling layer to reduce spatial dimensionality. Batch normalization provides some regularization by standardizing the inputs to the dense layer. First dense layer contains 32 neurons with ReLU activation function followed by dropout of 0.5. Finally, the last dense layer with sigmoid activation function classifies input tweets into spam or ham labels. The overall CNN network structure is illustrated in Figure 20(b).

5.1.3 Transformer

Transformer has different network architecture aligns to different types of NLP tasks as discussed in Section 3.3.4. In this project, Transformer with single encoder architecture implemented by [50] was selected for text classification task. Input text sequence was passed to token and position embedding layers to produce dense vectors and positional information of the words, respectively. The embedding outputs are passed to the Transformer block which is made up of multi-head attention layer, dropout layers, layer normalization layers, and fully-connected layers. The Transformer block is followed by global average pooling layer, fully-connected layers, and output layer. The network structure of Transformer is shown in Figure 21.

The model parameters used in this experiment were the same as the original implementation except two parameters, namely embedding dimension and activation function of output layer. Embedding dimension was changed from 32 to 50, and softmax activation function was changed to sigmoid activation function in the output layer.

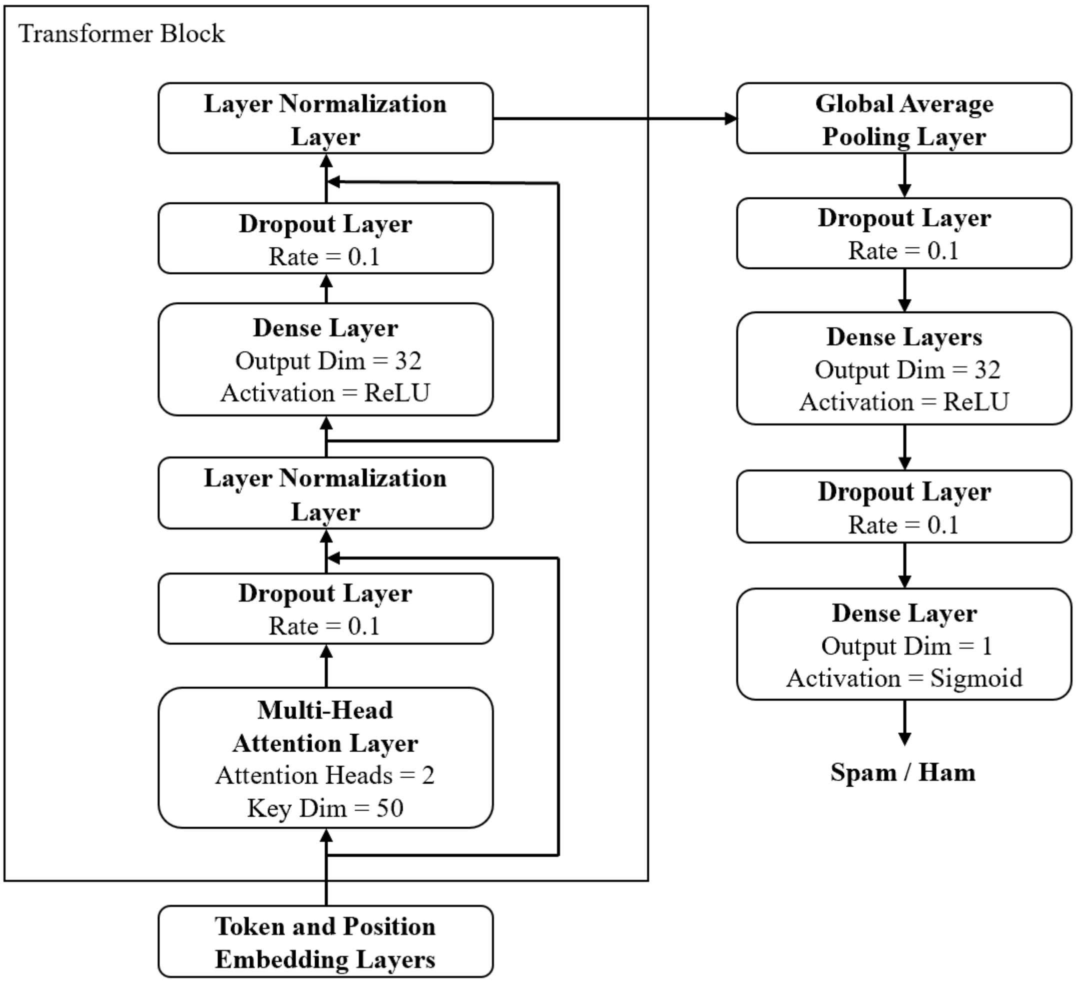


Figure 21: Transformer Network Structure.

5.1.4 Training Parameters

The overview of neural network structures and training parameters is summarized in Table 9. Neural networks were compiled using Adam optimizer [51] with a learning rate of 0.001 and errors were calculated by binary cross entropy loss function. The model is fitted on 20 training epochs and a batch size of 512 is used to segment the training data into smaller parts for the network to train.

Table 9: Network Structures and Training Parameters of LSTM, CNN, and Transformer.

	LSTM	**CNN**	**Transformer**
Network Structure	- Embedding Layer - LSTM Layer - Dense Layer - Dropout Layer - Dense Layer (Classification)	- Embedding Layer - Convolution Layer - Global Max Pooling Layer - Batch Normalization Layer - Dense Layer - Dropout Layer - Dense Layer (Classification)	- Token Embedding Layer - Position Embedding Layer - Multi Head Attention Layer - Dense Layer - Layer Normalization Layer - Dropout Layer - Global Average Pooling Layer - Dropout Layer - Dense Layer - Dropout Layer - Dense Layer (Classification)
Optimizer	Adam		
Learning Rate	0.001		
Loss Function	Binary Cross Entropy		
Number of Training Epochs	20		
Mini Batch Size	512		

5.2 Evaluation Metrics

Confusion matrix showed in Figure 22 is used to compute the evaluation metrics, namely accuracy, precision, recall, and f-score.

Actual Label

TN	FP
FN	TP

Predicted Label

Figure 22: Confusion Matrix.

True Negative (TN): Ham tweets that are correctly predicted.
True Positive (TP): Spam tweets that are correctly predicted.
False Negative (FN): Spam tweets that are falsely predicted.
False Positive (FP): Ham tweets that are falsely predicted.

The formulae to compute the evaluation metrics are as follows:

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

(Overall test accuracy of model)

$$Precision \ = \frac{TN}{TN + FN}$$

(Precision for Ham label)

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

(Precision for Spam label)

$$Specificity = \frac{TN}{TN + FP}$$

(Recall for Ham label)

$$Sensitivity = \frac{TP}{TP + FN}$$

(Recall for Spam label)

$$F - score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

(Weighted average of precision and recall)

5.3 Results and Discussion

Based on the classification results of LSTM obtained as shown in Figure 23, random embedding weights initialization outperformed the other word embeddings on all metrics. This result has highlighted the importance of assigning embedding weights to all vocabulary words, including unknown words. In pre-trained embedding weights, there is no embedding weights for unknown words which are not learnt by Word2Vec model and in GloVe vectors. As a result, this will cause poor generalization of model to new test data.

Apart from word embeddings, text processing and model's hyper-parameters tuning also have great impact on the classification results. As compared to the baseline result shown in Figure 24, there is improvement on all metrics for Word2Vec (CBOW) and random weights initialization. This improvement is the result of strategic text processing procedure and hyper-parameters tuning. Some modifications on the model's hyper-parameters include experimenting different output dimension of word embedding (50, 100, 200), number of LSTM layers, adding dropout layer, and adjusting dropout rate. In addition, training parameters such as learning rate, number of epochs, and mini-batch size (64, 128, 256, 512, 1024) were experimented.

Experimental results have shown that reducing the output dimension of word embedding have improved overfitting problem and reduced model training time. The increment in number of LSTM layers to 2 and using Bi-LSTM layer resulted in poor classification results. The reason is high model complexity forces the model to overfit the training data which leads to poor generalization to test data. Furthermore, the training time is doubled. Overfitting problem is solved by adding dropout layer or applying early stopping to terminate training when a criterion is reached. The latter is usually not ideal because optimal performance may be missed. For example, early stopping criterion is set to terminate training when validation loss does not improve after 5 epochs. However, a lower validation loss is observed at the 6th epoch. Furthermore, early stopping criterion is difficult to define. On the other hand, dropout rate adjustment is less complex. Larger number of training epochs allows the model to reach global minimum loss function. Mini-batch sizes of 64, 128, 256, 512, and 1024 were experimented. Batch sizes below 512 requires longer training time with minimal changes to training results. On the contrary, overfitting was increased for a batch size of 1024. Learning rate such as $5 \times 10^{-4}, 2 \times 10^{-3}$, and 3×10^{-3} were experimented to compare with the baseline learning rate of 1×10^{-3}. High learning rates resulted in higher training loss, whereas small learning rate has no improvement in training loss but longer training time. Therefore, baseline learning rate of 1×10^{-3} was preserved.

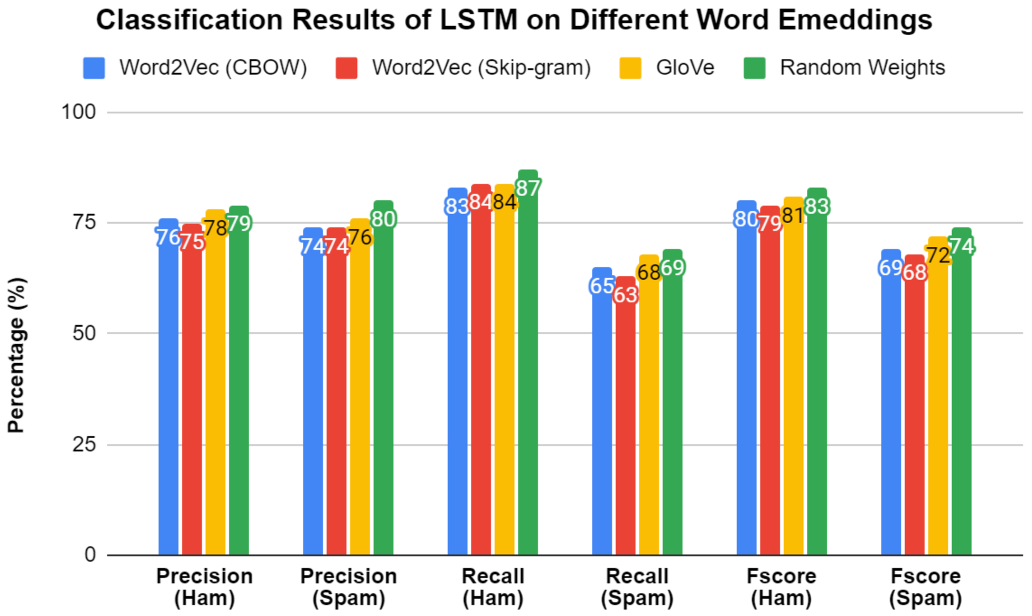


Figure 23: Classification Results of LSTM on Different Word Embeddings.

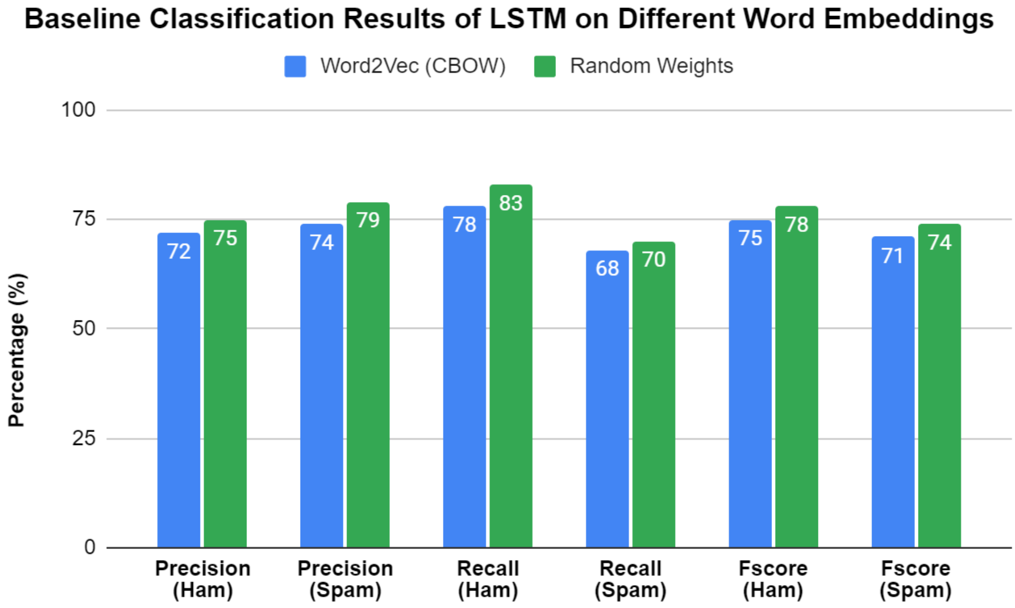


Figure 24: Baseline Classification Results of LSTM on Different Word Embeddings.

The improved classification results have high precisions at 79% and 80% for ham and spam labels, respectively. Specificity is high, but sensitivity is low. Low sensitivity score is due to high false negative rate. The number of misclassified and correctly classified test samples can be visualized using a confusion matrix as shown in Figure 25. Spam labels are denoted as 1 and ham labels are denoted as 0.

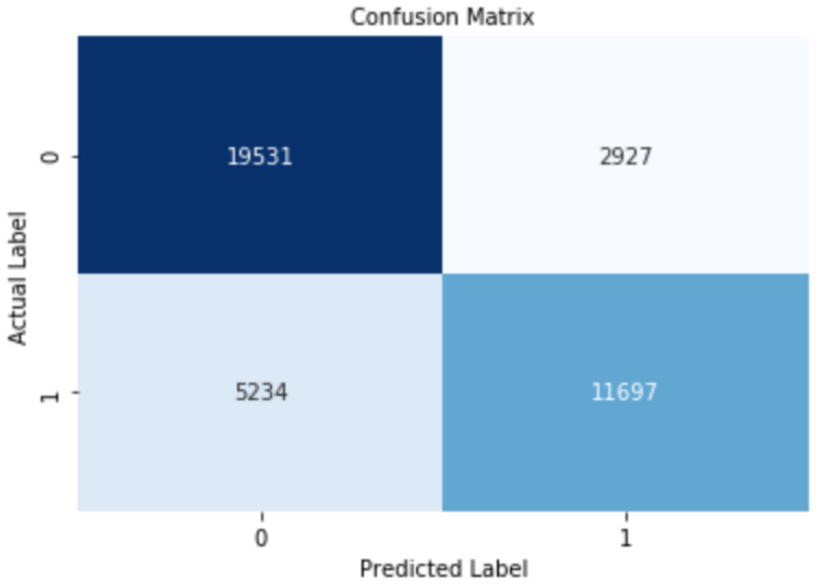


Figure 25: Confusion Matrix of LSTM with Random Embedding Weights Initialisation on Test Dataset.

Misclassified and correctly classified tweets were extracted and analyzed to investigate the model's prediction limitations. The source code for retrieving predictions can be found in Appendix G. Based on my analysis, examples of the most common false negatives are extracted and shown in Table 10. False negative tweets contain features such as 1) Retweet and multiple usermentions, 2) mixture of usermention, retweet, and URL, and 3) plain normal text without those common tweet features. These occurrences were expected because their features are matched to the features in true negative tweets as shown in Table 11. Despite false negatives, there are some common features which are correctly classified as spam tweets as shown in Table 12. Most common spam feature is having URL attached at the back of the sentence. Nevertheless, LSTM model can successfully classify spam tweets which do not contain common spam features such as URL and user mention.

Table 10: Examples of Common Spam Tweets Predicted as Ham Tweets.

Spam Tweets	Feature
retweet usermention ready usermention usermention usermention usermention usermention usermention usermention usermention	Retweet and multiple usermention
retweet usermention usermention next video interview series url	Mixture of usermention, retweet, and URL
legalize make sure everything business legit paper work	Plain normal text

Table 11: Examples of True Negative Tweets.

Spam Tweets	Feature
retweet usermention sexy lady usermention usermention usermention usermention usermention usermention usermention usermention	Retweet and multiple usermention
retweet usermention retweet usermention new york time cut newsroom job staff url	Mixture of usermention, retweet, and URL
must get back healthy eating plan far recently really feeling	Plain normal text

Table 12: Examples of True Positive Tweets.

Spam Tweets	Feature
movie maker peter king middle earth becomes knight url	URL at the back of sentence
retweet usermention heiress mistress sweetness lady two land url	Mixture of usermention, retweet, and URL
finally awesome making money program whole year money making business also change	Plain normal text

The second experiment was performed using CNN to compare with LSTM's classification results. In this experiment, only GloVe and random weights embedding were used because their performance on LSTM were superior to Word2Vec embedding. Based on Figure 26, CNN has poorer performance as compared to LSTM across all metrics. There was a significant drop of 14% in sensitivity when random weights embedding is used.

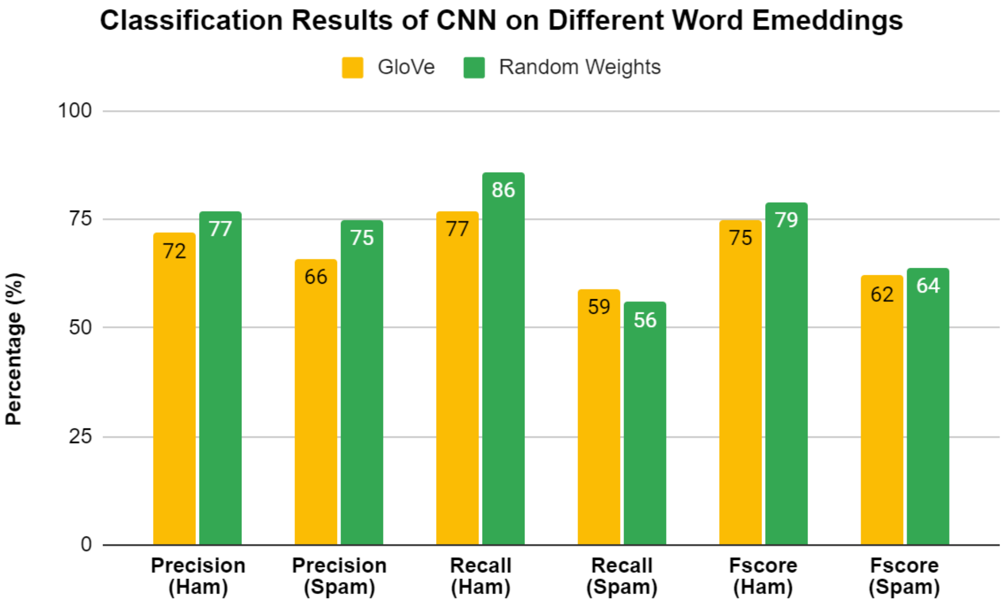


Figure 26: Classification Results of CNN on Different Word Embeddings.

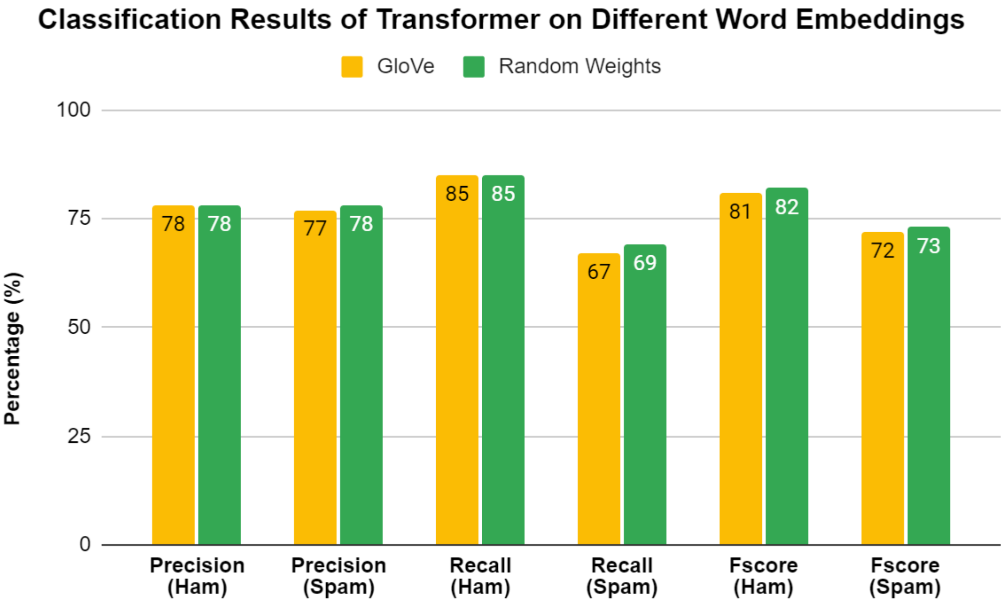


Figure 27: Classification Results of Transformer on Different Word Embeddings.

In the third experiment, GloVe and random weights embedding were used with Transformer. Based on Figure 27, Transformer outperformed CNN on the same datasets. On the other hand, the classification results of Transformer are comparable to LSTM in terms of precisions, specificity, and f-score. Although the sensitivity is 1-2% below LSTM's, Transformer has displayed potential to outperform LSTM before hyper-parameters tuning is performed. The downside of Transformer is its long training time. As shown in Figure 28, Transformer took about 1.5 to 3 times longer than LSTM and CNN for 20 training epochs.

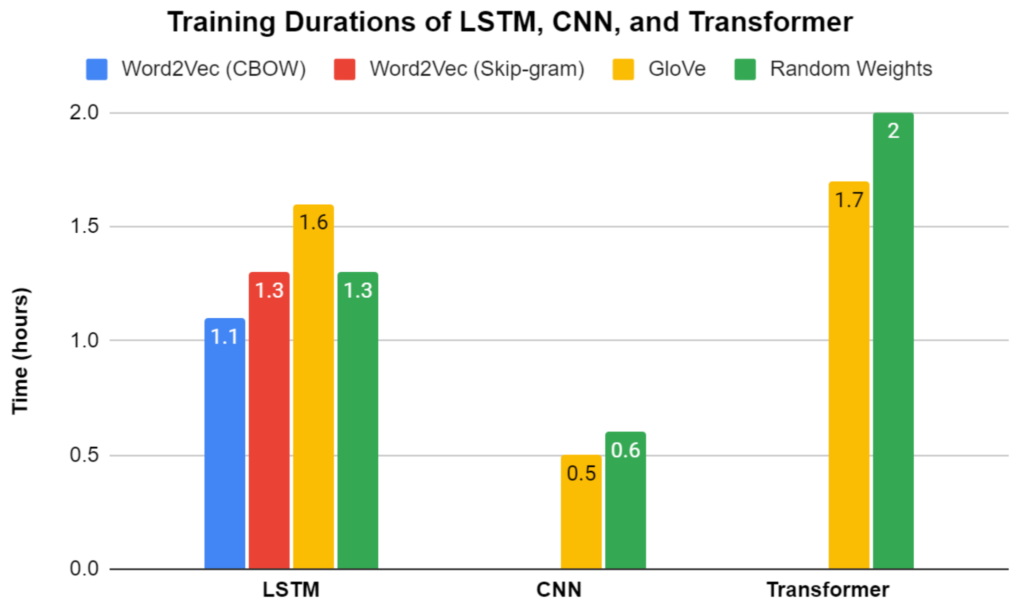


Figure 28: Training Durations of LSTM, CNN, and Transformer.

Based on the experimental results obtained, LSTM with random embedding weights initialisation achieved the best spam detection performance. It was decided as the best model based on its high spam precision and specificity with a score of 80% and 87%, respectively. Based on the metrics formulae, false positive is correlated to spam precision and specificity. In spam detection task, it is important to achieve low false positive score. This is to ensure that non-spam tweets will not be classified as spam tweets to prevent legitimate Twitter users being penalized. This does not imply that false negative is not important. In spam detection task, it is acceptable to allow some false negatives to pass through because Twitter users are still able to make the final judgement. On the contrary, it is disastrous to classify a non-spam tweet as spam tweet because this will cause a loss of important information. Therefore, LSTM with random embedding weights initialisation is my finalized proposed approach to Twitter spam detection.

Table 13: Comparison of Proposed Approach to Related Works.

Paper	**Model**	**Accuracy**	**Precision**	**Recall**	**F-Score**
Wang et al. [4]	Random Forest	N.A.	79.7%	65.6%	72.0%
Ashour et al. [8]	Logistic Regression	N.A.	79.5%	**79.4%**	79.4%
Martinelli et al. [21]	MLP with 3 hidden layers	N.A.	79.7%	73.4%	76.4%
Alom et al. [9]	CNN	76.7%	**86.5%**	73.8%	**79.7%**
Proposed Approach	LSTM	**79.0%**	80.0 %	69.0 %	74.0 %

Based on Table 13, my proposed approach has shown comparable results to related works in terms of accuracy, spam precision, sensitivity, and spam f-score. The main difference between my work and related past works is the size of datasets used. In my proposed work, million tweets were experimented. On the contrary, tens of thousands of tweets were used in past works [4, 8, 9, 21]. The distribution of number of tweets used in different approaches is shown in Figure 29. In general, deep learning applications require large amount of data to train a neural network for good generalization. Therefore, I used all the tweets in original dataset [31] instead of randomly sample a few thousands only.

Alom et al. [9] adopted an approach which randomly sampled 2,461 tweets from the original dataset for spam detection. Although this approach achieved a spam precision of 86.5%, it might not be able to generalize well on real world tweets because the model was trained on a small dataset. Wang et al. [4] and Ashour et al. [8] used similar approach to randomly sample a tweet from a Twitter user from the original dataset and produce close to 40,000 tweets for spam detection experiments. The size of the dataset used in these two approaches is relatively large given the fact that the experiment was conducted using a CPU processor instead of GPU. This is because larger dataset will require a GPU which has more computational power. Martinelli et al. [21] utilized the original dataset for spam detection experiment without data processing such as removing duplicates or redundant content features.

In my opinion, this is not an ideal approach for machine learning tasks because it can cause the model to overfit to the training dataset which leads to poor generalization to real word dataset.

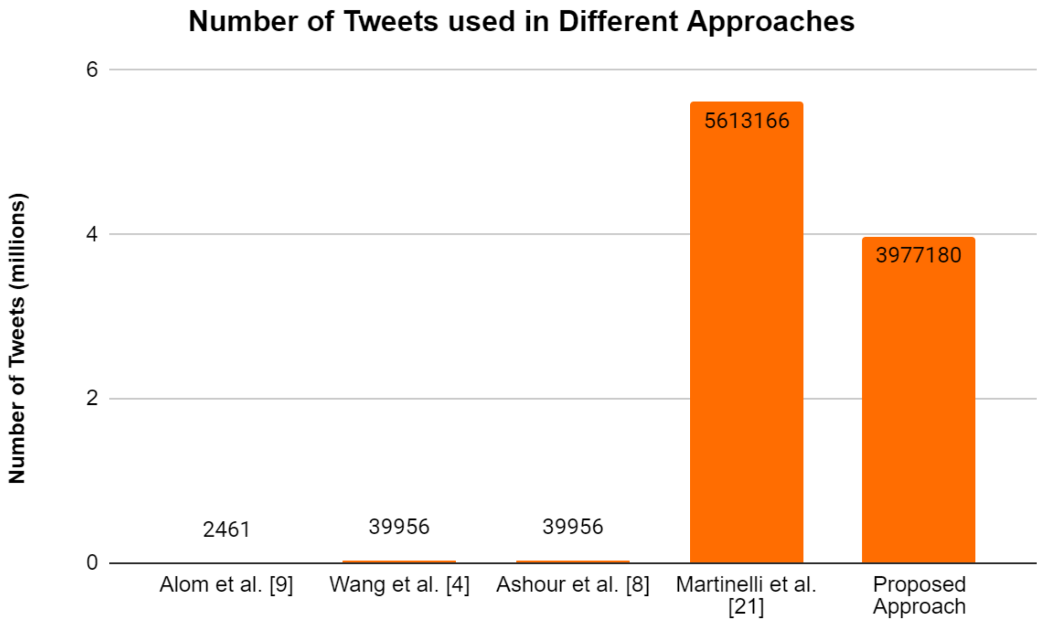


Figure 29: Distribution of Number of Tweets used in Different Approaches.

The advantage of using large training dataset is the ability to train a neural network from scratch. In comparison to past works whose utilized pre-trained word embeddings such as Word2Vec model and GloVe vectors to achieve good results, my proposed approach has shown improvement when random embedding weights initialisation was used instead. In my experiment, GloVe vectors were used as pre-trained word embedding weights too. The main difference between the GloVe vector implementation in my work and related text classification works is the dimension of GloVe vectors. For example, Alom et al. [9] used a vector dimension of 200 to represent each word in 200 dimensional space. On the contrary, a vector dimension of 50 was used in my approach because this dimension helps to reduce overfitting problem and computational resources. Lastly, my proposed approach showed that deep learning model, LSTM, outperformed machine learning model such as Random Forest classifier in spam detection [4].

Chapter 6: Conclusion and Future Work

6.1 Conclusion

In this project, three deep learning models such as LSTM, CNN, and Transformer were used to evaluate spam detection on Twitter dataset. Experimental results have shown that LSTM and Transformer performed better than machine learning model such as Random Forest classifier in terms of spam precision, recall, and f-score. The size of training data used in my proposed approach is about 100 times larger than the size used in past works. As a result, my proposed approach has better generalization to new tweets with a spam precision of 80%. Random embedding weight initialisation has shown better spam detection performance than pre-trained word embeddings because it can assign non-zero dense vectors to unknown words for machine learning. In conclusion, LSTM with random embedding weight initialisation has the best spam detection performance in terms of spam precision and specificity.

6.2 Future Work

The spam detection performance can be further improved by exploring other variants of Transformer model. The attention feature in Transformer has shown that it can put focus on word positions and word importance in a text sequence. The default Transformer network architecture has achieved a spam precision of 78% which is close to my proposed model, LSTM. Therefore, Transformer model's hyperparameters can be tuned and explore further to improve spam detection on Twitter.

Furthermore, the spam detection performance of pre-trained LSTM model in this project can be evaluated on latest tweets. The dataset used in this project was originated from December 30, 2009 to August 2, 2010. Since spammers' techniques are constantly evolving, it is critical to gather latest tweets via web crawler to maintain spam detection efficiency.

References

[1] *Twitter*, TwitterInc., 25 Jan 2021. [Online]. Available: https://twitter.com

[2] R. Yoel and H. Del. "How Twitter is Fighting Spam and Malicious Automation." https://blog.twitter.com/en_us/topics/company/2018/how-twitter-is-fighting-spam-and-malicious-automation.html (accessed 1 Nov, 2020).

[3] X. Ban, C. Chen, S. Liu, Y. Wang, and J. Zhang, "Deep-learnt features for Twitter spam detection," in *2018 International Symposium on Security and Privacy in Social Networks and Big Data (SocialSec)*, 10-11 Dec. 2018 2018, pp. 208-212, doi: 10.1109/SocialSec.2018.8760377.

[4] B. Wang, A. Zubiaga, M. Liakata, and R. Procter, "Making the most of tweet-inherent features for social spam detection on Twitter," *arXiv preprint arXiv:1503.07405,* 2015.

[5] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, 2010, vol. 6, no. 2010, p. 12.

[6] V. Vapnik, I. Guyon, and T. Hastie, "Support vector machines," *Mach. Learn,* vol. 20, no. 3, pp. 273-297, 1995.

[7] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001/10/01 2001, doi: 10.1023/A:1010933404324.

[8] M. Ashour, C. Salama, and M. W. El-Kharashi, "Detecting Spam Tweets using Character N-gram Features," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*, 18-19 Dec. 2018 2018, pp. 190-195, doi: 10.1109/ICCES.2018.8639297.

[9] Z. Alom, B. Carminati, and E. Ferrari, "A deep learning model for Twitter spam detection," *Online Social Networks and Media,* vol. 18, p. 100079, 2020/07/01/ 2020, doi: https://doi.org/10.1016/j.osnem.2020.100079.

[10] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine,* vol. 13, no. 3, pp. 55-75, 2018.

[11] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks,* vol. 3361, no. 10, p. 1995, 1995.

[12] J. L. Elman, "Finding structure in time," *Cognitive science,* vol. 14, no. 2, pp. 179-211, 1990.

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

[14] A. Vaswani *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762,* 2017.

[15] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, 1998, vol. 752, no. 1: Citeseer, pp. 41-48.

[16] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician,* vol. 46, no. 3, pp. 175-185, 1992.

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

[18] J. S. Cramer, "The origins of logistic regression," 2002.

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

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

[21] F. Martinelli, F. Mercaldo, and A. Santone, "Social Network Polluting Contents Detection through Deep Learning Techniques," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 14-19 July 2019 2019, pp. 1-10, doi: 10.1109/IJCNN.2019.8852080.

[22] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[23] F. Wei and U. T. Nguyen, "Twitter Bot Detection Using Bidirectional Long Short-Term Memory Neural Networks and Word Embeddings," in *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 12-14 Dec. 2019 2019, pp. 101-109, doi: 10.1109/TPS-ISA48467.2019.00021.

[24] A. M. Founta, D. Chatzakou, N. Kourtellis, J. Blackburn, A. Vakali, and I. Leontiadis, "A Unified Deep Learning Architecture for Abuse Detection," presented at the Proceedings of the 10th ACM Conference on Web Science, Boston, Massachusetts, USA, 2019. [Online]. Available: https://doi.org.remotexs.ntu.edu.sg/10.1145/3292522.3326028.

[25] K. Archchitha and E. Y. A. Charles, "Opinion Spam Detection in Online Reviews Using Neural Networks," in *2019 19th International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2-5 Sept. 2019 2019, vol. 250, pp. 1-6, doi: 10.1109/ICTer48817.2019.9023695.

[26] J.-H. Wang, T.-W. Liu, X. Luo, and L. Wang, "An LSTM approach to short text sentiment classification with word embeddings," in *Proceedings of the 30th conference on computational linguistics and speech processing (ROCLING 2018)*, 2018, pp. 214-223.

[27] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing,* vol. 70, no. 1-3, pp. 489-501, 2006.

[28] Z. Ding, R. Xia, J. Yu, X. Li, and J. Yang, "Densely connected bidirectional lstm with applications to sentence classification," in *CCF International Conference on Natural Language Processing and Chinese Computing*, 2018: Springer, pp. 278-287.

[29] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A C-LSTM neural network for text classification," *arXiv preprint arXiv:1511.08630,* 2015.

[30] D. G. Kumar, M. K. Rao, and K. Premnath, "A Recurrent Neural Network Model for Spam Message Detection," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 10-12 June 2020 2020, pp. 1042-1045, doi: 10.1109/ICCES48766.2020.9137940.

[31] K. Lee, B. Eoff, and J. Caverlee, "Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter," in *ICWSM*, 2011.

[32] S. Bird, E. Klein, and L. Edward, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[33] M. F. Porter, "An algorithm for suffix stripping," *Program,* 1980.

[34] G. A. Miller, "WordNet: a lexical database for English," *Commun. ACM,* vol. 38, no. 11, pp. 39–41, 1995, doi: 10.1145/219717.219748.

[35] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *arXiv preprint arXiv:1309.4168,* 2013.

[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, no. 6088, pp. 533-536, 1986/10/01 1986, doi: 10.1038/323533a0.

[37] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE international symposium on circuits and systems*, 2010: IEEE, pp. 253-256.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[39] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials,* vol. 13, no. 4, pp. 27-31, 1994, doi: 10.1109/45.329294.

[40] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning* (6.2.2.3 Softmax Units for Multinoulli Output Distributions, no. 2). MIT press Cambridge, 2016, pp. 180-184.

[41] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550-1560, 1990.

[42] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*, 1995: Springer, pp. 195-201.

[43] C. Olah. "Understanding LSTM Networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed 31 March, 2021).

[44] G. Van Rossum, "The Python Library Reference, release 3.8. 2," *Python Software Foundation,* p. 36, 2020.

[45] W. McKinney, *Data Structures for Statistical Computing in Python* (Proceedings of the 9th Python in Science Conference). 2010.

[46] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238,* 2013.

[47] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467,* 2016.

[48] *Keras*, F. Chollet, 2015. [Online]. Available: https://keras.io

[49] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *In Proceedings of the LREC 2010 workshop on new challenges for NLP frameworks*, 2010: Citeseer.

[50] A. Nandan. "Text Classification with Transformer." https://keras.io/examples/nlp/text_classification_with_transformer/ (accessed 04 April, 2021).

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