WORD2VEC HYPERPARAMETERS TUNING FOR EFFICIENT DEEP NEURAL NETWORK CLASSIFICATION

TASNEEM GAMAL ABDELLAH MOHAMMED ALY

UNIVERSITI KEBANGSAAN MALAYSIA

PENGGUNAAN HYPERPARAMETER WORD2VEC UNTUK KLASIFIKASI RANGKAIAN NEURAL YANG BERKESAN

TASNEEM GAMAL ABDELLAH MOHAMMED ALY

PROJECT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE (ARTIFICIAL INTELLIGENCE)

FACULTY OF SCIENCE AND TECHNOLOGY
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2021

PENGGUNAAN HYPERPARAMETER WORD2VEC UNTUK KLASIFIKASI RANGKAIAN NEURAL YANG BERKESAN

TASNEEM GAMAL ABDELLAH MOHAMMED ALY

PROJEK YANG DIKEMUKAKAN UNTUK MEMENUHI SEBAHAGIAN DARIPADA SYARAT MEMPEROLEH UNTUK IJAZAH SARJANA SAINS KOMPUTER (KECERDASAN BUATAN)

FAKULTI TEKNOLOGI SAINS DAN SAINS MAKLUMAT
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2021

DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries, which have been duly acknowledged.

05 October 2021 TASNEEM GAMAL ABDELLAH MOHAMMED ALY
P106220

ACKNOWLEDGEMENT

First and foremost, praise be to Almighty Allah for all His blessings for giving me patience and good health throughout the duration of this master research.

It is my pleasure to express my sincere gratitude and deepest thanks to my supervisor Prof. Dr Nazlia Omar for her encouragement, endless support, and guidance provided during my study at UKM. Her valuable discussions, comments, and suggestions have greatly improved the content and the presentation of this thesis.

Moreover, I am grateful to all the UKM staff and works,

I would like to thank all postgraduate students of UKM for their help, friendship, and creating a pleasant working environment throughout my years in UKM.

Last but not least, To my dearest family, my beloved Friends.

ABSTRACT

Word embedding is an emerging topic in recent years. The application of deep learning in word embedding is expanding and being used by top corporations like Google and Facebook in various applications such as search and recommendation engines. Some several techniques and libraries have been published like Word2Vec, FastText and GloVe. Word embedding usually defines the word by its accompanying words. Researchers often choose the default hyperparameters mentioned in the original paper for word embedding by Mikolov, such as Word2Vec. The default hyperparameters have different effects on different downstream tasks. One simply cannot use the same hyperparameters for word embedding for downstream classification tasks and semantic analogy downstream tasks. The default and recommended hyperparameters work the best with specific downstream tasks of analogy and semantic meaning. Complex downstream tasks such as classification might require different hyperparameters. This research aims to study hyperparameters' impact on the downstream task of classification. This will help the researchers to identify the combination of the word embedding hyperparameters that will fit their downstream tasks. In this research, different deep neural network (DNN) classifiers were used for the downstream classification tasks such as Recurrent Neural Network, Convolution Neural Network and Long Short-Term Memory Neural Network. The DNN networks were fed by different word embedding vector spaces trained using different hyperparameters. The results show that by navigating the hyperparameters space, the downstream task of classification accuracy can increase by 4.8%. The results also show that LSTM networks are more resilient to changes in the hyperparameters. However, CNN with a specific combination of word embedding hyperparameters can achieve the highest accuracy in the classification downstream task.

ABSTRAK

Penyisipan perkataan adalah topik yang muncul dalam beberapa tahun terakhir. Penerapan pembelajaran mendalam dalam penyisipan kata semakin berkembang dan digunakan oleh syarikat teratas seperti Google dan Facebook dalam berbagai aplikasi seperti mesin pencari dan cadangan. Beberapa teknik dan perpustakaan telah diterbitkan, seperti Word2Vec, FastText dan GloVe. Penyisipan kata biasanya mentakrifkan kata dengan kata-kata yang berhampiran. Penyelidik sering memilih hiperparameter lalai yang disebut dalam kertas asal oleh Mikolov untuk penyisipan kata seperti Word2Vec. Hiperparameter lalai mempunyai kesan yang berbeza pada tugas hiliran yang berbeza. Penyelidik tidak boleh menggunakan hiperparameter yang sama untuk penyisipan kata untuk klasifikasi tugas hilir dan untuk tugas hilir analogi semantik. Hiperparameter lalai dan yang disyorkan berfungsi paling baik dengan tugas hilir tertentu analogi dan makna semantik. Tugas hiliran yang kompleks seperti klasifikasi mungkin memerlukan hiperparameter yang berbeza. Penyelidikan ini bertujuan untuk mengkaji kesan hiperparameter terhadap tugas klasifikasi hiliran. Ini akan membantu para penyelidik untuk mengenal pasti kombinasi perkataan penyisipan hiperparameter yang sesuai dengan tugas hiliran mereka. Dalam penyelidikan ini pengkelasan Rangkaian Neural Mendalam (RNM) yang berbeza digunakan untuk tugas klasifikasi hilir seperti Rangkaian Neural Berulang (RNB), Rangkaian Neural Konvolusi (RNK) dan Rangkaian Neural Memori Jangka Panjang (RNMJP). Rangkaian JNJ diberi input dengan ruang vektor penyisipan kata yang terlatih menggunakan hiperparameter yang berbeza. Hasilnya menunjukkan dengan menavigasi ruang hiperparameter, tugas hiliran ketepatan klasifikasi meningkat sebanyak 4.8%. Hasilnya juga menunjukkan bahawa rangkaian RNMJP lebih tahan terhadap perubahan hiperparameter. Walau bagaimanapun, CNN dengan kombinasi khusus hyperparameter penyisipan kata dapat mencapai ketepatan tertinggi dalam tugas hiliran klasifikasi.

TABLE OF CONTENTS

		Page
DECLARATION		**iii**
ACKNOWLEDGEMENT		**iv**
ABSTRACT		**v**
ABSTRAK		**vi**
TABLE OF CONTENTS		**vii**
LIST OF TABLES		**x**
LIST OF ILLUSTRATIONS		**xi**
LIST OF ABBREVIATIONS		**xii**
CHAPTER I	**INTRODUCTION**	
1.1	Background	1
1.1	Problem Statement	1
1.2	Research Objectives	2
1.3	Research Scope	3
1.4	Significance Of Project	3
1.5	Report Overview	4
1.6	Summary	4
CHAPTER II	**LITERATURE REVIEW**	
2.1	Introduction	5
1.2	Word Embedding	5
	1.2.1 One-Hot Representation	6
	1.2.2 Distributed Representation	6
1.3	Word2Vec	8
	2.1.1 Word2Vec Limitations	9
	1.3.1 Comparison between TF_IDF and Word2Vec	9
2.2	Different Techniques For Word Embedding	10
2.3	Neural Networks	12
	2.3.1 Recurrent Neural Network	12
	2.3.2 Convolutional Neural Network	12
	2.3.3 Long Short-Term Memory	13
2.4	Related Work	13

	2.4.1	Tuning Word2Vec for Large Scale Recommendation Systems	15
	2.4.2	Word2Vec Applied to Recommendation: Hyperparameters Matter	16
	2.4.3	Word2Vec: Optimal Hyper-Parameters and Their Impact on NLP Downstream Tasks	16
	2.4.4	Learning Quality Improved Word Embedding with Assessment of Hyperparameters	16
2.5	Summary		18
CHAPTER III	**RESEARCH METHODOLOGY**		
3.1	Introduction		19
3.2	Research design		19
3.3	Dataset		20
3.4	Pre-processing		22
	3.4.1	Data Loading	22
	3.4.2	Data Selection	22
	3.4.3	Ranking Buckets	24
	3.4.4	Data Concatenation	24
	3.4.5	Text Cleaning	25
	3.4.6	Data Classification Preparation	25
3.5	AWS cloud environment experimental settings		26
	3.5.1	What is AWS Cloud Environment?	26
	3.5.2	AWS S3 Bucket	26
	3.5.3	AWS SageMaker	27
	3.5.4	AWS Glue	27
	3.5.5	AWS Athena	28
3.6	The Hyperparameter Tuning of Word2Vec		28
	3.6.1	Implementation Example of Word2Vec	30
	3.6.2	Continuous Bag-Of-Words Vs Skip-Gram	30
	3.6.3	Word2Vec Similarity Feature	31
	3.6.4	Word Embedding Visualization	32
	3.6.5	Word2Vec Hyperparameter	32
3.7	Classification		37
	3.7.1	Recurrent Neural Network parameters and layers	37
	3.7.2	Convolutional Neural Network parameters and layers	40
	3.7.3	Long Short Term Memory parameters and layers	43
3.8	Evaluation		46
	3.8.1	Accuracy	46
	3.8.2	F1 Score	46
	3.8.3	Precision	47

	3.8.4	Recall	47
3.9	Summary		47
CHAPTER II	**EXPERIMENTAL RESULTS**		
4.1	Introduction		49
4.2	Experiment Setting		49
	4.2.1	Random Generation of Hyperparameters	50
	4.2.2	Experiment AWS architecture	51
4.3	Word2Vec results		53
	4.3.1	DNN Performance and Hyperparameters Variation	54
	4.3.2	Best Hyperparameters Combinations	55
	4.3.3	Worst Hyperparameters Combinations	56
4.4	Dimension Size And Avg F1 Score		58
4.5	CBOW Vs Skip-Gram Average F1 Score		58
4.6	Summary		59
CHAPTER IV	**CONCLUSION AND FUTURE WORK**		
5.1	Contribution of the Study		60
5.2	Objectives Achievement		60
5.3	Limitations		60
5.4	Future Work		61
5.5	Conclusion		61
REFERENCES			**62**

LIST OF TABLES

Table No.		Page
Table 2.1	Summary of previous related work	17
Table 3.1	Dataset selected columns explanation	23
Table 3.2	An example of the dataset columns	23
Table 3.3	An example of the dataset after concatenated	24
Table 3.4	Statistic of the dataset	26
Table 3.5	Accuracy metrics	46
Table 4.1	Programming package	50
Table 4.2	Experimental setting	50
Table 4.3	Random range value of the hyperparameters	51
Table 4.4	F1 score value (min, max, average, and median) for (RNN, CNN, LSTM)	54
Table 4.5	The best performer combinations of Gensim Word2Vec hyperparameters	57
Table 4.6	The worst performer combinations of Gensim Word2Vec hyperparameters	57

LIST OF ILLUSTRATIONS

Figure No.		Page
Figure 2.1	One-Hot Representation of the word SAT in the small given text.	6
Figure 2.2	Distributed representation of the word 'SAT'	8
Figure 3.1	Amazon Customer Reviews dataset sample	23
Figure 3.2	An Example of a customer review after text cleaning	25
Figure 3.3	Neural Network represents Word2Vec Neural Network	29
Figure 3.4	CBOW vs Skip-Ngram	31
Figure 3.5	TensorFlow data visualization of word embedding	32
Figure 3.6	Illustrates the window size of Word2Vec	34
Figure 3.7	Simple RNN module summary	38
Figure 3.8	Simple RNN module architecture	39
Figure 3.9	Simple CNN module summary	41
Figure 3.10	Simple CNN module architecture	42
Figure 3.11	LSTM module summary	44
Figure 3.12	LSTM module architecture	45
Figure 4.1	AWS end-to-end system architecture	53
Figure 4.2	Visual representation of F1 score value (min, max, average and median) for (RNN, CNN, LSTM)	54
Figure 4.3	F1 score variations for RNN, CNN and LSTM for different combinations of Word2Vec hyperparameters	55
Figure 4.4	Dimension size impact on the overall average F1 score	58

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Service
BERT	Bidirectional Encoder Representations From Transformers
RoBERTa	Optimized BERT Pre-training Approach
CBoW	Continuous Bag Of Word
CNN	Convolutional Neural Network
DNN	Deep Neural Networks
GloVe	Global Vectors For Word Representation
IoT	Internet Of Things
LSTM	Long Short Term Memory
NLP	Natural Language Processing
RNN	Recurrent Neural Network
SA	Sentiment Analysis
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
UKM	Universiti Kebangsaan Malaysia
URL	Uniform Resource Locator
LDA	Latent Dirichlet Allocation
IMDb	Internet Movie Database

CHAPTER I

INTRODUCTION

1.1 BACKGROUND

Neural information retrieval is extensively being used in web search engines and social media data processing. Its accuracy degradation occurs due to the nature of using unsupervised machine learning techniques in processing its unstructured corpus. Sensitive scientific studies limit neural information retrieval applications due to the degradation of its accuracy. Search engines and social media frameworks recommendation systems are examples of a successful approximation of neural information retrieval.

Word Embedding is a dense representation of words in a low dimensional vector space. As a concept, it can hold the same meaning as word vectors or distributed word representation. Word embedding is a mathematical representation of words into numbers and vectors that can be processed in several vector spaces. Processing text by neural network requires a form of numerical transformation of text, and in this case, it will be numerical vectors. Works by (Mikolov et al. 2013d, 2013a, 2013b) discussed and presented promising solutions to enable the machine to learn high quality distributed vector representations to capture syntactic and semantic relations among words in the corpus.

1.1 PROBLEM STATEMENT

Word embedding hyperparameters are often taken with fixated values from the literature. The hyperparameters values mention in the literature by (Mikolov et al. 2013c) are usually designed for a specific downstream task such as semantic analogies. Understanding Word2Vec fully might be overwhelming and time-consuming, specifically for researchers who are focusing on other tasks such as

combining word2vec with SVM or others. Accordingly, shed light on this problem will allow the scientific community to have the means to understand how important the hyperparameters tuning task is for the downstream tasks.

Some researchers assume that the values of the same hyperparameters can be reused for other downstream tasks such as classification. Former publications (Barkan & Koenigstein 2016) and (Grbovic et al. 2016) that used Word2Vec rarely discussed the values of the hyperparameters. This often led to a decrease in the accuracy presented in the downstream task such as classification. This work demonstrates the accuracy degradation when hyperparameters tuning for Word2Vec are ignored. The main reason for the inaccuracy decline is the low quality of the word embeddings fed to the classification network (Barkan & Koenigstein 2016; Grbovic et al. 2016). This work will also demonstrate how the quality word embedding affects the overall accuracy of the downstream task.

A high-quality word embedding can be used for downstream classification tasks such as sentiment analysis. Generating high-quality word embeddings depends on the quality of the input corpus and the hyperparameters of the model (Caselles-Dupré et al. 2018). The definition of high-quality word embeddings varies between different downstream tasks, which means that the high-quality word embeddings designed to be used in the semantic analogy are different from the high-quality word embeddings that are designed to be used in classification.

Analyzing different combinations of word embeddings hyperparameters and their effect contribute towards the classification tasks. This can be measured by the accuracy metric of the classification neural network. Thus, examining the impact of the change of hyperparameters of the word embeddings on the classification tasks is necessary.

1.2 RESEARCH OBJECTIVES

The objectives of this research can be summarised as follows:

i. To find the best hyperparameters combination to generate high-quality word embeddings on the classification task.

ii. To measure the quality of the generated word embeddings via classification task performance using several types of Deep Neural Networks such as Recurrent Neural Network, Convolutional Neural Network and Long-Term Short Memory.

1.3 RESEARCH SCOPE

The hyperparameter tuning of Word2Vec will be the primary focus of this study. We evaluate the impact of changing the combination values of Word2Vec hyperparameters on the downstream task of classification by using different Deep Neural Network classifiers of Recurrent Neural Network, Convolution Neural Network and Long Short-Term Memory Neural Network. The main research scope is to improve the accuracy of the classification downstream task by modifying hyperparameters combinations. This research uses the Amazon Custom Reviews dataset that contains textual data representing the customers' opinion, and it is a star rating for various products that were sold on Amazon.com.

1.4 SIGNIFICANCE OF PROJECT

The results of this work can help researchers understand the customer product experience and construct a high-quality understanding of the evaluation of the natural language opinion and the variation in the perception of different products. This has been significantly used for marketing research and promotional and target marketing and addresses several problems such as bias in reviews and different opinions. The study contributes directly to the empirical testing of different combinations of Word2Vec hyperparameters and their impact on the classification tasks. Researchers can replicate and carefully select the best hyperparameters combinations of Word2Vec that suits their experimental design.

1.5 REPORT OVERVIEW

The methodologies and their stages would be described in the following:

i. Chapter I: this introduces the research idea and describes the study's background. Furthermore, the research gap is stated in the problem statement, which focuses on the problem that will be solved as a result of this study's contribution. Furthermore, the study's objectives and the research scope as well.

ii. Chapter II investigates the literature review of the study in which word embedding is being described. This chapter highlights the other techniques used and comparison to Word2Vec proposed technique. Finally, a critical analysis of the related work is being provided by this chapter.

iii. Chapter III illustrated the research methodology, which consists of the dataset used in this research, pre-processing tasks for the dataset, AWS cloud environment used for processing the data, proposed Word2Vec method has been declared by explaining the baseline methods, hyperparameter, classification and the evaluation.

iv. Chapter IV discusses the experimental results in further detail. After identifying the random generation of the hyperparameters of the experiments, this chapter displays the results of the model.

v. Chapter V summarises the research by providing a final summary that summarises all the thesis. In addition, the contribution of the study, Research limitation and Finally, the future works.

1.6 SUMMARY

This study examines the improvement in the accuracy of the classification downstream task by modifying hyperparameters combinations Word2Vec for the Amazon Custom Reviews dataset.

CHAPTER II

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter will describe different word embedding techniques. The primary approach that this research is focusing on is Word2Vec. Then will discuss the details of word embedding as a neural network and how it is used in different downstream tasks. The chapter will also discuss word embedding techniques, compare them to Word2Vec, and list word embedding usage. Lastly, a brief review of some related works to hyperparameters tunning with Word2Vec for different downstream tasks is also reviewed. This chapter is organized as follows: Section 2.2 introduces the definition of word embedding and its representation. Then will discuss Word2Vec in further detail. It begins with the definitions, limitations, and comparison Word2Vec and TF-IDF, which can be seen in Section 2.3. In addition, Section 2.4 illustrates different techniques of word embedding in more detail (like GloVe and FastText). Finally, in Section 2.5, Related works to Word2Vec hyperparameter tunning applied to a different downstream task or recommendation systems. Section 2.6 gives a summary of this chapter.

1.2 WORD EMBEDDING

The basic concept of word embedding is to transform the word representation into numerical values that can be processed and compared to the context and extract semantic and analogies from the corpus. The following section will discuss two different basic techniques that transform the corpus into numerical representations. The techniques are One-Hot representation and distributed distribution.

1.2.1 One-Hot Representation

The One-Hot representation is the simplest method of representing the word in the corpus. For each unique word of the corpus, there is one vector that represents this word. The vector is composed of two binary values [0,1]. Based on the order of the corpus and the word position in the corpus, the relevant bit in the vector will be turned into one. Furthermore, every other bit will be turned to zero. Figure 2.1 illustrates the vector projection of the word representation in the sentence.

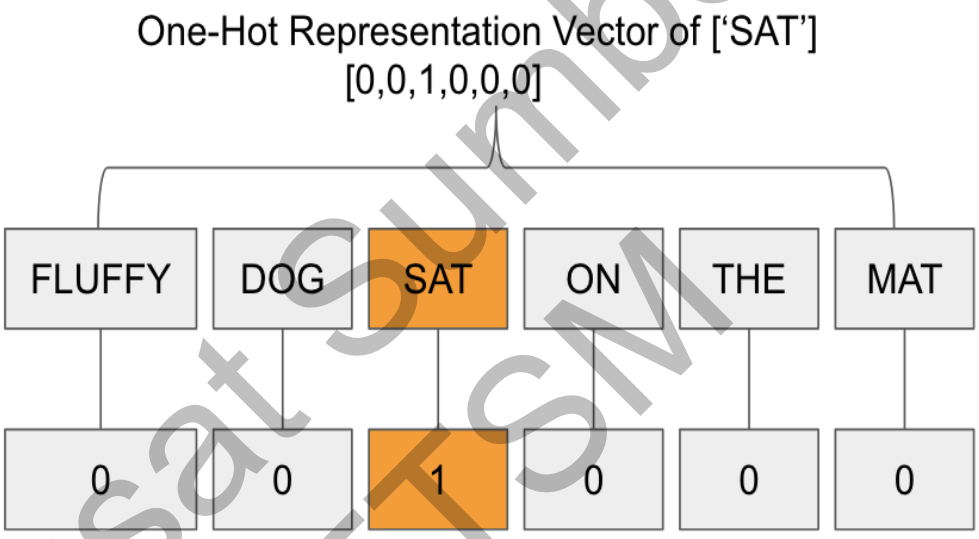


Figure 2.1 One-Hot Representation of the word SAT in the small given text.

1.2.2 Distributed Representation

The distributed representation of a word is a mathematical representation of the word vector in the corpus. One-Hot representation does not contain valuable information about the word except for its index, which might be considered arbitrary as it may or may not be determined by the word's position in a particular context. However, the distributed representation of the word contains different values for each word in the corpus and its relation to other words in the corpus. The distributed representation can be used mathematically to extract analogies and semantics of the word given its context. Figure 2.2 illustrates the distributed representation of the word 'SAT'. The

mathematical numbers in the vector are challenging to be interpreted by a human. The vector space of all the distributed representations of the words in the corpus is interrelated. Mathematical operations in the space vector result in different meanings like semantic, interchangeability, relatedness, and analogies.

The advantage of distributed representation versus One-Hot representation is the dimensionality and the learning of semantics of the words. The One-Hot representation is very high in dimensionality as it explicitly maps every unique word in the corpus. Perform operations like the co-occurrence matrix will be gigantic in the case of a big corpus. However, the distributed representation will reduce the dimensionality and decrease the computational expense of the processing matrix. It is important to note that generating the vector space of distributed representation is computationally expensive. However, once obtained, processing it will be significantly fast even for online computation for production applications**.** The number of dimensions of distributed representation can vary, but ideally, it is from one hundred to three hundred dimensions. The meaning of a word in the distributed representation is distributed over all the dimensions. For example, the vector ['SAT'] meaning is distributed over all other vectors of V['fluffy'], V['DOG'], V['ON'], V['THE'] and V['MAT'] as shown in figure1.2. The simple mathematical approach behind distributed representation is to obtain lower-dimensional space to represent the meaning of all words in the text. Word embedding core is the distributed representation.

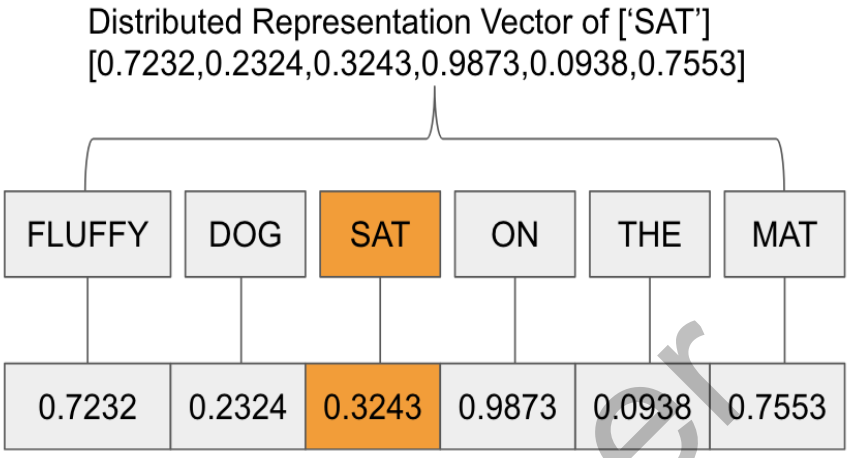


Figure 2.2 Distributed representation of the word 'SAT'

1.3 WORD2VEC

Word2Vec is a neural network that acts as a word embedding algorithm. It turns the corpus of text into vector space that can be processed mathematically. Google labs developed Word2Vec by (Mikolov et al. 2013e) , and it is claimed to be used by Google search engine, among other algorithms. Mikolov, Chen, et al. (2013) released the code online, and the trained model of 1.6 billion words described as state-of-the-art performance for measuring syntactic and semantic word similarity.

Since Word2Vec is a neural network, it goes with the exact mechanism of input, output, and hidden layers. The layers in the neural network of Word2Vec are word embedding. The weights of the neural network of Word2Vec are initialized to random numbers. Then the network keeps adjusting the weights by going through sentence by sentence (sentences that contain the designated word) until the weights are optimized to represent this word in the corpus best. The embedding typically is the weight vectors themselves, which often coincides with the activation pattern of the hidden layer. The outstanding findings of Word2Vec are the resultant weights vectors in the hidden layer. The resultant vector of different words can be mathematically calculated against each other to answer different linguistics questions like similarity, relationship, and semantic meaning. The most common example used to describe this is "king,

man, queen, woman" vectors. If the corpus is big enough, the "king" and "man" vectors should be close together.

Moreover, the vector of "woman" and "queen" should be close together. The elegance of Word2Vec comes in the ability to decipher this equation mathematically "Vector[king] + Vector[queen] - Vector[man]=?" the answer is a vector [woman] explained (Mikolov et al. 2013d). The base and simple outcome are that Word2Vec has successfully represented the words in any given corpus by a mathematical vector which is the outcome of the hidden layers of the trained neural network. This vector space can be used mathematically to extract meaning and quantify the relationship between different entities in the given corpus.

When comparing different word embedding techniques such as Word2Vec, Glove and FastText, Word2Vec seems to be the most researched. By searching for the terms “word2vec NLP”, “glove NLP”, and “FastText NLP” in Google Scholar, the number of publications was 26100, 22000 and 9060, respectively. Adding the word NLP” to the word embedding techniques was important to get relevant results. Based on these results, numerous studies have been published for word2vec in the NLP more than other word embedding technique.

2.1.1 Word2Vec Limitations

Word2Vec works well with one single word. However, many entities are defined with more than one word. There are several research articles and solutions that tried to solve this problem with no outstanding results. The Facebook research group published a research article about InferSent where they proposed sentence representations successfully with high accuracy (Conneau et al. 2017). This concept was represented earlier with a group of researchers in 2015 called Skip-Thoughts (Kiros et al. 2015)

1.3.1 Comparison between TF_IDF and Word2Vec

There are many elementary approaches to solve multiword sentences or entities. The mathematical average of two-word embedded vectors can result in decent

representations of the multiword entity. Document classification can be done with TF-IDF (Term Frequency-Inverse Document Frequency). Several libraries can deploy a simple baseline. TF-IDF works with counting the frequency of a specific word in a given document, adjusting the value to the inverse of counting the same word in all documents. This method has been used for years, and it seems to be working well. TF-IDF accuracy in document classification can be compared to Doc2Vec (Maslova & Potapov 2017). Using TF-IDF, weight averaging can be used in document classification.

Another example of the difference between word embedding and TF-IDF is the complexity of the model. In the case of the classical baseline model of TF-IDF, it is a basic counting and calculation which are easier to modify, debug and control. However, for complicated models like Word2Vec and Doc2Vec, the debugging and the modification is more complicated, and, in some cases, it is impossible to understand the error source. Word embedding might not be the most suitable solution for all problems. However, it opens other possibilities that were not achievable before. Using TF-IDF is easier to run and understand however word embedding is much more complicated than conventional methods.

Another limitation of Word2Vec is its trained model. It is complicated to expand it by adding more text to the trained model. Till this moment, none of the existing techniques enables those extensions. This means the model expansion is very computationally expensive and must amend the new text to the original corpus and retrain the model from scratch.

2.2 DIFFERENT TECHNIQUES FOR WORD EMBEDDING

Several other word embedding techniques can cover some of Word2Vec limitations. Such as and not limited to GloVe and FastText. However, those techniques have their limitations as well. In this section, some of those techniques are discussed, along with a good comparison between techniques.

GloVe: (Global Vectors for Word Representation (Pennington et al. 2014) is an unsupervised learning technique for obtaining word embedding. Its implementation is different from Word2Vec and FastText. GloVe needs to prepare the whole co-occurrence matrix at once and load it in the memory, which might be very memory expensive. In the case study of Wikipedia or Google News, GloVe requires very high RAM specifications to pre-process the corpus and load the co-occurrence matrix to the memory. Due to the computational cost, it is not efficient to be used with the big corpus. However, it will perform very well with a smaller corpus (Pennington et al. 2014).

FastText is a word embedding algorithm that the Facebook AI research group released in August 2016 (Bojanowski et al. 2017). The gensim API for FastText is very similar to Word2Vec gensim API. The experience of running the code is very similar. FastText is written in C++. It is available with Python API as well. By the end of 2017, Gensim released a native Python library for FastText to enhance the implementation experience of C++ core code. Against several claims, Facebook AI research claims that FastText can be trained with one billion words in less than ten minutes with a standard multicore CPU. The research was published in 2016 by Joulin, Grave, Bojanowski and Mikolov (2016). The speed of FastText in the memory was a difficult obstacle, and (Joulin et al. 2016) proposed a solution later in 2016 by using product quantization to store word embedding in the memory, claiming (Joulin et al. 2016) that it outperforms the state-of-art by a good margin.

FastText, in general, is much slower than Word2Vec, even if it is written in C++. The main reason for the slowness of FastText is related to how it works. The similarity in FastText goes deeper than Word2Vec. The similarity function in FastText is calculated by measuring the similarity between the subwords. For example, the word “superpower” will be divided to su + up + pe + er + rp + po + ow + we + er. The algorithm calculates the similarity among all those subwords. This is the main reason behind its slowness. This is proven to be beneficial since some chunks of words can be more related to the whole word morphologically. So, if two words share many sub words, they should be more similar than if the whole words are not. FastText (Joulin et al. 2017) has much better results considering the morphology of the words. The

interchangeability of the words has much more value than Word2Vec as there is a tendency for interchangeable words to share sub words.

The slowness of FastText (Joulin et al. 2016) is one of the drawbacks of the algorithm. As mentioned above, the slowness comes from the extra calculation of the subwords. Nevertheless, it gives a FastText morphological edge against other word embedding algorithms. However, the sub-word calculations impose a significant error as it might consider the word "night" and "knight" similar, which is semantically wrong. Another example is "can" as a container or "can" as a verb. Using the spaCy library in Python can be beneficial to distinguish the difference in meaning. It is important to note that even when the maximum length parameter is set to zero, which means that there are no sub words taken into consideration, still FastText is much slower than Word2Vec.

Word2Vec outputs a numerical vector that represents the word appearance within a context. However, it never classifies the sequence of words together. For example, how to calculate the similarity between two sequences of words "sentences". The simplest method to do this is to average the vector representation of each word.

2.3 NEURAL NETWORKS

2.3.1 Recurrent Neural Network

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable-length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

2.3.2 Convolutional Neural Network

A Convolutional Neural Network is a deep learning network developed for image classification and text classification (sentence prediction) (Zhang & Wallace 2015). It

consists of a series of convolutions and sub-sampling (pooling) operations to analyze only relevant information (e.g. borders and shapes of an image) and simplify the initial data. That will overcome overfitting data problems that could affect a multi-layer perceptron (MLP) network.

2.3.3 Long Short-Term Memory

A Convolutional Neural Network is a deep learning network developed for image classification and text classification (sentence prediction) (Zhang & Wallace 2015). It consists of a series of convolutions and sub-sampling (pooling) operations to analyze only relevant information (e.g. borders and shapes of an image) and simplify the initial data. That will overcome overfitting data problems that could affect a multi-layer perceptron (MLP) network.

Long Short-Term Memory (LSTM) networks were proposed by (Hochreiter & Schmidhuber 1997) are suitable to classify serialized objects such as sarcasm detection. LSTM is an extension for recurrent neural networks, which extends their memory. So, it is suitable easier to remember past data in memory. LSTM is well-suited to classify, process and predict time series and sequential text data. Many applications that the LSTM is suitable for Text generation, language translation, and handwriting recognition are the widespread application of LSTM. The LSTM repeat module is more difficult in Representing. Instead of making a single layer of the neural network, four layers communicate in a specific way. It has two states besides: hidden state and cell state.

2.4 RELATED WORK

The literature has shown great interest in the task of Word2Vec hyperparameter tunning. This section aims to discuss and presents the prior researchers' works. These studies were performed in different ways by applying Word2Vec hyperparameter tunning on some recommendation systems, other downstream tasks and in different languages.

The word embedding is designed to analyze the corpus of text and classify it eliminates the need for users to remark anything. It is a kind of unsupervised machine learning that learns from current words and sentences without requiring interpretation. The specific abilities of word embedding are determined by how it is applied. Here are a few illustrations of how Word embedding has been used to various problems.

- Automated text tagging**:** Nikfarjam et al. (2015) used word embedding to extract drug response characteristics from a social media corpus, claiming an accuracy of 82 %, which is an increase over the baseline assessed.

- Recommendation Engines: Ozsoy (2016) researched the architecture of a recommendation system utilizing Word2Vec word embedding in the Foursquare check-in dataset and found significant improvement for word embedding recommendation engines.

- Machine translation: Mikolov, Le, et al. (2013)using two alternative translations for the same content to train the word embedding algorithm. Showed how monolingual data might be mapped to bilingual data. The distributed representation can generate vector space similarities and successfully translate them. Mikolov's experiment obtained an accuracy of 90% of precision for translation between English and Spanish. Chelba et al. (2013) released a new one-billion-word data set that will be used to measure statistical language modelling. This dataset has the potential to be implemented for both translation and word embedding assessments.

- Question and answers: Even though there has been much study on automating an AI agent to answer human queries, Weston et al. (2015) believe no complete system can do it yet. They claimed that having an automated AI question and answer system might be achieved by combining word embedding with an improved memory network model.

- Sentiment analysis is an excellent illustration of just how Word2Vec can be used. It is possible that categorizing user reviews will take a long time. In the

classification of sentiment analysis, there are various supervised learning approaches. Using Word2Vec, on the other hand, can be a more straightforward method for sentiment analysis. The IMDB movie review dataset was researched in 2015 by a Facebook AI research group led by Mesnil et al. (2014). Several integrated machine learning techniques were used to study the IMDB movie review dataset. In their study, they studied the NB-SVM, RNN-LM, and sentence vector methods. They have made their code public to make it simple to replicate their results and increase their accountability.

2.4.1 Tuning Word2Vec for Large Scale Recommendation Systems

Chamberlain et al. (2020) study Word2Vec as an effective system mastering device that emerged from Natural Language Processing (NLP) and is now carried out in more than one domain, inclusive of recommender systems forecasting and community analysis. As Word2Vec is frequently used off the shelf, the researchers studied whether the default hyperparameters are appropriate for recommender systems. In this work, the researchers first elucidate the significance of hyperparameter optimization and display that unconstrained optimization yields a mean 221% development in hit charge over the default parameters. However, unconstrained optimization results in hyperparameter settings that can be very high priced and no longer viable for massive scale advice tasks. The researchers display 138% common development in hit charge with a runtime budget-restricted hyperparameter optimization.

Furthermore, to make hyperparameter optimization relevant for massive scale advice troubles in which the goal dataset is simply too massive to look over, the researchers look at generalizing hyperparameters settings from samples. They displayed that using restricted hyperparameter optimization, the usage of handiest a 10% pattern of the records nonetheless yields a 91% common development in hit charge over the default parameters whilst carried out to the whole datasets. Finally, the researchers observed hyperparameters discovered using the technique of restricted optimization on a pattern to the Who to Follow advice carrier at Twitter and are capable of boom compliance with prices with the aid of using 15%.

2.4.2 Word2Vec Applied to Recommendation: Hyperparameters Matter

Caselles-Dupré et al. (2018) studied the effect of Skip-gram with negative sampling. This is a common hyperparameter used in Word2Vec to create a high-quality work embedding and vector representation of a given corpus. Given that it is commonly used, it is not precisely accurate to use the same tuned model for different tasks such as classification, recommendation, sentiment analysis and others. The hyperparameters are often tuned dependently based on the given dataset and the downstream tasks. Results show that optimizing neglected hyperparameters, namely negative sampling distribution, number of epochs, subsampling parameters and window size, significantly improves performance on a recommendation task and can increase it by order of magnitude. The outstanding finding of this research is that the hyperparameters tuning functions in natural language processing tasks and Recommendation tasks are noticeably different.

2.4.3 Word2Vec: Optimal Hyper-Parameters and Their Impact on NLP Downstream Tasks

Adewumi et al. (2020) confirmed empirically that the combination of hyperparameters is highly dependent on the downstream tasks. They have used different combinations of hyperparameters to create a high-quality vector representation that can be used to build a state-of-the-art downstream task such as classifications. They have tested their hypotheses with intrinsic and extrinsic (downstream) evaluations, including named entity recognition (NER) and sentiment analysis (SA). Outstanding findings such as high analogy scores do not necessarily correlate positively with F1 scores, and the same applies to focus on data alone. Increasing vector dimension size after a point leads to poor quality or performance. They also noted that the size of the corpora might be irrelevant, and it is more about the content and the high-quality representation of the vector space.

2.4.4 Learning Quality Improved Word Embedding with Assessment of Hyperparameters

Yildiz & Tezgider (2020) studied the hyperparameters tuning for Word2Vec to generate high-quality word embedding and vector spaces. The research focused on the

parameters of the minimum word count, vector size, window size, and the number of iterations. They have also introduced two main methods: computationally more efficient than grid search and random search. They used around 300 million words. The downstream tasks were developed using deep learning classifiers. The task was to classify documents into ten different classes. The classification task was used to evaluate the quality of the generated word embedding and vector spaces. Their results show a 9% increase in the overall classification accuracy, which inherently provides undeniable proof that the parameters tuning can result in high-quality word embeddings that can be translated and propagated to the accuracy of the downstream tasks of deep neural networks classification tasks. Table 2.1 summarises all the studies mentioned earlier.

Table 2.1 Summary of previous related work

Author	Parameters	Feature	Dataset
Chamberlain et al. (2020)	▪ Window size ▪ Embedding dimension ▪ Negative sampling ▪ Negative samples ▪ Initial learning rate	▪ Recommendation task	▪ Twitter retweet ▪ Twitter follow
Yildiz & Tezgider (2020)	▪ Window size ▪ The minimum word count ▪ Vector size ▪ Iterations numbers	▪ Classification model	▪ 3 million Turkish texts
Adewumi et al. (2020)	▪ Window size ▪ Dimension size ▪ Epochs	▪ Named entity recognition ▪ Sentiment analysis	▪ Internet movie database (IMDB) of movie reviews
Caselles-Dupré et al. (2018)	▪ Window size ▪ Negative sampling ▪ Number of epochs ▪ Embedding size ▪ Learning rate ▪ Sub-sampling parameter	▪ Recommendation ▪ Task	▪ Music datasets ▪ E-commerce dataset ▪ Click-stream dataset

The hyperparameters are often tuned dependently based on the given dataset and the downstream tasks. Several studies proposed different values for hyperparameter tunning for different downstream tasks applied on the dataset—most of the studies proved to achieve a higher accuracy rate. For the hyperparameter used, All of the studies have used the window size parameter ((Chamberlain et al. 2020); (Yildiz & Tezgider 2020);(Adewumi et al. 2020);(Caselles-Dupré et al. 2018)). Two

of the studies by ((Chamberlain et al. 2020)and (Caselles-Dupré et al. 2018)) have used the Word2Vec hyperparameter tunning for recommendation tasks on different datasets and achieved a higher accuracy on the recommendation. A deep learning model was developed for the classification model to evaluate the quality of the word embedding by(Yildiz & Tezgider 2020) A BiLSTM network was trained on the IMDb dataset for sentiment analysis (Adewumi et al. (2020). (Chamberlain et al. 2020) have used a shallow network for the recommendation task on the Twitter follows and Twitter Retweet dataset.

2.5 SUMMARY

The chapter reviews related past studies on word embedding and the relevant techniques for word embeddings such as Word2Vec, GloVe and FastText. In addition, analyze those selected related research work for word embedding and Word2Vec technique applied by researchers on different classification tasks and other downstream tasks. The analysis of these studies will introduce and clarify that using Word2Vec hyperparameters tunning significantly impacts classification accuracy.

CHAPTER III

RESEARCH METHODOLOGY

3.1 INTRODUCTION

This chapter introduces the methodology of Word2Vec hyperparameter tuning and classification downstream tasks using several types of Deep Neural Networks such as RNN, CNN, and LSTM. Furthermore, it presents the study's research design, including information about the dataset and its pre-processing task. The implementation of algorithms depends on the Python programming language using the AWS platform. This chapter starts explaining the dataset going through the pre-processing task and describes the evaluation metrics to assess the performance of the algorithms. Finally, it explores the hyperparameter tunning in detail.

3.2 RESEARCH DESIGN

The methodology of this study consists of five phases. Phase one is the preparation of the Amazon Custom Reviews dataset. This preparation involves the specific technique that is used to extract and filter the data. Phase two is the data pre-processing. This phase focuses on the data loading and the random selection of the subset of that data that will be used in training. Phase three is the AWS cloud environment experimental settings. Phase four is the hyperparameter tuning of Word2Vec in preparation for phase five, the downstream classification task.

As mentioned before, the experiment objective is divided into two main sections. First, to find the best hyperparameters to generate high-quality word embeddings. Then the generated word embeddings are evaluated by testing its accuracy through downstream tasks of classification using several types of deep neural networks such as RNN, CNN, and LSTM. Comparison of the result of the word embedding with the same dataset through more than one deep neural network can give

a great insight on which of the neural network can achieve more accuracy rate with hyperparameter tuning and which neural network is more resilient to the hyperparameter tuning.

The first section aims to train the Word2Vec model to generate a high-quality vector space. The experiment is designed to try hundreds of several combinations of hyperparameters of Word2Vec to obtain the highest possible quality of vector space of word embedding. The resultant vector space of word embedding will be used in the second section of the experiment. It is the downstream task of classifying the Amazon Custom Reviews into the predefined ranking of the reviews.

The second section is the classification task that will use different types of Deep Neural Networks to find the highest possible accuracy in combination with different word embeddings. The first layer in the Deep Neural Networks will be the embedding layer. The resultant word embeddings from section one will be converted to an embedding matrix that fits into the Keras embedding layer. This layer will be defined as a non-trainable layer. Not letting Keras change the weights of the embedding layer will measure the effectiveness of the quality of the generated layer from section one.

3.3 DATASET

This research uses the Amazon Customer Reviews Dataset(“Amazon Customer Reviews Dataset” n.d.) (a.k.a. Product Reviews). This data has been collected over two decades since 1995. The data contains millions of Amazon customers that provided hundreds of millions of reviews. The reviews contain the customer opinion about their experience with the products and the purchase order through the amazon.com website. The dataset of Amazon Customer Reviews has been used several times in academic research , specifically in Natural Language processing by((Nandal et al. 2020);(Srujan et al. 2018); (Pankaj et al. 2019)). The field of Information Retrieval and Machine learning has significantly used the dataset.

- The dataset has helped several researchers to understand the customer product experience and construct a high-quality understanding of the evaluation of the

natural language opinion and the variation in the perception of different products across various geographical regions. This has been significantly used for marketing research and promotional and target marketing and addresses several problems such as bias in reviews and different ways of expressing the opinion. If you use the AWS Command Line Interface, you can list data in the Bucket with the "ls" command: aws s3 ls s3://amazon-reviews-pds/tsv/

The following section will be list in detail the dataset description:

- The link for the dataset “Amazon Customer Reviews Dataset” , 2015, https://s3.amazonaws.com/amazon-reviews-pds/readme.html
- The dataset is composed of over 130 million reviews. This research used only 100K reviews.
- The data is stored on Amazon Web Services S3 bucket US East Region.
- The data is available in TSV format.
- If you use the AWS Command Line Interface, you can list data in the Bucket with the "ls" command: aws s3 ls s3://amazon-reviews-pds/tsv/
- Each individual customer review is presented in one line.
- The dataset of Amazon Customer Reviews contains the review text itself. It also contains the metadata of the data. The metadata describes the data in the customer review. The metadata is composed of the below significant sections
- The reviews are collected from the Amazon.com website associated with data from 1995 to 2015. This dataset contains more than 130M customer reviews. It represented the human expression of the customer experience and how people evaluate and express their opinions on the products.
- The dataset of Amazon Customer Reviews contains product reviews in multiple languages. This can be used in multilingual research to understand how people evaluate and express their opinions about the same product in different languages. The reviews collected in different languages from five countries are counted as more than 200K reviews. This research only focuses

on the English language due to the limitation of time and the computational resources.

- Some of the reviews have been marked as non-compliant due to a violation of Amazon policies. This dataset can detect biased reviews and possible promotional reviews that aim to create a fake representation of actual customers. This dataset was not used in our research as the focus was only on generating a high-quality word embedding representation by tuning different hyperparameters of Word2Vec.

3.4 PRE-PROCESSING

In this stage, the data is prepared by first splitting the text when running on a set of pre-processing algorithms to prepare it for the following stages. The pre-processing tasks can be described as follows:

3.4.1 Data Loading

Data loading is the process of retrieving the data from the S3 Bucket and loading it directly to Amazon Athena. After Creating the Athena environment on the AWS account, run the below ETL job that will load the data from S3 Bucket.

3.4.2 Data Selection

The selection process was performed over AWS Athena. Random selection has been performed to ensure no bias in data selection. Due to the limitation in the computational expenses, 100K reviews are used with a maximum of 40 words per review. Figure 3.1 show a sample of the dataset.

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_votes	total_votes	vine	verified_purchase	review_headline	review_body
2	US	18778586	RDIJS7QY6XNR	B00EDBY7X8	122952789	Monopoly Junior Board Game	Toys	5	0	0	N	Y	Five Stars	Excellent!!!
3	US	24769659	R36ED1U38IELG8	B00D7JFOPC	952062646	56 Pieces of Wooden Train Track Compatible with All Major Train Brands	Toys	5	0	0	N	Y	Good quality track at excellent price	Great quality wooden track (better t
4	US	44331596	R1UE3RPRGCOLD	B002LHA74O	818126353	Super Jumbo Playing Cards by S&S Worldwide	Toys	2	1	1	N	Y	Two Stars	Cards are not as big as pictured.
5	US	23310293	R298788GS6I901	B00ARPLCGY	261944918	Barbie Doll and Fashions Barbie Gift Set	Toys	5	0	0	N	Y	my daughter loved it and i liked the p	my daughter loved it and i liked the
6	US	38745832	RNK4EXOBBPN5	B00UZOPOFW	717410439	Emazing Lights eLite Flow Glow Sticks - Spinning Light LED Toy	Toys	1	1	1	N	Y	DONT BUY THESE!	Do not buy these! They break very f
7	US	13394189	R3BPETL222LMIM	B009B7F6CA	873028700	Melissa & Doug Water Wow Coloring Book - Vehicles	Toys	5	0	0	N	Y	Five Stars	Great item. Pictures pop thru and a
8	US	2749569	R3SORMPJZO3F2J	B0101EHRSM	723424342	Big Bang Cosmic Pegasus (Pegasis) Metal 4D High Performance Generic Bat	Toys	3	2	2	N	Y	Three Stars	To keep together, had to use crazy
9	US	41137196	R2RDOJQ0WBZCF6	B00407S11Y	383363775	Fun Express Insect Finger Puppets 12ct Toy	Toys	5	0	0	N	Y	Five Stars	I was pleased with the product.
10	US	433677	R2B8VBEPB4YEZ7	B00FGPU7U2	780517568	Fisher-Price Octonauts Shellington's On-The-Go Pod Toy	Toys	5	0	0	N	Y	Five Stars	Children like it
11	US	1297934	R1CB783I7B0U52	B0013CYOS0	269360126	Claw Climber Goliath/ Disney's Gargoyles	Toys	1	0	1	N	Y	Shame on the seller !!!	Showed up not how it's shown . Wa
12	US	52006292	R2D90RQQ3V8LH	B00519PJTW	493486387	100 Foot Multicolor Pennant Banner	Toys	5	0	0	N	Y	Five Stars	Really liked these. They were a little
13	US	32071052	R1Y4ZOUGFMJ327	B001TCY2DO	459122467	Pig Jumbo Foil Balloon	Toys	5	0	0	N	Y	Nice huge balloon	Nice huge balloon! Had my local gr
14	US	7360347	R2BUV9QJI2A0DX	B00DOQCWF8	226984155	Minecraft Animal Toy (6-Pack)	Toys	5	0	1	N	Y	Five Stars	Great deal
15	US	11613707	RSUHRJFJIRB3Z	B004C04I4I	375659886	Disney Baby: Eeyore Large Plush	Toys	4	0	0	N	Y	Four Stars	As Advertised
16	US	13545982	R1T96CG98BBA15	B00NWGEKBY	933734136	Team Losi 8IGHT-E RTR AVC Electric 4WD Buggy Vehicle (1/8 Scale)	Toys	3	2	4	N	Y	... servo so expect to spend 150 mo	Comes w a 15$ servo so expect to
17	US	43880421	R2ATXF4QQ30YW	B00000JS5S	341842639	Hot Wheels 48- Car storage Case With Easy Grip Carrying Case	Toys	5	0	0	N	Y	Five Stars	awesome ! Thanks!
18	US	1662075	R1YS3DS218NNMD	B00XPWXYDK	210135375	ZuZo 2.4GHz 4 CH 6 Axis Gyro RC Quadcopter Drone with Camera & LED Li	Toys	5	4	4	N	N	The closest relevance I have to item	I got this item for me and my son to
19	US	18461411	R2SDXLTLF92O0H	B00VPXX92W	705054378	Teenage Mutant Ninja Turtles T-Machines Tiger Claw in Safari Truck Diecast V	Toys	5	0	0	N	Y	Five Stars	It was a birthday present for my gra
20	US	27225859	R4R337CCDWLNG	B00YRA3H4U	223420727	Franklin Sports MLB Fold Away Batting Tee	Toys	3	0	1	Y	N	Got wrong product in the shipment	Got a wrong product from Amazon
21	US	20494593	R32Z6UA4S5Q630	B009T8BSQY	787701676	Alien Frontiers: Factions	Toys	1	0	0	N	Y	Overpriced.	You need expansion packs 3-5 if yc
22	US	6762003	R1H1HOVB44808I	B00PXWS1CY	996611871	Holy Stone F180C Mini RC Quadcopter Drone with Camera 2.4GHz 6-Axis Gy	Toys	5	1	1	N	N	Five Stars	Awesome customer service and a c
23	US	25402244	R4UVQIRZ5T1FM	159174952	741582499	Klutz Sticker Design Studio: Create Your Own Custom Stickers Craft Kit	Toys	4	1	2	N	Y	Great product for little girls!	I got these for my daughters for pla

Figure 3.1 Amazon Customer Reviews dataset sample

Then, the four-column that is relevant to our experiments are selected, which are the review id, review headline, review body and the star rating. These columns and their explanation are shown in Table 3.1. The four columns are selected from the Amazon Customer Reviews dataset sample, as shown in Table 3.2.

Table 3.1 Dataset selected columns explanation

Data Column	Explanation
Review_id	The id of the review
Review_headline	The title of the review
Review_body	The review texts
Star_rating	The 1–5-star rating of the review

Table 3.2 An example of the dataset columns

Review_id	Star_rating	Review_headline	Review_body
R298788GS6I901	5	my daughter loved it and i liked the price and it came ...	my daughter loved it and i liked the price and it came to me rather than shopping with a ton of people around me. Amazon is the Best way to shop!
R2SDXLTLF92O0H	5	Five Stars	It was a birthday present for my grandson and he LOVES IT!!

To be continued…

…continuation

RNX4EXOBBPN5	1	DONT BUY THESE!	Do not buy these! They break very fast I spun then for 15 minutes and the end flew off don't waste your money. They are made from cheap plastic and have cracks in them.
R1UE3RPRGCOLD	2	Two Stars	Cards are not as big as pictured.
R1JS8G26X4RM2G	5	Five Stars	Great gift!

3.4.3 Ranking Buckets

The customer rating comes with 5-star ranks. For simplicity, the five ranks are divided into two ranks. The lower rank obtains a one and two-star ranking, and the higher rank contains the three, four and five-star ranking. Python Jupyter NoteBook is used to perform this process on AWS SageMaker.

3.4.4 Data Concatenation

The processed data has been concatenated to form a new comma-separated values (CSV) file, which is a delimited text file that uses a comma to separate values. The product review title and the review body have been concatenated to provide comprehensive customer opinion data. The concatenated text has been merged with the new ranking buckets to form a new dataset format of 100K review. An Example of the dataset after concatenated is shown in Table 3.3.

Table 3.3 An example of the dataset after concatenated

star_rating	**Review_headline + Review_body**
5	my daughter loved it and i liked the price and it came ... my daughter loved it and i liked the price and it came to me rather than shopping with a ton of people around me. Amazon is the Best way to shop!
5	Five Stars It was a birthday present for my grandson and he LOVES IT!!
1	DONT BUY THESE! Do not buy these! They break very fast I spun then for 15 minutes and the end flew off don't waste your money. They are made from cheap plastic and have cracks in them.
2	Two Stars , Cards are not as big as pictured.
5	Five Stars Great gift!

3.4.5 Text Cleaning

The resultant CSV file was converted to a pandas data frame. The data frame was prepared for the data cleansing process. First, all punctuations and special characters were removed from the text. Then the English Stop words in computing which are words that are filtered out before or after processing natural language data (text), were removed from the corpus (Rajaraman & Ullman 2011) Typically, they are the common words like "a," "the," "and" an "of", which appear in the text of the customer review. All the corpus letters have been converted to lowercase letters. The cleaned data is uploaded to a new data frame and ready for the next stage. An example of a review after the text cleaning is shown in

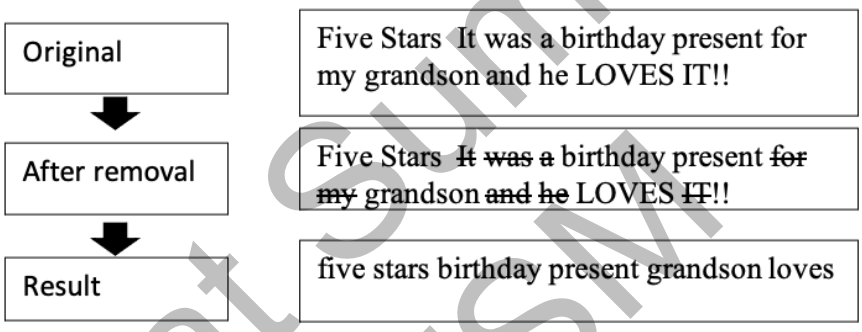


Figure 3.2 An Example of a customer review after text cleaning

3.4.6 Data Classification Preparation

The data cleansing process created a clean text corpus to be used in upstream and downstream tasks. The data frame created will then be passed in the tokenization process and sequence padding to ensure that all sentences have a similar length and then passed to the downstream task classifier. Table 3.4 shows the dataset number of characters before and after dataset cleaning. The data was split into 75% for training and 25% for testing. A close number to the default configuration of MATLAB for data split into Machine Learning tasks by (Lanka et al. 2020) has been chosen for the research experiment.

Table 3.4 Statistic of the dataset

Dataset	Number of characters
Dataset before cleaning (Number of Characters)	48,639,420
Dataset after cleaning (Number of Characters)	47,070,622
Removed Characters	1,568,798

3.5 AWS CLOUD ENVIRONMENT EXPERIMENTAL SETTINGS

In this research, the AWS environment has been utilized first to process the data. The data pipeline is created, the models trained and then perform the hyperparameters tuning task along with the data analysis. The following section will explain in detail the AWS cloud environment, AWS S3 bucket, AWS SageMaker, AWS Glue and AWS Athena. The reason this study used AWS is that when using Google Colab, various problems were presented such as being out of memory and exceeding the GPU time limit. AWS was the optimal solution to run the experiment.

3.5.1 What is AWS Cloud Environment?

Amazon Web Service (AWS) is the world's leading and most commonly and compressive used cloud platform. It offers more than 200 features and services and operates throughout several data centres around the globe. AWS serves millions of customers ranging from the largest enterprise and government agencies to small start-ups and researchers.

3.5.2 AWS S3 Bucket

Amazon Simple Storage Service (AWS 2020) is an object storage service. S3 buckets are one of the oldest services of AWS that offers an extensive range of features such as scalability, security, performance and data high availability.

In this research, the Amazon Custom Reviews dataset was initially stored in the S3 Bucket. Using the S3 Bucket in this project was a must to copy the data from the original location to the staging area to start the pre-processing procedures and build the experiments. Also, S3 buckets have been used in all stages of development and training the models. This will be demonstrated later in the system architecture.

There are several classes in which the data based can be store on its frequency of access and data lifecycle.

3.5.3 AWS SageMaker

Amazon SageMaker ("Amazon SageMaker – Machine Learning – Amazon Web Services" n.d.) is a fully managed service provided by AWS. It is explicitly designed for data scientists and developers to build, train, and deploy machine learning models to have a high-performance platform and cost-effective solution. AWS SageMaker offers a scalable environment that allows the researchers and data scientists to build the right environment size and pay only for their requirements.

In this research, AWS SageMaker was used to develop and build the models on SageMaker Notebook. Amazon SageMaker provides hosted Jupyter notebooks that are easily used to explore the data and visualization tasks. AWS SageMaker is enabled with direct connectors to AWS S3 buckets, allowing an easy way to access the data without using the local hard disks for storage or data processing.

During the development phase, a small environment for a small dataset was built. This allows significantly low fees to be paid for small machines. During the training phase, the size of the machine is increased for the period of the training only. This makes AWS SageMaker a very efficient and cost-effective solution for researchers.

3.5.4 AWS Glue

AWS Glue (*AWS Glue - Managed ETL Service - Amazon Web Services*, n.d.) is a fully managed ETL (extract, transform, and load) service. What makes AWS Glue cost-efficient is that it is pay-as-you-go. So, no need to pay an upfront cost to run the service. The dataset of Amazon Custom Reviews is provided in the S3 Bucket. Later how the data loaded to AWS Athena will be explained. This process required an ETL data pipeline that was built using AWS Glue.

3.5.5 AWS Athena

Amazon Athena (Amazon 2020) is an interactive query service. The service is generally used to analyze the data that is stored in Amazon S3. Amazon Custom Reviews dataset is offered in Amazon S3 Bucket. Amazon Athena Is used to accessing the dataset and load it using the AWS Glue service into a standard SQL schema. Athena is a serverless service that AWS fully manages. There is no infrastructure to manage, and it is a pay-as-you-go service based on the queries that run only. The data is stored in Amazon S3 Buckets, and the results of all the queries are also stored in Amazon S3 Buckets. This means that while using Amazon Athena, getting a low price of the storage of S3 Bucket without the need to have upfront costs incurred by building a large database.

3.6 THE HYPERPARAMETER TUNING OF WORD2VEC

In the following section, word embedding design will be discussed, and the detailed methodology of how word embedding is used in the research. This will require a detailed analysis of the algorithm's structure and implementation. The Word2Vec internal mechanism is very important to understand the underlying hyperparameters that this research addresses. For example, the window size mechanism in the model is a very crucial hyperparameter that decides the architecture of the neural network. This will be explained in the coming section.

Word2Vec is a learning algorithm to predict words from a given context. By providing a context, the network will be able to predict what is the most likely word that occurs in the context. This section explains in detail the technical implementation of the Word2Vec neural network. Word2Vec can always be explained as a simple input, output, and hidden layers. (Kimothi et al. 2020) illustrated that The size of the layer is dependent on the input and the output of each layer. For example, in Figure 3.3 below W(VxN) is equal to the input size of (x), which is (V) multiplied by the size of the hidden layer (h), which is (X). The same goes for the W(NxV), where N is the size of the hidden layer (h) and (V) is the size of the output layer (y) which is (V).

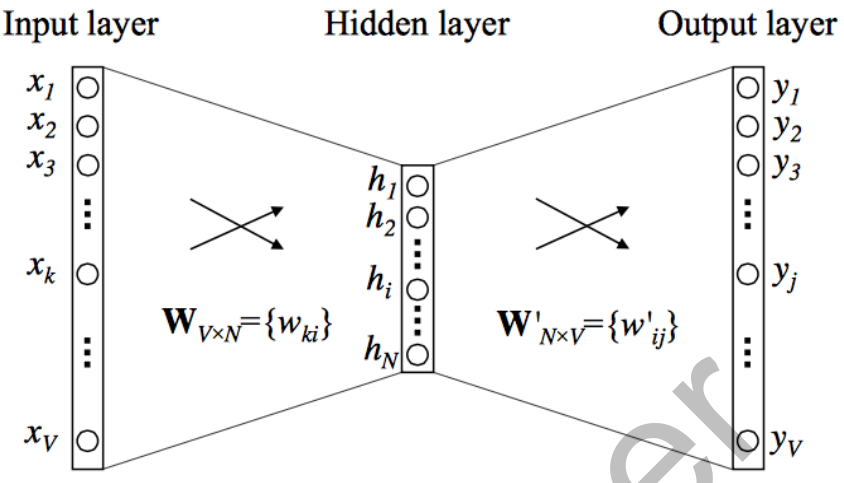


Figure 3.3 Neural Network represents Word2Vec Neural Network

Input layer: The input layer has one neuron for each and every unique word in the corpus. The input layer represents what is called One-hot representation. It is a very long vector "depending on the corpus" representing the index of a particular word in the corpus. All its values are zeros except for the bit representing the represented word, and it will turn into one. It is essential to know that the input layer of word embedding is only a vector and not a probability.

Output layer: The neuron and words representing them appear in the input layer are the same as the output layer. The output vector is the expected vector to appear when the input vector is presented. So, for example, in Figure 1.3. the word x3 and y1 appears next to each other in the corpus. So, if the input is set to the vector of x3, the output is the vector of y1. Vector of x3 means all neurons are zeros except the neuron that represents the word x3 and the same goes for the output vector of y1. The process starts with imposing the input of the x3 vector, setting random values for the weights for the hidden layer and checking the output vector and measuring the error by comparing the resultant vector and the target vector at the output layer. Then the error is propagated back to the network, and the weights matrices are adjusted in the hidden layers such that the output vector resembles the target vector. Then this operation is repeated for the entire training set till the highest accuracy is achieved.

Hidden layer: The hidden layer relies on the distributional hypothesis derived from the saying, "You shall know a word by the company it keeps" by Firth (1957). The hidden layer is a vector that represents a specific word in the corpus with specific numbers from -1 to 1. This hidden layer weight matrix size usually is hundreds (optimally from 100 to 300). When multiplying the input vector with the weight hidden vector, the output is the probability distribution of the given input word vector within its context. The word embedding is in the hidden layers of the weight matrix of Word2Vec.

3.6.1 Implementation Example of Word2Vec

The example mentioned above of simple two consecutive words in the corpus is a simplified example of the network design. The actual implementation of Word2Vec contains a training window. For example, in the sentence "This product is very reliable", The input layer will receive four different vectors V[This], V[product], V[very] and V[reliable], and the expected output should be V[is]. These settings ensemble the idea of predicting the word [is] based on the surrounding words with a window size of two words to the left and two words to the right. That is why the vector of [is] was not mentioned in the calculation

3.6.2 Continuous Bag-Of-Words Vs Skip-Gram

Continuous bag-of-words (CBOW): The input is a set of words, and the network task is to predict the most probable word that accompanies the input words. Continuous skip-gram will use a one-word vector to predict the accompanying words. CBOW is relatively faster than skip-gram. However, skip-gram is proven to be more accurate. Bornstein (2018) illustrates the architectural difference between the training methods in (Figure 3.4). This research explores both skip-gram and CBOW settings as a part of the hyperparameters navigations.

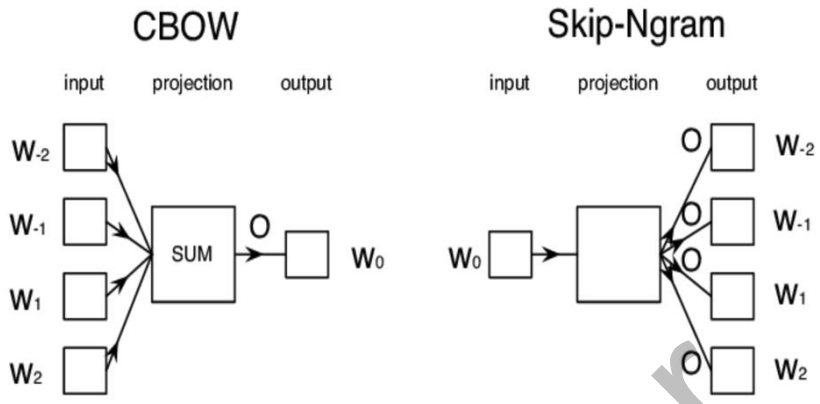


Figure 3.4 CBOW vs Skip-Ngram

3.6.3 Word2Vec Similarity Feature

One of the most famous outcomes of Word2Vec is the similarity function between two words or, to be specific, the similarity between two probability vectors representing two different words in their context window. There are several ways to calculate the similarity between two vectors, like Euclidean distance and cosine similarity. Word2Vec chooses to use the cosine similarity of two vectors which is the dot product of two vectors. Cosine similarity is the default function for similarity calculation in Gensim Library. The simplistic idea to visualize them is to measure the angle between two vectors in the vector space. The range of the cosine is from -1 to 1. Then it uses the SoftMax function to convert it to a probability that should be ranged from 0 to 1. The similarity feature is equivalent to the co-occurrence count matrix calculation.

Several trained models exist and are available online for download. Google has trained 100 billion unlabelled words for Google news and availed the vector space for the download. This research trained the model from scratch using the existing dataset from Amazon Customer Reviews. The use of the pre-trained model would have biased the model to the hyperparameters used in the pre-trained model.

3.6.4 Word Embedding Visualization

The distributed representation of word embedding is a relatively low dimensional vector space representing the meaning of the words in the corpus. The dimensionality is expected to be hundreds rather than thousands. This research navigates the dimension parameter from dimension size of 10 to 500. Visualizing hundreds of dimensions is an impossible task for 2D or even 3D diagrams. Principal component analysis (PCA) (Gorban et al., 2008) presents solutions to visually illustrate the vector space's dimensions. The diagram in Figure 3.5. illustrates the dimensionality of word embedding using a projector tool in the TensorFlow online library (Smilkov et al. 2016)**.** The visualization concept facilitates the understanding of the clustering of a word and its vectors. However, mathematically, it is impossible to extract exact meaning or use it for analogy identification due to the dimensionality reduction.

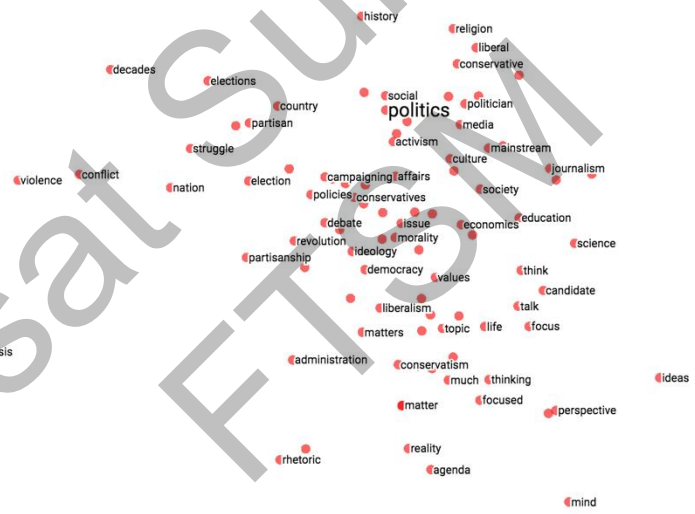


Figure 3.5 TensorFlow data visualization of word embedding

3.6.5 Word2Vec Hyperparameter

Gensim is a Python library that is free and open source. It is generally used to represent documents as mathematical vectors that carry semantic meaning. It is designed to be computationally efficient and to reduce the development effort as much as possible. Gensim accepts unstructured raw text and uses an unsupervised neural network and machine learning algorithm to generate the semantic mathematical representation.

Gensim library is considered to be the earliest implementation of Word2Vec. The library is comprehensive and has been maintained over the years. The Gensim version of 4.0.1 is used, which is the latest version at the time of the experiment. The following section will explain and detail how trained and fine-tuned the hyperparameters of the Word2Vec model.

The coming section explains the essential hyperparameters in the Word2Vec model in detail: the window size and the dimension size.

a. Word2Vec Window Size

The window size parameter is one of the most critical parameters in the Word2Vec algorithm. It defines how many words to the left and the right of the targeted word will be considered in the calculations. Figure 3.6 indicates the size of the window considering the target word "sat". If the window size is too small, like "1" only, then there is a 100% probability that the word "dog" appears next to the word "sat". However, if the window size changed to "2", then there will be a 50% probability that the word "dog" appears next to the word "sat". This probability will be equal to the probability that the word "fluffy" appears next to the word "sat". When more sentences are processed to calculate the space vector, the probability will change. So, the probability values mentioned here are for illustration of a single sentence only. When the window size increases "for example, to fifty", the context of the word increases, and the studied topic will become much wider. This will increase the attribute of the targeted words and include more entities to them. This can be helpful if the features of the words do not frequently appear too close to the targeted word. The bigger the window size, the less interchangeability of the target word, so when the window size decreases, it does better with interchangeability. When it increases, it does better with relatedness analogies.

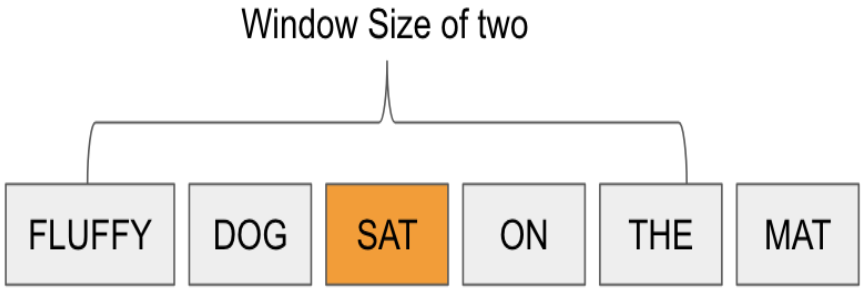


Figure 3.6 Illustrates the window size of Word2Vec

b. Word2Vec Dimension Size

The dimension size parameter reflects on the size of the distributed representation, which is the hidden layer. Typically, the dimension size varies from one to three hundred. The bigger the dimension size, the easier it will be overfitted on the training set and bad performance on the test. Tuning this parameter requires high accuracy on the training set and low accuracy on the testing set. This means that the dimension size is too big, and reducing it might solve the overfitting problem of the model.

The coming section will explain in detail all hyperparameters of Word2Vec models and how to navigate the hyperparameters space seeking the best combination for the downstream task.

c. Hyperparameters Navigation

Word2Vec algorithm implementation in the Gensim library is dependent on several hyperparameters. Each hyperparameter is used to have a technical or semantic impact on the generated vector space. In the coming section, list the hyperparameters of the Word2Vec Algorithm in the Gensim Python library:

- Vector_size: Dimensionality of the word vectors. The dimension size of the vector can be seen as a compressed meaning of numbers that are relevant to each other. For example, in a well trained Word2Vec model, the vector of "man" will be close to the vector of "King". Similarly, the vector of "woman" will be close to the vector of "queen" in such a distance that the vectors of

"man" and "king" are closer. According to the original paper of (Mikolov et al. 2013a), the best dimension size is 300. The dimension size is considered one of the essential hyperparameters. Ideally, when the dimension size increases, the vector representation will represent more information in-depth about each word. In contrast, the mathematical vector representation will be more abstract when the dimension size decreases and generalize more. For a visual representation, the standard method used is dimensionality reduction to reduce the dimensions to 3 or 2 dimensions presented in 2 or 3 dimensions diagrams.

- Window_size: The window size is the sliding window that the model uses to slide on the corpus. As explained earlier, the increase of window size will result in a holistic understanding of the sentence together, which can be explained by relatedness. However, if the window size decreases, the vector representation will have more interchangeability analogies. The commonly known number of window sizes is 5.
- Word2Vec_epochs: Number of iterations (epochs) over the corpus. (Formerly: iter). Ideally, when the number of epochs increases, the weight of the word embeddings will be more representative of the corpus.
- Word2Vec_sg: #({0, 1}, optional) – Training algorithm: value equal to 1 for skip-gram; otherwise CBOW. The difference was explained earlier.
- Word2Vec_hs: #({0, 1}, optional) – If value is 1, hierarchical softmax will be used for model training. Suppose the value is 0, and the negative is non-zero. In that case, negative sampling will be used
- Word2Vec_negative: #negative (int, optional) – If more than 0, negative sampling will be used. The int for negative specifies how many "noise words" should be drawn (usually between 5-20). If set to 0, no negative sampling is used. Negative sampling enables the machine learning and neural networks only to modify a small percentage of the weights, rather than all of them for each training sample.
- Word2Vec_ns_exponent = 0.75 #ns_exponent (float, optional) – The exponent used to shape the negative sampling distribution. A value of 1.0 samples exactly in proportion to the frequencies, 0.0 samples all words equally, while a

negative value samples low-frequency words more than high-frequency words. The original Word2Vec paper chose the popular default value of 0.75.

- cbow_mean ({0, 1}, optional) If it is equal to 0, use the sum of the context word vectors. If it is equal to 1, use the mean, it only applies when CBOW is used.
- hashfxn (function, optional) – Hash function to randomly initialize weights for increased training reproducibility.

d. Random Search Hyperparameter

In this experiment, an unconstrained random search algorithm was used to find the optimal combination of the hyperparameters and then test the optimal combination of those hyperparameters on the classification task.

Gensim library was used to construct Word2Vec word vectors. Word vectors are created by performing an unconstrained hyperparameter search. The five most important Word2vec hyperparameters are the dimension size of the learned vectors with default value 100, the value of maximum sliding window size is 5, the negative sampling exponent value is 0.75, and the negative sampling number is equal to 5.

A wide range for each hyperparameter has been initialized. A range starting from 10 up to 500 values as the Dimension size, values as the window size is from 1 to 50, value as negative sampling exponent is 0.75, and negative sampling range from 50 to 20.

The total number of hyperparameter combinations based on the hyperparameters and their ranges will result in 2,352,000,000 combinations. Each hyperparameter combination will take a long time and due to the computational resource limitation present. One thousand different combinations for the hyperparameters have been randomly chosen from the total combination for this research. At the last stage, parameters that produce the best results are saved as the best hyperparameter set. The study of the impact of the change for a single hyperparameter is not the primary concern of this study.

3.7 CLASSIFICATION

The second part of the experiment is the classification task. It takes the word embedding vector space as an input. Then train a deep neural network classifier on a downstream task classification task. The output accuracy is, of course, dependable on the DNN models. However, the aim is to find the discrepancy that might change the overall accuracy of changing the input word embedding layers.

The word embedding vector space can be used as a simple classifier by itself. Mathematical vector aggregation is generally used to find an average vector that represents a sentence or a document. Comparing this average vector to multiple vectors that represent multiple classes usually succeeds in identifying the classification tasks. However, using a Deep Neural Network in concatenation with word embedding usually results in higher accuracy.

In this experiment, the resultant word embeddings vector space was fed to several Deep Neural Networks such as (CNN), (RNN)and (LSTM). Then, the results of each accuracy are compared to identify the best hyperparameters combination for each model. Multiple methods of measuring the accuracy of the models were used, such as accuracy, precision, recall and F1 score for both testing and validation sets.

3.7.1 Recurrent Neural Network parameters and layers

This section describes the layers and the parameters used in Recurrent neural networks.

The diagram below Figure 3.7 and Figure 3.8 illustrates the RNN parameters and layers.

- The first layer is the embedding layer imported from the resultant vector space word embedding of Gensim Word2Vec in section one. Change in the dimension size will change the size of the embedding layer.
- The second layer is the spatial dropout that usually helps promote independence between feature maps.

- The third layer is the simple RNN layer.
- The fourth layer is batch normalization, a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This stabilizes the learning process and dramatically reduces the number of training epochs required to train deep networks.
- The fifth layer is a dropout, which refers to ignoring units (i.e. neurons) during a specific set of neurons chosen at random during the training phase.
- The sixth layer is a method of downsampling the whole feature map to a single value known as global max pooling. Setting the pool size to the size of the input feature map would achieve the same result.
- Two classes are represented by the seventh layer, which is a dense layer resulting in two neurons' output layers.


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 200, 128)          29081088
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 200, 128)          0
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 200, 25)           3850
_________________________________________________________________
batch_normalization_3 (Batch (None, 200, 25)           100
_________________________________________________________________
dropout_3 (Dropout)          (None, 200, 25)           0
_________________________________________________________________
global_max_pooling1d_3 (Glob (None, 25)                0
_________________________________________________________________
dense_6 (Dense)              (None, 50)                1300
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 102
=================================================================
Total params: 29,086,440
Trainable params: 5,302
Non-trainable params: 29,081,138
_________________________________________________________________
```

Figure 3.7 Simple RNN module summary

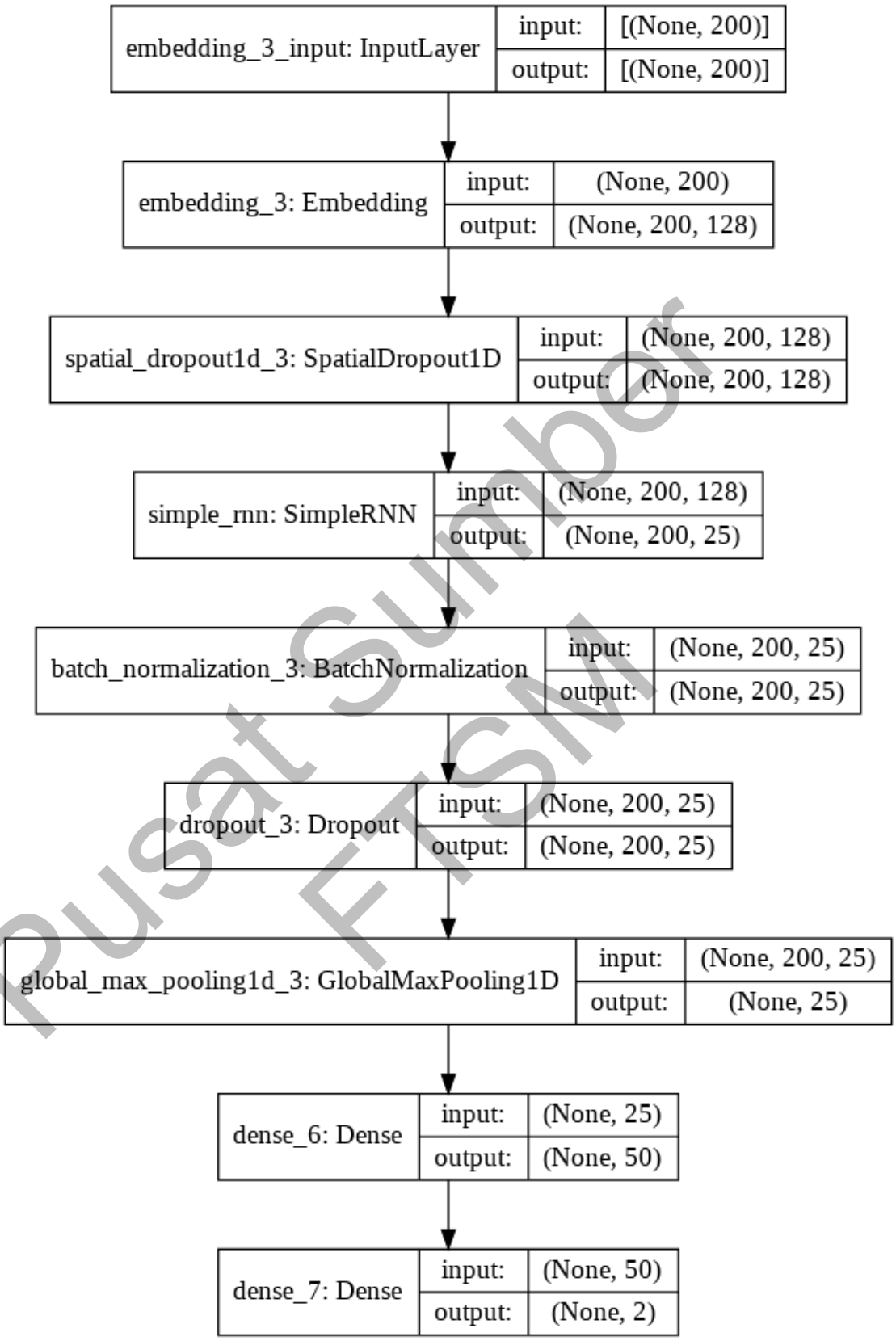


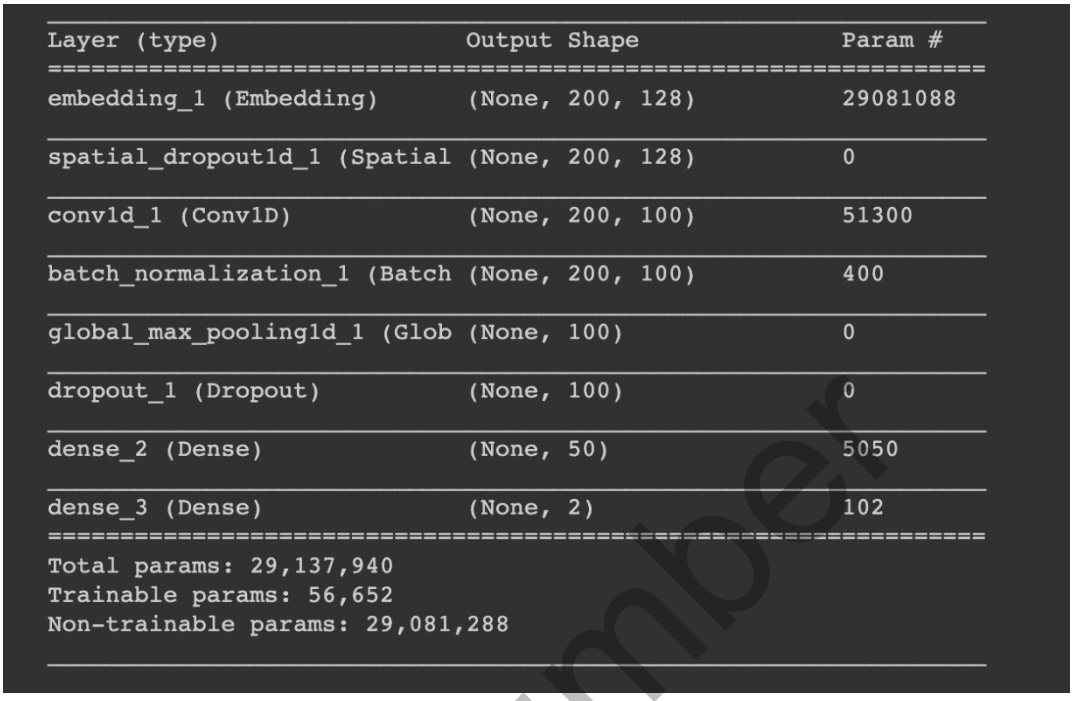
Figure 3.8 Simple RNN module architecture

3.7.2 Convolutional Neural Network parameters and layers

This section describes the layers and the parameters used in Convolutional neural networks.

The diagram below in Figure 3.9 and Figure 3.10 illustrates the architecture of CNN.

- The first layer is the embedding layer imported from the resultant vector space word embedding of Gensim Word2Vec in section one.
- The second layer is the spatial dropout that usually helps promote independence between feature maps.
- The third layer is the Convolution layer.
- The fourth layer is batch normalization, a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This stabilizes the learning process and dramatically reduces the number of training epochs required to train deep networks.
- The fifth layer is a dropout, which refers to ignoring units (i.e. neurons) during a particular set of neurons chosen at random during the training phase.
- The sixth layer is a method of down sampling the whole feature map to a single value known as global max pooling. Setting the pool size to the size of the input feature map would achieve the same result.
- Two classes are represented by the seventh layer, which is a dense layer resulting in two neurons' output layers.



Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 200, 128)	29081088
spatial_dropout1d_1 (Spatial	(None, 200, 128)	0
conv1d_1 (Conv1D)	(None, 200, 100)	51300
batch_normalization_1 (Batch	(None, 200, 100)	400
global_max_pooling1d_1 (Glob	(None, 100)	0
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 2)	102

Total params: 29,137,940
Trainable params: 56,652
Non-trainable params: 29,081,288

Figure 3.9 Simple CNN module summary