DZONGKHA WORD SEGMENTATION USING DEEP LEARNING

YESHI JAMTSHO

A Thesis Submitted to the Graduate School of Naresuan University
in Partial Fulfillment of the Requirements
for the Master of Engineering in (Computer Engineering - (Type A 2))
2019

Copyright by Naresuan University

DZONGKHA WORD SEGMENTATION USING DEEP LEARNING

YESHI JAMTSHO

A Thesis Submitted to the Graduate School of Naresuan University
in Partial Fulfillment of the Requirements
for the Master of Engineering in (Computer Engineering - (Type A 2))
2019
Copyright by Naresuan University

Thesis entitled "Dzongkha Word Segmentation Using Deep Learning"
By YESHI JAMTSHO

has been approved by the Graduate School as partial fulfillment of the requirements for the Master of Engineering in Computer Engineering - (Type A 2) of Naresuan University

Oral Defense Committee

.... Chair
(Associate Professor Chakchai So-In, Ph.D.)

.... Advisor
(Professor Paisarn Muneesawang, Ph.D.)

.... Internal Examiner
(Assistant Professor Phongphun Kijsanayothin, Ph.D.)

.... Internal Examiner
(Assistant Professor Panomkhawn Riyamongkol, Ph.D.)

Approved

....
(Professor Paisarn Muneesawang, Ph.D.)

for Dean of the Graduate School

Title DZONGKHA WORD SEGMENTATION USING DEEP LEARNING

Author YESHI JAMTSHO

Advisor Professor Paisarn Muneesawang, Ph.D.

Academic Paper Thesis M.Eng. in Computer Engineering - (Type A 2), Naresuan University, 2019

Keywords Dzongkha word segmentation, Deep Learning, Natural Language Processing, Window approach, Deep Neural Network, Bi-LSTM RNN, Syllable tagging

ABSTRACT

Dzongkha is the national language of Bhutan. The preservation and promotion of the national language are of the utmost importance because the language represents the identity of the country. Focusing and advancing in the field of Natural Language Processing (NLP) and its applications can be the technological movements toward the said goal. However, there is no advancement seen in the field of Dzongkha language processing and its research. Also, the development of NLP applications is challenging because the Dzongkha is written as a string of syllables without an explicit word delimiter.

For such language, the word segmentation is the first and fundamental step towards building NLP applications. The word forms the basic constituent for the NLP task such as translator and the participation of the word in the given sentence or phrase determines the meaning. In this thesis, the Dzongkha word segmentation is formulated as the syllable tagging problem because the word is formed as a combination of one or more syllables. The tag of the syllable represents the position of the syllable in a word. There are many techniques for tagging ranging from dictionary-based to modern approaches. The deep learning algorithm, particularly Deep Neural Network (DNN) and Bi-directional Long Short-Term Memory (Bi-LSTM) were proposed. The usage of deep learning algorithms avoids the need for manual feature engineering.

In our experiments for the DNN model, the window approach was implemented to incorporate contextual information of the target syllable. The context size ranging from 0 to 3 were considered to determine the most suitable context size for the Dzongkha language. Two experimental sets were designed based on the usage of pretrained syllable embedding. Each set comprises of four models of various context sizes. Amongst the eight models, the model with context 2 using pretrained syllable embedding achieved the highest accuracy of 94.35% and F1-score of 94.40% with 94.47% precision and 94.35% recall.

There is no thumb rule to determine the optimal hyperparameters for the deep learning algorithms. We have designed 24 Bi-LSTM models with different configurations, which can be broadly classified into two experimental sets, based on the neuron size: 256 and 512. Amongst these models, the model with the configuration of 256 neuron size, embedding dimensions of 128, the learning rate of 0.01 and without dropout achieved the highest accuracy of 95.25%, which is 0.90% higher than the DNN based model. Further, the proposed deep learning models have been compared with traditional machine learning algorithms like CRF and SVM, which shows the proposed model outperformed the traditional machine learning approaches.

Out-of-vocabulary (OOV) is are the most prominent issue to be considered for language processing. Both of the models were designed to handle the OOV syllables. My work is the first of its to apply Deep Learning algorithms in the field of Dzongkha language processing and I consider the performance achieved by both of the models as the significant one.

ACKNOWLEDGEMENTS

I am really out of words to express my heartfelt gratitude to His Majesty the King of Bhutan, Jigme Khesar Namgay Wangchuk, for showering me with the prestigious postgraduate scholarship. Also, I extend my sincere gratitude to Naresuan University for offering me the scholarship to study at the university with free access to every facility. Thank you, NU, for making my study period homely.

I would like to offer my deepest and sincere appreciation to my advisor Prof. Dr. Paisarn Muneesawang for his continued support rendered during my study and research, patience, motivation, enthusiasm, and immense knowledge. His constant guidance helped me to reach this far, for which I consider him as the most precious mentor in my study.

Further, I am extremely grateful to my committee for their critical recommendations and suggestion which is the basis for my personal development and particularly, my thesis: Assoc. Prof. Dr. Chakchai So-In, Asst. Prof. Dr. Panomkhawn Riyamongkol, and Asst. Prof. Dr. Phongphun Kijsanayothin.

I cannot forget to thank Dzongkha Development Commission (DDC) and particularly, Senior ICT officer, Mr. Tenzin Namgyel for providing the syllable tagged corpus as the dataset for my experiments. Your unwavering support has enabled me to complete my experiment on time.

My sincere acknowledgment also extends to the Royal University of Bhutan, and in particular, to the management of the College of Science and Technology, for being very supportive and granting study leave for my postgraduate studies.

Lastly, I could not have completed this thesis without the support from my family, relatives, colleagues, friends and all the well-wishers. Thank you for your prayers and love. Your presence gave me the strength to work harder and inspired me to become a productive and responsible student.

YESHI JAMTSHO

TABLE OF CONTENTS

มหาวิทยาลัยนเรศวร

List of Tables

Page

List of Figures

Page

CHAPTER I

INTRODUCTION

Introduction

Natural Language Processing (NLP) is the subfield of Artificial Intelligent concerned with the cognitive development of the computer system to understand natural (human) language. NLP has witnessed its application in machine translation, spell and grammar checker, Text to speech system and many more. The word forms the fundamental constituent of processing natural language. However, identification of words in languages without explicit word boundaries, unlike in the English language is challenging. Word Segmentation is the process of breaking down the input text sequences into its constituent words. The segmentation task is considered as the fundamental steps for building the NLP applications, for the languages without explicit word boundaries. The words in the natural language share semantic and syntactic relationships, consequently, their participation determines the meaning of the phrase or text.

Dzongkha language, the national language of Bhutan, is one of the languages without explicit word delimiters. The scripts are written as a string of smallest token called syllable. This chapter introduces an overview of my thesis on "Dzongkha Word Segmentation using Deep Learning". This chapter is organized into sections as Background and significance of the study, the purpose of the study, problem statement, scope of the study, basic assumption, hypothesis of the study, summary of the work and the contributions of the work.

Background and Significance of the Study

Dzongkha is the national language of Bhutan since 1971 (DDC, 2019). The national language represents the identity and sovereignty of the country; thus, it is important to preserve and promote the language. The Royal Government of Bhutan (RGOB) under the royal command of His Majesties the third king, late Jigme Dorji Wangchuk and His Majesty the fourth king, Jigme Singye Wangchuk, has initiated various steps for the preservation, development, and promotion of language thorough formation of the committee such as Dzongkha Development Division under Ministry of Education and Dzongkha Advisory Committee in 1986, and introducing Dzongkha as a subject in the schools (DDC, 2019). In 1989, Dzongkha Development Commission (DDC) was established that functions autonomously for the development and promotion of the language in the country with a vision "*To make Dzongkha the main medium of communication for every Bhutanese in order to promote harmony, cohesion and stability in the country*" (DDC, 2019).

Preservation and Promotion through the technology perspective are never witness for the Dzongkha language. Natural Language Processing (NLP) is a sub-field of Artificial Intelligence (AI) that deals with extending the capability of computers to understand the statement or words of the human language or cognitive development of the computer system to understand the natural language. The technological advances in computational power and machine learning algorithms, and the availability and accessibility to organized linguistic data has enabled NLP to gained much attention in the research and development (Hirschberg & Manning, 2015; Young, Hazarika, Poria, & Cambria, 2018). In recent times, it has spread its applications in various fields such as machine translation, email spam detection, information extraction, summarization, medical, spell or grammar checker, Text to Speech (TTS), Speech to Text System, question answer system and many more.

The computerization of the Dzongkha language can be considered as the technological movement towards the preservation and promotion of the language, and further enabling the language as a medium of communication for the foreigners in the country and the Bhutanese. According to the Tourism Council of Bhutan (2018) report, Bhutan has seen an increasing tourist arrival rate in the country with a total of 274,097 visitor arrivals in 2018 which is 7.61% increase over the past years. The application of NLP such as machine translation and speech synthesis would benefit the tourist to interact with the Bhutanese people in the village and farm to get more insights into the culture and traditions. This would enable the tourist to feel homely in the alien country.

Further, NLP applications such as TTS and translator would enable easier access to a wide range of information irrespective of one's qualification and capability. For example, daily news published in English can be read by those with no knowledge of English by using the "translator" and a visually impaired person (without braille knowledge) can write his thoughts by using speech to text system.

However, advancement has not been witnessed in terms of Dzongkha language processing. It is due to the fact that Dzongkha is written as a string of syllables, without explicit word delimiter unlike in English. The word forms the basic constituent towards the task of NLP applications. The participation of the words in the given phrase, sentence or text plays an important role in representing the meaning of the text. The meaning of the phrase, sentence and paragraph is always dependent on the context. Thus, increasing the complexity of developing NLP applications. The computer does not have common sense knowledge and reasoning capability (Arun et al., 2016) as we do have. Therefore, the word segmentation system plays an important role in recognizing the words in the given input sentence and it would enable the development of other NLP toolkits and its application.

Word segmentation is the process of breaking the given phrase or sentence into its constituent words. For example, the sentence "ཁྱོད་ཀྱི་སྐོར་ལས་ག་ར་བསྟན་འཛིན་གྱིས་སླབ་དེས།" which means "*Tenzin has shared everything about you.*" can be segmented as "ཁྱོད་ | ཀྱི་ | སྐོར་ལས་ | ག་ར་ | བསྟན་འཛིན་ | གྱིས་ | སླབ་ | དེས | །", where vertical stroke "|" delimits the segmented words of the given input sentence. Thus, a robust and intelligent word segmentation system using the most recent efficient algorithm is desired for the Dzongkha language.

This research aims to assist DDC in terms of Dzongkha language processing and young researchers who wish to research in the field of Dzongkha language processing, thereby promoting and preserving the language. Many researches and development have been done for other languages like Thai, Chinese and many other languages. However, there is not much research conducted in Dzongkha word segmentation other than work of (Norbu, Choejey, Dendup, Hussain, & Muaz, 2010). The research on Dzongkha word segmentation using deep learning can be the first of its kind.

Purposes of the Study

The main goal of this research to build a robust and intelligent Dzongkha word segmentation system that can be used in future research for further development of

Dzongkha language toolkits such as POS tagging, information retrieval, Machine translation and many more, by fulfilling the following purpose:

- Since Dzongkha language is written as a string of syllables, formulate the Dzongkha segmentation task as syllable tagging task
- Develop deep learning model for Dzongkha word segmentation using Deep Neural Network (DNN) and Bi-directional Long Short-Term Memory (Bi-LSTM),
- The model is designed for the accurate identification of the position of a syllable in a word,
- Compare the performance of Deep Learning model with the performance of traditional machine learning models such as Support Vector Machine (SVM) and Conditional Random Field (CRF), and
- The model can handle the out-of-vocabulary word efficiently.

Problem statement

Dzongkha Development Commission (DDC), the autonomous government organization that is considered as the premier institution responsible for developing and promoting Dzongkha in the country has realized the potential of NLP in promoting the language and enabling the language as one of the communication mediums in disseminating the information. The office explores various opportunities for collaboration with academic institutions like College of Science of Technology under the Royal University of Bhutan and other renowned institutions in India and abroad, to work on Dzongkha language Processing and come up with various NLP application for Dzongkha like Spell and Grammar Checker, Machine translation, Automatic Speech recognition, Speech Synthesis system, Optical Character Recognition and many more. However, despite their unwavering effort, nothing much has been observed advancing in the field of Dzongkha Language Processing.

As discussed in the background section, Dzongkha is written as a string of syllables without explicit word delimiters which makes the computerization of Dzongkha challenging. Dzongkha sentences or phrases are written in the form of continuous syllable separated by *Tsheg* without proper word boundary, unlike English. Word forms semantic and syntactic constituent for language processing and thus, it plays an important role in the language processing task. Therefore, word segmentation is considered one of the fundamental steps towards effective Language Processing (Norbu et al., 2010; Theeramunkong & Usanavasin, 2001; C. Wang & Xu, 2017) for those languages without word delimiters. The intelligent and robust word segmentation system is desired for the advancement of Dzongkha in the field of NLP.

Scope of the Study

The scope of the research is to develop the intelligent and robust Dzongkha word segmentation with the following milestones

1. Conduct a literature review in the field of word segmentation,
2. Explore deep learning algorithms,
3. Apply deep learning algorithms in the field of NLP, particularly Dzongkha Word Segmentation,
4. Provide performance comparison between some of the traditional machine learning approach and the proposed deep learning approach, and
5. Optimize the deep learning models using batch normalization and learning rate scheduler.

Basic Assumption

During the research following assumptions will be made:

1. The model can intelligently identify and handle out-of-vocabulary (OOV) word,
2. The Deep Neural Network model performs better with windows approach,
3. The proposed deep learning model outperforms the traditional machine learning approach, and
4. The tagging model is trained with enough datasets

Hypothesis of the Study

1. Does the proposed deep learning model perform segmentation with high accuracy?
2. How does the context contribute to enhance the efficiency of the model?
3. Does the proposed deep learning model outperform the traditional learning approaches such as SVM and CRF? and

4. Does the proposed model handle the OOV word efficiently?

Summary of the work

The Dzongkha script is written as a string of syllables without explicit word boundaries. The Dzongkha word segmentation task was formulated as a syllable tagging task. The tag of the syllable represents the position of the syllable in a word. Three tags were used in this thesis such as: '*beg*' tag represents the first syllable of a word or a syllable that forms the word by itself, '*end*' tag represents the last syllable of a word while the '*mid*' tag marks the syllable in between the syllables with '*beg*' and '*end*' tags.

The traditional machine learning approaches heavily depend on manual feature engineering, which can be incomplete, time-consuming and requires linguistic knowledge. The performance of such models depends on the effectiveness of manual feature engineering. In this thesis, Deep learning algorithms, in particular, Deep Neural Network (DNN) and Bi-directional Long Short-Term Memory (Bi-LSTM) were proposed for the syllable tagging task. A *word2vec* model from the *scikit-learn* library was used to generate the syllable vectors or embedding.

Windows approach was incorporated for the DNN approach as the tag of the syllable depends on its surrounding syllable. Window or context sizes from 0 to 3 were used for the experiment. DNN experiments were conducted in two sets with four experiments in each set. The set is categorized by the usage of pretrained syllable embedding from the *word2vec* model. The DNN with context size 2 which is trained with pretrained syllable embedding achieved the highest accuracy of 94.35%.

The DNN approach considers each input as an independent entity, where in reality, each syllable in the input sentences shares the dependency. Thus, Recurrent Neural Network-based Bi-LSTM was proposed for the syllable tagging task. In this case, the previous information is used for tagging the current syllable. In this case, 24 different experimental configurations were proposed, among which one of the configurations provided the highest accuracy of 95.25%, outperforming the DNN based approach by 0.90%.

In addition to our deep learning-based proposed models, experiments employing traditional machine learning algorithms such as Support Vector Machine

(SVM) and Conditional Random Field (CRF) were conducted. The features are manually derived, unlike in the Deep learning approach. The performance of CRF was observed higher than the SVM models by 12.70%. Further, the CRF performance was compared with the proposed deep learning models. DNN and Bi-LSTM based models achieved 2.55% and 1.65% higher than traditional CRF models, respectively. All these models were trained using the dataset provided by the Dzongkha Development Commission. Finally, the words are formed by combining the syllable based on their tag.

The Contributions of the work

The experiments proposed for Dzongkha word segmentation in this thesis were successfully conducted and the main contributions of my work are as follows:

1. The Deep Learning algorithms (DNN and Bi-LSTM) have been applied to the field of Dzongkha word segmentation. The models avoid the need for manual feature engineering, unlike in traditional machine learning approaches. Application of Deep Learning approaches to the Dzongkha language is the first of its kind.
2. The windows approach was explored for deriving the contextual information of the target syllable for the tagging task using DNN.
3. The exploration of Dzongkha word segmentation using traditional approaches are also the first of its kind in the field of Dzongkha word segmentation. The performance of traditional machine learning approaches was compared with the deep learning approaches, showing the superiority of Deep learning algorithms.
4. Part of Speech tagging (NER) and Named Entity Recognition (NER) are also important aspects of Natural Language Processing. The proposed models can be applied for this task.
5. The work presented in this thesis would assist the Dzongkha Development Commission (DDC) in making plans for the advancement of the Dzongkha language. Further, the results obtained in this research can be used as the basis for upcoming research.

CHAPTER II

RELATED WORKS AND STUDIES

Introduction

This chapter provides an Overview of the Dzongkha language, Literature review on the Segmentation task and further, it discusses Deep Learning algorithms.

Overview of the Dzongkha language

Bhutan is linguistically rich (Wangdi, 2015) country with 21 spoken languages (Driem, 1992) and geographically small landlocked country with a total population of 735,553 as of May 30, 2017 (NSB, 2017) in the Asia continent, sharing its border with India in the south and China in the north. Dzongkha is adopted as the official and national language of Bhutan since His Majesty the third King of Bhutan, Jigme Dorji Wangchuk passed a royal decree in 1971 and according to Article 1(8) “the Constitution of Kingdom of Bhutan”. Further, the people of eight western dzongkhags or districts (Thimphu, Paro, Ha, Gasa, Chukha, Punakha, Wangduphodrang, and Dagana) use Dzongkha as their native language. According to Dorjee (2014), Dzongkha is the widely spoken language among other spoken languages in the country with an estimated native speaker of 160,000. It is adopted as the *lingua franca* where most of the Bhutanese use as a common language for communication with others using different native.

Dzongkha is defined as the language spoken in the fortresses (*kha* - language and *Dzong* - fortress). All spoken languages except *Lhotsamkha* or *Nepali* are Tibeto-Burman Languages. The consonants, vowels, and digits of the Dzongkha language are shown in Figure 1.

Dzongkha Consonants

ཀ	ཁ	ག	ང	ཅ	ཆ	ཇ	ཇ	ཉ	ཏ	ཐ	ད
ན	པ	ཕ	བ	མ	ཙ	ཚ	ཛ	ཝ	ཞ	ཟ	འ
ཡ	ར	ལ	ཤ	ས	ཧ	ཨ					

Dzongkha vowels

◌ི	◌◌ུ	◌ེ	◌ོ

Dzongkha digits

༠	༡	༢	༣	༤	༥	༦	༧	༨	༩

Figure 1 Dzongkha character sets

A sample of Dzongkha sentence is shown below:

རྫོང་ཁ་གོང་འཕེལ་ལྷན་ཚོགས་ཀྱིས་ ཡོངས་གྲགས་ཡོད་པའི་གློག་རིག་རིམ་ལུགས་ཚུ་ནང་ལུ་ གནས་ཚད་གཞི་བཞག་གི་རྒྱབ་སྐྱོར་མཐུན་རྐྱེན་ཚུ་བཟོ་ནི་ལུ་བརྩོན་ཤུགས་བསྐྱེད་དོ་ཡོདཔ་ཨིན།

which translates as "*The Dzongkha Development Commission is dedicated to developing standard-based support and increased functionality on popular computing platforms.".*

The Dzongkha sentence or script is written in the form of continuous syllables as seen in the sample sentence. A syllable is the smallest token in the Dzongkha script formed by a single or collection of characters. The syllable "རྫོང་" is a collection of four characters excluding the special dot character while "ཁ་" is a syllable of a single character. Each of the syllables in the script is separated by a special dot character called *Tsheg* (་). A Dzongkha word can be represented either by a single syllable or a combination of syllables as shown in Table 1. The vertical stroke (|) in the table was used to separate the syllables. In most cases, a vertical stroke called *Shad* (།), is used to terminate the sentence. Figure 2 shows a Dzongkha word illustrating syllable, *Tsheg*, and *Shad*.

Table 1 A sample Dzongkha words with its number of syllables

Dzongkha words	**Number of the syllable (s)**	**Translation**
ང་	1	I
རྒྱལ་ཁབ་ (རྒྱལ་ | ཁབ་)	2	Country
སློབ་སྦྱོང་པ་ (སློབ་ | སྦྱོང་ | པ་)	3	Trainee
ཡོངས་འབྲེལ་རིམ་ལུགས་ (ཡོངས་ | འབྲེལ་ | རིམ་ | ལུགས་)	4	Internet

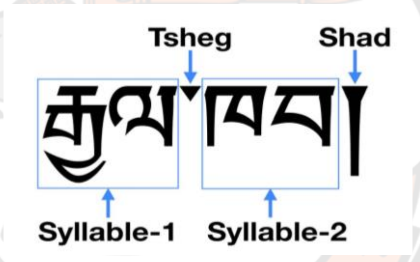


Figure 2 A labeled Dzongkha Word

Literature Review

The robustness and effectiveness of the Natural Language Processing (NLP) applications like a translator, Spell Checker, Question and answering system and many more depend on the perfectness of the segmentation system (Noyunsan, Haruechaiyasak, Poltree, & Saikaew, 2014). The segmentation system allows the system or computer to identify the word in the sequential data sentences or paragraphs. The word segmentation is considered as the initial step (Sproat, Gale, Shih, & Chang, 1996) and the fundamental steps in the building the NLP applications (Chirawichitchai, 2014; Tanaya & Adriani, 2016).

However, the segmentation is not a big deal for the language like English because one can easily consider *whitespace* as the word boundaries, to delimit the words. But most of the Asian languages such as Chinese (Sproat et al., 1996), Thai and Japanese are written without in the continuous form without a proper word delimiter. The word segmentation for these languages is important but it is challenging (Peng, Feng, & McCallum, 2004).

Similarly, Dzongkha is written as a sequence of syllables without word explicit delimiters. However, the syllables are separated with a dot-like character called '*Tsheg*' while vertical stroke called '*Tshad*', which usually marks the end of the sentences. But in some cases, the Dzongkha sentences end without a vertical stroke, making it more complex.

Considering the importance of the word segmentation, many researches have been conducted on word segmentation for those languages (Thai, Chinese, Japanese) without explicit word delimiters adopting various algorithms or approaches starting from a dictionary-based approach to Deep Learning (DL) approach. In contrast, there is not much research carried out for Dzongkha word segmentation other than the work of Norbu et al. (2010). The complexity of the language and unavailability of the public corpus or dataset might have contributed to the researcher for not taking up the study on Dzongkha word segmentation.

According to Xue (2003) and M. Wang, Li, Wei, Zhi, and Wang (2018), the approaches adopted for word segmentation can be categorized as follows:

Dictionary-based approach

The dictionary-based approach uses the predefined dictionary that contains a set of words in the language. It employs a greedy search routine called a maximum matching (MM) algorithm that scans through the syllables or characters in the given sentences. This algorithm always favors the longest string as a word. For example, consider the word 'སེམས་' (heart) and 'སེམས་རྟོགས་ཁ་' (*Semtokha*- name of Place) in the dictionary. Given an input 'སེམས་རྟོགས་ཁ་' (Semtokha), the MM algorithm will output 'སེམས་རྟོགས་ཁ' at first scan and later 'སེམས་'. The MM algorithm can perfectly handle the words that are available in the dictionary (Theeramunkong & Usanavasin, 2001). However, in the above example, it fails to identify the most appropriate word when it has more than one-word candidate, rather all the candidates are returned as the correct segmented word. Therefore, the MM approach cannot select a suitable word given in the context.

Statistical approach

The statistical approach relies on the probability of the adjacent characters to decide whether the combination of syllables forms a word. Given a sequence of syllables 'S_1, S_2, ……, S_n', the pair of adjacent characters with the largest mutual information greater than a predefined threshold are grouped as a word. This process is repeated until there are no more pairs of adjacent characters with the mutual information value greater than the threshold. The dictionary is not required in the statistical approach because the probabilities can be computed from the easily available unsegmented data, but this doesn't give better accuracy (Xue, 2003).

Hybrid approach

The hybrid approach uses both MM and statistical approaches. When MM produces more than one possible word, the statistical approach can be applied to identify the correct word based on contextual probabilities. This approach outperforms the other two standalone approaches because the approach is guided by the dictionary along with mutual information.

The research carried out by Norbu et al. (2010) and (Dhungyel & Grundspeņķis, 2017) are the only work that is carried out in pursuit of Dzongkha word segmentation. In their work, they have proposed Dzongkha word segmentation based on Maximal Matching followed by bigram techniques and reported an overall accuracy of 91.5%. Their model was built and tested using data from the various domain of information such as astrology, newsletter, notification, religion, song lyrics and many more.

It is understood that it is easier to implement to Segmentation based on the Maximal Matching technique followed by the bigram technique and provides higher accuracy when it deals with the words in the corpus (Chen, Zhao, & Yang, 2017). However, in a real-time scenario, it is expected to come across new words that are not available in the training corpus or dictionary. Such unknown words are known as out-of-vocabulary words. Therefore, in such a situation, it is reported by Theeramunkong and Usanavasin (2001) that the accuracy decreases as the number of unknown words or out-of-vocabulary words increases.

The new words are still emerging in Dzongkha Language to sustain the language with the developing world and technology. The formation of new words in Dzongkha comes from the name (the name of the places, people and animals) which are formed by an unprecedented combination of the syllables. Secondly, the world is advancing with high technology which arises with new technological terms. The technological terms are transcribed that leads to the formation of new words, such as Artificial Intelligence (བཙོས་རིག།) or Algorithm (རྩིས་ཐབས།) or computer (ཀམ་པིའུ་ཊ།). Thus, an efficient technique needs to be studied for Dzongkha word segmentation that can effectively handle words that is out of vocabulary.

The Dzongkha language is in the infant stage in terms of Language computerization compared to other languages. For instance, the Google Translate, the multilingual translation service provided by Google, supports more than 100 languages in the world like English, Thai, Arabic, Hindi, Chinese, Danish, Estonian, etc. However, the translation for Dzongkha is not supported in the application, which indicates research and development are required to be carried out in the field of Dzongkha language processing.

Most Asian languages (Thai, Japanese and Chinese) are written continuously without word separator (Cai & Zhao, 2016; Theeramunkong & Usanavasin, 2001), the technique proposed for those languages can be adopted for Dzongkha word segmentation. In 2003, Xue has proposed the Chinese word segmentation as a character tagging problem (Xue, 2003). The identification of character position in a word is considered to be the most crucial step for effective Chinese word segmentation since it is written in a sequence of strings without explicit delimiter like in English. In his work, four tags were used to mark the position of the Chinese character. The character that forms a word by itself is tagged as 'LR', 'LL' for those characters who appear to appear at the left side of a word while 'RR' represents the character in the right side of the word and the characters that appear in the middle of the word are tagged as 'MM'. The characters are later concatenated based on their corresponding character-tag to form a valid word.

Similarly, Dzongkha words are formed by the sequence of syllables. The syllables can be tagged in a similar fashion, as adopted by (Xue, 2003). For example, consider the word ‘སེམས་རྟོགས་ཁ’ which can be represented as {‘སེམས’, ‘རྟོགས’, and ‘ཁ’}. The labels can be assigned to these syllables according to their position of appearance in the word. Thus, the Dzongkha word segmentation problem can be formulated as a syllable tagging problem.

Many studies were done towards the effective word segmentation of Chinese and Thai Languages using decision tree technique (Theeramunkong & Usanavasin, 2001), maximum entropy (Low, Ng, & Guo, 2005), Conditional Random Field (CRF) (Zhao, Huang, & Li, 2006). Further, a study on Tibetan word segmentation using CRF was conducted and reported F1-score of 95.12% on the 131,903 training sentences and 1000 test sentences (Hu & Liu, 2017). The techniques discussed in this part heavily depend on the hand-crafted features and featuring engineering consumes lots of time (Cai & Zhao, 2016; C. Wang & Xu, 2017; Young et al., 2018; Zheng, Chen, & Xu, 2013). Also, the task requires linguistic knowledge to identify features, which can be incomplete. The system performance is directly proportional to the effectiveness of feature engineering.

Therefore, a novel approach is desired for Dzongkha word segmentation where the need for feature engineering is avoided and the systems automatically extract features on its own. Neural Network approaches were proposed for Chinese word segmentation such as in (Cai & Zhao, 2016; C. Wang & Xu, 2017; Zheng et al., 2013) that learn features automatically from a large unlabeled training data.

Deep learning in NLP

Deep learning (DL) is a subfield of machine learning (ML), depends on a set of algorithms to learn multiple levels of representation to find a model for high-level abstractions in data. DL tries to mimic the human brain, by constructing an architecture that consists of an input layer and an output layer with many hidden layers between them. These hidden layers are responsible for doing complex computations to extract features from the raw data to obtain a better representation. Deep learning architectures and algorithms have already made impressive advances in fields such as computer vision and pattern recognition (Young et al., 2018). Following the trend, deep learning has been increasingly used and witness many breakthroughs in NLP.

Many DL techniques are used for NLP, such as deep neural networks (DNN), Convolutional Neural Networks (CNN), and recurrent neural networks (RNN). The application of Deep Learning in NLP can be seen chatbot (Google Assistant), speech recognition, Document summarization, machine translation and Question and Answer system (QA).

Deep Neural Networks (DNN)

The first artificial neural network was invented in 1958 by psychologist Frank Rosenblatt. A neural network is a collection of layered neurons that are connected to compute and derive meaningful insights from the given inputs. The computation is carried out inside neurons. The typical neural network (NN) (Rauber, Fadel, Falcao, & Telea, 2016) is shown in Figure 3 which comprises three layers: *input layer (Layer L_1), hidden layer (Layer L_2) and output layer (Layer L_3).*

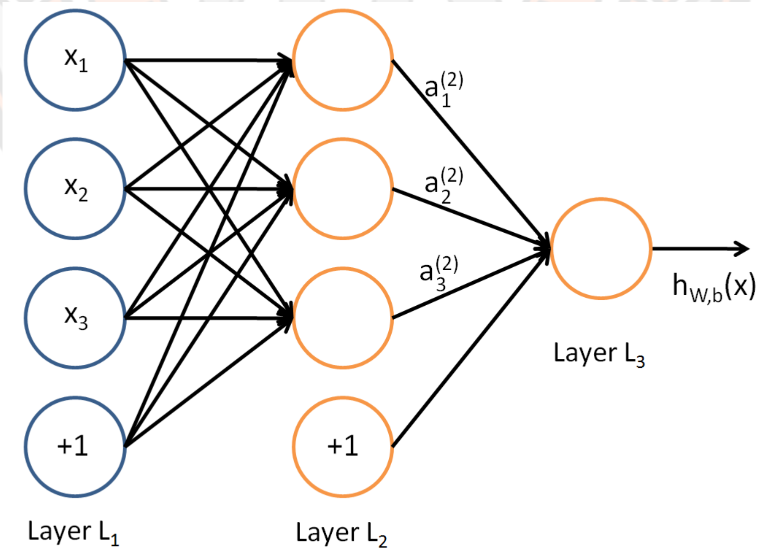


Figure 3 Typical Neural Network

Source: http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

As seen in figure 3, X_1, X_2 and X_3 are the input that is fed to the hidden layer L_2 with 3 neurons. The input can be represented as X_i and the number of neurons in each hidden layer is called the hidden size. Each neuron is associated with its weight *w* and bias *b*. The computation in each hidden layer can be seen as $Z_i = X_i * w$, so the output of the hidden layer with 3 neurons is represented as the weighted sum of all the output of each neuron Thus, each hidden unit outputs in *l*th layer output $Z_j^l = \sum_{i=1}^{n}(X_i * w_{ji}) + b$, *j* is the *j*th unit in *l*th layer while w_{ji} is the weight associated with *i*th input and *j*th unit of the *l*th layer.

The output value from the hidden neurons ranges from -α to +α. The neuron doesn't know what value should be considered to be fired for the next layers. Thus, activation function is deployed to enable neurons to make an efficient decision (Nwankpa, Ijomah, Gachagan, & Marshall, 2018). The output of the hidden neurons is passed over the non-linear activation function *f*, in which the final output of the hidden can be represented as $a_j^l = f(Z_j^l)$, which is taken as the input to the next layer. Some of the activation functions, particularly step function, tanh, sigmoid, and rectified linear unit (ReLU) are discussed in the last section of this subchapter. The summary of these activations is shown in figure 4.

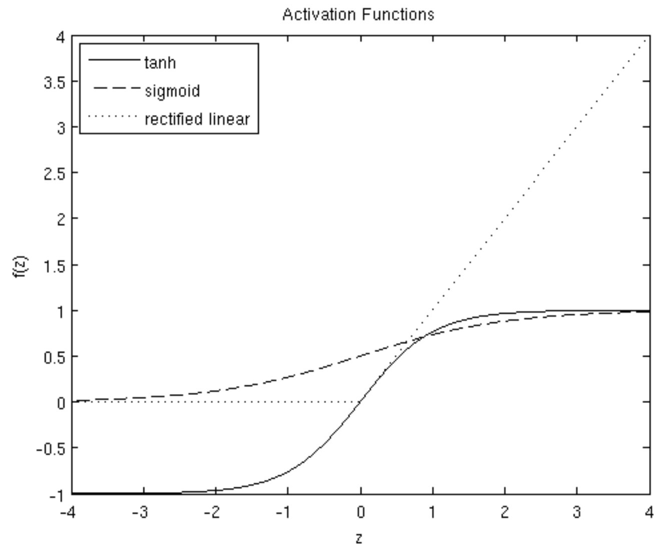


Figure 4 Activation functions

Source: http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

The same process is repeated until the output layer and such a step is called as forward propagation. The Weight *W* and bias *b* are the trainable parameters, which can be trained over backpropagation.

Convolutional Neural Networks (CNN)

A Convolutional Neural Network is also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology. CNN is very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and is used for extracting high-level features of the input.

The three layers of CNN architecture are Convolutional Layer, Pooling Layer, and Fully-Connected Layer (Gu et al., 2018). These layers to form a full CNN architecture. The CNN architecture in sentence classification is shown in figure 5.

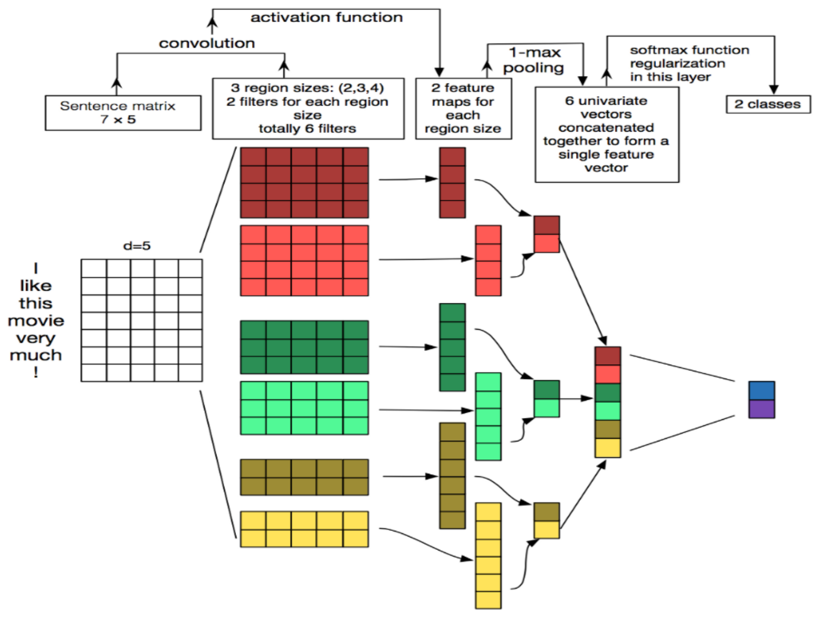

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 x 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 5 Convolutional Neural Network Architecture

Source: Zhang & Wallace, 2015

The convolution layer is the core building block of CNN. It carries the main portion of the network’s computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters known as a kernel, and the other matrix is the restricted portion of the receptive field. During the forward pass, the kernel slides across the height and width of the input producing the high-level representation of that receptive region as shown in figure 6. This produces a two-dimensional representation of the input known as a feature map. The sliding size of the kernel is called a stride.

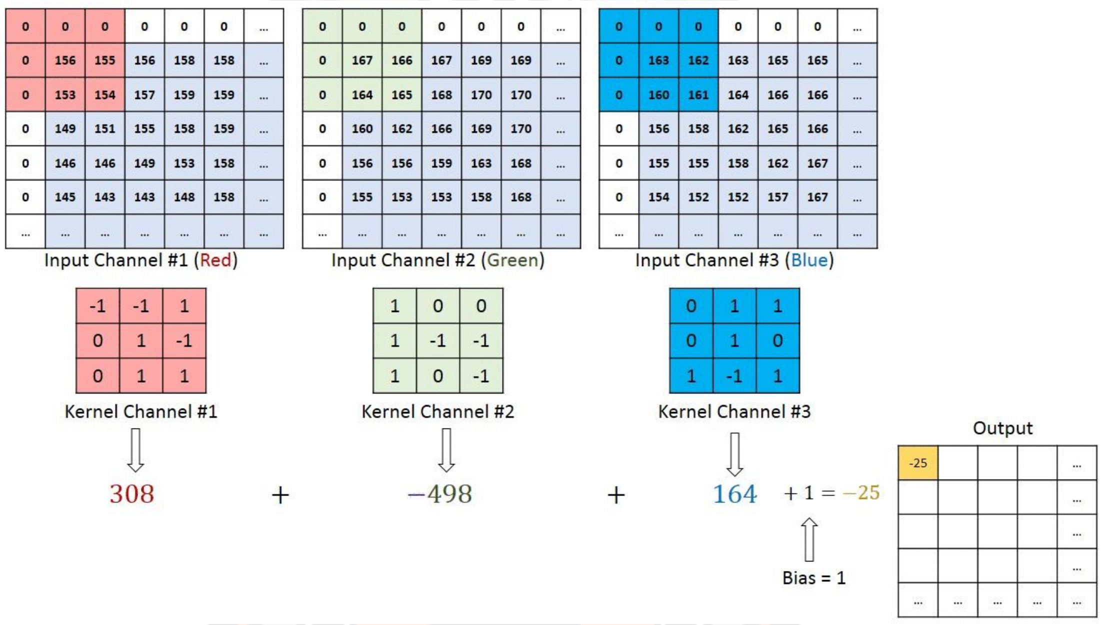


Figure 6 Convolutional operations

Source: Sumit Saha, 2018

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. The pooling operation is shown in figure 7. However, the most popular process is max pooling, which reports the maximum output from the neighborhood.

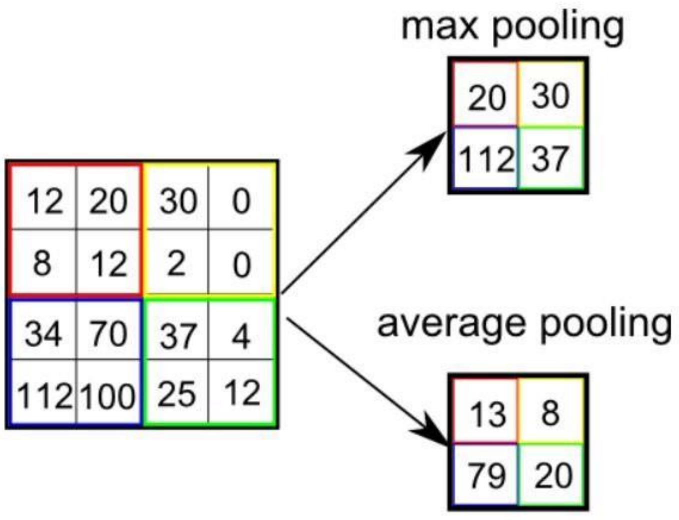


Figure 7 Types of the pooling layer

Source: Sumit Saha, 2018

The Fully Connected layer helps map the representation between the input and the output.

Recurrent Neural Networks (RNN)

RNN (Zaremba, Sutskever, & Vinyals, 2014) is a kind of neural network that uses the previous information ($a^{< t-1 >}$) to do current computation. RNN is commonly applicable for sequential data like the NLP task, where sentences are inputted in the form of a continuous word or word sequence. The RNN can be visualized as multiple copies of the single network as shown in figure 8, which takes input x_t and $a^{<t-1>}$ to output $y^{<t>}$ *and* $a^{<t>}$ which are then fed to its successor network.

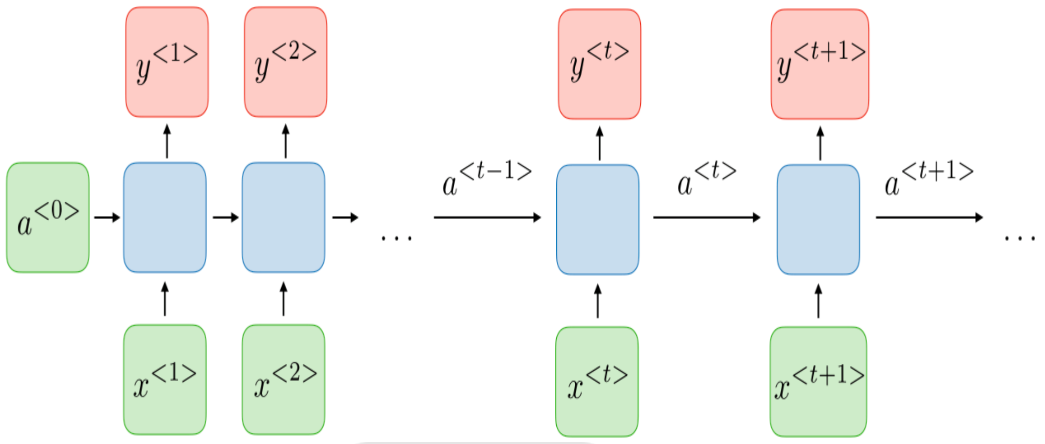


Figure 8 RNN architecture

Source: Afshine Amidi and Shervine Amidi, Stanford

For each timestamp *t*, the output $y^{<t>}$ and hidden state are expressed as

$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$ *and* $y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$.

Where W_{aa}, W_{ax}, W_{ya}, b_a and b_y are shared temporally and g_1, g_2 are the activation functions.

The standard RNN encounters a long-term dependency problem where it cannot retrieve information from a long context due to phenomena of vanishing or exploding gradient(Pascanu, Mikolov, & Bengio, 2012; Sundermeyer, Schlüter, & Ney, 2012). For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the sky,” we don’t need any further context – it’s pretty obvious the next word is going to be the sky. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use past information.

But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France. I speak fluent _____.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large. In such cases, RNNs become unable to learn to connect the information.

Long Short-term Memory units (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho, Van Merriënboer, Bahdanau, & Bengio, 2014) were introduced such problems with standard RNN by usage gates in RNN (Pascanu et al., 2012). In LSTM, cell state C_t and gates such as input gates i_t , forget gate f_t and output gate O_t. The cell state can be thought as the memory of the network. It carries relative information all the way from the sequential processing. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. The information is accumulated as the cell state goes through the sequential processing. The gates decide the information to be allowed on the cell state, in other words, it decides what to keep and what to forget during the training phase of the network. The detailed LSTM architecture is shown in figure 9.

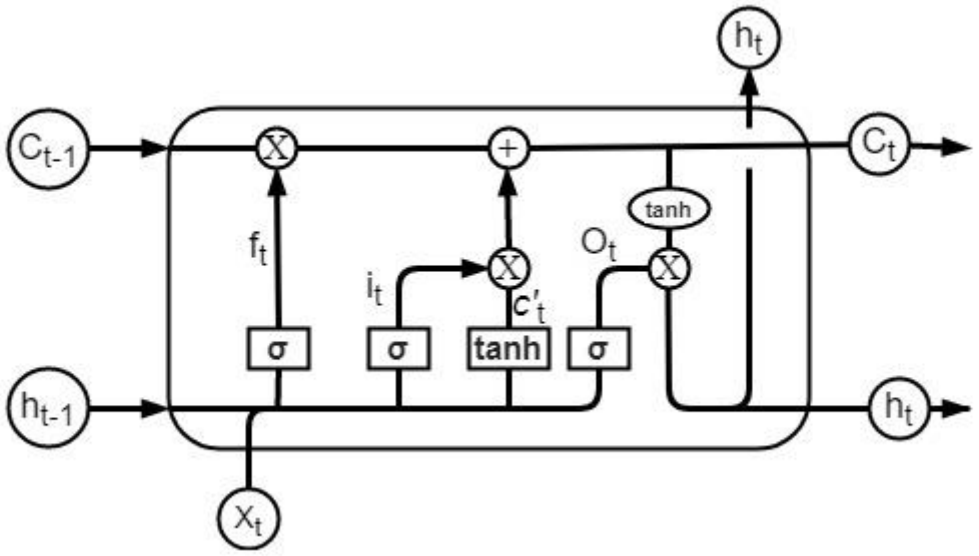


Figure 9 Detailed LSTM architecture and its gates

The forget is the first gate in the LSTM block, which decides what information should be discarded or stored. The previous hidden state h_{t-1} and current input information X_t is fed through a sigmoid function (σ), producing the output in the rage 0 and 1. The value close to 0 will be forgotten and vice versa. The forget gate f_t is mathematically represented as shown in the following equation:

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f)$$

where W_f denotes the weight and $b_f$is bias at the forget gate.

The second gate, the input gate updates the cell state by picking important information while discarding the useless information. This is achieved by passing previous hidden state h_{t-1} and current input information X_t through sigmoid function (σ). The intermediate resulting closer to 1 will be considered as important and vice versa. In addition, previous hidden state h_{t-1} and current input information X_t is passed through *tanh* function whose result will be in the range -1 to 1. The sigmoid output f_i and tanh output C'_t are then multiplied, where sigmoid output decides the importance of the information. The computation in this gate is mathematically represented as follows:

$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i)$$

$$C'_t = tanh\,(W_c[h_{t-1}, X_t] + b_c)$$

where W_i and W_c represents weight associated with sigmoid and tanh operations, respectively, b_i is the bias associated with sigmoid operation and b_c is the bias of tanh operation.

Once, using the information from input and forget gate, the cell state C_t is calculated as $C_t = (C_{t-1} * f_t) + (i_t * C'_t)$, where * denotes the pointwise multiplication and + represents the pointwise addition operation. The cell state information may be dropped if $f_t = 0$. The pointwise addition operation will result in a new cell state C_t . The cell state C_t passed to the next state at *(t+1)* timestamp.

The output gate is the last gate in LSTM architecture. It decides the information in the next hidden state. The hidden state is used for predictions in the next state, so the hidden state should contain the information from previous input. First, we pass the previous hidden state h_{t-1} and the current input X_t into a sigmoid function which is mathematically represented as follows:

$$O_t = \sigma(W_o[h_{t-1}, X_t] + b_o)$$

where O_t represents the sigmoid output, W_o is the weight associated with output gate and b_o is the bias related to the output gate. Then the newly formed cell state is passed through tanh function. The tanh function output and sigmoid output O_t are multiplied to produce a new hidden state that can be passed to the state of (t+1) timestamp. The hidden state h_t is calculated as follows:

$$h_t = O_t * tanh(C_t)$$

Activation functions

All the deep learning algorithms discussed in previous sections use the activation function. Activation functions are mathematical equations that determine the output of a neural network, its accuracy, and the efficiency of the model. In addition, it also has a major effect on the convergence of the model. It is attached to each neuron in the network, and determines whether it should be activated ("fired") or not, based on the relevancy of the neuron's input for the predictions. Also, it normalizes the output of each neuron to a range between 1 and 0 or between -1 and 1. The activation function transforms the input into a non-linear form. The non-linear transformation helps in building a powerful system.

In general, there are three types of activation functions, namely: binary step function, linear activation, and non-linear activation function. Amongst which the non-linear is used popularly in the deep learning algorithm. It is due to the following reasons:

- Binary step function doesn't support multi-value outputs, since it has only two options, i.e., 'fired' if the input is above the threshold value and vice versa,
- Though multiple output value may be supported in linear activation function, it doesn't support backpropagation and
- Non-linear activations support complex mapping between input and output which is considered an important aspect in building a powerful automated system.

Some of the commonly used non-linear activation functions in deep learning models are discussed in this section.

Sigmoid activation function

The sigmoid function is also known as the Logistic activation function. It is especially used for models where we have to predict the probability as an output.

Since the probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. It is computed as:

$$f(x) = \frac{1}{1+e^{-x}}$$

which produces the output between 0 and 1. Figure 10 illustrates the sigmoid activation function.

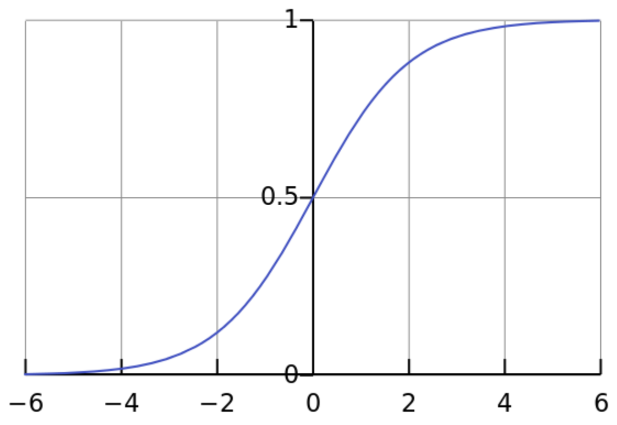


Figure 10 sigmoid activation function

Source: Avinash Sharma V, 2017

The Hyperbolic Tangent activation function

Hyperbolic Tangent activation function is also referred to as TanH activation function. It looks similar to sigmoid function but it has range of values from -1 to 1. It is computed as:

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Figure 11 illustrates the hyperbolic activation function.

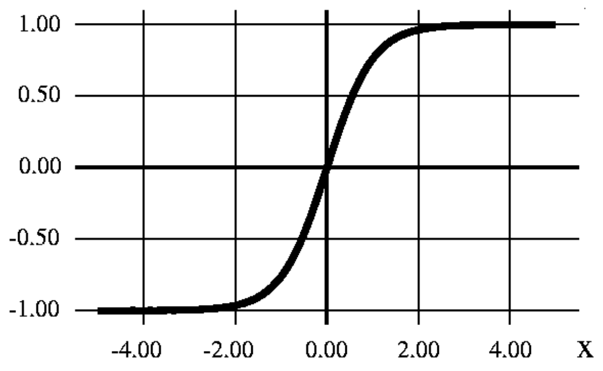


Figure 11 TanH activation function

Source: Anish Singh Walia, 2017

Rectified Linear Unit

Rectified Linear Unit abbreviated as ReLU, is commonly used in many of the deep learning models such as CNN (Gulcehre, Moczulski, Denil, & Bengio, 2016). It is computed as follows:

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

It is illustrated in figure 12.

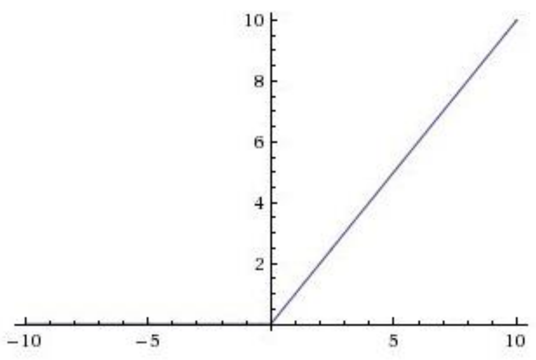


Figure 12 ReLU activation function

Source: Avinash Sharma V, 2017

Watermark: university emblem

Softmax activation function

It is considered as a more generalized logistic activation function (Basatini & Chinipardaz, 2014; Martins & Astudillo, 2016). It computes probability distribution over *n* number of possible target values, ranging the value between 0 and 1. It is computed as:

$$\sigma(z)_j = \frac{e^{zj}}{\sum_{k=1}^{K} e^{zk}}$$

CHAPTER III

RESEARCH METHODOLOGY

Introduction

The Dzongkha word segmentation was formulated as a syllable tagging problem. Since Dzongkha is written in the form of continuous syllable without the proper word delimiter, the identification of syllable position in the word is important in solving the segmentation problem of the Dzongkha language. The Dzongkha syllables were tagged as '*beg*', '*mid*' or '*end*', depending on the position of occurrence of a syllable in a word. This section presents procedures to perform Dzongkha word segmentation. The Dzongkha word segmentation can be performed in four stages. These stages are discussed in the first section of this chapter, while the second section discusses data collection.

System Overview

In our work, four stages were adopted for Dzongkha word segmentation as shown in Figure 13. Each of these stages is considered crucial for the effectiveness of the system. The stages were building word embedding (*Word2Vec*) model, preprocessing, syllable tagger and Segmentation generator. Each of these stages is discussed in the following subsections.

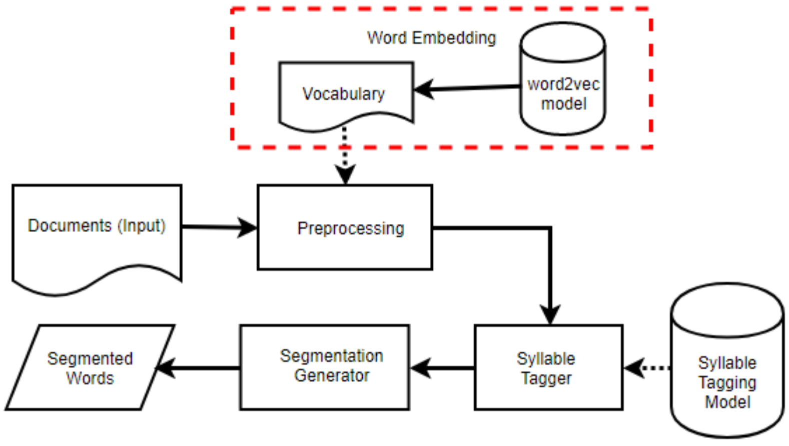


Figure 13 Segmentation system overview

Word Embedding (word2vec)

Images are represented in their pixel form that can be easily fed to the neural network. However, in terms of language processing, the input is in the form of continuous characters in which the neural network does not have the capability to interpret it. The neural network takes input only in the form of digits or numbers. Thus, the text or words have to be converted to its vector form. There are many ways to convert the text into its vector form as discusses in the following sections.

Count-based approach

One Hot encoding is the simplest form of representing the categorical variables into its vector form which can be used by the neural network as an input and process to achieve the task of interest. In one hot encoding, the words are represented in the form of *n*-dimensional space.

To understand one-hot encoding, let us consider the following similar sentences.

- **Sentence 1**: *Have a good day.*
- **Sentence 2**: *Have a great day.*

And the vocabulary *V* from the given sentences can be *V*= {*Have, a, good, great, day*}.

Now, let us create a one-hot encoded vector for each of these words in *V*. Since, the length of our vocabulary is 5, the length of our one-hot encoded vector is 5. In this case, we would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary, in which that particular element would be one. The encodings in table 2 would explain this better. The encoding can be visualized in a 5-dimensional space, where each word occupies one of the dimensional spaces and has nothing to do with the rest (no projection along the other dimensions).

Table 2 One-hot encoding

word	Have	a	Good	Great	Day
Have	1	0	0	0	0
a	0	1	0	0	0
good	0	0	1	0	0
great	0	0	0	1	0
day	0	0	0	0	1

One hot encoding can be of two variants: a bag of words (BOW) and term frequency-inverse document frequency (TF-IDF). In BOW representation, the vectors of that particular index (word) are represented as the count of occurrences of that particular word in the documents while TF-IDF is a statistical measure used to evaluate how important a word is to a document in a collection of documents or corpus. TF is a scoring of the frequency of the word in the current document. Since every document is different in length, it is possible that a term would appear more times in long documents than shorter ones. Therefore, the term frequency is divided by the document length to normalize.

$$\text{TF}\ (w) = \frac{\text{Count of Word } (w) \text{ in a document}}{\text{Total number of words in a document}} \qquad (1)$$

Inverse Document Frequency (IDF): is a scoring of how rare the word is across documents. Rarer the term, more is the IDF score.

$$\text{IDF}(w) = \log_{\text{e}}\left(\frac{\text{Total number of documents}}{\text{number of word '}w\text{' in it}}\right) \qquad (2)$$

Thus TF-IDF score =TF * IDF.

In language, each of the words shares a semantic and syntactic relationship with other words. Even though the above approaches provide the vector representation for each word in the vocabulary that can be used for processing by the neural network, it does not consider the semantic and syntactic relationship of words. Further, the dimensions of the vectors grow with an increase in the size of the vocabulary which would lead to the curse of dimensionality.

Thus, Continuous bags of word (CBOW) and skip-gram models were proposed by Mikolov, Chen, Corrado, and Dean (2013) to consider the syntactic and semantic relationship between words. These models are called *predictive models* while prior approaches are referred to as the *count-based model*.

Predictive model

The predictive model is unsupervised neural models used to compute and generate high quality, distributed and continuous dense vector representation of the words from the massive unlabelled corpora. It is also used to create a vocabulary of all possible words. Usually, you can specify the size of the word embedding vectors and the total number of vectors is essentially the size of the vocabulary. This makes the dimensionality of this dense vector space much lower than the high-dimensional sparse vector space built using the count-based approach. This model can be of two variants:

Continuous bags of word (CBOW)

This method predicts the target word from the corresponding context words or surrounding words. For instance, consider the sentence '*I am going to college*'. To predict the target word '*going*' with 2 as the context size (window size), it can be seen as a pair as ({*am*,*to*}, *going*) where *am* and *to* are the left and right context of the target word '*going*'. The neural architecture of CBOW is shown in figure 14.

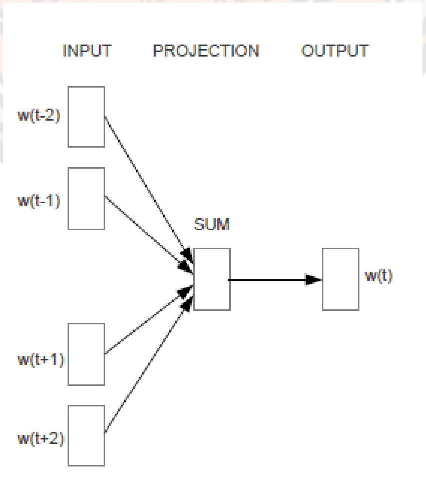


Figure 14 CBOW model

Source: Mikolov et al. 2003

The above model takes *C* context words. The vectors of the target words can be calculated as an average over all these *C* context word inputs.

Skip-gram model

The skip-gram model predicts the context words of the corresponding target word. For instance, consider the sentence '*I am going to college*'. Given the target word '*going*' with context size as 2, it tries to predict '*am*' as its left context and '*to*' as its right context. The neural architecture of the Skip-gram model is shown in figure 15.

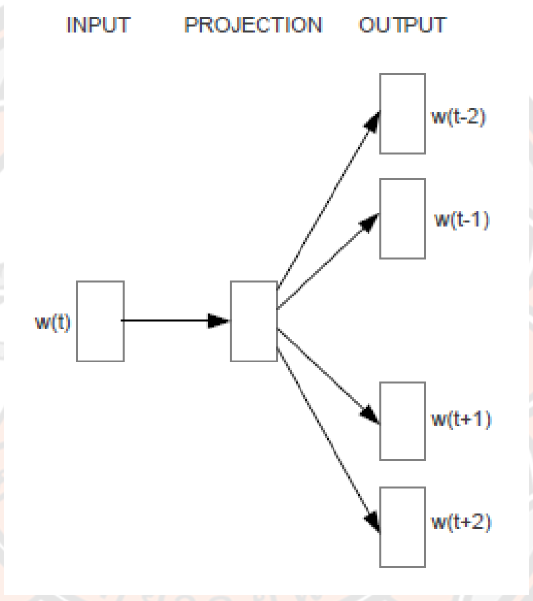


Figure 15 Skip-gram model

Source: Mikolov et al. 2003

In this thesis, the skip-gram model was adopted to generate the continuously distributed syllable vectors because Skip Gram works well with a small amount of data and it is found to represent rare words well (Mikolov et al., 2013). The unlabeled data (sentence) was fed into the embedding model (*word2vec*) to generate the syllable embedding matrix $\mathcal{M} \in \mathbb{R}^{|V| \, x \, d}$, where |V| is the vocabulary size and *d* is the dimension of the embedding matrix. Further, the trained embedding matrix can be used to generate the syllable vocabulary $V \in \mathbb{R}^{|V| \times 1}$, where $|V|$ is the size of the vocabulary.

The following code snippet shows the implementation of syllable embedding:

```
from gensim.models import Word2Vec
EMB_DIM = 300
w2v = Word2Vec(Sentences, size = EMB_DIM, window =5,min_count =1
, negative = 15, iter = 10, workers = 1)
```

The corpus `Sentences` with 10,255 sentences were fed to the *word2vec* model. Every sentence was tokenized into its syllabic form as shown below:

```
['མོང་', 'སྒར་', 'ཁྲོམ་', 'གྱི་', 'སྒྲུག་', 'ལུ་', 'ཡོད་', 'པའི་', 'དཔལ་', 'སྒྲོན་', 'ཚང་',
'ཁང་, 'ནང་', 'ཐུགས་', 'རྗེ་', 'དབང་', 'ཕྱུག་', 'གིས་', 'ཚང་', 'གཏོང་', 'ཁར་', 'བཀང་',
'།']
```

The following shows the *300-dimensional* syllable embedding for the syllable ‘གིས་’ which is retrieved from the syllable embedding matrix $\mathcal{M}$.

```
[ 0.3210197  -0.15351024  0.95191234 -0.69813704 -0.07671746  0.07590735
 -0.9091967   0.32311854  0.33156452 -0.32314035  0.2615459   0.14687999
  0.5473725  -0.71909726  0.53287125  0.03456965  0.30288315 -0.04023279
  0.19642457  0.13165298  0.04969903  0.00307721  0.30487853 -0.06859454
 -0.3712254   0.5475902   0.34564188 -0.42273006 -0.01707851  0.04979415
  0.09182609  0.46747556 -0.49909383 -0.13065352 -0.1027848   0.13512476
  0.20513588  0.18210682 -0.27366987  0.18192793  0.07094382 -0.88443625
 -0.19615975 -0.13093862  0.66337115 -0.6658243  -0.4254617   0.30282584
  0.5080116  -0.10835751  0.0742984   0.33183134 -0.2949197   0.522677
 -0.04473643  0.3196213  -0.07143073 -0.59285563  0.5399393  -0.1871956
 -0.29831967 -0.17071249 -0.05257144  0.19703664 -1.0632615  -0.16080467
 -0.18753618  0.5776778   0.5304845  -0.1659883  -0.1324222   0.43425635
 -0.50684386  0.08895883 -0.3234413  -0.31502834  0.31158307  0.06663076
 -0.44911718  0.4479407   0.3574822  -0.13294731  0.04071715 -0.35661343
 -0.10559443 -0.21920484  0.38865426  0.61690795  0.14190854  0.92985463
  0.38688555 -0.5396631  -0.27903783 -0.25335598 -0.05868918 -0.3991832
 -0.51297975  0.11210173 -0.8140017  -0.18571657 -0.6295385  -0.3508472
  0.19671534  0.48802882  0.41331294  0.46507463  0.55321956  0.26814324
 -0.27244323 -0.75347644 -0.17870347 -0.34811586  0.6973272  -0.09584054
  0.34359345 -0.16714601  0.2391204   0.603859   -0.26545346 -0.5815423
  0.84040296  0.8684258  -0.48318216 -1.0251215   0.67045975 -0.28564698
  1.3181759   0.238932   -0.31026012 -0.5631942   0.33890858 -0.81634647
 -0.13116628 -0.5689911   0.21023095  0.13752754 -0.23049593 -0.396373
 -0.17098397  0.52437806 -0.11991276 -0.17967495  0.35036764 -0.02136617
 -0.3808628  -0.82136095 -0.5103798   0.19593334  0.0920986  -0.24427408
  0.40123457 -0.2820268  -0.28051072 -0.5928526  -0.01019426 -0.87303203
  0.1217309  -0.11852136 -0.26868933 -0.1444161  -0.7486063  -0.19866931
 -0.37238127  0.06388521 -0.35379276 -0.13490604  0.08158439 -0.09031488
 -0.16991304 -0.95759964 -0.16343851  0.09244142  0.55896074  0.11754514
  0.8666826  -0.85200155 -0.5000392  -0.10773891 -0.96491873  0.2043062
 -0.20884645 -0.53790313 -0.472793   -0.12941085  0.33376974 -0.18331663
  0.14290327  0.2669529  -0.04817933  0.1961905   0.01975059 -0.34626755
 -0.1688545  -0.62168086  0.19479963  0.5991113  -0.23918003  0.5081801
 -0.14233941  0.12468854  0.82995105  0.02796978  0.4268009  -0.3133791
 -0.7182112   0.04522754  0.13505438  0.17482986 -0.5277166   0.04686747
 -0.5022211  -0.08384314 -0.06026209  0.534554    0.7027847   1.0531831
 -0.06110565 -0.09441035 -0.29154858  0.04263121 -1.2222071   0.69154465
  0.15146658  0.63643485 -0.15173855  0.31316218  0.05443235 -0.6697509
 -0.83844763  0.12627576  0.3469649   0.7906073  -0.12472709 -0.13051951
 -0.37473735  0.05644423  0.11256064 -0.73327583  0.28946555 -0.60252875
  0.07282681  0.38723153 -0.03525896 -0.18535335 -0.7231747   0.54515857
  0.56857216 -0.03015586 -0.38713259 -0.15400992 -0.33501333  0.48441866
 -0.14256254  0.48276302 -0.37047175  0.5438715  -0.70645    -0.21443231
  0.35043532  0.2619092  -0.08084039  0.51055723 -0.0469946   0.49831977
 -0.69439036  0.43322366 -0.0129585  -0.2776995  -0.01880828 -0.22086047
 -0.23338751  0.39741778 -0.16438815 -0.08405622  0.06063091 -0.43516147
  0.8238099  -0.5553637  -0.2939282   0.19725098 -0.15464969  0.77290225
 -0.10740466 -0.9243945   0.07814645 -0.09383044  0.5910903   0.34824884
  0.08855297  0.29560855  0.11905991 -0.0518215   0.11963987 -0.52313364
  0.28260273  0.7639619  -0.64613444  0.47562882  0.17544304 -0.3824722]
```

One of the main goals of this research to handle OOV words effectively which may affect the performance of the model if it is not taken care of. A unique token '*UNK*' was added to the vocabulary *V*, with an index 0. The vectors for unknown or new word will be assigned as the average of all the vectors in the embedding matrix $\mathcal{M}$. Further, during the context generation or windows approach, the End of Sequence (*EOS*) token was added in the vocabulary with index 1. The vocabulary of 3676 unique syllables was generated. Part of the vocabulary is shown below.

```
'འདུད': 2692, 'ཅོག་': 2693, 'སྙེ་': 3576, 'གཏད': 3577, 'ཆགསན་': 3578, 'ཏུ་':
2694, 'ཏུ': 2317, 'འོངསམ': 2695, 'ཤོབ': 2696, 'རྗོབ་': 3579, 'སེན་': 3580,
'སྐྱབས་': 3581, 'འགྲེདཔ': 3582, 'སོམས་': 3583'བསྐྱི་': 1675, 'གཏའམ་': 1794, 'རྩེཕ
': 3584, 'ཤབ་': 2697, 'དཀྲུམ་': 3585, 'ཊའི': 3586, 'བྲུནམ': 3587, 'མནྡལ་':
1096, 'བྲིངས་': 3588, 'ཁུ': 3589, 'རས': 3590, 'སླབནི་': 3591, 'བགའལྷ་':
3592, 'མནྡལ': 1589, 'ཚ': 3593, 'དྲགཔོ་': 3594, 'ཙ': 3595, 'གསུངས': 3596,
'བེལཀེ་': 3597, 'བླལ་': 3598, 'བེག': 3599, 'ཀཱ་': 3600, 'དྷར་': 3601, 'འདེད་':
1210, 'བདེ': 2698, 'ཆེད་': 2699, 'ཡར་': 3602, 'སྦྱོདཔ་': 1915, 'བསྐྱིཕ་':
3603, 'vt_ne': 2318, 'རྗེངམ': 3604, 'ཕུས་': 3605, 'བརྡུངསམ་': 3606, 'བཞགསཔ་
': 3607, 'ཅོག': 3608, 'སྙེ': 2700, 'ལྷཕ': 3609, 'འགྲོནི་': 3610, 'དཔ':
3611, 'ncc': 2701, 'གནི་': 3612, 'ངོམས': 3613, 'ཚོཤོག': 3614, '།?':
3615, 'eep': 2319, 'ཟེརསུའི་': 3616, 'ཟེརསུའི': 3617, 'སྦོབུ་': 3618, 'པཔོ་':
3619, 'ona': 2702, 'གགཅིག': 3620, 'ij': 3621, 'ཨོངའུང་': 3622, 'ཏཛ་':
2703, 'ཨོངཞུང་': 3623, 'ཆེཔ་': 3624, 'གབཟུང་': 3625, 'མཡར་': 3626, 'སློས':
3627, 'cag': 3628, 'འཛོཇུན་': 2704, 'neg': 3629, 'ཞིནམམ་': 3630, 'གགཏང
': 3631, 'དཔའོ': 3632, 'བཀའལྷ་': 3633, 'སྨེའབད་': 3634, 'དེམ': 3635, 'འཇིདི་':
3636, 'ཕྱིས་': 2705, 'phr': 3637, 'ཨོཙོང་': 3638, 'mp': 2706, 'sm':
3639, 'ཨའ་': 3640, 'འབྲིན་': 3641, 'ངལྷ་': 2707, 'pfv': 3642, 'ཏེའདི་':
3643, 'cf': 3644, 'nm': 3645, 'avt': 3646, 'icq': 3647, 'སྤོབཏོན་':
3648, 'Kiba': 3649, 'འདི་': 2708, 'ཏེརྟེ་': 3650, '_cac': 3651, 'ཚོགཤོག':
3652, 'འ': 3653, 'བསྟོཏོན་': 3654, 'བསྐྱེལ': 3655, 'ཤུདའི་': 3656, 'sb':
3657, 'བགཏང་': 2709, 'གོསྒོ་': 2320, 'ང': 3658, 'eh': 3659, 'དེའདི་':
3660, 'et': 3661, 'གབཏང་': 3662, 'or': 3663, 'vt': 2321, 'བསྐྱིས་':
3664, 'དེན': 3665, 'འདུགནི་': 3666, 'དོགཔ་': 3667, 'འཛོལཕ་': 3668, 'སྣུངམ་':
3669, 'འབདནི་': 3670, 'ལྷར་': 3671, 'བསྐྱལ': 3672, 'སྦོདནི་': 3673, 'དབངམོ':
3674, 'ཕངས': 3675, 'UNK': 0, 'EOS': 1
```

The integer values in the vocabulary represent the index of the corresponding syllables.

In addition, tag vocabulary was constructed. In this thesis, vocabulary *T* of size 3 is constructed from the dataset. The tags used in marking the position of the word are discussed in table 3 where the index represents the index of the tag in the tag vocabulary.

Table 3 Tag Vocabulary and description

Tag	Description	Index	Example
beg	the single syllable word	0	‘ང་’ (I) is a single syllable word which will be tagged as ‘beg’
	or first syllable of a word		‘རྒྱལ་ཁབ་’ (country) is a two-syllable word where ‘རྒྱལ་’ will be tagged as ‘beg’ as it is the first syllable of a word ‘རྒྱལ་ཁབ་’
end	the last syllable of a word	1	‘རྒྱལ་ཁབ་’ (country) is a two-syllable word where ‘ཁབ་’ will be tagged as ‘end’ as it is the last syllable of the word ‘རྒྱལ་ཁབ་’.
mid	Syllables in between the beginning and the last syllable of a word	2	‘སློབ་སྦྱོང་པ་’ (Trainee) is a three-syllable word where the middle syllable ‘སྦྱོང་’ is tagged ‘mid’

Pre-processing

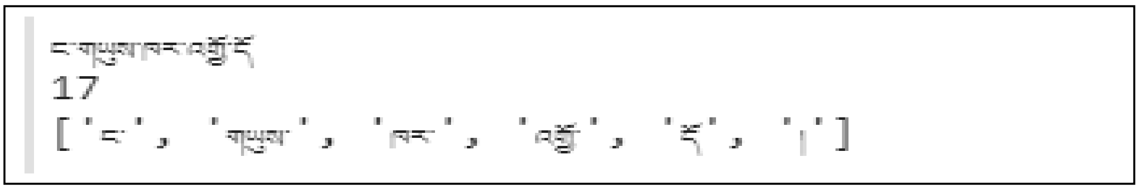
Text pre-processing is one of the fundamental steps in machine learning to create a meaningful and quality data, on which the model can work. In this stage, pre-processing was carried out in three stages: 1) Syllable segmentation, 2) Context Generation and 3) mapping syllables to its indices in Vocabulary.

Syllable Segmentation

The first and foremost step is to split the sentences or documents to its syllabic form, the smallest token of Dzongkha word. For example, the sentence ‘ང་གསུམ་ཁར་འགྱོ་དོ།’ can be broken down as ‘ང་’, ‘གསུམ་’, ‘ཁར་’, ‘འགྱོ་’, ‘དོ’, and ‘།’. The following code snippet was used to split a sentence into its syllabic form.

```
import re

# define the document
# text = 'དེ་དང་གཅིག་ཁར་  རྫོང་ཁ་སླབ་སྦྱང་གི་དཔེ་དེབ་ཚུ་འབྲི་ནི་དང་པར་སྐྲུན་འབད་ནིའི་དོན་ལུ་  ཤེས་རིག་ལྷན་ཁག་འོག་ལུ་རྫོང་ཁ་གོང་འཕེལ་སྡེ་ཚན་ཅིག་
ཡང་གཞི་བཙུགས་གནང་སྟེ་  སློབ་གྲྭ་ཁག་ལས་པར་རྫོང་ཁ་འདི་སྐད་ཡིག་དང་རྩོམ་རིག་གི་གྲས་ཁར་སློབ་སྟོན་གྱི་ཆོས་ཚན་ཅིག་སྦེ་འགོ་བཙུགས་གནང་ནུག།'
# text = 'དཔལ་འབྱོར་རིག་པའི་ཆོས་ཚན་འདི་སློབ་སྦྱོང་འབདཔ་ད་ལྷ་ཁག་ཡོད།'
text ='ང་གསུམ་ཁར་འགྱོ་དོ'

pre=text.find('།')!=-1
cou=text.count('།')

# tokenize the document
text= re.sub('(།|[a-z0-9]|' ')','',text)
  result = text.split('་')
  for i in range(len(result)):
    if(i<(len(result)-1)):
      result[i]=result[i].strip()
      result[i]=result[i]+'\u0f0c'
  if pre:
    for i in range(cou):
      result.append('།')

print(result)
```

The above code snippet produced the output as follows:



```
ང་གསུམ་ཁར་འགྱོ་དོ
17
['ང་', 'གསུམ་', 'ཁར་', 'འགྱོ་', 'དོ', '།']
```

Context generation

A syllable is considered as the smallest token of a word in Dzongkha. A word can be either a syllable or a combination of syllables. The identification of syllable position in a word is considered as the most important in segmentation formulated as a syllable tagging problem. Thus, it is understandable that the tag of a syllable depends on its neighboring syllable (Zheng et al., 2013).

For this purpose, the right and left context for a target syllable were considered for a given sentence. The left context for the first syllable and right context for the last syllable of a given sentence is marked as ‘End of Sequence’ (EOS). The EOS token was added in the vocabulary with an index as 1 which would help us to identify the beginning and end of a sentence. Further, the addition of EOS token eased to generate the context by making the even length of a sentence. In our thesis, context size ‘*N*’, where *N* = 0, 1, 2 and 3 were considered to determine the effective context size in syllable tagging problem for word segmentation. A context size 0 can be understood as without considering the context, while the other means that *N* right and *N* left context would be considered for the target syllable.

For instance, consider a sentence ‘ང་གཡུས་ཁར་འགྱོ་དོ།’ which can be broken down into its syllabic form as ‘ང་’, ‘གཡུས་’, ‘ཁར་’, ‘འགྱོ་’, ‘དོ’, and ‘།’ in the pre-processing stage and generate its context for every syllable with context size *N* = 1, 2, and 3. EOS token was padded in the beginning and end of the sentence, depending upon the context size. *N* EOS token was padded as a left context for the first syllable and right context for the last syllable of the sentence. The context for each of the target syllable in the given sentence using various context size is shown in table 4, 5 and 6.

Table 4 Context of Each syllable with N=1

L_1	**Target**	**R_1**
EOS	**ང་**	གཡུས་
ང་	**གཡུས་**	ཁར་
གཡུས་	**ཁར་**	འགྱོ་
ཁར་	**འགྱོ་**	དོ
འགྱོ་	**དོ**	।
དོ’	**།**	EOS

Table 5 Context of each syllable with N = 2

L2	**L1**	**Target**	**L1**	**L2**
EOS	EOS	**ང་**	གཤུས་	ཁར་
EOS	ང་	**གཤུས་**	ཁར་	འགྲོ་
ང་	གཤུས་	**ཁར་**	འགྲོ་	དོ’
གཤུས་	ཁར་	**འགྲོ་**	དོ	।
ཁར་	འགྲོ་	**དོ**	।	EOS
འགྲོ་	དོ’	**།**	EOS	EOS

Table 6 Context of each syllable with N = 3

L3	**L2**	**L1**	**Target**	**R1**	**R2**	**R3**
EOS	EOS	EOS	**ང་**	གཤུས་	ཁར་	འགྲོ་
EOS	EOS	ང་	**གཤུས་**	ཁར་	འགྲོ་	དོ’
EOS	ང་	གཤུས་	**ཁར་**	འགྲོ་	དོ’	।
ང་	གཤུས་	ཁར་	**འགྲོ་**	དོ	।	EOS
གཤུས་	ཁར་	འགྲོ་	**དོ**	।	EOS	EOS
ཁར་	འགྲོ་	དོ	**།**	EOS	EOS	EOS

In the above tables, L_1, L_2, and L_3 represent the syllables in the left which is an immediate neighbour of the target syllable in the given while R_1, R_2, and R_3 are the right neighbours. Mathematically, consider ‘*S*’ as the list of syllables in a sentence, L_i for left context and R_i for right context for the target syllable ‘S_j’ at j^{th} index of the sentence, where i = 1, 2,..N, which is the context size to be considered. Thus, L_i and R_i can be denoted as $L_i = S_{(j-i)}$ and $R_i = S_{(j+i)}$.

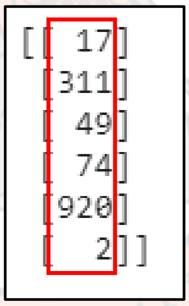
Note: <u>Context generation step is not applicable for syllable tagger using Recurrent Neural Network</u>

Mapping syllables to its indices in the vocabulary

The syllable is still in the text form. The syllables were converted to a number form. The vocabulary *V* constructed in the previous stage ‘word embedding’ was used to get integer equivalent of a syllable, which is the index of a syllable in the |*V*| sized Vocabulary V. Then the syllables are fed into the model as indices. In our vocabulary *V*, the syllables ‘ང་’, ‘གཡུས་’, ‘ཁར་’, ‘འགྲོ་’, ‘ནོ’, and ‘།’ were stored as shown below.

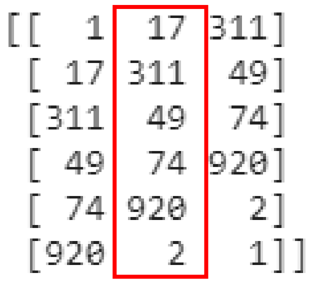
```
'ང་': 15, 'གཡུས་': 311, 'ཁར་': 49, 'འགྲོ་': 74, 'ནོ': 920, '།': 2,
'UNK': 0, 'EOS': 1
```

The context generated in the previous section was now mapped to its indices as shown in the following figures. The red boxes denote the target syllables.



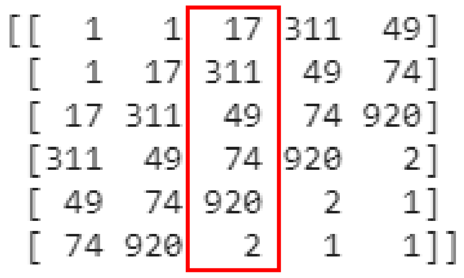
```
[[ 17]
 [311]
 [ 49]
 [ 74]
 [920]
 [  2]]
```

Figure 16 Input sentence mapped to its indices when *N* = 0



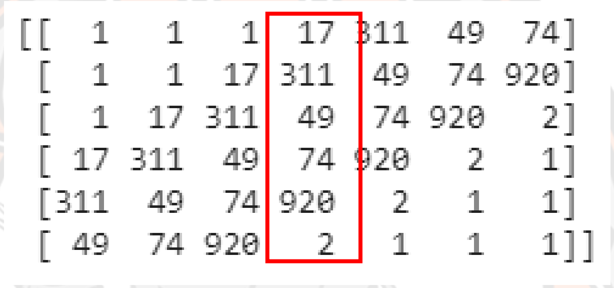
```
[[  1  17 311]
 [ 17 311  49]
 [311  49  74]
 [ 49  74 920]
 [ 74 920   2]
 [920   2   1]]
```

Figure 17 Input sentence mapped to its indices when *N* = 1



```
[[  1   1  17 311  49]
 [  1  17 311  49  74]
 [ 17 311  49  74 920]
 [311  49  74 920   2]
 [ 49  74 920   2   1]
 [ 74 920   2   1   1]]
```

Figure 18 Input sentence mapped to its indices when *N* = 2



```
[[  1   1   1  17 311  49  74]
 [  1   1  17 311  49  74 920]
 [  1  17 311  49  74 920   2]
 [ 17 311  49  74 920   2   1]
 [311  49  74 920   2   1   1]
 [ 49  74 920   2   1   1   1]]
```

Figure 19 Input sentence mapped to its indices when *N* = 3.

Syllable Tagger

The Dzongkha word segmentation was formulated as a syllable tagging problem where the labels were assigned for each of the syllables in a sentence. In my thesis, two deep learning algorithms were proposed. The first model is based on Deep Neural Network (DNN) and the later use Bidirectional Long Short-Term Memory Recurrent Neural Network ((Bi-LSTM RNN). The first model is discussed in the first section and the second section of this chapter discusses the later model.

Deep Neural Network

The neural network architecture for syllable tagging is shown in figure 20. The network is a variant of neural architecture which was proposed by Collobert et al. (2011) for POS tagging, Named Entity Recognition (NER), chunking and semantic role labeling. The model learns feature on its own, without having to depend much on the hand-crafted features.

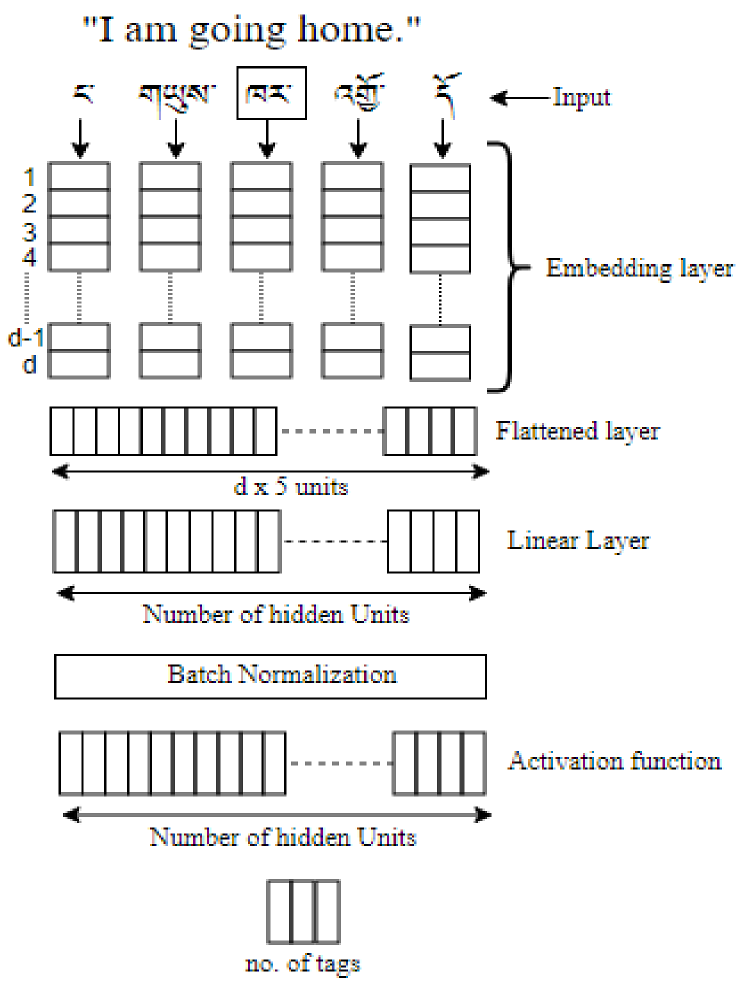


Figure 20 Neural Network Architecture

The syllables were fed into the network as indices as seen in mapping sections of this chapter. The first layer extracts the *d* dimensional features for each syllable. It is called a lookup operation where the indices of the syllables are mapped to its vector representation in the embedding matrix $\mathcal{M}$. The lookup layer would output *n x d* dimensional features, where *n* is the no of input. The second layer flattens the *n x d* dimensional feature to a 1-dimensional feature which was fed to the next linear layer. The linear layer extracts the features from the windows of syllables. And finally outputs the tag for each syllable in the given sentences.

A neural network can be considered as a function $f_{\theta}^{l}(.)$ with parameters θ. Any feed-forward neural with L-layers can be seen as a composition of functions $f_{\theta}^{l}(.)$ defined for each layer *l:*

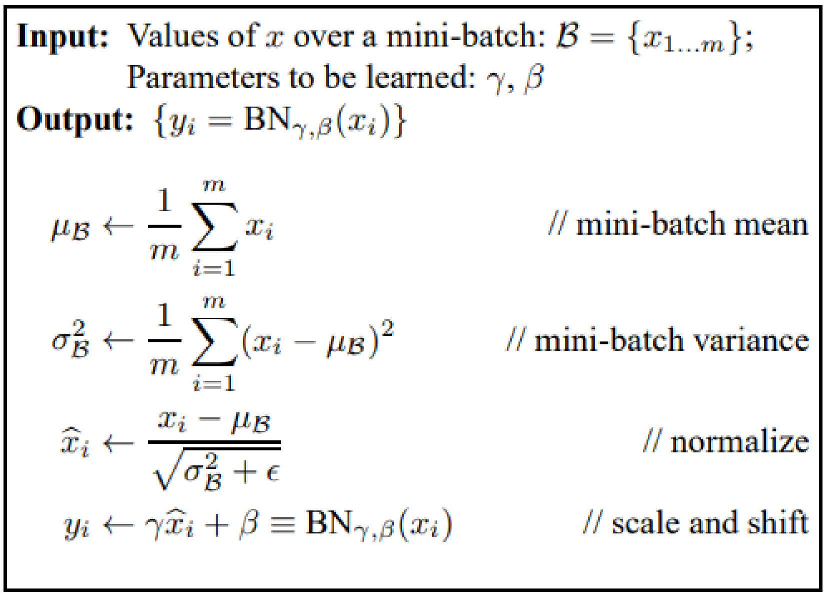
$$f_{\theta}^{l}(.) = f_{\theta}^{L}(f_{\theta}^{L-1}(\ldots\ldots f_{\theta}^{1}(.)\ldots)) \qquad (3)$$

The output of the first layer is fed to two linear standard linear layers that successively perform affine transformation over f_{θ}^{1}, interleaved with non-linearity function g(.). Given a set of tags T for the task of interest, the network outputs a vector size of |T| for each syllable at position i, interpreted as the score for each tag in T and each syllable S_i in the sentence.

$$f_{\theta}^{l}(S_i) = f_{\theta}^{3}\left(g\left(f_{\theta}^{2}\left(f_{\theta}^{1}(S_i)\right)\right)\right)$$
$$= W^3 g(W^2 f_{\theta}^{1}(S_i) + b^2) + b^3 \qquad (4)$$

where the matrices $W^2 \in \mathbb{R}^{H \times (nd)}$, $b^2 \in \mathbb{R}^{H}$, $W^3 \in \mathbb{R}^{H \times /T/}$, and $b^3 \in \mathbb{R}^{/T/}$ are the parameters to be trained where H is the hyperparameters, usually called the number of hidden layers.

Backpropagation will be used to train the network. During the backpropagation, the parameters of the layer change which leads to change the input of the next layer (Ioffe & Szegedy, 2015), thus increasing the complexity of the training phase because the learning system has to be adapted with new inputs. This phenomenon is called as ‘Covariate shift’. Batch Normalization was added in the tagging neural architecture to accelerate speed the training of the model and as a regularizer. The batch normalization algorithm is shown in figure 21.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 21 Batch Normalization algorithm

Source: Ioffe and Szegedy, 2015

When we have unbalanced class distribution, the accuracy metric may not be the right metric to evaluate the performance of the model because there are chances that the accuracy may be dominated by the class that is higher in number. Therefore, the F1 score metric will be used along with the accuracy metric for evaluating the performance of the model. Consider the confusion matrix as given in table 7 to understand the Recall, Precision and F1 score.

Table 7 Confusion Matrix

		PREDICTED	
		Negative	**Positive**
ACTUAL	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Recall: Ratio of correctly predicted positive observations to the total number of observations in the actual class as shown in equation 5.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \qquad (5)$$

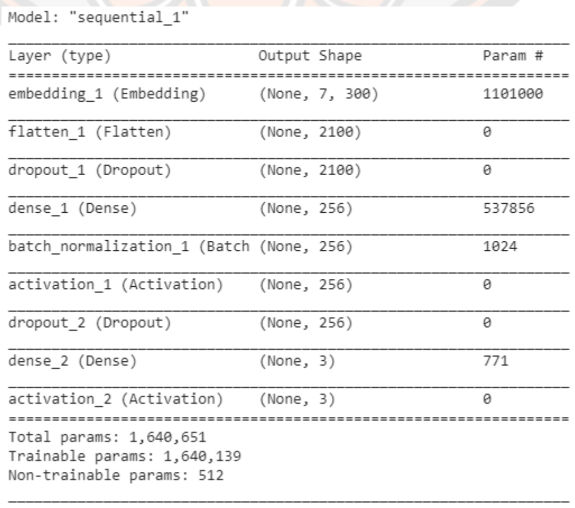
Precision: Ratio of correctly predicted positive observations to the total number of positive predicted observations as shown in equation 6.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}} \qquad (6)$$

Finally, the F1 score is computed as:

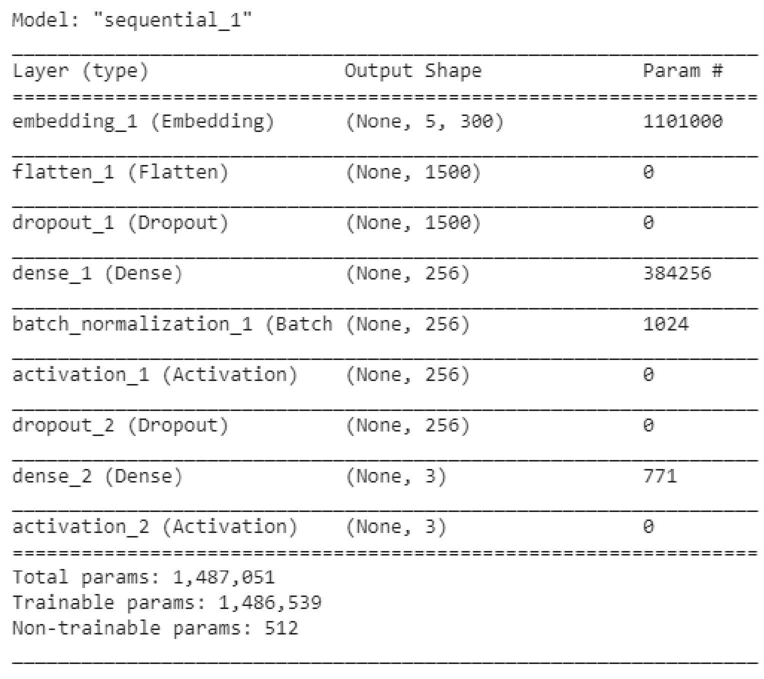
$$\text{F1 score} = \frac{\text{2 x (Precision x Recall)}}{\text{Precision + Recall}} \qquad (7)$$

As discussed in this section, two sets of experiments were designed. In each set, four experiments were conducted with models of varied context sizes ranging from 0 to 3. The first experiment set uses pretrained syllable embedding, while the other learns embedding during the network training, which is a shut shell, it does not use pretrained syllable embedding. The main idea of this experimental design is to understand the impact of the pretrained embedding matrix. It was reported by C. Wang and Xu (2017) that pretrained embedding increases the performance of the model. The model summary for all the experimental sets is shown in the figures below.



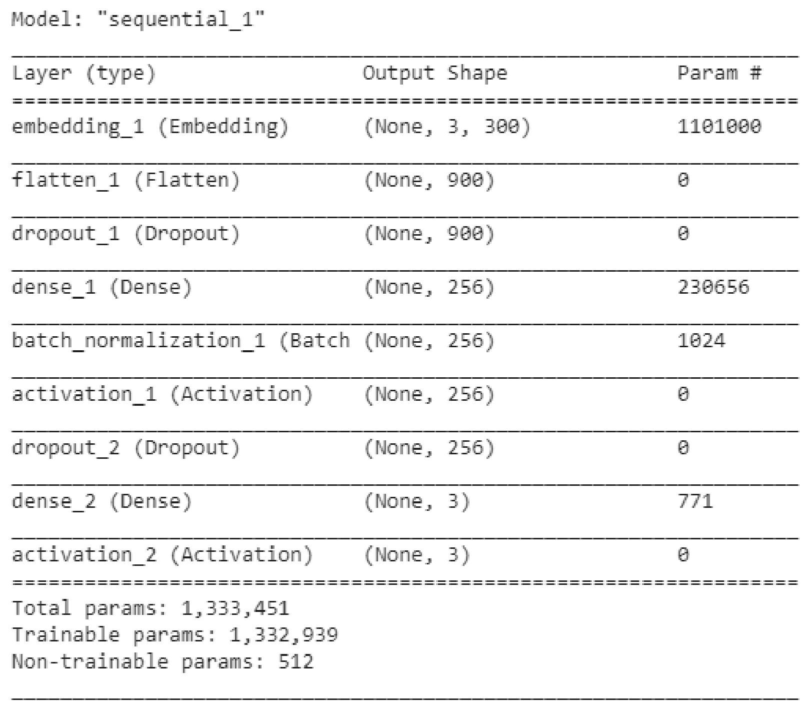
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 7, 300)            1101000
_________________________________________________________________
flatten_1 (Flatten)          (None, 2100)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2100)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               537856
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,640,651
Trainable params: 1,640,139
Non-trainable params: 512
_________________________________________________________________
```

Figure 22 Model summary for *N*=3 without pretrained embedding



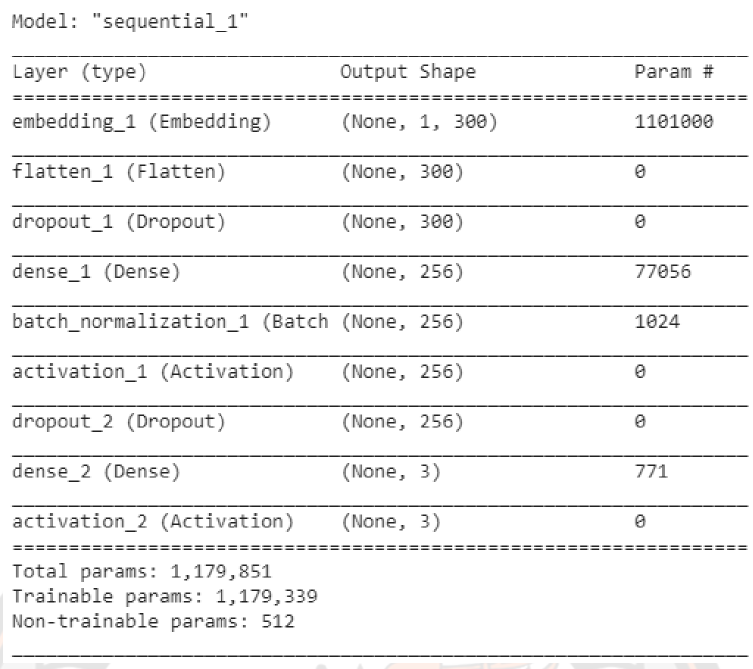
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 5, 300)            1101000
_________________________________________________________________
flatten_1 (Flatten)          (None, 1500)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1500)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               384256
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,487,051
Trainable params: 1,486,539
Non-trainable params: 512
_________________________________________________________________
```

Figure 23 **Model summary for *N*=2 without pretrained embedding**



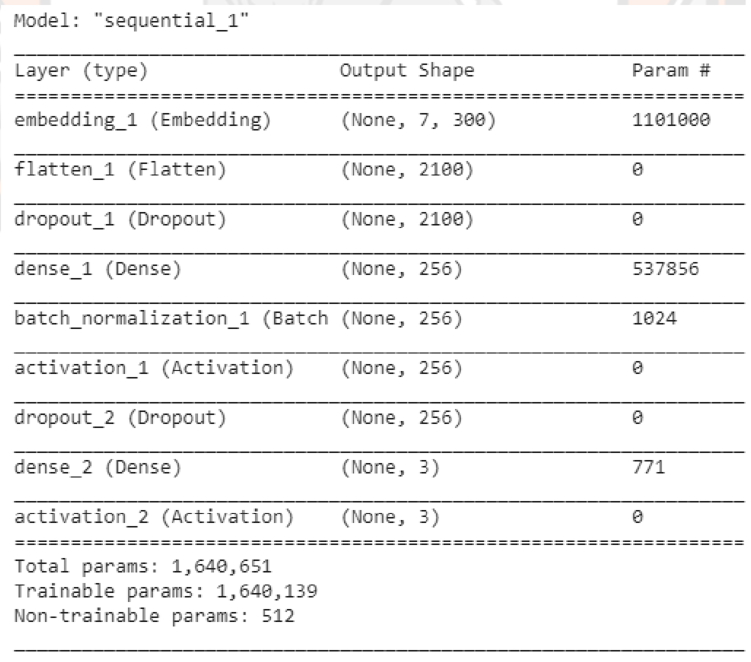
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 3, 300)            1101000
_________________________________________________________________
flatten_1 (Flatten)          (None, 900)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 900)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               230656
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,333,451
Trainable params: 1,332,939
Non-trainable params: 512
_________________________________________________________________
```

Figure 24 **Model summary for *N*=1 without pretrained embedding**



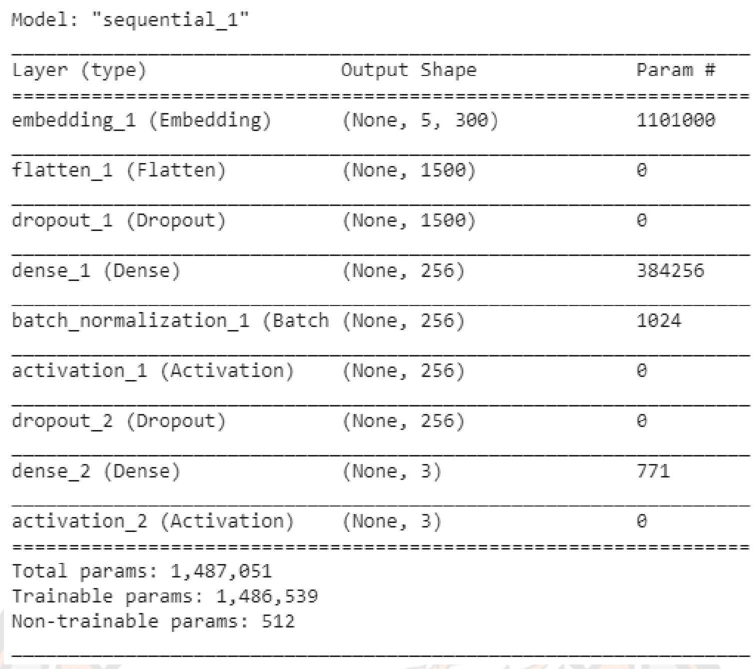
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1, 300)            1101000
_________________________________________________________________
flatten_1 (Flatten)          (None, 300)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               77056
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,179,851
Trainable params: 1,179,339
Non-trainable params: 512
_________________________________________________________________
```

Figure 25 Model summary for *N*=0 without pretrained embedding



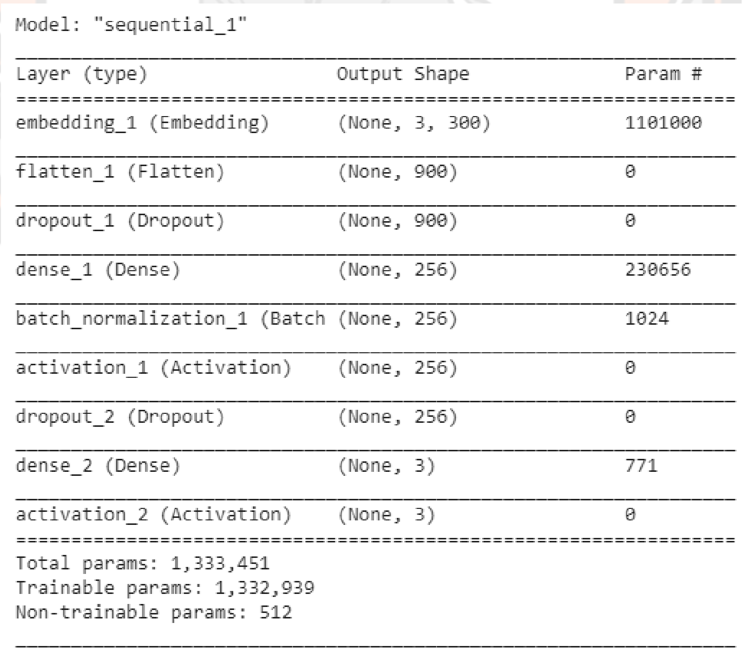
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 7, 300)            1101000
_________________________________________________________________
flatten_1 (Flatten)          (None, 2100)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2100)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               537856
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,640,651
Trainable params: 1,640,139
Non-trainable params: 512
_________________________________________________________________
```

Figure 26 Model summary for *N*=3 with pretrained embedding



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 5, 300)            1101000   
_________________________________________________________________
flatten_1 (Flatten)          (None, 1500)              0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 1500)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               384256    
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771       
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0         
=================================================================
Total params: 1,487,051
Trainable params: 1,486,539
Non-trainable params: 512
_________________________________________________________________
```

Figure 27 Model summary for N=2 with pretrained embedding



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 3, 300)            1101000   
_________________________________________________________________
flatten_1 (Flatten)          (None, 900)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 900)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               230656    
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771       
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0         
=================================================================
Total params: 1,333,451
Trainable params: 1,332,939
Non-trainable params: 512
_________________________________________________________________
```

Figure 28 Model summary for *N=1* with pretrained embedding

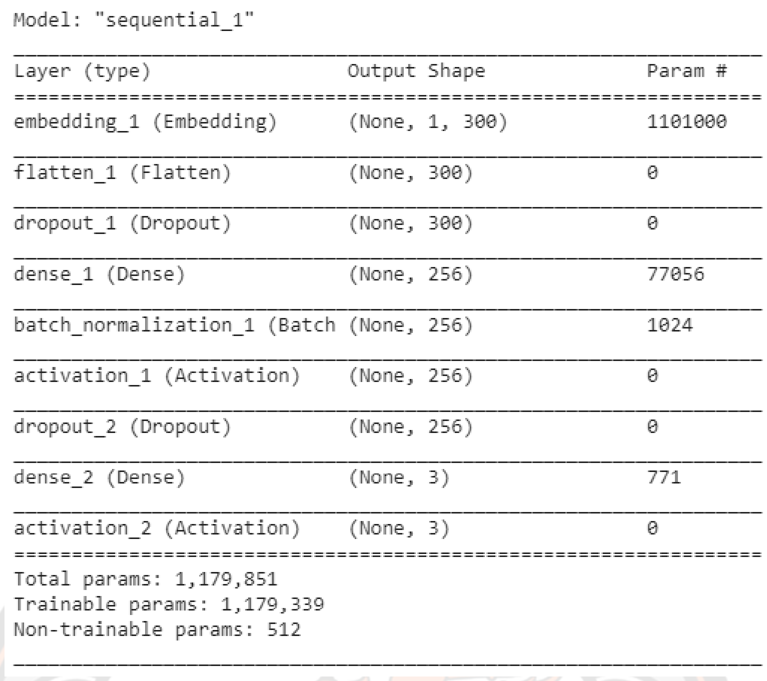

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 1, 300)            1101000   
_________________________________________________________________
flatten_1 (Flatten)          (None, 300)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               77056     
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771       
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0         
=================================================================
Total params: 1,179,851
Trainable params: 1,179,339
Non-trainable params: 512
_________________________________________________________________
```

Figure 29 Model summary for *N=0* with pretrained embedding

The model hyperparameters are the most important properties the govern the behavior of the network. It is defined as properties that control the entire training process of the model. The choice of right hyperparameters contributes to the efficiency of the model (Domhan, Springenberg, & Hutter, 2015). The hyperparameters used in these experiments are presented in table 8.

Table 8 Model Hyperparameters for DNN models

Hyperparameters	**Value**
Hidden layer	256
Dropout rate	0.1
Initial Learning rate	0.002
Maximum Learning rate	0.02
Embedding dimension	300
Embedding Window size	5

Amongst the hyperparameters presented in table 8, the learning rate is the most important one to be critically considered to optimize the model. The usage of higher learning rate diverges from the objective function while the smaller learning rate slows the learning process of the model, thereby increasing the training time (Zeiler, 2012). For this consideration, the choice of the right learning rate is important. However, it is challenging because there is no thumb rule for choosing the right learning rate. In our work, the cyclical learning rate (CLR) schedule (Smith, 2017), also known as the Triangular learning rate schedule was used along with the stochastic gradient descent (SGD) optimizer (Amari, 1993).

In the CLR schedule, the minimum and maximum learning rates are set. The learning rate during the network training varies cyclically between these two bounds. In our experiment, the two bounds were set to 0.02 and 0.002 (presented table 8) with a step size of 250 epochs. The CLR schedule is illustrated in figure 30.

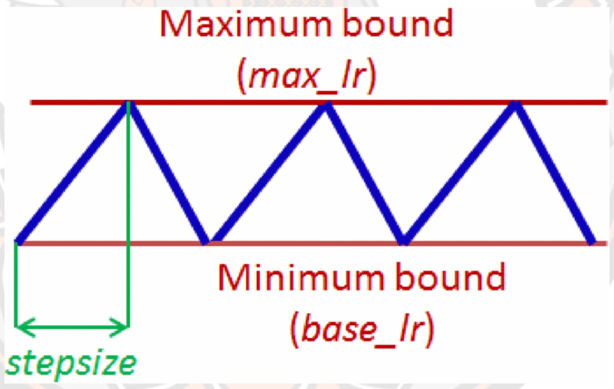


Figure 30 CLR learning rate schedule

Source: Smith, 2017

Bi-LSTM RNN

The Bidirectional Long Short-Term Memory (Bi-LSTM RNN) architecture for tagging a syllable in a word is presented in figure 31. Given an input sequence $S = (S_1, S_2, S_3, S_4, \ldots\ldots, S_n)$, the network computes hidden state $h = (h_1, h_2, h_3, \ldots\ldots, h_n)$ and outputs $T = (T_1, T_2, T_3, \ldots\ldots, T_n)$, where T_i represents the tag or position of an i^{th} syllable S_i in a word.

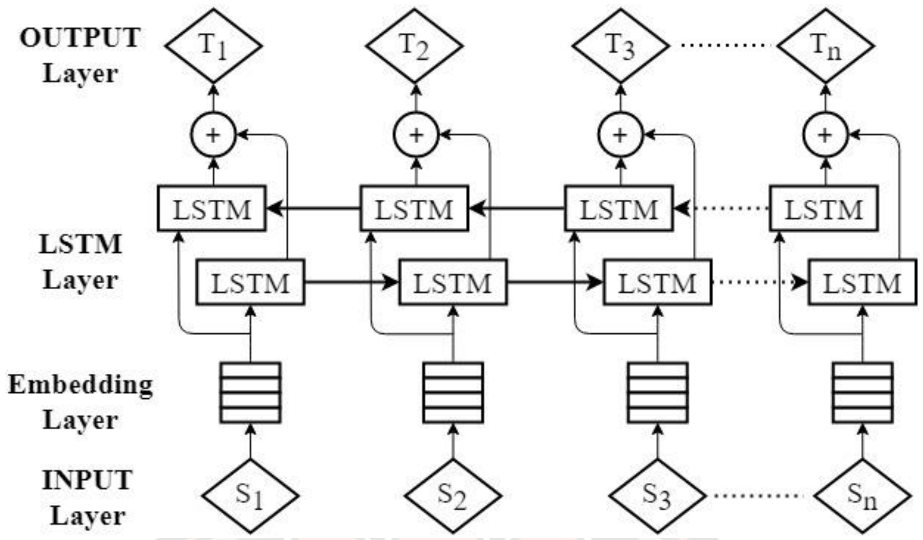


Figure 31 Bi-LSTM RNN model architecture for syllable tagging

The Bi-LSTM (Shabanian, Arpit, Trischler, & Bengio, 2017) network has two LSTM layer. The first LSTM layer computes forward hidden layer vector $\vec{h}$ from *i* =1 until *n*, while the second layer computes backward hidden vector $\overleftarrow{h}$ from *i* = *n* to 1. The iterative process of updating the output layer *i* can be expressed as follows:

$$\vec{h}_i = \mathrm{H}\left(W_{\overrightarrow{sh}} S_i + W_{\vec{h}\vec{h}} \overrightarrow{h}_{i-1} + b_{\vec{h}}\right),$$

$$\overleftarrow{h}_i = H(W_{\overleftarrow{sh}} S_n + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{i-1} + b_{\overleftarrow{h}}),$$

$$T_i = (W_{\vec{h}y}\vec{h}_i + W_{\overleftarrow{h}y}\overleftarrow{h}_i + b_y).$$

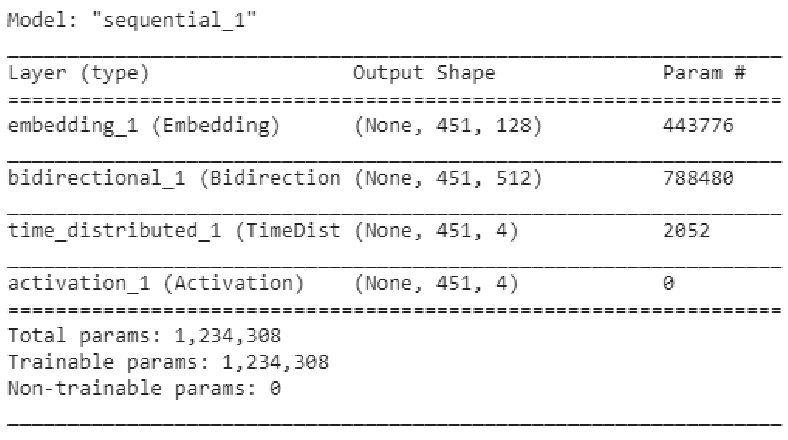
W represents weight matrices between layers, $b_{\vec{h}}$, $b_{\overleftarrow{h}}$ and by are respectively the bias vectors of the hidden of the forward LSTM, backward LSTM, and output layers, H is the activation function of the output layer.

In our experiment, two experimental sets were designed based on the usage of dropouts. The first set uses a dropout rate of 0.2, while the other is without the usage of dropout. Further, each of these sets was divided into two subsets based on embedding dimensions. Embedding dimensions (Emb_Dim) of 128 and 256 were considered in our experiment. In each subset, three models with different learning rates (LR) were developed. Each model in the subset was developed with two different hidden sizes (size) of 256 and 512 neurons. In total, 24 models were built considering the summary of configurations presented in table 9. This enables to find the most optimized model for the task of Dzongkha word segmentation.

Table 9 Experimental configurations for Bi-LSTM architecture

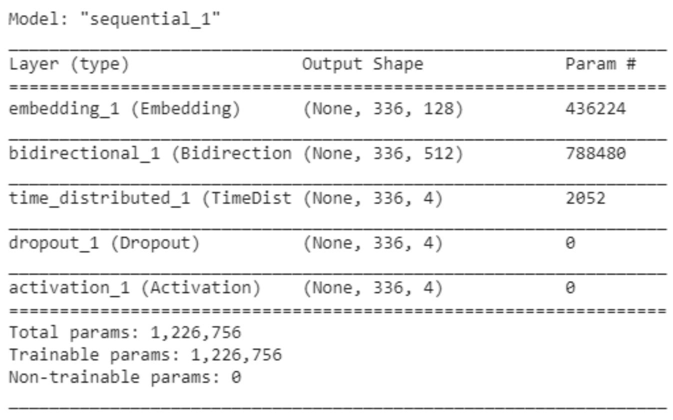
Size	**Dropout**	Yes		No	
	Emb_dim	128	256	128	256
256	**LR**	0.001	0.001	0.001	0.001
		0.010	0.010	0.010	0.010
		0.020	0.020	0.020	0.020
512	**LR**	0.001	0.0010	0.001	0.001
		0.010	0.010	0.010	0.010
		0.020	0.020	0.020	0.020

The figures below illustrate the model summaries of the models build using the configurations presented in table 9.



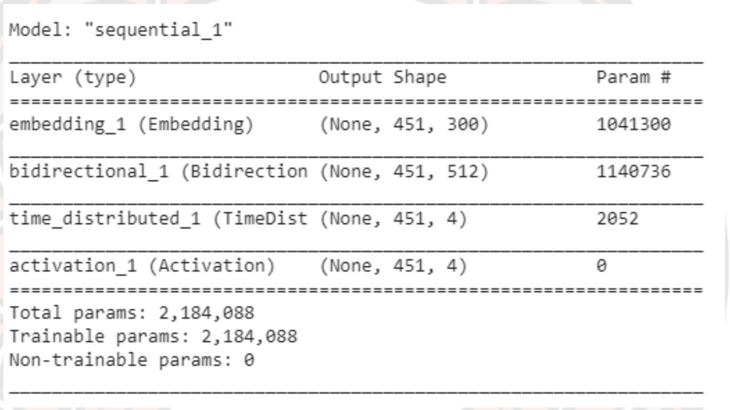
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 451, 128)          443776
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 512)          788480
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            2052
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0
=================================================================
Total params: 1,234,308
Trainable params: 1,234,308
Non-trainable params: 0
_________________________________________________________________
```

Figure 32 Model summary for 128 Emb_Dim and 256 Neurons without dropout



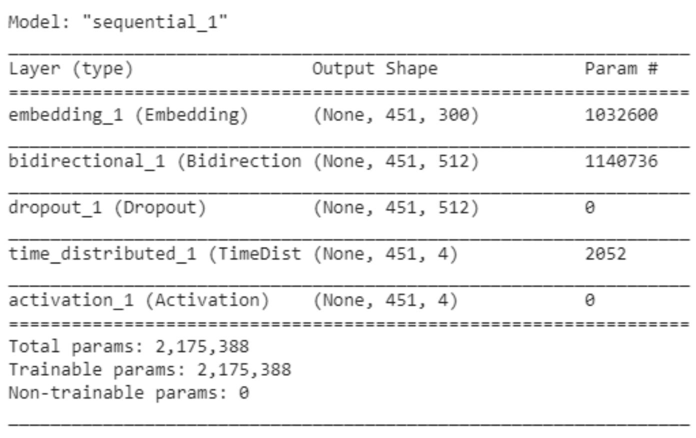
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 336, 128)          436224
_________________________________________________________________
bidirectional_1 (Bidirection (None, 336, 512)          788480
_________________________________________________________________
time_distributed_1 (TimeDist (None, 336, 4)            2052
_________________________________________________________________
dropout_1 (Dropout)          (None, 336, 4)            0
_________________________________________________________________
activation_1 (Activation)    (None, 336, 4)            0
=================================================================
Total params: 1,226,756
Trainable params: 1,226,756
Non-trainable params: 0
_________________________________________________________________
```

Figure 33 Model summary for 128 Emb_Dim and 256 Neurons with dropout



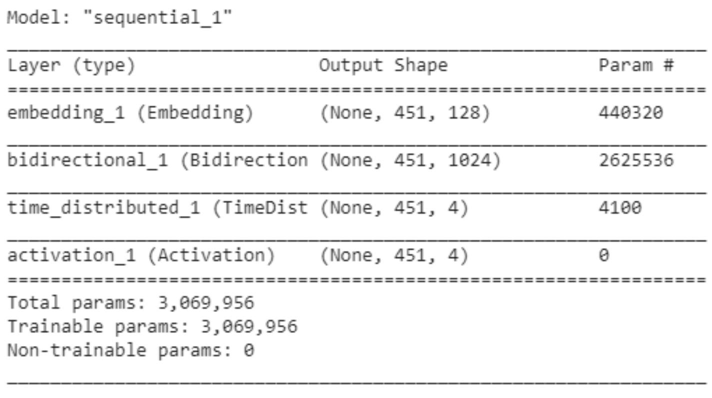
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 451, 300)          1041300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 512)          1140736
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            2052
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0
=================================================================
Total params: 2,184,088
Trainable params: 2,184,088
Non-trainable params: 0
_________________________________________________________________
```

Figure 34 Model summary for 300 Emb_Dim and 256 Neurons without dropout



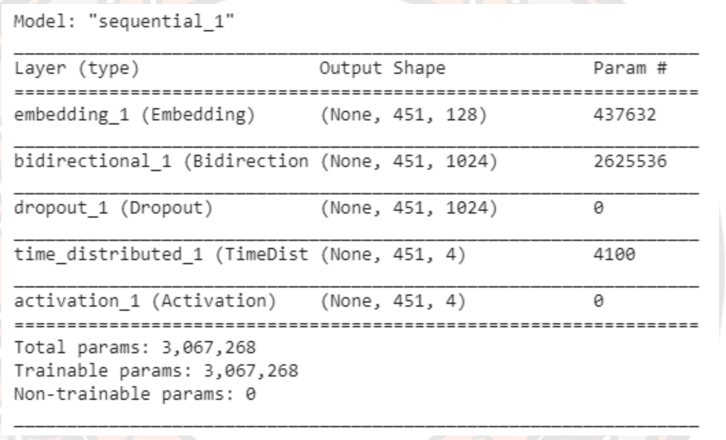
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 451, 300)          1032600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 512)          1140736
_________________________________________________________________
dropout_1 (Dropout)          (None, 451, 512)          0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            2052
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0
=================================================================
Total params: 2,175,388
Trainable params: 2,175,388
Non-trainable params: 0
_________________________________________________________________
```

Figure 35 Model summary for 300 Emb_Dim and 256 Neurons with dropout



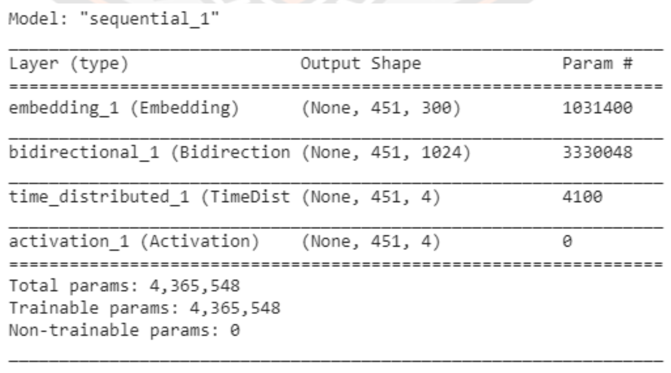
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 451, 128)          440320    
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 1024)         2625536   
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            4100      
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0         
=================================================================
Total params: 3,069,956
Trainable params: 3,069,956
Non-trainable params: 0
_________________________________________________________________
```

Figure 36 Model summary for 128 Emb_Dim and 512 Neurons without dropout



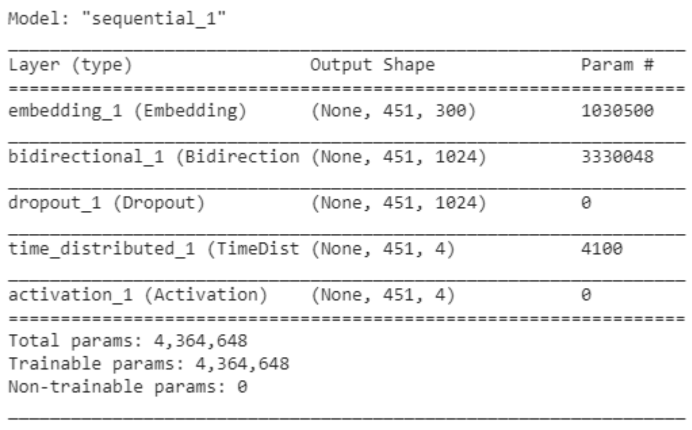
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 451, 128)          437632    
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 1024)         2625536   
_________________________________________________________________
dropout_1 (Dropout)          (None, 451, 1024)         0         
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            4100      
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0         
=================================================================
Total params: 3,067,268
Trainable params: 3,067,268
Non-trainable params: 0
_________________________________________________________________
```

Figure 37 Model summary for 128 Emb_Dim and 512 Neurons with dropout



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 451, 300)          1031400   
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 1024)         3330048   
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            4100      
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0         
=================================================================
Total params: 4,365,548
Trainable params: 4,365,548
Non-trainable params: 0
_________________________________________________________________
```

Figure 38 Model summary for 300 Emb_Dim and 512 Neurons without dropout



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 451, 300)          1030500   
_________________________________________________________________
bidirectional_1 (Bidirection (None, 451, 1024)         3330048   
_________________________________________________________________
dropout_1 (Dropout)          (None, 451, 1024)         0         
_________________________________________________________________
time_distributed_1 (TimeDist (None, 451, 4)            4100      
_________________________________________________________________
activation_1 (Activation)    (None, 451, 4)            0         
=================================================================
Total params: 4,364,648
Trainable params: 4,364,648
Non-trainable params: 0
_________________________________________________________________
```

Figure 39 Model summary for 300 Emb_Dim and 512 Neurons with dropout

Segmentation Generator

The trained models produce the sequence of tags for the given sequence of syllables. Consider the sequence of syllables as ‘ང་’, ‘གཡུས་’, ‘ཁར་’, ‘འགྲོ་’, ‘དོ’, and ‘།’are fed to the trained model and Y as the predicted tag sequence for the given sequence of syllable, *Y* = {0, 0, 1, 0, 0, 0}, where 0 represents tag ‘*beg*’, 1 as tag ‘*end*’ and 2 as tag ‘*mid*’. Then the given input sequence can be tagged as ‘ང་/beg’, ‘གཡུས་/beg’, ‘ཁར་/end’, ‘འགྲོ་/beg’, ‘དོ/beg’, and ‘།/beg’. Using the tag information, the final segmentation result of the given sentences can be viewed as ‘‘ང་’, ‘གཡུས་ཁར་’, ‘འགྲོ་’, ‘དོ’, and ‘།’. The syllable ‘གཡུས་’, and ‘ཁར་’are concatenated to get the word ‘གཡུས་ཁར་’ (village) since ‘གཡུས་’ has been tagged as the beginning syllable of the word and ‘ཁར་’ as the last syllable of a word. The complete flowchart for word formation using the predicted tags (Y) for the given input sequence is explained in figure 40 and the algorithm is illustrated below.

```
Input: Syllable sequence S = ($S_1$, $S_2$, $S_3$, $S_4$, .. , $S_n$ ) and output sequence T =
($T_1$, $T_2$, $T_3$, ………, $T_n$ )
word = [ ]
for i ⟵ 1 to n:
    if $t_i$ == 'beg' do
        word_candi = $s_i$
        for j ⟵ i+1 to n do
            if $t_i$ == 'beg' do
                break;
            else:
                word_candi = word_candi + $s_i$
    word. append(word_candi)
```

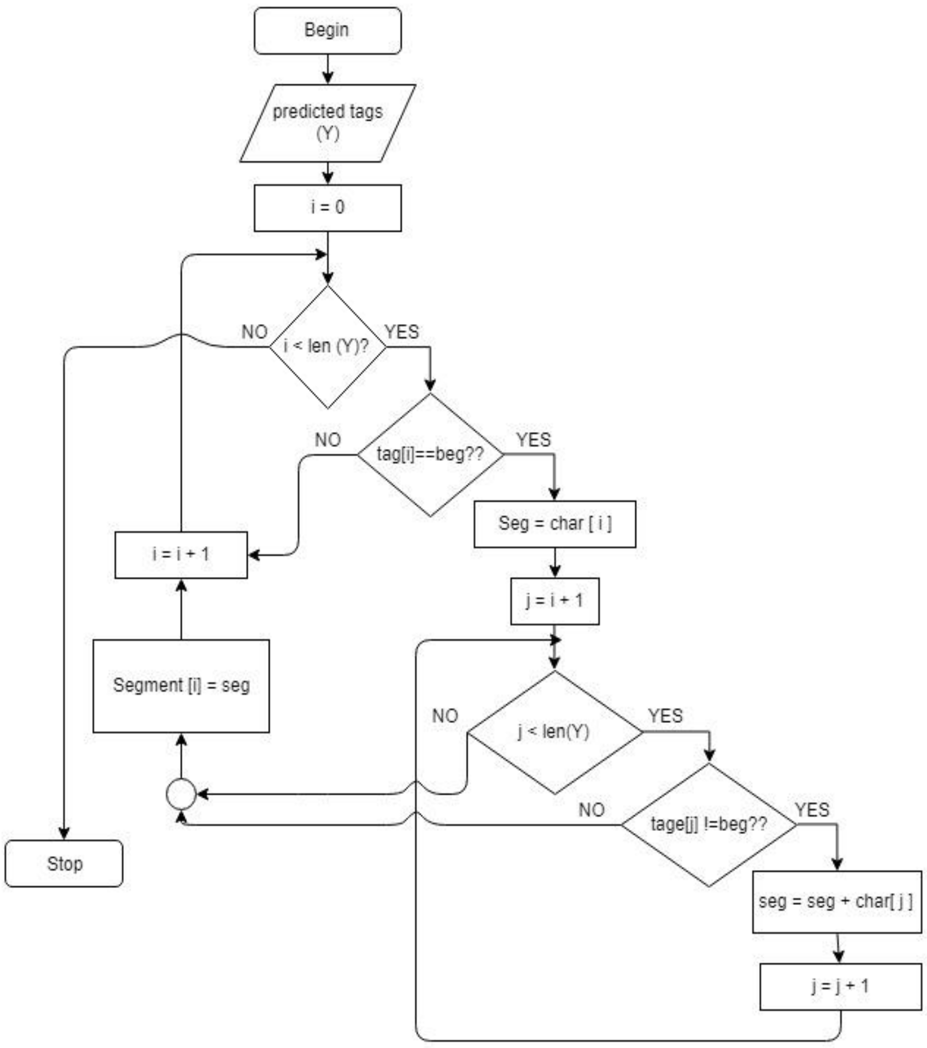


Figure 40 The flowchart of word formation using the predicted tag sequence of the model

Data Collection

The standard dataset is not available for Dzongkha. However, I have contacted the officials working in the Dzongkha Development Commission (DDC), the autonomous government organization responsible for the promotion and preservation of the languages in Bhutan, to help me with the datasets for this research. The officials including the Secretary were kind enough to share the dataset for segmentation. The dataset contains more than 10,226 sentences in which each syllable in the sentence is tagged either '*beg*' or '*mid*' or '*end*'. The sample dataset is shown below in Table 10.

Table 10 Dataset Sample

Sample 1		Sample 2	
ཪོ་	beg	ཁྱོད་	beg
ཀྲནམ་	end	ཀྱི་	beg
གིས་	beg	ཨ་	beg
བརྒྱུད་	beg	རོགས་	mid
འཕྲིན་	end	དགའ་	mid
འབྲུ་	beg	རོགས་	end
སྐྱེ་	beg	ཨིན་	beg
སློ་	beg	པས	beg
ཨ་	beg	ཟེར་	beg
ཚེ་	mid	བརྒྱུད་	beg
ཅིག་	end	འཕྲིན་	end
སླབ་	beg	ང་	beg
ཞིནམ་	beg	ལུ་	beg
ལས་	end	སྤྱོད་	beg
དཔལ་	beg	དེས	beg
བཟང་	end	।	beg

Source: DDC, 2019

The syllables in every sentence were tagged according to their position in the word. The datasets contained rows of tagged syllables in a TSV file as shown in Table 10. The figure 41 illustrates the statistics of syllables tagged with each tag.

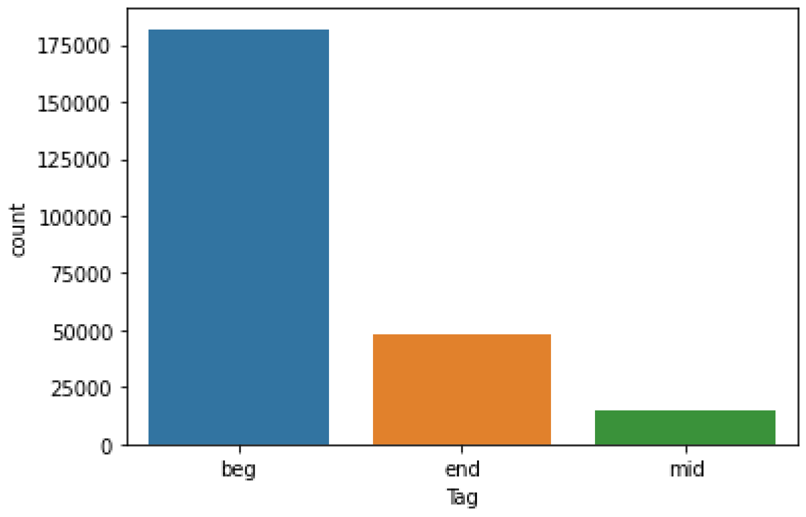
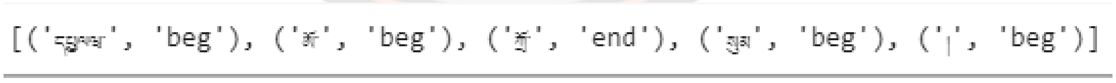


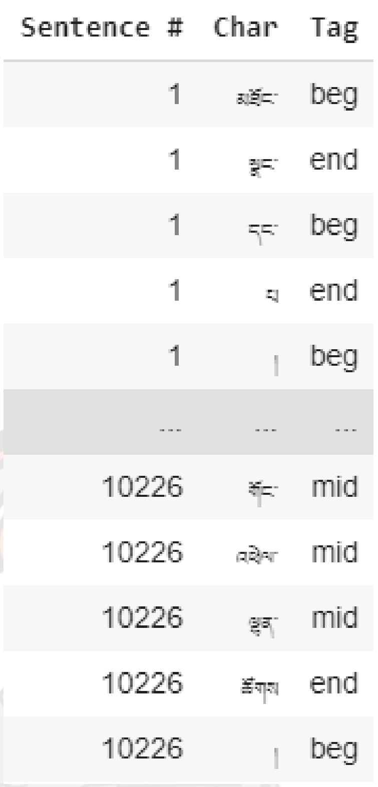
Figure 41 Statistics of tags in the dataset

The sentences in the dataset is delimited with white space. Using white space as the delimiter, the sentence number for each of the tagged syllable is appended as shown in figure 43. This could enable us to create tagged sentence in a simpler way. The tagged sentence is shown in Figure 42



```
[('[illegible]', 'beg'), ('ཀ', 'beg'), ('[illegible]', 'end'), ('[illegible]', 'beg'), ('།', 'beg')]
```

Figure 42 sample of Tagged sentence



Sentence #	Char	Tag
1	མཚོང་	beg
1	སྐང་	end
1	དང་	beg
1	པ	end
1	།	beg
...	...	...
10226	རྟོང་	mid
10226	འཕེལ་	mid
10226	རྒྱུན་	mid
10226	ཚོགས	end
10226	།	beg

Figure 43 Sentence Number appended to every tagged syllable

The dataset is split into training and test. 90% of the sentences in the dataset will be taken as a training dataset while remaining will be used as the test set. Further, the training set is split into the training set and validation set. 20% of the sentences in the training set will be considered as the validation set while remaining will be used as a training dataset. Figure 44 shows the proportion of the dataset for various sets. However, the data ratio of 80:20 is maintained for RNN models.

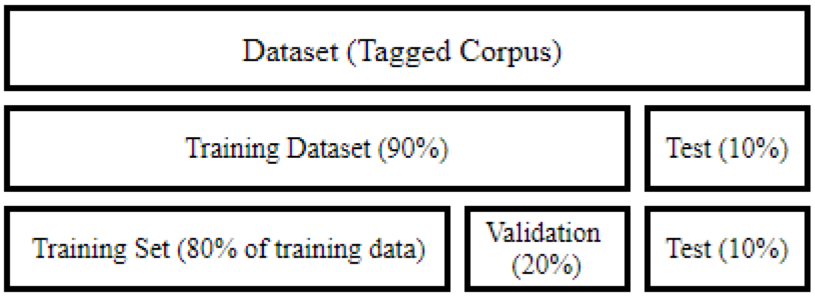


Figure 44 Proportion of datasets for training, validation and test sets

CHAPTER IV

RESULTS AND DISCUSSIONS

Introduction

The Dzongkha word segmentation was formulated as a syllable tagging problem in chapter III. The models designed for syllable were trained on Google Collaboratory (abbreviated as Google Colab) incorporating configurations presented in chapter III. Google Colab is created by Google research project to disseminate education on machine learning and its research. It is a free cloud-based Jupyter notebook environment that runs on Tesla K80 GPU with 12GB RAM. The Google Colab is simple to use because all the required libraries are inbuilt into it. This chapter presents the results of the designed experiments and discussion on the results. The dataset presented in chapter two was used for the experiments. These results are presented in two sections as Deep Neural Network and Bi-LSTM. Then, the comparative analysis between these experiments based on these two algorithms is presented in the last section.

Deep Neural Network

The DNN based tagger was proposed to a syllable in a word (Jamtsho & Muneesawang, 2020). The tag represents the position of the word, which can be then concatenated using the tag information of the syllable to form a valid Dzongkha word. The experiments were conducted as proposed in chapter III under the DNN section. The various context sizes for the target syllable were considered. This is mainly to determine the best context size for Dzongkha word segmentation based on syllable tagging task and also to study how the context of different size contributes to the task. Eight syllable tagger models were built for various context sizes and configurations. Each of the models was trained for 500 epochs.

The precision determines how precise is our models while recall determines the actual correctly predicted. The F1-score seek a balance between precision and recall. These matrices are calculated using the formulae presented in Chapter III under DNN based model section. Instead of coding to compute these matrices, *classification_report* class provided by *scikit-learn* library was used. Besides F1-score, we have also considered recording the accuracy of every DNN based model. This is to enable model comparison with the Bi-LSTM model. However, for this section F1-score will be considered for the discussion.

Table 11 presents the experimental results for various configurations. Our DNN based models achieved high accuracy as stated in Hypothesis 1 in Chapter I under the Hypothesis section. The 'Yes' or 'No' under the Pretrained Embedding column header illustrates the incorporation of pretrained embedding in the model.

Table 11 Experimental result for DNN based models

Pretrained Embedding Usage	**Context Size (N)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**	**Accuracy (%)**
Yes	0	83.46	85.13	83.98	85.13
	1	94.19	94.02	94.08	94.02
	2	**94.47**	**94.35**	**94.40**	**94.35**
	3	93.63	93.58	93.60	93.58
No	0	83.19	85.29	83.69	85.29
	1	94.19	93.86	93.99	93.86
	2	**94.20**	**93.95**	**94.05**	**93.95**
	3	93.91	93.56	93.70	93.56

From model configuration that does not use a pretrained embedding matrix, which is represented as 'No', the model with context size 2 achieved the highest F1-score of 94.05% with 94.20% precision and 93.95% recall. While the model with context size achieved the lowest F1-score of 83.69% with 83.19% precision and 85.29% recall. The second highest F1-score was obtained by the model with context size 1 which is followed by the model with context size 3.

Further, in another experimental set that uses pretrained syllable embedding, the same hierarchy of performance was maintained. The model with context size 2 achieved the highest F1-score of 94.40% with 94.47% precision and 94.35% recall which the model with context 2 achieved the lowest F1-score of 83.98% with 83.46% precision and 85.16% recall. The usage of pretrained syllable embedding has contributed to increasing the performance of the model as reported in (C. Wang & Xu, 2017). The performance gain for every context size of 0, 1, 2 and 3 is calculated as the difference of the F1-score between model without and with pretrained embedding. The performance gain is presented in table 12.

Table 12 Performance gain between two experimental sets

Context Size (N)	F1-score for models with pretrained embedding (A) (%)	F1-score for models without pretrained embedding (B) (%)	Performance Gain (A-B) (%)
0	83.98	83.69	0.29
1	94.08	93.99	0.09
2	94.40	94.05	0.35
3	93.60	93.70	-0.10

However, performance gain for a model with context size two was not observed. But in general, it is considered that performance gain is observed since it is shown from another context sizes.

Further, confusion matrices for every model of both experimental sets were computed using the *confusion_matrix* class from the *scikit-learn* library. The confusion matrices are presented in table 13. From the table, it is understood that model with context size 0 can handle the syllable with '*beg*' tag efficiently but performs badly with the syllables with '*mid*' tag while almost 47% of syllables with '*mid*' tag are mistagged. In addition, almost 52% of the syllables were wrongly tagged by the same model without pretrained embedding. It can be also observed that the model with context size 2 in both of the experimental sets was able to correctly tag the highest number of syllables with '*beg*' tag. For '*end*' tag, the context 2 model using a pretrained syllable embedding model and context 1 model without pretrained embedding correctly tagged the highest number of syllables while context 1 model in both of the experimental sets was able to correctly tagged the highest number of syllables with '*mid*' tag.

Table 13 Confusion matrix for two experimental sets using various context size

Pretrained embedding usage	**Context size (N)**	**Confusion Matrix**			
			beg	**end**	**mid**
YES	0	**beg**	**10566**	569	80
		end	1024	**1180**	15
		mid	269	105	**61**
	1	**beg**	**10833**	273	109
		end	297	**1852**	70
		mid	38	43	**354**
	2	**beg**	**10842**	277	96
		end	265	**1901**	53
		mid	43	50	**342**
	3	**beg**	**10830**	289	96
		end	321	**1837**	61
		mid	57	66	**312**
NO	0	**beg**	**10715**	426	74
		end	1142	**1065**	12
		mid	271	115	**49**
	1	**beg**	**10743**	358	114
		end	231	**1923**	65
		mid	41	42	**352**
	2	**beg**	**107**73	337	105
		end	240	**1916**	63
		mid	42	52	**341**
	3	**beg**	**10720**	377	118
		end	238	**1914**	67
		mid	46	47	**342**

From the experimental results, we concluded that context size 2 is the most suitable context size for the task of Dzongkha word segmentation which was formulated as a syllable tagging task. This is because the said model achieved the highest F1-score than other models and in the Dzongkha language, most of the words are with less than 5 syllables.

The model was designed to handle the out-of-vocabulary word. However, the statistics could not be provided because all the sentences provided in the dataset were used for building the syllable embedding and subsequently, the syllable vocabulary.

The training and validation accuracy during the training phase of the models are presented in the figures below.

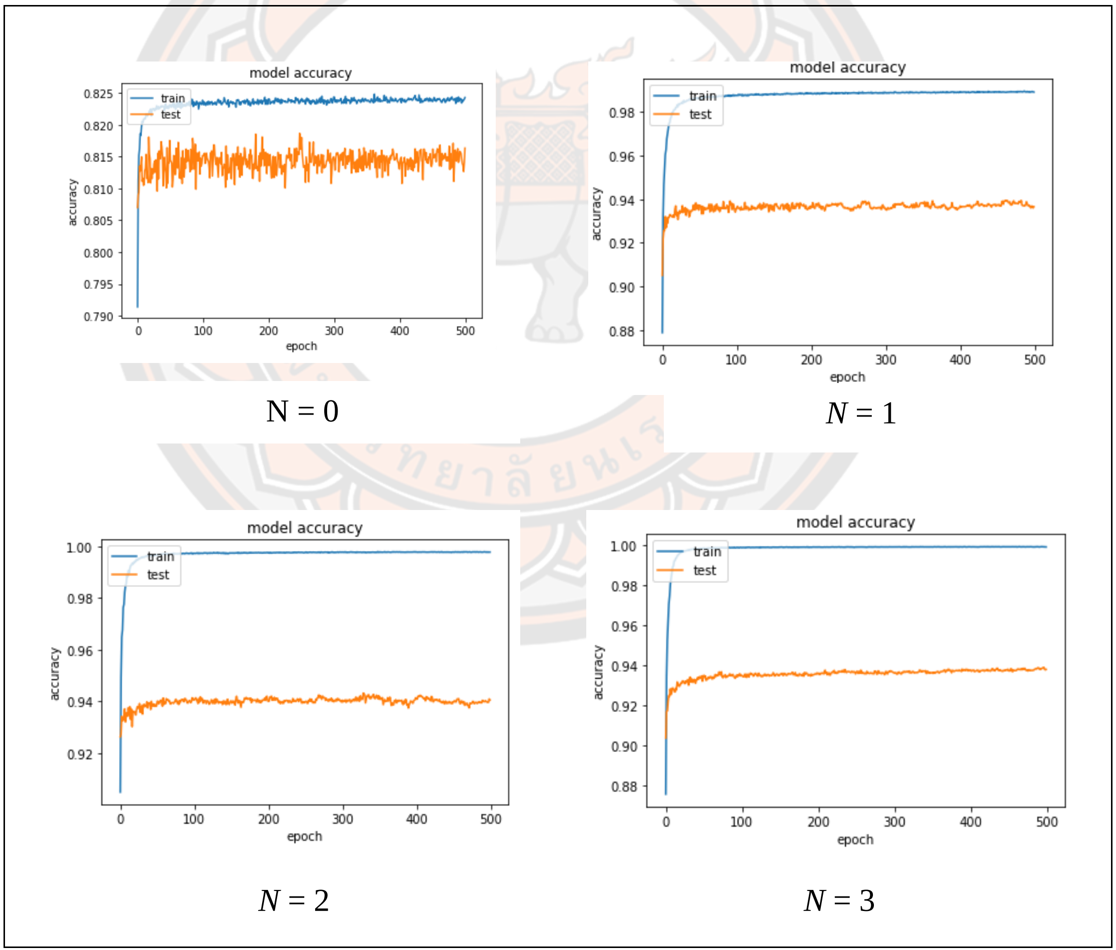


Figure 45 Training and validation accuracy for models without pretrained syllable embedding

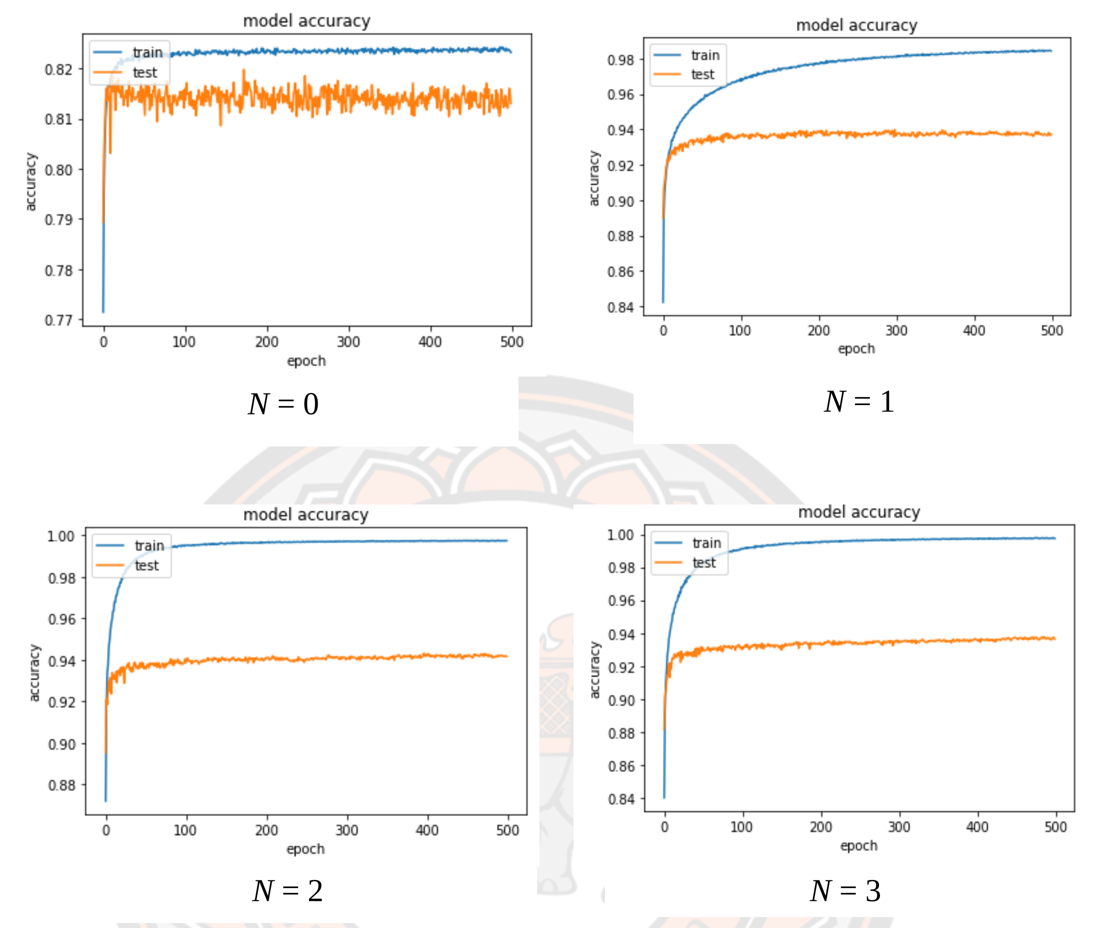


Figure 46 Training and validation accuracy for models with pretrained syllable embedding

Bi-LSTM

Another experiment was conducted on Dzongkha word segmentation which is formulated as a syllable tagging task. This time, a powerful deep algorithm that is applicable to sequential inputs (Young et al., 2018) was implemented. In our experiments, 24 experimental models were designed which can be broadly categorized into two sets based on the number of hidden neurons, which are 512 or 256. The experiments were set with different configurations. The attributes of configuration are Learning rate (LR), Dropout (0.2) and embedding dimensions (Emb). Such arrangements were made because there is no thumb rule to determine the appropriate hyperparameters for the given task. Further, Adam optimizer (Kingma & Ba, 2014) was adopted for this experiment. Our goal is to find the appropriate configurations for the task of Dzongkha word segmentation through the trial and error method. The experimental results for models using the Bi-LSTM algorithm with various configurations are presented in table 14.

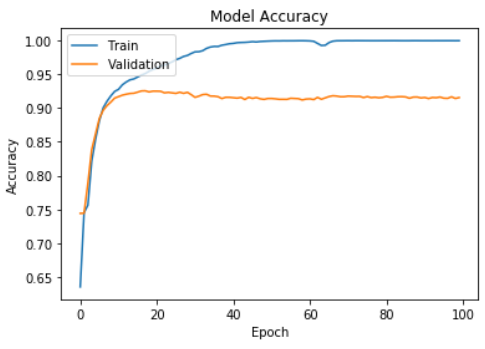
Table 14 Bi-LSTM model accuracy (%) for various configurations

Neurons	LR	Without Dropout		With Dropout (0.2)	
		128 Emb	300 Emb	128 Emb	300 Emb
512	0.001	91.76	92.12	91.45	91.81
	0.010	94.24	93.50	91.25	84.86
	0.020	88.90	89.90	84.70	86.44
256	0.001	91.05	91.75	91.23	91.38
	0.010	**95.25**	**94.40**	**94.64**	**94.58**
	0.020	95.21	94.24	94.07	92.22

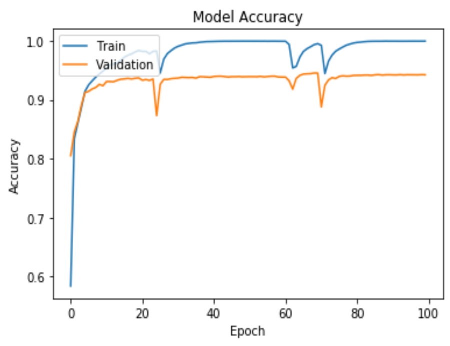
From the above table, it is observed that the model configurations with 256 neurons, learning rate of 0.010 and embedding dimension of 128 without usage of dropout rate 0.2 achieved the highest accuracy of 95.25%, while the lowest of 84.70% was recorded with the model configurations of 512 neurons, learning rate of 0.020 and embedding dimension of 128 with dropout rate of 0.2.

The performance of the models is considered to be significant because all the model performance was above 80.00%. Amongst the models, we considered the model with 95.35% as the best and suitable model for the Dzongkha word segmentation. Further, the same model with different embedding dimensions and dropout outperformed the other models in their categories as illustrated in the table using bold text. All the configurations under each embedding dimension are considered as one category.

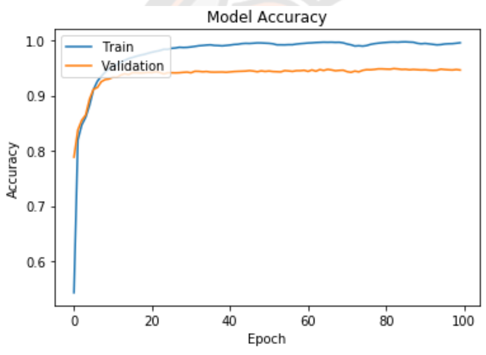
The figures below show the training and validation accuracy recorded during the training phase of the Bi-LSTM models with configuration 256 neurons and 300 embedding dimensions.



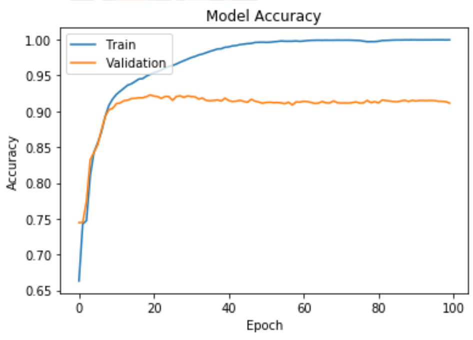
LR = 0.001 and without Dropout



LR = 0.01 and without Dropout



LR = 0.01 with Dropout rate 0.2



LR = 0.001 with Dropout rate 0.2

Figure 47 Training and validation accuracy for models for various configurations

Comparative Analysis

This section presents a comparison between the two models being presented in earlier sections of this chapter.

In the DNN based model, the output was based on the current input. The past information is lost when the next input placed into the input layer because it does not have the memory to store the past information, thereby it is not suitable for long term dependencies (Sak, Senior, & Beaufays, 2014). The window approach (context) has to be applied for DNN based model if one wishes to incorporate the information around the target input. On the other hand, the RNN based models such as LSTM (Hochreiter

& Schmidhuber, 1997) are designed for handling sequential data and long-term dependencies because LSTM maintains a cell state which functions as the memory to store the past input information.

We could not provide the direct comparison with the research done by Norbu et al. (2010) and (Dhungyel & Grundspeņķis, 2017), which were the only previous research for the Dzongkha word segmentation. However, their models heavily depend on dictionary (C. Wang & Xu, 2017)and the completeness of the vocabulary determines the robustness of the model (Theeramunkong & Usanavasin, 2001). The new words are continuously evolving in the Dzongkha language to sustain with technological development and unprecedented names for animals, human, etc. On the other hand, the models proposed in my research is research is robust. It can effectively handle out-of-vocabulary words. Further, requirement for feature engineering is avoided since the model learns features on its own.

The Bi-LSTM model configurations with 256 neurons, the learning rate of 0.010 and embedding dimension of 128 without the usage of dropout rate 0.2 outperformed the model accuracy DNN based model. The performance of Bi-LSTM was seen 0.85% higher than the highest accuracy of DNN based model. Table 15 illustrates the comparative accuracy of the DNN based model that was selected as the most suitable model in section II of this chapter, with all the configuration of the Bi-LSTM model that provides the highest accuracy in their categories.

Table 15 Comparative results for two models

	Without Dropout		**With Dropout (0.2)**	
Bi-LSTM (A)	**128 Emb**	**300 Emb**	**128 Emb**	**300 Emb**
	95.25	94.40	94.64	94.58
DNN (B)	94.35	94.35	94.35	94.35
(A-B)	**0.90**	**0.05**	**0.29%**	**0.23**

The syllables that does not exist in the dictionary or vocabulary can be considered as the noise or unknown syllables. In both of these deep learning models, all the unknown syllables or words can be annotated as ‘UNK’ token. A special ‘UNK’ token was introduced in the dictionary for handing the unknown words.

Further, traditional machine learning algorithms such as Support Vector Machine (SVM) (Antony, Mohan, & Soman, 2010; Vishwanathan & Murty, 2002) and Conditional Random Field (CRF) (Liu, Nuo, Ma, Wu, & He, 2011; PVS & Karthik, 2007) were used for syllabling tagging to enable to efficiency with of our model. The experimental results for SVM and CRF is presented in table 16. The CRF model has achieved the highest performance among traditional algorithms which is 12.70% higher than the accuracy of SVM models.

Table 16 Traditional machine learning model performance

Algorithms	**Accuracy**
Support Vector Machine (SVM)	80.00%
Conditional Random Field (CRF)	**92.70%**

CRF is the most commonly used traditional machine learning approach for sequence tagging task (Li, Savova, & Kipper, 2008) like Part of Speech (POS) tagging and Named Entity Recognition (NER). The deep learning models are computationally expensive to train while the opposite is seen the traditional approaches. However, the traditional approaches heavily depend on feature engineering (Gu et al., 2018). The manually driven features can be incomplete, time consuming and requires linguistic knowledge(C. Wang & Xu, 2017; M. Wang et al., 2018). The features used for CRF is presented below.

On other hand, although the deep algorithms are computational expensive to build, it provides an amazing performance as compared with the performance of the traditional algorithms without the need of manual feature engineering as the models learns the features automatically from the input and output set fed during the training of the models. The performance analysis between the traditional algorithm and deep learning algorithms is presented in table 17. The CRF has provided the highest accuracy of 92.70% among the traditional algorithms, whose accuracy is 2.55% and 1.65% lesser than the performance of RNN and DNN models, respectively. Thus, proposed deep learning algorithms are superior than traditional machine learning model.

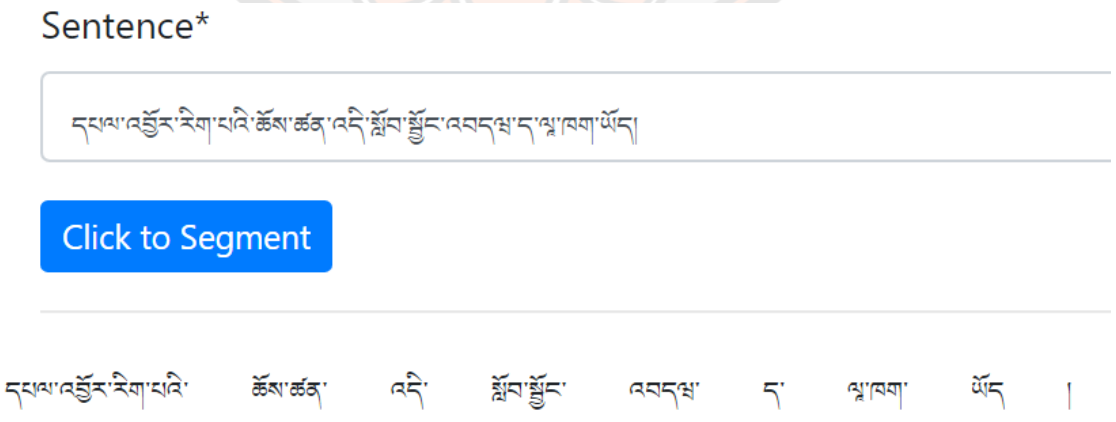
```
'is_first_syllable': int(index==0),
'is_last_syllable':int(index==len(sentence)-1),
'prev_word_1':'' if index==0 else sentence[index-1],
'prev_word_2':'' if index==0 or index==1 else sentence[
index-2],
'prev_word_3':'' if index==0 or index==1 or index==2 el
se sentence[index-3],
'next_word_1':'' if index==len(sentence)-
1 else sentence[index+1],
'next_word_2':'' if index==len(sentence)-
1 or index==len(sentence)-2 else sentence[index+2],
'next_word_3':'' if index==len(sentence)-
1 or index==len(sentence)-2 or index==len(sentence)-
3   else sentence[index+3],
'is_numeric':int(sentence[index].isdigit()),
'is_alphanumeric': int(bool((re.match('^(?=.*[0-
9]$)(?=.*[a-zA-Z])',sentence[index])))),
'prefix_1':sentence[index][0],
'prefix_2': sentence[index][:2],
'prefix_3':sentence[index][:3],
'prefix_4':sentence[index][:4],
'suffix_1':sentence[index][-1],
'suffix_2':sentence[index][-2:],
'suffix_3':sentence[index][-3:],
'suffix_4':sentence[index][-4:],
'word_has_hyphen': 1 if '-' in sentence[index] else 0
```

Table 17 Performance between traditional and deep learning algorithms

algorithms	Performance Accuracy (%)			
	Without Dropout		With Dropout (0.2)	
Bi-LSTM (A)	**128 Emb**	**300 Emb**	**128 Emb**	**300 Emb**
	95.25	94.40	94.64	94.58
DNN (B)	94.35	94.35	94.35	94.35
CRF (C)	92.70	92.70	92.70	92.70
(A-C)	**2.55**	**1.7**	**1.94**	**1.88**
(B-C)	**1.65**	**1.65**	**1.65**	**1.65**

Word Interpretation

The syllable tagger model produces the sequence of tags for the input sequences. The valid word in the given sentences can be identified using the tag sequence which was produced by the tagger model. The algorithm adopted in Chapter III was used for the word interpretation. The OOV syllable(s) will be annotated with 'UNK' token with the index 0 in the vocabulary. The output illustrated in figure 48 represents the segmented word for the input sequence 'དཔལ་འབྱོར་རིག་པའི་ཆོས་ཚན་འདི་སློབ་སྦྱོང་འབདཝ་ད་ལྷ་ཁག་ཡོད།'


Sentence*

དཔལ་འབྱོར་རིག་པའི་ཆོས་ཚན་འདི་སློབ་སྦྱོང་འབདཝ་ད་ལྷ་ཁག་ཡོད།

དཔལ་འབྱོར་རིག་པའི་ ཆོས་ཚན་ འདི་ སློབ་སྦྱོང་ འབདཝ་ ད་ ལྷ་ཁག་ ཡོད །

Figure 48 Segmented words using tag information produced by the tagger

CHAPTER V

CONCLUSION

Introduction

This is the chapter of my thesis. The summary of the main part of the thesis is presented in Summary section. The remainder of the chapter is organized as implication and recommendation of the study, limitations of the study and the future research work.

Summary

The Dzongkha script is written as a string of syllables without explicit word delimiters, unlike in English. For such language, the word segmentation is considered to be the fundamental steps in building NLP applications such as translator, text to speech system, spell, and grammar checker, etc. This is because word forms the basic constituent of any language. The meaning of the sentence or phrase depends on the participation of the word.

Most of the Asian languages are written without explicit word delimiters. Many researches have been done for those languages. However, there is not much research done for the Dzongkha word segmentation. In this thesis, the Dzongkha word segmentation was formulated as the syllable tagging task, where each of the syllables was tagged either 'beg', 'mid' and 'end' depending on their position in a word. The identification of the position of a syllable is very important in this task. The 'beg' tag represents the syllable at the beginning of the word or the syllable that forms word by itself, 'end' tags represents the syllable that marks the end of the word while the 'mid' tag represents the syllable in between the syllables with 'beg' and 'end' tag.

The syllable tagging algorithm can be of two variants. The traditional approach such as CRF. The traditional method heavily depends on manual feature engineering which is considered to be time-consuming and sometimes incomplete. The efficiency of the model depends on the effectiveness of feature engineering. In this thesis, modern approaches were used where deep learning algorithms were studied and applied for the task of Dzongkha word segmentation. The modern approaches skip the need for manual feature engineering. The Deep Neural Network and Bi-directional Long Short-term memory algorithms were proposed for syllable tagging task in this study.

In our experiment, the syllable vectors were computed using the *word2vec* model which is based on the skip-gram algorithm. Two experimental sets based on usage of pretrained syllable vectors were designed for DNN algorithms and each set comprised of four models of varied context size from 0 to 3. The contextual information was used because the tag of the syllable depends on its surrounding syllables. Amongst the eight DNN based models, the model with context size 2 achieved the highest F1-score of 94.40% or accuracy of 93.95%. The DNN based models are not suitable for long term dependencies because it does not have a special memory to record the previous layer information.

The special neural network called 'Bi-LSTM' was used in the second experiment. The algorithm has a cell state that records the precious that can be used for prediction in the next layer, thereby making it suitable for long term dependencies. A total of 24 models were trained using different configurations. This is carried out as the trial and error technique to determine the best model with an optimized configuration for the task of Dzongkha word segmentation since there is no hard rule to determine the optimal hyperparameters. Amongst these models, the model with 256 neurons, a learning rate of 0.01 and embedding dimensions of 128 without dropout regularization achieved the highest accuracy of 95.25%, which outperformed the performance of DNN based model. An increase in 0.90% was observed. The final segmented words were interpreted using the tag information obtained from the syllable tagger models.

The implication of the research and recommendations

The syllable that marks the beginning of the word and forms the word by itself were tagged with the '*beg*' tag. The performance of the model may increase if a sperate tag is introduced for the tag that marks the syllable. Further, there is no maximum amount of dataset for the deep learning models. The model performance will increase with a higher amount of data.

The Dzongkha segmentation system that has been proposed in this study can be used for further advancement in the field of the Dzongkha language processing. This system is considered as the initial step for the development of Dzongkha parser. The development of the NLP application would help to promote and preserve the national language of Bhutan. Further, the applications would enable effective communications between the foreigner and the native Bhutanese.

Limitations of the study

The following are the limitations of my study:

1. Dzongkha language is a morphically rich language (DDC, 2019; Wangdi, 2015) where the spelling of the word depends on its context. For example, ང་ཆུ་འཐུངམ་ཨིན། which means 'I am drinking some water'. The word 'འཐུངམ་' is the word derived from its stem word 'འཐུང་' to represent factual present. Such type of syllables or words are not morphologically considered in our work,
2. Besides being morphologically rich language, Dzongkha script has style of writing in the short form or abbreviated form. This enable people for quick writing. For example, the name of the person 'བཀྲ་ཤིས་' which can be transcribed as 'Ta-shi' can be written in an abbreviated form as 'བཀྲིས་'. Such styles of abbreviations are not considered in this work.
3. During the course of the experiments in this study, the focus was made only on the performance of the models. The different models presented in this work can be differentiated on the ground of computational cost such as parameter size, model size, and training time. The study on computational cost of the models is not considered, and
4. The segmentation models obtained from the experiments conducted in this study can be of higher storage and processing requirement, which it will not be applicable to deployment on devices with limited storage and processing power such as Raspberry Pi 3 (Termritthikun, Jamtsho, & Muneesawang, 2019).

Future Research

The limitations discussed in the above section are the basis for the future research. Some of the future research as follows:

1. Increasing the size of the datasets has to be considered because the effectiveness of the deep learning models depends on the size of its training dataset. Further,

the dataset should be able to handle abbreviated and morphological words, incorporating separate tag for syllable that forms the word by itself and the syllable that marks the beginning of the word.

2. Development of deep learning models suitable for deployment on devices with limited storage and processing power.
3. Application of Convolutional Neural Network (CNN) in the field of Dzongkha word segmentation. CNN has the advantage of efficiently capturing of n-gram features through its convolutional layer (C. Wang & Xu, 2017).
4. Further, the hybrid deep learning models can be proposed for the task of Dzongkha word segmentation. For example, a model with CNN and CRF. The Convolutional layer of CNN can be used to automatically captures features of the text and then, it can be fed to CRF for making the predictions.

REFERENCES

Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing, 5*(4-5), 185-196.

Antony, P., Mohan, S. P., & Soman, K. (2010). *SVM based part of speech tagger for Malayalam.* Paper presented at the 2010 International Conference on Recent Trends in Information, Telecommunication and Computing.

Arun, P., Parshu, D., Karma, W., Kesang, W., Uttar, R., & Yeshi, J. (2016). Automatic answer evaluation: NLP approach: ResearchGate.

Basatini, F. M., & Chinipardaz, R. (2014). Softmax Model as Generalization upon Logistic Discrimination Suffers from Overfitting. *J. Journal of Data Science, 4*, 563-574.

Cai, D., & Zhao, H. (2016). Neural word segmentation learning for Chinese. *arXiv preprint arXiv:1606.04300*.

Chen, M., Zhao, S., & Yang, K. (2017). *Neural architecture for tibetan word segmentation.* Paper presented at the 2017 International Conference on Asian Language Processing (IALP).

Chirawichitchai, N. (2014). *Emotion classification of Thai text based using term weighting and machine learning techniques.* Paper presented at the 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE).

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research, 12*(Aug), 2493-2537.

DDC. (2019). About Dzongkha Development Commission. *Dzongkha Development Commission.* from https://www.dzongkha.gov.bt/en/aboutus/about-dzongkha-development-commisiso

Dhungyel, P. R., & Grundspeņķis, J. (2017). Analysing the Methods of Dzongkha Word Segmentation. *Applied Computer Systems, 21*(1), 61-65.

Domhan, T., Springenberg, J. T., & Hutter, F. (2015). *Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves.* Paper presented at the Twenty-Fourth International Joint Conference on Artificial Intelligence.

Dorjee, K. (2014). Linguistic landscape of Bhutan: An overview of number of languages, language policy, language education, and language use in Bhutan. *Bhutan Journal of Research & Development, 3*(1), 79-101.

Driem, G. v. (1992). The grammar of Dzongkha. *Thimphu: Dzongkha Development Commission of the Royal Government of Bhutan*.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., . . . Cai, J. (2018). Recent advances in convolutional neural networks. *Pattern Recognition, 77,* 354-377.

Gulcehre, C., Moczulski, M., Denil, M., & Bengio, Y. (2016). *Noisy activation functions.* Paper presented at the International conference on machine learning.

Hirschberg, J., & Manning, C. D. (2015). Advances in natural language processing. *Science, 349*(6245), 261-266.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735-1780.

Hu, J., & Liu, Q. (2017). CASICT Tibetan Word Segmentation System for MLWS2017. *arXiv preprint arXiv:1710.06112*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167.*

Jamtsho, Y., & Muneesawang, P. (2020, 29 Jan.-1 Feb. 2020). *Dzongkha Word Segmentation using Deep Learning.* Paper presented at the 2020 12th International Conference on Knowledge and Smart Technology (KST).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, D., Savova, G., & Kipper, K. (2008). *Conditional random fields and support vector machines for disorder named entity recognition in clinical texts.* Paper presented at the Proceedings of the workshop on current trends in biomedical natural language processing.

Liu, H., Nuo, M., Ma, L., Wu, J., & He, Y. (2011). *Tibetan word segmentation as syllable tagging using conditional random field.* Paper presented at the Proceedings of the 25th Pacific Asia conference on language, information and computation.

Low, J. K., Ng, H. T., & Guo, W. (2005). *A maximum entropy approach to Chinese word segmentation.* Paper presented at the Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing.

Martins, A., & Astudillo, R. (2016). *From softmax to sparsemax: A sparse model of attention and multi-label classification.* Paper presented at the International Conference on Machine Learning.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Norbu, S., Choejey, P., Dendup, T., Hussain, S., & Muaz, A. (2010). *Dzongkha word segmentation.* Paper presented at the Proceedings of the Eighth Workshop on Asian Language Resouces.

Noyunsan, C., Haruechaiyasak, C., Poltree, S., & Saikaew, K. R. (2014). *A Multi-Aspect Comparison and Evaluation on Thai Word Segmentation Programs.* Paper presented at the JIST (Workshops & Posters).

NSB. (2017). 2017 POPULATION & HOUSING CENSUS OF BHUTAN: National Statistics Bureau of Bhutan.

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Pascanu, R., Mikolov, T., & Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR, abs/1211.5063, 2*, 417.

Peng, F., Feng, F., & McCallum, A. (2004). *Chinese segmentation and new word detection using conditional random fields.* Paper presented at the Proceedings of the 20th international conference on Computational Linguistics.

PVS, A., & Karthik, G. (2007). Part-of-speech tagging and chunking using conditional random fields and transformation based learning. *Shallow Parsing for South Asian Languages, 21*, 21-24.

Rauber, P. E., Fadel, S. G., Falcao, A. X., & Telea, A. C. (2016). Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics, 23*(1), 101-110.

Sak, H., Senior, A. W., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling.

Shabanian, S., Arpit, D., Trischler, A., & Bengio, Y. (2017). Variational bi-lstms. *arXiv preprint arXiv:1711.05717*.

Smith, L. N. (2017). *Cyclical learning rates for training neural networks.* Paper presented at the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV).

Sproat, R., Gale, W., Shih, C., & Chang, N. (1996). A stochastic finite-state word-segmentation algorithm for Chinese. *Computational linguistics, 22*(3), 377-404.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). *LSTM neural networks for language modeling.* Paper presented at the Thirteenth annual conference of the international speech communication association.

Tanaya, D., & Adriani, M. (2016). Dictionary-based Word Segmentation for Javanese. *Procedia Computer Science, 81*, 208-213.

TCB. (2018). BHUTAN TOURISM MONITOR 2018: Tourism Council of Bhutan.

Termritthikun, C., Jamtsho, Y., & Muneesawang, P. (2019). On-device facial verification using NUF-Net model of deep learning. *Engineering Applications of Artificial Intelligence, 85*, 579-589.

Theeramunkong, T., & Usanavasin, S. (2001). *Non-dictionary-based thai word segmentation using decision trees.* Paper presented at the Proceedings of the first international conference on Human language technology research.

Vishwanathan, S., & Murty, M. N. (2002). *SSVM: a simple SVM algorithm.* Paper presented at the Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290).

Wang, C., & Xu, B. (2017). Convolutional neural network with word embeddings for Chinese word segmentation. *arXiv preprint arXiv:1711.04411*.

Wang, M., Li, X., Wei, Z., Zhi, S., & Wang, H. (2018). *Chinese Word Segmentation Based on Deep Learning.* Paper presented at the Proceedings of the 2018 10th International Conference on Machine Learning and Computing.

Wangdi, P. (2015). Language Policy and Planning in Bhutan. *Retrieved April, 26*, 2016.

Xue, N. (2003). Chinese Word Segmentation as Character Tagging. *International Journal of Computational Linguistics & Chinese Language Processing, 8*, 29–48.

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine, 13*(3), 55-75.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhao, H., Huang, C.-N., & Li, M. (2006). *An improved Chinese word segmentation system with conditional random field.* Paper presented at the Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing.

Zheng, X., Chen, H., & Xu, T. (2013). *Deep learning for Chinese word segmentation and POS tagging.* Paper presented at the Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing.