

Generating Contextual Word Embeddings for Sinhala

J. A. S. N. Silva

2019



Generating Contextual Word Embeddings for Sinhala

**A Thesis Submitted for the Degree of Master of
Business Analytics**

J. A. S. N. Silva

University of Colombo School of Computing

2020



DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety.
I have duly acknowledged all the sources of information which have been used in the thesis.
This thesis has also not been submitted for any degree in any university previously.

Student Name: J. A. S. N. Silva

Registration Number: 2018BA032

Index Number: 18880323



14/09/2021

Signature of the Student & Date

This is to certify that this thesis is based on the work of Ms. J. A. S. N. Silva under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name: Dr. A. R. Weerasinghe



14/09/2021

Signature of the Supervisor & Date

I would like to dedicate this thesis to
My family, friends, colleagues and teachers who were an immense source of support and
guidance throughout

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Dr. A. R. Weerasinghe, who provided valuable assistance, encouragement and guidance throughout the process of my research. Without his assistance and feedback it could have been impossible to complete my research within exact time limits.

Also, I would like to convey my warm gratitude to my parents, my brother and family members, who always provided moral support and encouragement to accomplish my academic goals in the best possible ways.

Last but not least, I would like to thank my all colleagues, for their enormous support and encouragement provided throughout the project.

ABSTRACT

Word embeddings are an important aspect in Natural Language Processing (NLP) which tries to find a better representation of a word. One such application of word embedding is predicting the next word of a given phrase. Mostly word embedding techniques are applied for high resource languages. Word embeddings done for low resource languages are very rare. Sinhala language can be considered as a low resource language. Therefore, in this study we addressed the issue of predicting the next word of given phrase in Sinhala language.

The data for the study were collected from the web and were preprocessed to remove the noise. The preprocessed data were fed into 3 types of models namely n-gram model, LSTM model (which builds the word embedding by ignoring the meaning of the word based on its context, therefore the true meaning of the sentence might get ignored) and RoBERTa model (which learns the meaning of the word based on its context). Best model has been selected by comparing the perplexity values.

According to the results obtained among the n-gram models created the tri-gram model obtained the lowest perplexity value of 502.34. From the LSTM models created the LSTM model which considered four previous words for the prediction had a better performance (perplexity: 44.53). From the results obtained for BERT model it can be concluded that distilBERT model performs better compared to BERTbase model. The lowest perplexity value for distilBERT model was 3.19 at the epoch 30. Further, it can be concluded that RoBERTa model is better for language modelling compared to n-gram model and LSTM model.

LIST OF FIGURES

Figure 1-1 Sinhala Alphabet	2
Figure 2-1 Results obtained from mBERT and E-mBERT models	8
Figure 3-1 Alan Turing	13
Figure 3-2 Wordpiece tokenization for Sinhala language	15
Figure 3-3 Flow of information in an LSTM model	17
Figure 3-4 Architecture of the BERT model	19
Figure 4-1 Method	23
Figure 4-2 Usage of the toolkit	26
Figure 4-3 Part of the dataset of len_2	27
Figure 5-1 Zipf's law behaviour of the dataset	32
Figure 5-2 Most common bigrams	33
Figure 5-3 Most common trigrams	33
Figure 5-4 Comparison of the perplexity values in BERT model	36

LIST OF TABLES

Table 2-1 Comparison of SCIBERT with other models	10
Table 4-1 Text distribution according to the source	25
Table 5-1 Highest frequent words	32
Table 5-2 Perplexity values of n-gram models	34
Table 5-3 Perplexity values of the LSTM models	35
Table 5-4 Perplexity values of the created BERT models	36

TABLE OF CONTENT

Acknowledgement	III
Abstract	IV
List of Figures	V
List of tables.....	V
Table of content	VI
CHAPTER 1: INTRODUCTION	1
1.1 Background of the study	1
1.2 Sinhala language	2
1.3 Motivation	3
1.4 Objective	3
1.5 Significance	3
1.6 Scope of the project.....	3
1.7 Structure of the dissertation.....	4
CHAPTER 2: LITERATURE REVIEW	5
CHAPTER 3: THEORETICAL FRAMEWORK.....	13
3.1 Natural Language processing (NLP).....	13
3.2 Text corpus	14
3.3 Text preprocessing	14
3.3.1 Lower casing text data	14
3.3.2 Stemming and lematization.....	14
3.3.3 Tokenization	14
3.3.3.1 WordPiece tokenization.....	15
3.4 Zipf's law	16
3.5 Language models.....	16
3.5.1 N-gram models.....	16
3.5.2 LSTM (Long-Short-Term-Memory) model.....	17

3.6	Word embedding	18
3.6.1	Traditional word embedding.....	18
3.6.2	Contextual word embedding.....	18
3.7	BERT model.....	18
3.7.1	Model architecture of BERT.....	19
3.7.2	MLM in pre training	19
3.7.3	Sentence prediction in pre training	20
3.7.4	Finetuning of the BERT model.....	20
3.8	mBERT model.....	20
3.9	RoBERTa model	21
3.10	Evaluating language models	21
3.10.1	Perplexity measure.....	21
3.10.2	Accuracy measure.....	22
CHAPTER 4: METHODOLOGY		23
4.1	Data collection.....	23
4.2	Method description.....	23
4.2.1	Data pre-processing	24
4.2.2	Creating the corpus	24
4.2.3	Analysing the pre-processed dataset.....	25
4.2.4	Creating the n-gram model	26
4.2.4.1	CMU toolkit.....	26
4.2.4.2	Building the n-gram model.....	26
4.2.5	Creating the LSTM model	27
4.2.5.1	Creating the dataset	27
4.2.5.2	Encoding the created sequences	28
4.2.5.3	Creating input sequences and output elements.....	28
4.2.5.4	Building the model	28

4.2.6	Creating the BERT model.....	29
4.2.6.1	Training the tokenizer.....	29
4.2.6.2	Building the model	30
4.2.6.3	Creating the training dataset.....	30
4.2.6.4	Data collator	30
4.2.6.5	Initializing the trainer and pretraining the model	31
4.2.6.6	Testing the model	31
CHAPTER 5: EVALUATION AND RESULTS		32
5.1	Zipf’s law behaviour	32
5.2	Frequent words in the dataset.....	33
5.3	Results from n-gram model.....	34
5.4	Results from LSTM model.....	35
5.5	Results from the BERT model	35
CHAPTER 6: CONCLUSION AND FUTURE WORK.....		37
APPENDIX.....		38
Appendix A: languages trained on mBERT		38
Appendix B: Codes		39
Code for pre-processing.....		39
Code for n-gram calculation		40
Code for calculating perplexity in n-gram model: Using toolkit.....		41
Code for Zipf’s law		41
Code for LSTM model.....		43
Code for BERT tokenizer		46
Code for BERT model		46
REFERENCES		49

ABBREVIATIONS

AI	–Artificial Intelligence
ALWE	– Attention based multilayer Word Embedding
BERT	–Bidirectional Encoder Representations from Transformers
mBERT	–Multilingual BERT
RoBERTa	–Robustly optimized BERT approach
CBOW	– Continuous Bag of Words
KNN	–K-Nearest Neighbor
MLE	–Maximum Likelihood Estimate
MLM	–Masked Language Model
NER	–Named Entity Recognition
NLP	–Natural Language Processing
PoS	–Part of speech
SVM	–Support Vector Machine
TF-IDF	–Term Frequency-Inverse Domain Frequency
LSTM	–Long-Short-Term-Memory

CHAPTER 1: INTRODUCTION

1.1 Background of the study

In the modern world computers play a vital role in day today life. From governing the security and financial aspects of a country to domestic usage, it is very useful. As computers become more popular people tries to make it more user friendly but however the communication between humans and computers is not based on human language. Making computers understand the human language is the goal of computational linguistics. When considering about computational linguistics, representing words in a human language is a problem. The earliest attempts performed can be recognized as vector space models and statistical approaches such as maximum likelihood estimates of the words and using n-grams. One of the earliest usages of this model is in the field of speech recognition. By using statistical approaches syntax of the word cannot be captured (only the semantics can be captured).

Therefore, to capture the syntaxes of the words a method named ‘bag-of-words’ were proposed by S. Harris, where the text is considered as a bag of its words. It will create a vocabulary and assign each word in the vocabulary a unique index. Each document is then represented by a vector of length n (n = no of words) with the number of occurrences in each word. Due to the large size of the vocabulary this will lead to high dimensional feature vectors. The Term Frequency-Inverse Document Frequency (TF-IDF) method is a variation of bag of words method. It reweights the frequency vectors with the inverse document frequency of each word.

According to the literature (Almeida & Xexeo, 2019) word embeddings are ‘dense, distributed, fixed length word vectors built using word co-occurrence statistics as per distributional hypothesis’. These vectors can capture relationships between words and their meanings. However, each word had a particular embedding which was independent of context. Traditional word embeddings (GloVe and word2Vec) are implemented by ignoring the meaning of the word based on its context. The problem here is since it ignores the contextual meaning of the word the original meaning of the sentence may be violated. Therefore, as a solution the contextual word embedding concept was introduced. It is implemented using transformer models (mostly based on BERT) and it attempt to model the word semantics based on its context.

Word embeddings is a good research area which tries to find a better representation of a word. It can be useful in many Natural Language Processing (NLP) tasks, such as question answering, word prediction and generating possible sentences. Today word embedding techniques are applied for high resource languages, but word embeddings done for low resource languages are very rare.

1.2 Sinhala language

Sinhala is the native language spoken by the Sinhalese people in Sri Lanka, which is the largest ethnic group in the island. It belongs to the Indo – Aryan group of languages. The oldest Sinhala script was found in third to second century BCE. Sinhala has its own writing style which comes from the Indian Brahmi script which contains 60 letters; 18 vowels and 42 consonants (figure 1.1), but only 57 (16 vowels and 41 consonants) are required for writing spoken Sinhala. In the writing system of Sinhala language the consonants are written with letters while the vowels are indicated with diacritics (*pilla*) on those consonants. Sinhala letters are divided into 2 sets named as ‘*sudda Sinhala*’ (pure Sinhala) and ‘*misra Sinhala*’ (mixed Sinhala) and most of the letters are circular. Sinhala sentences are written from left to right.

සිංහල වර්ණ මාලාව

අ	ආ	ඇ	ඈ	ඉ	ඊ
උ	ඌ	ඍ	ඎ	ඏ	ඐ
එ	ඒ	ඓ	ඔ	ඕ	ඖ
ඞ	ඟ				
ක	ඛ	ග	ඃ	඄	අ
ආ	භ	ඈ	ඉ	ඊ	උ
ඌ	ඍ	ඎ	ඏ	ඐ	එ
ඒ	ඓ	ඔ	ඕ	ඖ	඗
඘	඙	ක	ඛ		
ඛ	ඞ	ඟ	ඈ	ඉ	ඊ

Figure 1-1 Sinhala Alphabet

1.3 Motivation

Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) which deals with the interaction between the computers and human using the natural language. A quite amount of research have been done using NLP for English language but NLP tools for Sinhala are very rare.

Various tasks can be performed using NLP. One such application is word prediction. It is a word processing feature that suggest the next word while you type a sentence. This facility can be found in smart phones in the modern world. Google has developed this facility for 102 languages including English language, but so far this facility is not available for Sinhala language.

1.4 Objective

The objective of this research is to generate a word embedding scheme for Sinhala language and then to develop a system that can be used to generate the next possible word for a given phrase and to generate Sinhala sentences. To meet this goal following sub objectives should be achieved

- Collect a representative corpus of Sinhala text
- Design a word embedding schemes using the dataset and compare them.
- Use proposed contextual embedding scheme to generate the next possible word for a given phrase.
- Explore the generation of longer sequences such as phrases and sentences using the proposed embedding.

1.5 Significance

Many upstream Sinhala language processing tasks and applications will be able to use the propose contextual embedding in order to improve their performance.

1.6 Scope of the project

- Word representations for written Sinhala text will only be considered.
- Only prose of the non-fiction genre will be considered.

1.7 Structure of the dissertation

Chapter 1 introduces the research based on NLP in human languages. Chapter 2 gives a brief description about few previous works done by other researchers around the world relevant to this study and about the NLP researches related to Sinhala language. Chapter 3 consists of the theory on which this research was conducted. Chapter 4 provides a clear idea about how the research was carried out, what instruments and software used and how the model was trained and tested. Chapter 5 contains the results obtained from the model and analysis of them. Discussions, limitations and hardships while doing this research are mentioned in chapter 6 with further development ideas. Final chapter contains the conclusion of the study and the summary.

CHAPTER 2: LITERATURE REVIEW

Human languages are unstructured. To make computer to understand a human language the concept of NLP was introduced in 1950s. It is a branch of AI which can be used to predict words, machine translations, chatbox (voice assistants such as Siri and Alexa) etc. Several studies have been performed for English language using NLP techniques are mentioned below.

One basic requirement for a machine to understand a human language is to understand its vocabulary. In NLP this can be referred as word embedding. The earliest methods used are Word2Vec, GloVe and FastText. Word2vec is a word embedding model which is trained using a feedforward neural network. It has 2 types of architectures namely, Continuous Bag Of Words (CBOW) and continuous skip-gram model (Mikolov, et al., 2013). In CBOW model it predicts which is the most likely word in the given context. Therefore, words with equal likelihood of appearing are considered as the same and occur closer in the dimension space while in the skip-gram model it predicts the context using the given word. An extension named FastText was introduced in 2017 to the skip-gram model (Bojanowski, et al., 2017) by considering the morphology of the words. Using this methodology unknown words also can be presented in vector form. The GloVe model was introduced by (Pennington, et al., 2014). It is trained on aggregated global word to word co-occurrence matrix from a given text collection of text documents. This co-occurrence matrix is decomposed to form denser and expressive vector representation.

Before 2016 the available word embedding methods were not optimized for sentence representations. Therefore as a solution for this matter, an estimation of sentence embeddings was done using Siamese Continuous Bag of Words (Siamese CBOW) model (Kenter, et al., 2016). Word embeddings were trained using Toronto Book Corpus which contained around 74 million preprocessed sentences, which was made from around 1 billion tokens. The final vocabulary contains 315,643 words. The tokens which appeared more than 5 times were only considered. The model was evaluated on 20 datasets from a variety of sources. According to the results obtained 14 out of 20, Siamese CBOW outperforms a word2vec baseline and a baseline based on the recently proposed skip-thought architecture.

A continuous-valued word embedding method have been used in (Minih & Kavukcuoglu, 2013) to capture semantic and syntactic information about words. The word

embeddings were learned by training log-bilinear models with noise-contrastive estimation. The dataset used was the English Wikipedia and a collection of 500 Gutenberg text that form canonical training data. After performing preprocessing techniques, the dataset consists with 1.5 billion words. To speed up the training only 1 word was predicted. The best results were shown by learning high dimensional embeddings from a large amount of data.

In (Aurnhammer, et al., 2019) a word prediction for natural sentence reading was performed using information and theoretic measures (suprisal and next-word entropy). To overcome the short comings lookahead information gain was introduced. During the study the information theoretic measures were compared with 3 datasets (ECG, eye tracking and self-paced). Initial and final words of a sentence, words before comma and words with clitics were excluded. The data were analyzed using linear mixed effect regression models. According to the computational results obtained in the predictive processing system, the cost of predicting outweighs the gains.

A study was performed in (Tien, et al., 2018) to encode the characteristics from set of word embeddings into a single word embedding. By using this newly created embedding the similarity between sentences can be learned. Another research (Babu & S, 2020) was done to find the similar queries or searches using word embeddings. The word embeddings were used on questions asked on Stack Overflow using the Cosine similarity.

The most popular method for contextual word embedding is the use of BERT (Bidirectional Encoder Representations from Transformers) model which was developed by Google in 2018. The first BERT model was created for English language and today Google has created the BERT model for 104 languages which is named as multilingual BERT (mBERT). A comparison of the geometry of BERT, ELMo and GPT-2 embeddings were performed in 2020 (Ethayarajh & Kawin, 2020). Some of the findings are mentioned below.

1. The contextualized word representations of the words are anisotropic in all layers and it is more anisotropic in higher layers.
2. The vector representations of the same word occurrences in different context are non-identical. In the upper layer these representations are very dissimilar. Therefore, task-specific representations are produced in upper layers of LSTMs.
3. Stopwords have the most context-specific representations.
4. The context-specificity manifests are different in the 3 models.
5. In ELMO, words in the same sentence are more similar to one another in upper layers.

6. In BERT, words in the same sentence are more dissimilar to one another in upper layers.
7. In GPT-2, word representations in the same sentence are no more similar to each other than randomly samples words.
8. Less than 5% of the variation in a word's contextualized representations can be explained using the first component by adjusting the effect of anisotropy.
9. Principle components of contextualized representations in lower layers outperform GloVe and FastText on many benchmarks.

As mentioned above the mBERT model is only focused on the top 104 languages in Wikipedia. Therefore, in 2020 a new approach to extend mBERT was proposed (E-mBERT) so that any unseen language can be accommodate (Wang, et al., 2020). The NER task was performed for 27 languages out of that 11 languages were not included in mBERT. The monolingual data of the target language was created using the same procedure in BERT. The words which appeared on the mBERT vocabulary was filtered out. The encoder and decoder weights has been extended so that it can work with the new vocabulary. According to the results obtained (Figure 2-1) it is clear that E-mBERT performs well compared to mBERT irrespective of the whether the language is present in mBERT or not. Further it can be concluded that E-mBERT performs well when the language is not present in the mBERT model.

In the same study the E-mBERT model was compared with the bilingual BERT (B-BERT) model. B-BERT is a model which is trained in the same way as mBERT, but it only contains 2 languages (English language and the target language). In this study the languages that are not included in mBERT were considered. According to the results obtained it can be concluded that E-mBERT has a better performance for most of the languages.

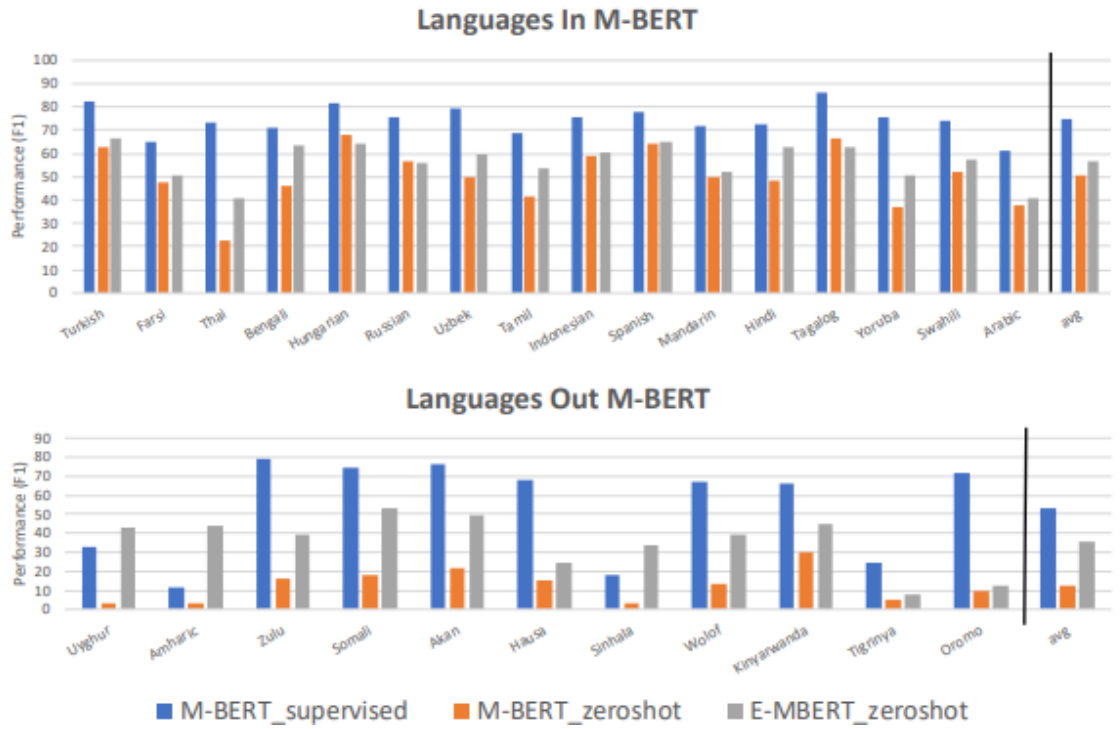


Figure 2-1 Results obtained from mBERT and E-mBERT models

The BERT model was used to analyze the language style in twitter in Italian language (Poligano, et al., 2019) which was named as ALBERTo. The dataset included 200 million tweets in Italian language. The model has been trained using the original BERT model. Model was evaluated for sentiment analysis for Italian language. According to the results obtained the system performed well and when the number of epochs of the learning phase of fine tuning was increased, the results were increased between 7% and 11%.

In (Canete, et al., 2020) a BERT based language model was pretrained on Spanish data. The data were collected from Wikipedia and some other sources (OPUS project) that contained Spanish data. The final training corpus contained 3 billion words. The collected corpora were comparable with the corpora which used in original BERT. The model was trained both for cased data and uncased data. In the training step each sentence had 10 masks (dynamic masking) and also Whole-Word Masking (WWM) were used. Larger batches were used for training when compared to the original BERT. According to the results obtained Spanish BERT outperform most of the multilingual BERT (mBERT).

The Dutch BERT model (BERTje) was introduced in (de Vries, et al., 2019). the data consist of Dutch text (taken from books, news corpus and Wikipedia). By considering the

overall quality of the data the data obtained from twitter were removed. After performing the necessary pre-processing techniques, the final dataset consists with 2.4 billion tokens. In this model the pretraining task was modified such as different technique to perform the Masked Language Model (MLM) task. In this study consecutive words belong to the same word were masked. 15% of the tokens were masked using this technique. According to the results obtained for word level NLP tasks the model outperforms the mBERT.

An upgraded version of BERT was released recently which is named as Whole Word Masking. In 2019 this method was adopted for Chinese BERT (Cui, et al., n.d.), using data collected from Wikipedia. Further the efficiency of the existing models of Chinese pretrained models (BERT, ERNIE, BERT-wwm, BERT-wwm-ext, RoBERTa-wwm-ext and RoBERTa-wwm-ext-large) were also analysed. the models were analysed using 3 datasets namely, CMRC 2018, DRCD and CJRC According to the analysis,

- BERT-wwm shows significant improvements on CMRC 2018 and DRCD.
- ERNIE does not perform well on DRCD.
- Traditional Chinese character are removed from the vocabulary of ERNIE.
- BERT-wwm performs well for CJRC compared to BERT and ERNIE.

In (Ma, et al., 2019) Attention-based multi-layer word embedding model (ALWE) was used to find the semantic structure of Chinese words. Data were collected from Chinese Wikipedia and as preprocessing techniques numbers and non-Chinese words were removed. According to the results obtained on word similarity, word analogy, and text classification, the proposed model showed a 1% in improvement accuracy compared to all other baselines.

In 2019 (Alsentzer, et al., 2019) the BERT model was implemented in related to bio medical field namely BioBERT. The dataset used was clinical text with 2 million notes. The model was trained for 2 types of datasets namely clinical BERT (text from all types of notes) and discharge summary BERT (only contain information about discharge summaries). The tensorflow implementation of original BERT model was used for pre-training. From the results obtained it can be concluded that clinically fine-tuned BioBERT show more performance compared to general BERT model. The clinical BERT had an accuracy of 82.7%. This model has some limitations, such as the created model was not experimented with advance model architectures and the data used only contained ICU data.

A study was performed using NLP techniques in scientific domain, as a result the BERT model which trained on scientific text namely SciBERT was released in 2019 (Beltagy, et al., 2019). It was a pre-trained language model based on BERT. The dataset included 1.14 million papers from semantic scholar (18% from computer science and 82% from biomedical domain). 4 different versions of SCIBERT was trained using the same configuration and size of the BERT-base. Improved results can be obtained by finetuning the model. As future work SCIBERT model can be developed based on the BERT-large model. Table 2-1 will provide a comparison of the results obtained by SCIBERT with state-of-the-art (SOTA) and BERT-base.

Table 2-1 Comparison of SCIBERT with other models

<i>Data domain</i>	<i>No of datasets</i>	<i>BERT-base</i>	<i>SOTA</i>
Biomedical	7	SCIBERT outperforms	For 3 datasets the performance of SCIBERT is worst.
Computer science	3	SCIBERT outperforms	SCIBERT achieves new SOTA results on ACL-ARC.
Multiple	2	SCIBERT outperforms	SCIBERT outperforms for the SciSite dataset.

In 2019 a study based on pretraining in BERT was done using 5 different English language corpora (Liu, et al., 2019). The created model was named as Robustly optimized BERT approach (RoBERTa). The pretrained model was evaluated using following,

1. The General Language Understanding Evaluation (GLUE): finetuning procedure is same as the original BERT model. According to the results obtained the RoBERTa model has a better accuracy compared to BERT model.
2. The Stanford Question Answering Dataset (SQuAD): 2 versions were evaluated. In the first version it always contains with an answer while the other version some questions were not answered. From the results it can be concluded that RoBERTa performs well for all the data except for 1.

3. The Reading comprehension from Examiners (RACE): the task is to select the correct answer for the question using the available 4 options. According to the results RoBERTa outperforms the standard BERT model.

Sinhala is the native language of Sinhalese people in Sri Lanka. It is a morphologically rich and low resourced language. Therefore, research work based on NLP techniques for Sinhala language are very rare compared to English language. According to the resources the earliest attempt on Sinhala NLP was done by S. Herath in 1991 and also the Language Technological Research Laboratory of UCSC has been involved in Sinhala language related NLP research since 2004.

To develop tools for machine translation, spelling/ grammar correction and speech recognition NLP techniques were used to analyze the Sinhala language (Gallege, n.d.). Data for the study were collected from LTRL Sinhala corpus, stories by Martin Wickramasinghe and online newspapers. The data were preprocessed in order to apply NLP algorithms. This study includes Maximum Likelihood Estimates (MLE) on Sinhala Characters, Language Identification using a Naïve Bayes Classifier, Zipf's Law Behavior, Topic Classification using Support Vector Machines (SVM) and Language Models. According to the study all NLP techniques applied produced satisfactory results

In 2020 the first evaluation of different types of word embeddings for Sinhala language was performed (Lakmal, et al., 2020) using 3 standard word embedding models (Word2Vec, FastText and GloVe). The methods were evaluated using 2 methods namely intrinsic evaluation and extrinsic evaluation. The data for the study were collected using common crawl and 2 corpora were created, with and without stopwords. Overall best results were reported for FastText word embedding with 300 vector dimensions.

A Sinhala question answering system named 'Mahoshadha' was created by (Jayakody, et al., 2015), the first QA system introduced for Sinhala language. The system created have 4 parts namely, document summarizing, document categorizing, question processing and answer processing. The document summarization was done using SVM classifier. K-nearest neighbour (KNN) was used for document categorization. The question processing is divided into 2 components as analyzing the question and identifying the answer. The n-gram model was used

for answer processing. According to the results it has proven that the generated outputs are accurate.

In 2013 (Karunaratne, et al., 2013) a study was performed to predict Sinhala sentences in SMS. The objective of this research was to develop an algorithm to reduce the typing effort and for time saving. The challenging part of this research was it was hard to develop an algorithm since the SMS writers do not follow language rules. The data were collected for 6 month period from 30 users belongs to 5 categories. The n-gram model was used in this study. From the results obtained it can be concluded that the developed algorithm has reduced typing time.

A survey was conducted on publicly available Sinhala NLP tools and research. According to the survey, the earliest attempt on Sinhala NLP was conducted by (Herath, et al., 2007). In this study the advanced machine translation capability problem was approached by examining the Sinhala natural language. A study based on some challenges and opportunities of using Sinhala language was done by (Nandasara & Mikami, 2016). At this point most of the studies in Sinhala NLP has been on Optical Character Recognition (OCR). The creation of the end-to-end Sinhala-to-English translator is the most important project on Sinhala NLP.

As a conclusion the most challenging part in Sinhala related in NLP tasks is collecting sufficient data (text corpus) because Sinhala text available in the internet is limited. Some available data cannot be extract because different web sites use different encodings and fonts. According to the above literature though there are huge number of NLP based implementations done for Sinhala language, they are in small scale compared with other high resource languages.

CHAPTER 3: THEORETICAL FRAMEWORK

3.1 Natural Language processing (NLP)

Human languages are unstructured. When considering 2 or more human languages it also does not have the same structure. Therefore, in order to make the computers to understand human languages the concept of NLP was introduced in 1950s by an English mathematician named Alan Turing (Figure 3.1). In 1950 he published an article named “Computing Machinery and Intelligence” (Turing, 1950). His experiment was known as “Turing test”, where he tried to test (whether it’s equal or not) the intelligence behaviour of machines with the human brain.



Figure 3-1 Alan Turing

NLP is a branch of Artificial Intelligence (AI) which help computers to interpret and understand the human language. The ability of reading speech, hearing speech and the interpretation of it are also provided by the computers with the use of NLP. Therefore, NLP can be considered as the science of studying about human languages using computer technology. Its ultimate goal is to develop computer models for humans.

3.2 Text corpus

A corpus can be considered as a large collection of structured set of texts. It may contain text data in a single language (monolingual corpus) or in multiple languages (multilingual corpus). A corpus can be used to perform statistical analysis (word frequencies), hypothesis testing or to validate linguistic rules in a language.

3.3 Text preprocessing

Text preprocessing can be defined as the process of cleaning text data before encoding. It is the first step to solve an NLP related problem is preprocessing data (text). Such as removing date formats, white space, stopwords and numbers or converting numbers into words. Commonly used preprocessing techniques are mentioned below.

3.3.1 Lower casing text data

The idea behind this technique is to convert all the input text into same casing format so that ‘word’, ‘Word’ and ‘WORD’ are considered in the same way. This technique is needed when dealing with word frequencies (when calculating TF-IDF vectors), hence same words will combine together, and duplicates will be removed and therefore correct frequency counts can be obtained.

3.3.2 Stemming and lematization

Stemming can be considered as reducing the word to their word stem, base or root. Therefore, the resultant word might be a non-existing word. The goal of lemmatization is to remove the inflections and map the word to its root form. This will always give a word that have some dictionary meaning. An example is given below

Troubled → Troubl (stemming)

Troubled → Troubled (lemmatization)

3.3.3 Tokenization

The goal of the tokenization process is to split the given text into smaller pieces named tokens. Tokens can be considered as words, numbers and punctuation marks.

3.3.3.1 WordPiece tokenization

WordPiece tokenization can be considered as breaking words into more than 1 sub-words. This creates a fixed size vocabulary of sub-words, individual characters and words that best fits the language data. This vocabulary consists with 4 things.

1. Whole words
2. Sub-words occurring in front of a word or in isolation
3. Sub-words not in front of a word (this is denoted by using ‘##’)
4. Individual characters

This tokenizer first checks whether the whole word is in the vocabulary. If not, it will divide the word into largest possible sub-words that are in the vocabulary, finally it will decompose the word into individual characters. Therefore, a word can represent as a collection of its individual characters. When an out of vocabulary (OOV) word is present it will be decomposed into sub-words and individual characters so that embeddings can be generated. Figure 3-2 shows an application of wordpiece tokenization for Sinhala language.

Eg: Possible sub-word tokens for the word ‘embeddings’ will be,

[‘em’, ‘##bed’, ‘##ding’, ‘##s’]

Source: (McCormick, 2020)

```
[10] tokenizer_bert.encode(" අනාවරනය කළේය. ").tokens
```

```
['අන', '##ාවර', '##නය', 'කළේය', '.']
```

```
[11] tokenizer_bert.encode(" මම ගෙදර යමි. ").tokens
```

```
['මම', 'ගෙ', '##ද', '##ර', 'ය', '##මි', '.']
```

Figure 3-2 Wordpiece tokenization for Sinhala language

3.4 Zipf's law

Zipf's law was introduced by an American linguist named George Kingsley Zipf. It is a statistical distribution of words in a linguistic corpus. Zipf's law states that given a text of corpus the frequencies of words are inversely proportional to their ranks. According to this law stop words will have a large portion of occurrences in a dataset. When the words are ranked according to their frequencies and when the frequency is plotted against the rank the resultant graph will be a logarithmic curve (when graph is plot on log scale it will result a straight line). This law can be applied for any natural language.

$$r.f = k$$

Where, r is the rank of the word, f is the frequency of the word and k is a constant

3.5 Language models

Language models can be considered as the backbone of NLP. It is a collection of statistical and probabilistic techniques which can be used to determine the probability of a given sequence of words occurring in a sentence by analyzing the text data. This can be used as a basis of a word prediction model. Language models can be used in speech recognition, machine translation and part of speech tagging.

3.5.1 N-gram models

n-gram model is a statistical language model which assigns probabilities to sentences and sequences of words given a sequence of n-1 words. It can be used to predict the next word in a sentence and also for speech recognition.

If the model relies on how often a word occurs without looking at previous words, it is called as a unigram model. If the model considers only the previous word to predict the current word, then it is called bigram model. If the model considers the two previous words, then it can be considered as trigram model, and so on.

Bigram probability is given by: $P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}$

3.5.2 LSTM (Long-Short-Term-Memory) model

LSTM is an artificial recurrent neural network (RNN) which was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber in mid 90s. LSTMs helps to maintain the error that can be propagated through time and layers which allows the recurrent nets to continue to learn over time steps. In a LSTM information is kept in a gated cell which can make decisions such as what to store, when to allow writes, reads and deletions. This is done by the analog gates which are implemented by element wise multiplication using sigmoids. The analog gates can be differentiable and hence it is suitable for backpropagation.

The behavior of a gate is similar to the nodes in a NN. When a signal is received by the gate they pass or block the relevant information based on its strength and import. This is done by filtering their own sets of weights. These weights are adjusted via the learning process of the recurrent network. Therefore, the cells will learn through an iterative process by making guesses, backpropagating error and also by adjusting weights via gradient descent. Figure 3-3 shows the flow of information in a cell and how the gates controlled it inside an LSTM. LSTMs can be used for speech recognition, handwriting recognition and also to make predictions based on time series data.

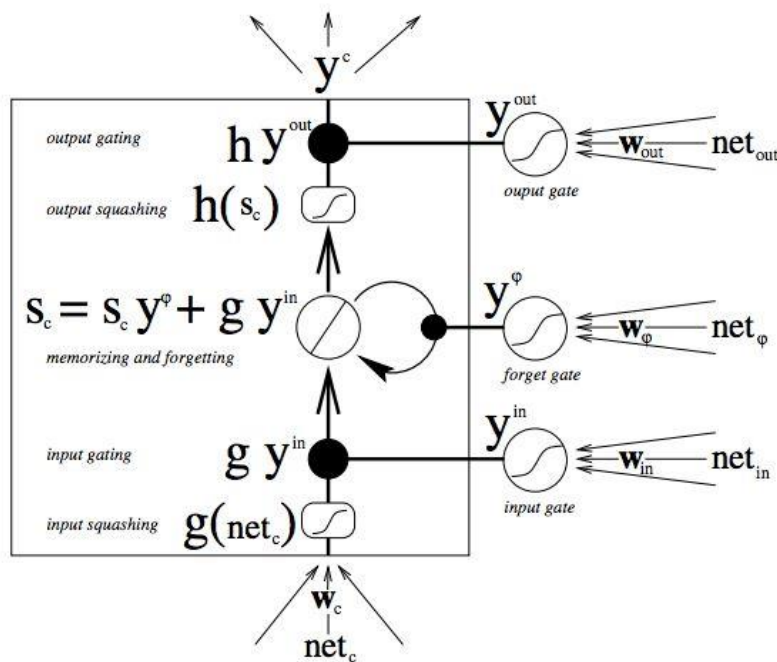


Figure 3-3 Flow of information in an LSTM model

3.6 Word embedding

Word embedding is a vector representation of a word, where words with similar meanings have a similar representation. The vector representation of a word can be considered as a one-hot encoded vector. (The position of the word where the word exists is represented as 1 and 0 otherwise). The word embedding can be divided into 2 parts as traditional word embedding and contextual word embedding.

3.6.1 Traditional word embedding

Traditional word embedding is done by ignoring the meaning of the word based on its context. The disadvantage of the traditional word embedding is that it ignores the contextual meaning and hence the original meaning of the sentence may be violated. For example, consider the following sentence,

පන්සලේ තිබූ පහන් දැක මගේ සිත පහන් විය.

In this sentence the first instance of the word ‘පහන්’ represent the lamp and the second instance represent a human feeling. In traditional word embedding only 1 representation is learnt for the word ‘පහන්’. Common methods used for traditional word embedding are Glove and word2vec (a statistical method).

3.6.2 Contextual word embedding

Through the careful consideration of the sequence of all the words in the specified documents, the sequence level semantics are learnt by it. Therefore, in the above example it will learn the word ‘පහන්’ based on its context. The most popular method for contextual word embedding is the use of the BERT model.

3.7 BERT model

BERT stands for Bidirectional Encoder Representations from Transformers. It is a machine learning algorithm created by Jacob Devlin and his colleagues from Google in 2018. As the name implies it is designed to pretrain deep bidirectional representations from unlabeled text data by considering both the left and right context in all layers. Therefore, the pretrained model can be finetuned by using one additional output layer. The BERT model can be used

create models based on NLP such as question answering, predic, abstract summarization and conversational response generation.

The original BERT model was created for English language and comes with 2 pre trained types namely BERT-base and BERT-large. Both models are trained on a book corpus with 800M words, and on the English Wikipedia with 2500M words. BERT model uses WordPiece tokenization strategy to tokenize the words. The BERT model is pretrained for Masked Language Modelling (MLM) and sentence prediction. Today Google has created the BERT model for 102 languages (mBERT) such as Arabic, Bengali, Chinese, Hindi and Tamil. So far Google has not provided this facility for Sinhala language.

3.7.1 Model architecture of BERT

The architecture of the BERT model is built on top of the Transformer encoder.

- BERT-base: 12 layers (transformer blocks), 12 attention heads, and 110M parameter neural network architecture.
- BERT-large: 24 layers (transformer blocks), 16 attention heads, 340M parameter neural network architecture.

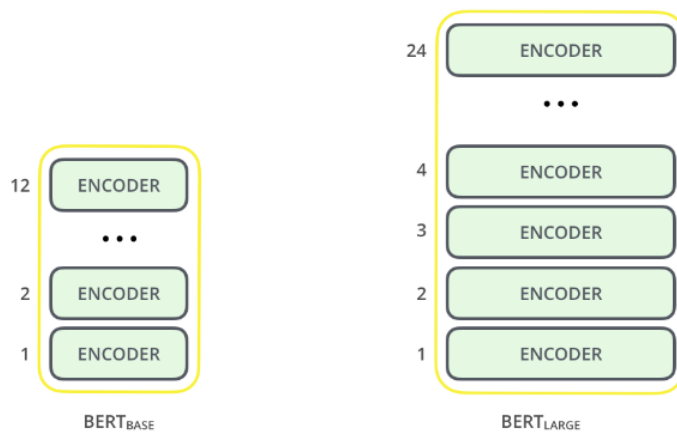


Figure 3-4 Architecture of the BERT model

3.7.2 MLM in pre training

Language modelling can be considered as predicting the next word in a given sequence of words. In MLM without predicting the next token a portion of input tokens are masked randomly and the masked tokens are predicted. These models tries to understand the relationship between words. In the BERT model MLM is implemented as follows.

First 15% of the tokens are chosen randomly so that the masked tokens can be seen before fine tuning. Then,

- 80% of the time tokens are actually replaced with the token [MASK].
- 10% of the time tokens are replaced with a random token.
- 10% of the time tokens are left unchanged.

3.7.3 Sentence prediction in pre training

This can be considered as a binary classification task. The data can be acquired from any corpus by splitting it into sentence pairs. BERT model performs this as follows,

- For 50% of the sentence pairs, the second sentence would actually be the next sentence to the first sentence. The label would be 'IsNext'
- For the remaining 50%, the second sentence is a random sentence from the corpus. The label would be 'NotNext'.

3.7.4 Finetuning of the BERT model

BERT has a huge neural network architecture which has around 100M to 300M parameters. Therefore, training the BERT model on a small dataset might result in overfitting. As a solution for this problem a BERT model which was trained on a huge dataset is used. The process of training the pretrained BERT model with a smaller dataset is known as model finetuning.

3.8 mBERT model

As mentioned earlier BERT model was built for English language. Therefore, a model which can accommodate many languages was introduced named as mBERT. This model was trained using 104 languages (see appendix A). All the data were collected from Wikipedia. Languages which had a less amount of data were oversampled and languages with large amount of data were under-sampled.

When training on many languages without using a separate vocabulary a shared vocabulary was used for all the languages. This will save space and the model will learn the root structure of language and the underlying structure.

3.9 RoBERTa model

RoBERTa model is an optimized BERT pre-training approach which was developed by researches at Facebook and Washington university. This model was trained using 5 datasets namely, book corpus, English Wikipedia dataset, CC news, open web text and stories. It is almost similar to the original BERT architecture. The changes done to improve the results of the BERT architecture are as follows,

- The next sentence prediction is removed: When it is removed it slightly improves the performance.
- Training was done with bigger batch sizes and longer sequences: The BERT model was trained for 1M steps with batch size of 256 sequences. The RoBERTa model was trained for 125 steps with 2K sequences and 31K steps with 8K sequences of batch size. When using large batches MLM objective and end task accuracy was improved.
- Masking pattern was changed dynamically: In the BERT model a single static mask was used. In the RoBERTa model training data was duplicated and masked 10 times, each time with different mask strategy. This was performed over 40 epochs. (4 epochs had the same mask). This method can be considered as a dynamic masking strategy.

3.10 Evaluating language models

3.10.1 Perplexity measure

Perplexity measures how a probability model or a probability distribution predicts a given test sample. A language model can be consider as a probability distribution of a set of sentences. Therefore, the most common method for evaluating a language model can be consider as measuring the perplexity value. Low perplexity value indicates a better model. Perplexity can be defined in various ways. Popular methods are mentioned below,

1. Inverse probability of the test set, normalized by the number of words

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1, w_2, \dots, w_n)}}$$

2. Using cross entropy: cross entropy is the average number of bits need to encode a word. Therefore, the number of words that can be encoded with those bits is known as perplexity.

$$PP(W) = 2^{H(W)} = 2^{-\frac{1}{N} \log_2 P(w_1, w_2, \dots, w_n)}$$

3. As a weighted branching factor

3.10.2 Accuracy measure

Accuracy measures the number of correct predictions made by the model. In a language model there are only 2 possible predictions, either correct prediction or wrong prediction. Therefore, the accuracy measure can be computed as follows.

$$Accuracy = \frac{\text{no of correct predictions}}{\text{total no of predictions}}$$

CHAPTER 4: METHODOLOGY

4.1 Data collection

Data for the study were collected by scraping Sinhala websites such as sinhala.news.lk, adaderana.lk and www.newsfirst.com. Open datasets available on Git Hub were also used (contained data extracted from Silumina paper, neth news, hiru news, blog data and film subtitles). The detailed description of the dataset is given in section 4.2.2.

4.2 Method description

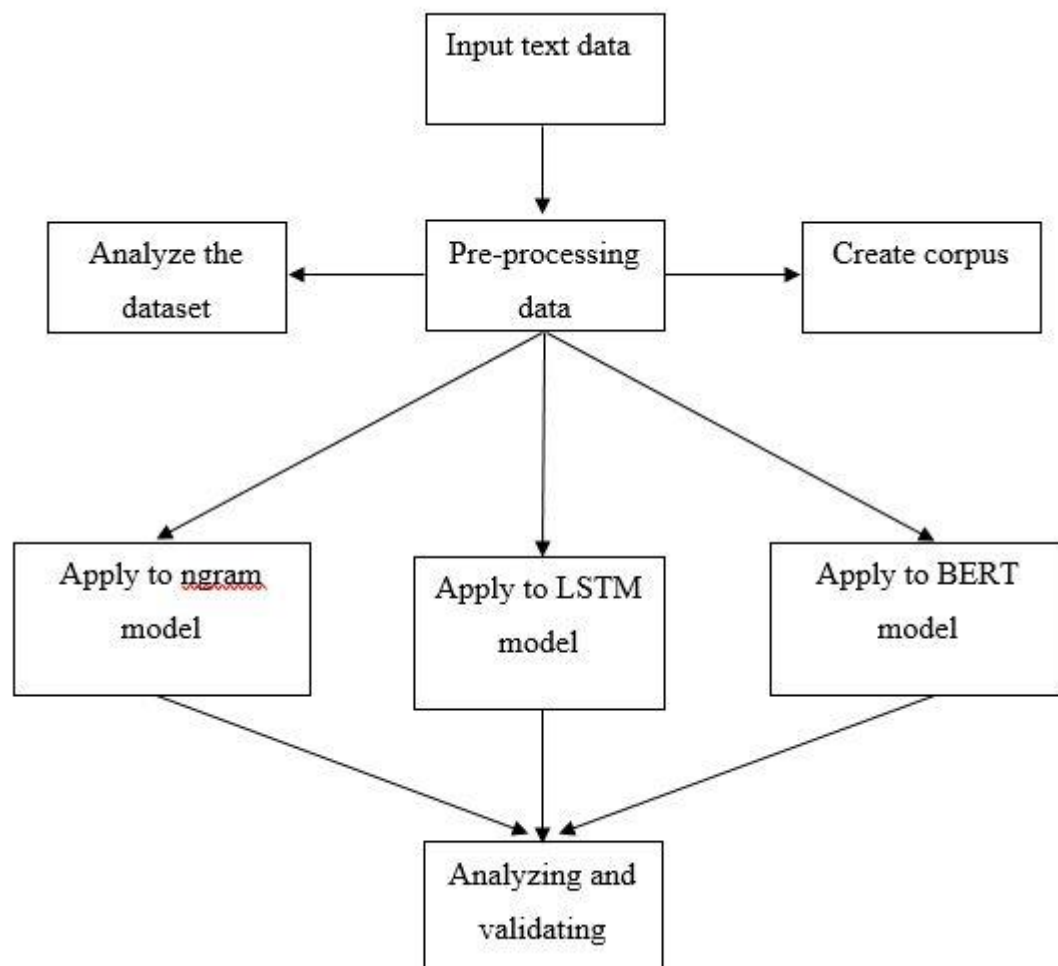


Figure 4-1 Method

A detailed description of Figure 4-1 is given below

4.2.1 Data pre-processing

The data collected for the study contained with noise. Therefore, to have clean dataset following steps were performed as preprocessing techniques

- Date formats: when observing the dataset, the news items contained with date formats. All the dates were in the form of YYYY/MM/DD or YY/MM/DD. Therefore, these were removed using regular expressions.
- There were non-Sinhala text (English and Tamil) in between Sinhala text. These were also removed using regular expressions.
- All Numbers were replaced using ‘num’ key word using regular expressions (this will include decimal numbers, percentage numbers, and numbers which contains commas in between them).
- Changing word boundaries of some words.
ගියා ය → ගියාය
අතර වූ → අතරවූ
- Finally stemming/ lemmatization was performed. Sinhala words were stemmed using the ‘Sinhala stem dictionary’ which was created by the Language Resource Technological Laboratory at University of Colombo School of Computing. This contains 27000+ Sinhala words with their stems.
අංකද → අංක
අංකය → අංක
අංකයක් → අංක
- Extra spaces were removed using regular expressions.
- Unrecognized characters were removed by using manual preprocessing techniques.

4.2.2 Creating the corpus

After performing the preprocessing steps all files were combined as a single text file by using ‘glob’ function. The data were tokenized, and only the Sinhala characters were considered when creating the corpus (punctuation marks were omitted). This was achieved by using the command ‘re.compile(u'^\u0D80-\u0DFF', re.UNICODE)’. Duplicate words were removed by using ‘list(set(tokens))’. Finally the words were sorted in the alphabetical order by using the command ‘sorted(tokens)’. A detailed description of the created corpus is given in Table 4-1.

Table 4-1 Text distribution according to the source

<i>Data source</i>	<i>No of words</i>	<i>No of unique words</i>	<i>No of sentences</i>
News paper	7,607,714	377,616	442,781
Blog data	463,277	67,595	37,862
Wikipedia	1,107,508	108,484	73,196
Common Crawl	108,569,473	3,907,507	7,542,008

4.2.3 Analysing the pre-processed dataset

To find the Zipf's law behavior of the dataset the dataset was tokenized as mentioned in section 4.2.2. Then the count of each word was calculated using `'[(w, tokens.count(w)) for w in tokens]'` command. The word list was then sorted with respect to their frequency using the command, `'sorted(counts,key=lambda l:l[1], reverse=True)'` and then A list was created to store the rank. As the next step the log values of the rank and the respective frequency was calculated using 'math' library (the first value of both lists were dropped as it contained the frequency count of the empty spaces). Finally `log(frequency)` against `log(rank)` was plotted. This was achieved as follows,

```
r = [math.log(x) for x in rank]
f = [math.log(x) for x in frq]
r1 = r[1:-1]
f1 = f[1:-1]
plt.plot(f1,r1)
plt.title("Zipf's Law Behavior")
plt.xlabel("log(rank) ")
plt.ylabel("log(frequency) ")
```

The most frequent unigram, bigram and trigram in the dataset was found using the n-gram model. This was done by using the ngrams in nltk library.

4.2.4 Creating the n-gram model

The n-gram model was created using the CMU toolkit. A description of the toolkit and the steps followed are given below.

4.2.4.1 CMU toolkit

The CMU toolkit is a statistical language modelling toolkit which was released in 1994. It is a set of Unix software tools which can be used for language modelling work. (Clarkson & Rosenfeld, 2000). The usage of the toolkit is illustrated in Figure 4.2.

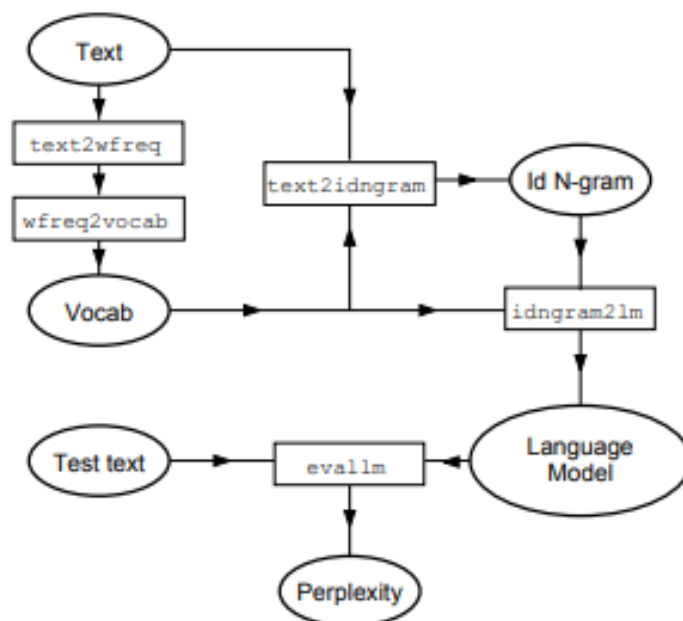


Figure 4-2 Usage of the toolkit

4.2.4.2 Building the n-gram model

As the first step the vocabulary file will be created. This is done by using the following tools.

- `Text2wfreq`: This computes the number of occurrences for each word in the given text.
- `Wfreq2vocab`: This will turn the list into a vocab file (the file will contain the most common 20000 words).

The next step is to construct the language model. First the training set is converted to a list of id n-grams. This was done using the following code.

```
!cat Train.txt | text2idngram -n 2 -vocab Train.vocab -buffer 100 -  
temp /tmp > Train.id2gram
```

The -n 2 indicates the 2-gram model. The amount of data to grab (in MB) by the program is specified by the -buffer option. Next the id n-gram will be converted into a binary language model file. This is done as follows.

```
!idngram2lm -idngram Train.id2gram -vocab Train.vocab -n 4 -  
binary Train.2gram.binlm -cutoffs 2 -witten_bell
```

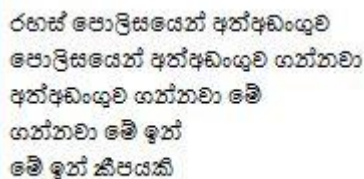
The cutoff value of 2-gram is set to 2, hence the counts less than 2 will be discarded. The Witten-Bell discounting is also specified. Finally, the perplexity of the test dataset is calculated using the eval_lm tool.

4.2.5 Creating the LSTM model

The steps for creating the LSTM model are mentioned below

4.2.5.1 Creating the dataset

The preprocessed dataset was loaded and was tokenized. Then the list of tokens were arranged into sequences and it was saved to a text files. In this study several models were created based on the sequence length. (len_2 is based on 2 previous words: Figure 4-3, len_4 is based on 4 previous words and so on)



රහස් පොලිසියෙන් අත්අඩංගුව
පොලිසියෙන් අත්අඩංගුව ගන්නවා
අත්අඩංගුව ගන්නවා මේ
ගන්නවා මේ ඉන්
මේ ඉන් කීපයකි

Figure 4-3 Part of the dataset of len_2

4.2.5.2 Encoding the created sequences

When feeding the sequences to the embedding layer it should be fed in the form of integer values. Therefore, each word was mapped to a unique integer value. This was done using the Tokenizer class available in Keras. The tokenizer was trained on the training dataset. Then by using the fit Tokenizer the training sequences were encoded to a list of integers.

4.2.5.3 Creating input sequences and output elements

The input(x) and output(y) were created by using array slicing. The output should be converted to an integer to a vector of 0 values, one for each word in the vocabulary with a 1 to indicate the specific word at the index of the words integer value (one hot encoding). This can be achieved by using `to_categorical()` function available in Keras library.

4.2.5.4 Building the model

LSTM recurrent network was used as the model. The model was created using two LSTM hidden layers with 100 memory cells. The size of the embedding vector space was set to 50. In interpreting the features extracted from the sequence, a dense fully connected layer consisting of 100 neurons which is connected to the LSTM hidden layer is used. The next word is predicted by the output layer with the help of a single vector which has the size of the vocabulary. The probability of the output vector is governed by the words in the vocabulary. To make sure that the outputs have the features of normalized probabilities, a softmax activation function is used.

Model was compiled using categorical cross entropy and the optimizer used was Adam. The model was trained for 50 epochs with the batch size of 64. Finally, accuracy and perplexity measures were obtained. A summary of the created model is showed in Figure <>.

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 2, 50)	977000
lstm (LSTM)	(None, 2, 100)	60400
lstm_1 (LSTM)	(None, 100)	80400
dense (Dense)	(None, 100)	10100
dense_1 (Dense)	(None, 19540)	1973540
Total params: 3,101,440		
Trainable params: 3,101,440		
Non-trainable params: 0		

Figure 4-4 Summary of the created model for 2_len

4.2.6 Creating the BERT model

The steps for creating the BERT model are described below

4.2.6.1 Training the tokenizer

The tokenizer was initialized using the ByteLevelBPETokenizer function available at huggingface 'tokenizers' library. It was trained on the entire training data set using some special tokens,

- [UNK]: This represents unknown tokens which is not in the vocabulary and cannot convert into an ID, hence it is set to this token.
- [S]: This represents the start token.
- [PAD]: This is used for padding.
- [/S]: This is the end token.
- [MASK]: The token used for masking values. Used in MLM.

The vocab_size was set to 52000 and the min_frequency was 2. Results obtained from the tokenizer are showed in the Figure 4-2


```
e = tokenizer.encode("අද මම යනවා.")
print('Encoded string:', (e.tokens))

d = tokenizer.decode(e.ids)
print('Decoded string:', d)

Encoded string: ['àṽḥàṽ', 'Ḡàṽ.àṽ.', 'Ḡàṽḡàṽ±à·Ḡ', 'à·1.']
Decoded string: අද මම යනවා.
```

Figure 4-5 Results from the ByteLevelBPETokenizer

The created tokenizer will generate 2 files namely,

1. Merges.txt: which contains the merged tokenized sub strings.
2. Vocab.json: which contains the indices of the tokenized sub strings.

4.2.6.2 Building the model

Two separate models were created. The configuration of the 1st model will be based on the DistilBERT transformer structure. Therefore, the model will have 12 attention heads and 6 layers. The 2nd model configuration was based on the BERTbase structure. Hence it will have 12 attention heads and 12 layers. In both models the vocabulary size was set to 52000. The model configuration can be initialized by using the ‘RobertaConfig’ function available at transformers library. Then the created configuration will be initialized for language modelling.

4.2.6.3 Creating the training dataset

The corpus size of the training set was 5.8M words. The training dataset was tokenized with the tokenizer created and it was feed in the form of line by line for batch training with the block_size of 64.

4.2.6.4 Data collator

The next step will be defining the data collator. It will take samples from the dataset and collate them into batches. The batched samples will be initialized for MLM by setting `mlm = True`. The `mlm_probability` value was set to 0.15. This value will determine the percentage of tokens masked during the pretraining process.

4.2.6.5 Initializing the trainer and pretraining the model

The trainer was initialized using the previously mentioned information (the model configuration, dataset and the data collator) along with the training arguments (number of training epochs and the batch size). Finally the model was trained using the `trainer.train()` function.

4.2.6.6 Testing the model

The model can be tested by using the `FillMaskPipeline` with the use of the pretrained model and the trained tokenizer. This will return the top 5 predictions with their probability values. The accuracy values and perplexity values were obtained.

CHAPTER 5: EVALUATION AND RESULTS

5.1 Zipf's law behaviour

The highest 10 words in the sorted list are given in the Table 5-1

Table 5-1 Highest frequent words

<i>Rank</i>	<i>Word</i>	<i>Frequency</i>
1	මේ	72077
2	බව	64961
3	ඒ	59433
4	ඇති	56072
5	සහ	48677
6	කර	44340
7	අතර	44021
8	වන	41400
9	හා	37261
10	කළ	34678

The graph of the Zipf's law behaviour is shown in Figure 5-1

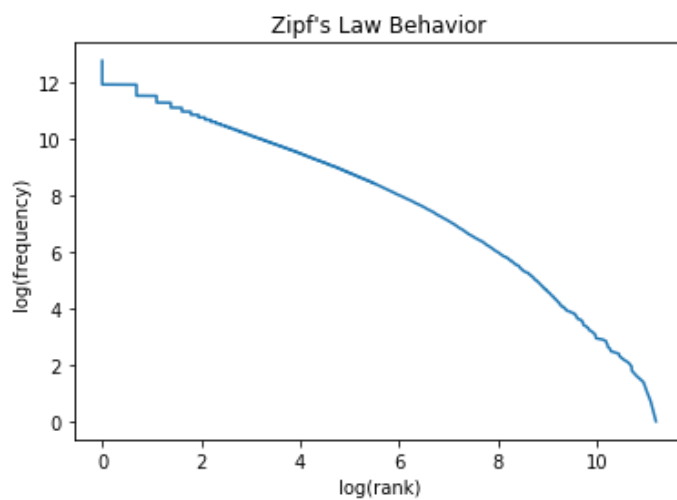


Figure 5-1 Zipf's law behaviour of the dataset

From Table 5-1 it can be concluded that the highest most frequent words represent the stopwords of the Sinhala language. According to the above graph (Figure 5-1) it can be observed that the line is approximately linear on log-log plot. Therefore, it can be concluded that the collected dataset follows the Zipf's law behaviour.

5.2 Frequent words in the dataset

The frequent bigrams and trigrams were compute using the ngram model. The results are showed in Figure 5-2 and Figure 5-3.

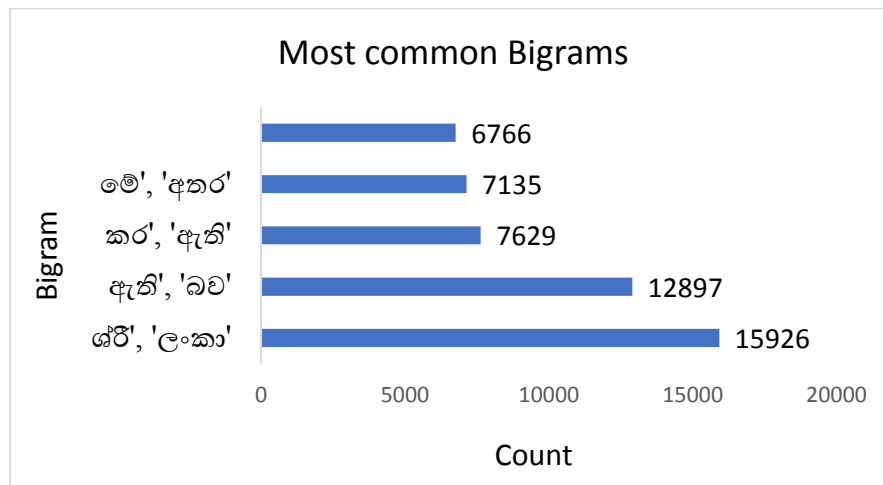


Figure 5-2 Most common bigrams

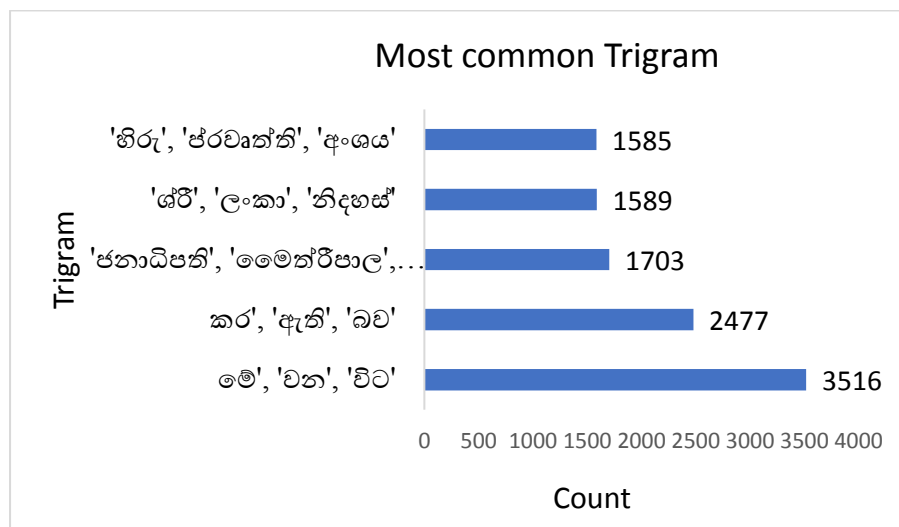


Figure 5-3 Most common trigrams

5.3 Results from n-gram model

Four n-gram models were created using the CMU toolkit (for $n = 2, 3, 4$ and 5). The models were trained using the training set and the perplexity values were obtained using the test set. In all 4 model the cut off value was set to 2. The perplexity values obtained are given in Table 5-2.

Table 5-2 Perplexity values of n-gram models

<i>Model</i>	<i>Perplexity</i>
2-gram	692.62
3-gram	502.34
4-gram	683.92
5-gram	686.13

According to the results in Table 5-2 the trigram model has a better performance with a perplexity value of 502.34. All the other models have a perplexity value in the range of 680-690. Among the models created the bigram model has a poor performance (perplexity 692.62).

According to the theory of n-gram model, when the 'n' value increases the models should perform better (perplexity values should be decreased when the value of 'n' increases). In general, a language has long distance dependencies and also when 'n' increases model does not contain sufficient data. Therefore, the above mentioned points might be a possible reason for high perplexity values obtained for 4-gram and 5-gram models.

5.4 Results from LSTM model

The perplexity values (showed in Table 5-3) for the created LSTM models were calculated using the cross entropy method.

Table 5-3 Perplexity values of the LSTM models

<i>Sequence length</i>	<i>Perplexity value</i>
Len_2	55.81
Len_4	44.53
Len_6	48.66
Len_8	80.56

From Table 5-3 it can be concluded that the LSTM model performs better when the sequence length is 4. The perplexity values increase after the sequential length is 4. This behaviour is very similar to the results obtained from the n-gram models. (In the n-gram model when is ‘n’ is greater than 3 the model starts to deviate from the standard theory)

5.5 Results from the BERT model

Two variations of BERT models (distilBERT: 6 layers and 12 attention heads, BERTbase: 12 layers and 12 attention heads) were created. The training corpus size was 6.4M and the testing corpus size was 2M. Both models were trained for 40 epochs. Finally the perplexity values and accuracy values for word prediction were calculated. The calculated perplexity values are given in Table 5-4.

Table 5-4 Perplexity values of the created BERT models

<i>No of epochs</i>	<i>Perplexity value: distilBERT</i>	<i>Perplexity value: BERT base</i>
10	14.5	15.71
20	3.45	16.45
30	3.19	20.85
40	8.72	23.21

The comparison of the perplexity values of the 2 models are showed in Figure 5-4.

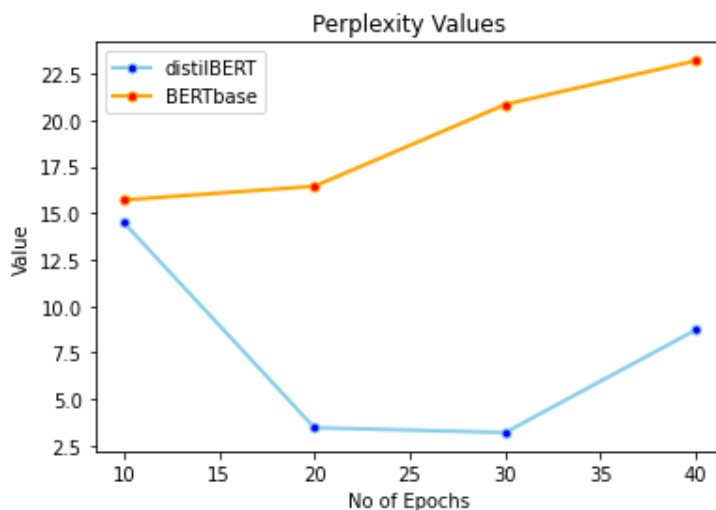


Figure 5-4 Comparison of the perplexity values in BERT model

According to figure 5-4 it can be concluded that the perplexity values for BERTbase model is higher than distilBERT model. When analysing the results of BERTbase model it can be seen that the perplexity values increases with the no of epochs (15.71 to 23.21). In the variation line of distilBERT the lowest perplexity value can be found when the model reached 30 epochs (value: 3.19). According to the theory of perplexity it is said that when the perplexity value is low the language model is better. Therefore, it can be concluded that distilBERT model has a better performance compared to BERTbase model.

CHAPTER 6: CONCLUSION AND FUTURE WORK

This study was based on word prediction in Sinhala language using word embedding methods. To achieve the goal three models were used namely ngram model. LSTM model and RoBERTa model. Sinhala is a low resource language therefore, finding a reasonable size of text corpora was a challenge. Most of the text related preprocessing techniques are for English language hence some of the preprocessing steps were done manually which was very time consuming.

When considering the results from n-gram model and LSTM model it can be concluded that when the value of 'n' increases the model deviates from the standard theory. Possible reason for this is lack of examples in the dataset when n is high. According to the results obtained, among the n-gram models created the tri-gram model obtained the lowest perplexity value of 502.34. From the LSTM models created the LSTM model which considered 4 previous words for the prediction had a better performance (perplexity: 44.53) compared to the other LSTM models created.

When training the RoBERTa model the huge challenge was feeding the data to the model because of memory capacity available in Google Colab. Therefore, the training data was divided into sub parts and was fed to the model. The training time for 1 epoch was around 3-4 hours. From the results it was found that the distilBERT model is more accurate when compared to BERTbase model. Further it can be concluded that the RoBERTa model have good performance compared to the n-gram model and LSTM model.

In a language there can be many meaningful words that can be used for a given mask and hence calculating accuracy is not a better idea. Various methods can be used to improve this study such as by feeding more data to the model, by performing more preprocessing techniques (removing stop words of the Sinhala language, spell checking the words).

The methodology presented in this research can be naturally extended for other languages also. Further this research can be extended to generate sentences in Sinhala language, question answering and also to predict the next sentence in a given phrase.

APPENDIX

Appendix A: languages trained on mBERT

- Afrikaans
- Albanian
- Arabic
- Aragonese
- Armenian
- Asturian
- Azerbaijani
- Bashkir
- -Basque
- Bavarian
- Belarusian
- Bengali
- Bishnupriya Manipuri
- Bosnian
- Breton
- Bulgarian
- Burmese
- Catalan
- Cebuano
- Chechen
- Chinese (Simplified)
- Chinese (Traditional)
- Chuvash
- Croatian
- Czech
- Danish
- Dutch
- English
- Estonian
- Finnish
- French
- Galician
- Georgian
- German
- Greek
- Gujarati
- Haitian
- Hebrew
- Hindi
- Hungarian
- Icelandic
- Ido
- Indonesian
- Irish
- Italian
- Japanese
- Javanese
- Kannada
- Kazakh
- Kirghiz
- Korean
- Latin
- Latvian
- Lithuanian
- Lombard
- Low Saxon
- Luxembourgish
- Macedonian
- Malagasy
- Malay
- Malayalam
- Marathi
- Minangkabau
- Mongolian

- Nepali
- Newar
- Norwegian (Bokmal)
- Norwegian (Nynorsk)
- Occitan
- Persian (Farsi)
- Piedmontese
- Polish
- Portuguese
- Punjabi
- Romanian
- Russian
- Scots
- Serbian
- Serbo-Croatian
- Sicilian
- Slovak
- Slovenian
- South Azerbaijani
- Spanish
- Sundanese
- Swahili
- Swedish
- Tagalog
- Tajik
- Tamil
- Tatar
- Telugu
- Thai
- Turkish
- Ukrainian
- Urdu
- Uzbek
- Vietnamese
- Volapük
- Waray-Waray
- Welsh
- West Frisian
- Western Punjabi
- Yoruba

Appendix B: Codes

Code for pre-processing

```
import re

def preprocess(infile, outfile):
    """ This will preprocess the given text """
    reg1 = '[0-9]{4}.*?[0-9]{2}.*?[0-9]{2}.*?[0-9]{2}.*?[0-9]{2}.*?[0-9]{2}.*?'
    reg2 = re.compile(u'^\u0D80-\u0DFF"num"', re.UNICODE)

    f1 = open(infile, 'r', encoding = 'utf-8')
```

```

txt = f1.read()

txt = re.sub(reg1, '', txt)          # Removing date formats
txt = re.sub('[A-Za-z]', '', txt)   # Removing English characters
txt = re.sub('\d+', 'num', txt)     # Replacing digits
sentences = txt.split('.')          # Splitting data to obtain sentences


f2 = open(outfile, 'w', encoding = 'utf-8')
for sentence in sentences:
    words = sentence.split()

    word = [reg2.sub('', w) for w in words] # Extracting Sinhala
characters

    sentence = ' '.join(word)

    f2.write(sentence)

    f2.write('\n')

f2.close()

f1.close()

```

Code for n-gram calculation

```

import nltk, re, string, collections

from nltk.util import ngrams


def ngram(infile):
    """ This will find the most common uni, bi and tri gram in the
    dataset """

    f = open(infile, 'r', encoding = 'utf-8')
    data = f.read()
    f.close()

    tokens = nltk.word_tokenize(data) # Extract only the sinhala words
    regex = re.compile(u'^\u0D80-\u0DFF', re.UNICODE)
    tokens = [regex.sub('', w) for w in tokens]

```

```

bi = ngrams(tokens, 2) # Bi gram
bi_freq = collections.Counter(bi)
tri = ngrams(tokens, 3) # Tri gram
tri_freq = collections.Counter(tri)

print('Most common bigrams') # Top 10 bi grams
print(bi_freq.most_common(10))
print('\n')
print('Most common trigrams') # Top 10 tri grams
print(tri_freq.most_common(10))

```

Code for calculating perplexity in n-gram model: Using toolkit

```

# CMU toolkit should be downloaded using http://www.speech.cs.cmu.edu/SLM/toolkit\_documentation.html
# Creating the vocab
!cat Train.txt | text2wfreq | wfreq2vocab -top 20000 > Train.vocab

# Building the model
!cat Train.txt | text2idngram -n 2 -vocab Train.vocab -buffer 100 -
temp /tmp | \
    idngram2lm -idngram Train.id2gram -vocab Train.vocab -n 2 \
    -binary Train2gram.binlm -cutoffs 2 -witten_bell

# Computing the perplexity
!echo "perplexity -text Test.txt" | evalm -binary Train.2gram.binlm

```

Code for Zipf's law

```

import nltk
import re
import matplotlib.pyplot as plt
import math

```

```

f = open('DATA_FINAL.txt','r',encoding = 'utf-8')
content = f.read()
f.close()
tokens = nltk.word_tokenize(content)

# get only sinhala unicode charactors
regex = re.compile(u'[^\\u0D80-\\u0DFF]', re.UNICODE)
tokens = [regex.sub('', w) for w in tokens]

# Obtaining the frequency and sorting
frequency = {}
for word in tokens:
    count = frequency.get(word,0)
    frequency[word] = count + 1

frequency = {key:value for key,value in frequency.items()}
frequency = sorted(frequency.items(), key=lambda x: x[1],
reverse=True)

# Printing the most frequent words in the dataset
top_10 = list(frequency)[1:11]
print(top_10)

# Plotting
frq = [c for (w,c) in frequency][1:-1]
rank = list(range(1,len(frq)+1))
r = [math.log(x) for x in rank]
f = [math.log(x) for x in frq]
plt.plot(f,r)
plt.title("Zipf's Law Behavior")
plt.xlabel("log(rank)")

```

```
plt.ylabel("log(frequency)")
```

Code for LSTM model

```
# Reading data
f = open('gdrive/MyDrive/train.txt', 'r', encoding = 'utf-8-sig')
sents = f.readlines()
f.close()

data = ""
for i in sents:
    data = ' '.join(sents)
data = data.replace('\n', ' ')

# Getting tokens
tokens = data.split()
number_of_unique_tokens = len(set(tokens))

sequence_length = 2

# organize into sequences of tokens of input words plus one output word
length = sequence_length + 1
sequences = list()
for i in range(length, len(tokens)):
    seq = tokens[i-length:i]
    line = ' '.join(seq)
    sequences.append(line)

# saving the tokens to file
def save_doc(lines, filename):
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```

```

out_file = 'gdrive/MyDrive/lstm/data2_sequences.txt'
save_doc(sequences, out_file)

# load the file
def load_doc(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text

in_file = 'gdrive/MyDrive/lstm/data8_sequences.txt'
doc = load_doc(in_file)
lines = doc.split('\n')

# Perplexity calculation
from tensorflow.keras import backend as K

def perplexity(y_true, y_pred):
    cross_entropy = K.categorical_crossentropy(y_true, y_pred)
    return K.pow(2.0, cross_entropy)

# integer encode sequences of words
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
sequences = tokenizer.texts_to_sequences(lines)

# vocabulary size
vocab_size = len(tokenizer.word_index) + 1

import numpy as np
from tensorflow.keras.utils import to_categorical
# separating input and output
sequences0 = np.array(sequences)

```

```

y = []
X = []
for l in sequences0:
    f = l[-1]
    X.append(l[:1])
    y.append(l[-1])
y = to_categorical(y, num_classes=vocab_size)
seq_length = np.asarray(X).shape[0]

X = np.array(X)
y = np.array(y)

from pickle import dump
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout

model = Sequential()
model.add(Embedding(vocab_size, 50, input_length=sequence_length))
# LSTM hidden layers with 100 memory cells each.
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100))
model.add(Dense(100, activation='relu'))
model.add(Dense(vocab_size, activation='softmax'))
print(model.summary())

model.compile(loss='categorical_crossentropy', optimizer='adam', met
rics=['accuracy',perplexity])
history = model.fit(X, y, validation_split=0.3, batch_size=64, epoch
s=50).history

```


Code for BERT tokenizer

```
# Installing the libraries
!pip install git+https://github.com/huggingface/transformers
!pip install tokenizers

from tokenizers import ByteLevelBPETokenizer
# Initialize a tokenizer
tokenizer = ByteLevelBPETokenizer()

paths = ['gdrive/MyDrive/Train/Train.txt']

# Customize training
tokenizer.train(files=paths, vocab_size=52_000, min_frequency=2,
               special_tokens=[
                   "<s>",
                   "<pad>",
                   "</s>",
                   "[UNK]",
                   "<mask>",
               ])

# Saving the tokenizer to the directory
tokenizer.save_model("gdrive/MyDrive")
```

Code for BERT model

```
#Installing the libraries
!pip install git+https://github.com/huggingface/transformers
!pip install tokenizers

from tokenizers.implementations import ByteLevelBPETokenizer

# Loading the created Tokenizer
tokenizer = ByteLevelBPETokenizer(
    "gdrive/MyDrive/dM1/vocab.json",
```

```

        "gdrive/MyDrive/dM1/merges.txt",
    )

    #Defining the model configuration
    from transformers import RobertaConfig
    config = RobertaConfig(
        vocab_size=52_000,
        max_position_embeddings=514,
        num_attention_heads=12,
        num_hidden_layers=6,
        type_vocab_size=1
    )

    #Reloading the tokenizer and initializing the model
    from transformers import LineByLineTextDataset
    from transformers import RobertaTokenizerFast

    tokenizer = RobertaTokenizerFast.from_pretrained("gdrive/MyDrive/dM1
    ", max_len=512)

    from transformers import RobertaForMaskedLM
    model = RobertaForMaskedLM(config=config)

    #Building the dataset
    dataset = LineByLineTextDataset(
        tokenizer = tokenizer,
        file_path = 'gdrive/MyDrive/Train/2.txt',
        block_size=128,
    )

    #Data collator
    from transformers import DataCollatorForLanguageModeling
    data_collator=DataCollatorForLanguageModeling(tokenizer=tokenizer, m
    lm=True, mlm_probability=0.15)

    #Initializing the trainer

```

```

from transformers import Trainer, TrainingArguments
training_args = TrainingArguments(output_dir='gdrive/MyDrive/dM1', o
verwrite_output_dir=True,

                                num_train_epochs=3,
                                per_device_train_batch_size=32,
                                save_steps=10_000,
                                save_total_limit=2,

)

trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=dataset,
)

#Training and saving the model
trainer.train()
trainer.save_model('gdrive/MyDrive/dM1')

#Language modelling
from transformers import pipeline

fill_mask=pipeline(
    'fill-mask',
    model='gdrive/MyDrive/M1',
    tokenizer='gdrive/MyDrive/M1'
)

fill_mask('<mask> ඉගෙනීමට ගියා')

```

REFERENCES

- Almeida , F. & Xexeo, G., 2019. *Word Embeddings: A Survey*. s.l., s.n.
- Alsentzer, E. et al., 2019. Publicly Available Clinical BERT Embeddings. *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pp. 72-78.
- Aurnhammer, Christoph & Frank, S. L., 2019. Evaluating information-theoretic measures of word prediction in naturalistic sentence reading. *Neuropsychologia*, Volume 134.
- Babu, J. & S, T., 2020. Finding the Duplicate Questions in Stack Overflow using Word Embeddings. *Procedia Computer Science*, pp. 2729-2733.
- Beltagy, I., Lo , K. & Cohan, A., 2019. *SciBERT: A Pretrained Language Model for Scientific Text*. s.l., s.n.
- Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T., 2017. *Enriching word vectors with subword information*. s.l., s.n.
- Canete, J. et al., 2020. *SPANISH PRE-TRAINED BERT MODEL*. s.l., s.n.
- Clarkson, P. & Rosenfeld, R., 2000. *Statistical Language Modelling Using the CMU-Cambridge Toolkit*. s.l., s.n.
- Cui, Y. et al., n.d. *Pre-Training with Whole Word Masking for Chinese BERT*. s.l., s.n.
- de Silva, N., 2019. *Survey on Publicly Available Sinhala Natural Language Processing Tools and Research*. s.l., s.n.
- de Vries, W., Cranenburgh, A. v., Bisazza, A. & Caselli, T., 2019. *BERTje: A Dutch BERT Model*. s.l., s.n.
- Ethayarajh & Kawin, 2020. *How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings*. s.l., s.n.
- Gallege, S., n.d. *Analysis of Sinhala Using Natural Language Processing Techniques*, s.l.: s.n.
- Herath, S. et al., 2007. Machine Processing Of Sinhala Natural Language: A Step Toward Intelligent Systems. *Cybernetics and Systems*, pp. 331-348.

- Jayakody, T., Gamlath, T. S. K., Lasantha, W. A. N. & Premachandra, K. M. K. P., 2015. "Mahoshadha ", *The Sinhala Tagged Corpus based Question Answering System*. s.l., s.n.
- Karunaratne, M. S., Nanayakkara, L. D. J. F. & Ponnampereuma, K., 2013. Sentence Prediction on SMS in Sinhala Language. *International Journal of Scientific and Research Publications*, 3(12), pp. 392-398.
- Kenter, T., Alexey, B. & Maarten, d. R., 2016. *Siamese CBOW: Optimizing Word Embeddings for Sentence Representations*. s.l., s.n.
- Lakmal, D., Ranathunga, S., Peramuna, S. & Herath, I., 2020. Word Embedding Evaluation for Sinhala. *Proceedings of the 12th Language Resources and Evaluation Conference*.
- Liu, Y., Ott, M., Goyal, N. & Du, J., 2019. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. s.l., s.n.
- Ma, B. et al., 2019. *Learning chinese word embeddings from character structural information*, s.l.: s.n.
- McCormick, C., 2020. *BERT word embeddings tutorial*. [Online]
Available at: <https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/>
- Mikolov, T., Chen, K., Corrado, G. & Dean, J., 2013. *Efficient estimation of word representations in vector space*. s.l., s.n.
- Minih, A. & Kavukcuoglu, K., 2013. *Learning word embeddings efficiently with noise-contrastive estimation*. s.l., s.n.
- Nandasara, S. T. & Mikami, Y., 2016. Bridging the digital divide in Sri Lanka: some challenges and opportunities in using Sinhala in ICT. *Advances in ICT for Emerging Regions (ICTer)*, pp. 1-13.
- Pennington, J., Socher, R. & Manning, C. D., 2014. Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, p. 1532–1543.
- Poligano, M., Basile, P., de Gemmis, M. & Semeraro, G., 2019. *ALBERTO: Italian BERT Language Understanding Model for NLP Challenging Tasks Based on Tweets*. s.l., s.n.

Tien, H. N., Le, M. N., Tomohiro, Y. & Tatsuya, I., 2018. *Sentence modeling via multiple word embeddings and multi-level comparison for semantic textual similarity*. s.l., s.n.

Turing, A. M., 1950. *Computing Machinery and Intelligence*. s.l., s.n.

Wang, Z., K, K., Mayhew, S. & Roth, D., 2020. Extending Multilingual BERT to Low-Resource Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, p. 2649–2656.