SCHOOL OF GRADUATE STUDIES

DEVELOPING SEMANTIC TEXTUAL SIMILARITY FOR GURAGIGNA LANGUAGE USING DEEP LEARNING APPROACH

MSc. THESIS

GETNET DEGEMU

May 28, 2024

WOLKITE, ETHIOPIA

Wolkite University

School of Graduate studies

Developing Semantic Textual Similarity for Guragigna Language using Deep Learning Approach

A Msc Thesis Submitted to School of Graduate Studies in Partial Fulfillment Requirement for the Degree of Master of computer Science in Computer and Engineering

Getnet Degemu Besir

Major Advisor: Sintayehu Hirpassa (Ph.D.)

Co-Advisor: Abdo Ababor (Msc)

May 28, 2024

Wolkite, Ethiopia

APPROVAL SHEET

Wolkite University

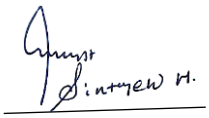
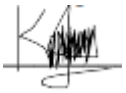
School of Graduate studies

As Thesis advisor, I hereby certify that all given comments by reviewers have considered, read, and evaluated the Thesis entitled Developing Semantic textual similarity for Guragigna language using Deep learning Approach.

Submitted by

Getnet Degemu	_____	5/28/2024
Name	Signature	Date

Approved by

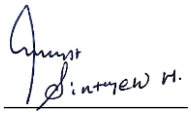
Sintayehu Hirpassa (Ph.D.)		25/05/2024
Name of Major Advisor	Signature	Date
Abdo Ababor (Msc)	_____	_____
Name of CO-Advisor	Signature	Date
Kindie Biredagn (Ph.D.)		25/05/2024
Name of extremal examiner	Signature	Date
Worku Muluye	_____	_____
Name of Internal examiner	Signature	Date
_____	_____	_____
Name of DGC chairman	Signature	Date
_____	_____	_____
Name of CPG coordinator	Signature	Date
_____	_____	_____
SGS Approval	Signature	Date

APPROVAL SHEET

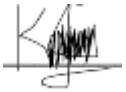
WOLKITE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby certify that we have read and evaluated this Thesis titled **"Developing Semantic textual similarity for Guragigna language using Deep learning approach"** prepared under our guidance by **Getnet Degemu Besir**. We recommend that the Thesis shall be submitted as fulfilling the requirements for the award of a MSc. degree in computer Science and Engineering.

Sintayehu Hirpassa (Ph.D.)		25/05/2024
Major Advisor	Signature	Date
Abdo Ababor (Msc)	_____	_____
Co-Advisor	Signature	Date

As members of the Board of Examiners of the Master of Science Thesis open defense examination, we have read and evaluated this Thesis prepared by **Getnet Degemu Besir** and examined the candidate. We hereby certify that, the thesis is accepted for fulfilling the requirements for the award of the degree of Master of Science (M.Sc.) in computer Science and Engineering.

Kindie Biredagn (Ph.D.)		25/05/2024
Name of extremal examiner	Signature	Date
Worku Muluye	_____	_____
Name of Internal examiner	Signature	Date
_____	_____	_____
Name of chainman	Signature	Date

Final approval and acceptance of the Thesis is contingent upon the submission of its final copy to the Council of Postgraduate Program (CPCS), through the candidate's department or school graduate committee (DGC or SGC).

DEDICATION

I dedicate the present work to my daughter, Ayana Getnet **(ኤያና)**

DECLARATION

By my signature below, I declare and affirm that this Thesis/Dissertation is my own work. I have followed all ethical principles of scholarship in the preparation, data collection, data analysis and completion of this thesis. All scholarly matter that is included in the thesis has been given recognition through citation. I affirm that I have cited and referenced all sources used in this document. Every serious effort has been made to avoid any plagiarism in the preparation of this thesis.

This thesis is submitted in partial fulfillment of the requirement for a degree from the School of Graduate Studies at Wolkite University. The thesis is deposited in the Wolkite University Library and is made available to borrowers under the rules of the library. I solemnly declare that this thesis has not been submitted to any other institution anywhere for the award of any academic degree, diploma or certificate.

Brief quotations from this Thesis/Dissertations may be used without special permission provided that accurate and complete acknowledgement of the source is made. Requests for permission for extended quotations from, or reproduction of, this thesis in whole or in part may be granted by the Head of the School or Department or the Deas of the School of Graduate Studies when in his or her judgment the proposed use of the material is in the interest of scholarship. In all other instances, however, permission must be obtained from the author of the thesis.

Name: Getnet Degemu

Signature: _____

Date: 05/28/24

Department: computer Science and Engineering

ACKNOWLEDGMENT

First of all, I give thanks to God and Saint Mary (የረሽ ማርያም) for all of their support in my life.

In next, I appreciative of Dr. Sintayehu H., my adviser, for his suggestions and assistance, and invaluable insights throughout this research. His expertise and encouragement have been instrumental in shaping the direction of this work and pushing me to achieve my best and I would like to convey my heartfelt appreciation to my co-advisor, Abdo Ababor (Msc), that have greatly influenced and enhanced my work and I am truly grateful for your deep guidance, constructive comments, and sharing of your experience and skills. Working with you has been a privilege, and I am fortunate to have had your support and mentorship throughout this work.

Also, I would like to convey my appreciation to the CCI college members of the Department of Software Engineering at Wolkite University for their continuous support and encouragement. Their dedication to academic excellence and their commitment to fostering a conducive learning environment have greatly enriched my educational experience.

Special thanks go to my family, colleagues (Alex) and friends (Temesgen, Tsegaye F, D Yalewu F, D Dawit F and D Cheru T) who have been a source of inspiration and motivation. Their insightful discussions, constructive feedback, and support in collection of data for this research.

Lastly; this acknowledgment to my beloved wife Marta W (ማዬ). Whose unwavering support, love, and understanding. Your presence in my life has been a constant source of inspiration and strength. Your unwavering belief in my abilities even during the most challenging times has propelled me forward and given me the confidence to follow my dreams.

Name: Getnet Degemu

Signature: _____

Date: 05/28/24

Department: computer Science and Engineering

ABBREVIATIONS AND ACRONYMS

BERT	Bidirectional Encoder Representation
BRNN	Bi-Directional Recurrent Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
EXP	Expert
GLOVE	Global Vectors For Word Representation
GLSA	Generalized Latent Semantic Analysis
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
IDE	Integrated Development Environment
IDF	Inverse Document Frequency
LSTM	Long Short-Term Memory
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Network
PDF	Portable Document Format
PNG	Portable Network Graphics
RELU	Rectified Linear Unit
RNN	Recurrent Neural Network
S1	Sentence One
S2	Sentence Two
SRNN	Stacked Recurrent Neural Network
STS	Semantic Textual Similarity
SVD	Singular Value Decomposition
SVG	Scalable Vector Graphics
TXT	Text File Extension
USE	Universal Sentence Encoder
UTF-8	Unicode Transformation Format-8-Bit
Word2vec	Word To Vector

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURE

LIST OF ALGORITHMS

LIST OF TABLES IN THE APPENDIX

LIST OF FIGURES IN THE APPENDIX

ABSTRACT

Natural language processing (NLP) is one part of how far the world has come in terms of technology. It is the process of teaching human language to machines and includes everything from Morphology Analysis to Pragmatic Analysis. Semantic Similarity is one of the highest levels of NLP. The Previous Semantic textual similarity (STS) studies have been conducted using from string-based similarity methods to deep learning methods. These studies have their limitations, and no research has been done for STS in the local language using deep learning. STS has significant advantages in NLP applications like information retrieval, information extraction, text summarization, data mining, machine translation, and other tasks. This thesis aims to present a deep learning approach for capturing semantic textual similarity (STS) in the Guragigna language. The methodology involves collecting a Guragigna language corpus and preprocessing the text data and text representation is done using the Universal Sentence Encoder (USE), along with word embedding techniques including Word2Vec and GloVe and mean Square Error (MSE) is used to measure the performance. In the experimentation phase, models like LSTM, Bidirectional RNN, GRU, and Stacked RNN are trained and evaluated using different embedding techniques. The results demonstrate the efficacy of the developed models in capturing semantic textual similarity in the Guragigna language. Across different embedding techniques, including Word2Vec, GloVe, and USE, the Bidirectional RNN model with USE embedding achieves the lowest MSE of 0.0950 and the highest accuracy of 0.9244. GloVe and Word2Vec embedding also show competitive performance with slightly higher MSE and lower accuracy. The Universal Sentence Encoder consistently emerges as the top-performing embedding across all RNN architectures. The research results demonstrate the effectiveness of LSTM, GRU, Bi RNN, and Stacked RNN models in measuring semantic textual similarity in the Guragigna language.

Keywords: Semantic textual similarity, Guragigna language, deep learning, corpus-based approaches, LSTM, GRU, Bidirectional RNN, Stacked RNN and Word embedding.

CHAPTER ONE

1. INTRODUCTION

1.1. Background of the Study

NLP means doing computations in natural language. Semantic analysis is one of the processes involved in natural language processing. When building the syntactic structure of the sentence the input sentence analysis does a semantic analysis of the sentence and Sentences are given meaning by semantic interpretation. Logical forms are mapped to knowledge representations by contextual interpretation. The semantic similarity of features in a vector model is the fundamental building block of semantic analysis.[1].

The comparison of text meaning known as semantic text similarity (STS) plays a vital role in various tasks within natural language processing (NLP) like information retrieval, categorization, content extraction, answering questions, and identifying plagiarism.

Text similarity between simple sentence is an important and necessary task in many information retrieval applications. Performance of many natural language processing (NLP) applications like text summarization, machine translation, plagiarism detection, and sentiment analysis. It also relies on similarity of text and meaning. Several other applications have used similarity such as text classification, feedback on relevancy, word disambiguation, subtopic mining, and web search[2].

Similarity measures for many languages such as English, Spanish and Arabic are available, and some have been organized by the organizers of SemEval ST for calculating similarity between multilingual and monolingual simple sentence research duties [2]. One typical approach for computing similarity is lexical matching between simple sentence. A similarity score is determined using the quantity of terms that belong to both text segments. These metrics however, are only able to calculate similarities at a very basic level. Furthermore, this matching can only estimate text similarity but not semantics.

Consider two simple sentence “**ሁት ሜና ነረን ባረም ተሳረምታ ቸነም**” (does he has a work? He asked) and “**ሁት ሜና ኤነን ባረም ተሳረምታ ቸነም**” (doesn’t he has a work? He asked). As indicated by the lexical assignment in both sentences he has two headwords ("ሁት" and "ሜና"). But these he has no semantic connection between the two simple sentence. Consider another pair of sentences: “**አት አርች ቸዋች ተሐረ አወገዳታ ብሽ ንስራነ ቧረንም**” (A boy went to cry with his good friend) and “**አማት አርች ሶሬሳ ተሐረ አወገዳታ ብሽ ንወነ**

ቧረንም" (A boy went to cry with his good friend). There is no clear terminology present in these two sentences. but there are clear semantic similarities [2].

Table 1-1: The weakness of lexical matching in capturing semantic similarity

Sentence 1	Sentence 2	Similarity
"ሁት ሜና ነረን ባረም ተሳረምታ ቸነም"	"ሁት ሜና ኤነን ባረም ተሳረምታ ቸነም"	Lexically similar but **not semantically**
"አት አርች ቸዋች ተሐረ አወገዳታ ብሽ ንስራነ ቧረንም"	"አማት አርች ሶሬሳ ተሐረ አወገዳታ ብሽ ንወነ ቧረንም"	Semantically similar but **not lexically**

Similarities between Guragigna simple sentence is more difficult than simple sentence in other languages. One of the main reasons is that the Guragigna resources are not comparable to those of any other language. Other text preprocessing includes the well-known tokenizers, stemmers, and lemmatizes used in almost every NLP task, and their performance is arguably even better. But on the contrary. This kind of tool is less common in Guragigna simple sentence. Additionally, there are well-organized resources such as WordNet, NLP POS-Tagger, and more. This improves the performance of similarity estimation methods and Guragigna text methods therefore, lack such tools and resources [2]. Trying to overcome the challenge of capturing semantic similarities between Guragigna text pairs. We introduced a method to measure the semantic similarity of Guragigna simple sentence Based on Deep learning techniques an efficient Guragigna algorithm for measuring semantic text similarity is used. Prepare a dataset that can be used to test the performance of the Guragigna text semantic similarity measure [2].

1.2. Motivation

Research in the field of natural language processing has been primarily motivated by they lead to a better understanding of the structure and function of human language Building natural language interfaces. It is used to facilitate communication between both human and computers. Recently, research on the similarity of semantic sentence similarity in international has been made. As an illustration, foreign languages have developed semantic text similarity such as English, Arabic, Spanish, and Bengali [2]. However, the research in local languages and Guragigna is very limited in order to direct Guragigna approach to technology. In particular, semantic textual similarities have not developed in the Guragigna language also in local language.

1.3. Statement of the Problem

An essential component of the processing of natural languages is Semantic Textual Similarity (STS) with significant implications for various tasks and applications. When retrieving information (IR), STS plays a vital role by measuring the similarity between user queries and documents, enabling precise retrieval of relevant information. STS is also valuable in information extraction, where it aids in mining text that is unorganized for useful information by measuring semantic similarity between different pieces of text. Another important application is text summarization, where STS helps identify similar or redundant content within a document, making it easier to create educational summaries. STS is also valuable in data mining, where it aids in clustering similar instances or identifying similar patterns by measuring semantic similarity. In machine translation, STS improves the accuracy of translations by capturing the semantic similarity between source and target language sentences. In question answering systems, STS helps determine the similarity between user queries and candidate answers, leading to more accurate responses. STS is also relevant in sentiment analysis, where it measures similarity between sentiment-bearing texts, aiding in tasks such as sentiment classification. Additionally, STS aids in paraphrase detection, which is crucial for tasks like plagiarism detection and text generation. Overall, STS is a fundamental concept in natural language processing that enhances the efficiency and accuracy of language understanding across various domains.

Guragigna is one of the most widely spoken languages in Ethiopia and is an Afro-Asiatic language of the Semitic Southern Ethiopian branch spoken by the Gurage people. According to the 2007 Census there are currently over 6.8 million native speakers of the language. The language is used in the middle grades of elementary school and in various institutions in the community[3]. It is also used in various fields such as Wolkite radio stations, Magazines, textbooks, and fiction are published in the language. The limited study in Guragigna language can be justified by several factors. NLP tasks require significant linguistic resources, such as annotated corpora and language models. Which are often developed for languages with greater demand and research backing. As a result Guragigna may lack the necessary resources to support advanced NLP research. Additionally, the availability of data plays a crucial role in NLP, and languages with limited study may suffer from a shortage of publicly available language resources. The problem of development and evaluation of NLP systems for Guragigna.

Moreover, the absence of practical applications or tools. For instance Machine translation systems or part-of-speech taggers for Guragigna, further indicates the limited research and development in these areas. Due to these reasons there have been few research studies conducted on the Guragigna language in various tasks involving natural language processing (NLP) instance part-of-speech tagging and machine translation [4], automatic Guragigna language character recognition [5], and others. International academic research websites like IEEE Xplore, ACM Digital Library, and Google Scholar, as well as local academic research websites associated with Ethiopian universities, linguistic research institutions, or language departments. Were explored using relevant keywords such as "Guragigna language" "semantic textual similarity" and "NLP" While no specific studies on STS for Guragigna were found? it is possible to come across related research in the broader field of NLP or studies on STS in other languages, such as Bengali [6], English [7], Arabic [8], and others. These studies showed promising results but also had certain limitations including not utilizing any RNN or CNN models, a large gap in accuracy compared to English STS models, a lack of a lexical standard for Arabic, insufficient experiment detail (failure to explain actual scores and model predictions) and comparison with unrelated works, a lack of information about pre-training embedding's, a shortage of annotated corpora, better performance on smaller datasets, and limited availability of training data. Additionally, in a study conducted on the Amharic language [9] an attempt was made to develop an Amharic-English CLSTSM system by utilizing a statistically topic modeling-based semantic text similarity measurement approach. This model, which uses statistical topic modeling approaches like LDA has a disadvantage in that it primarily relies on word co-occurrence statistics and fails to incorporate the semantic meaning of the words. As a result the topics generated by the model may not always align perfectly with human interpretation or understanding of the underlying themes. Therefore, Based on these problem conduct research on semantic textual similarity is mandatory and important for information retrieval, Information extraction, Text summarization, Data-mining, machine translation and other issues, so we intend to do this Study.

1.4. Research Questions

At the end of this study, the following research questions are answered and investigated.

RQ1. Which word embedding techniques can be used for Model development that can determine the effectiveness and robustness of Semantic Text Similarity (STS)?

RQ2. Which deep learning model is the most effective in performing Semantic Text Similarity (STS) analyses for the Guragigna language?

1.5. Objective

1.5.1. General objective

- The general objective is to develop a semantic textual similarity analyzer using a deep learning approach for Guragigna language.

1.5.2. Specific objective

- To prepare a semantic text similarity corpus for the Guragigna language.
- To develop word embedding techniques for Guragigna Semantic Text Similarity (STS) analyzer.
- To develop a deep learning model for Guragigna semantic text similarity (STS) analyzer.
- To measure the performance of word embedding techniques in conjunction with deep learning model.
- To measure the effects of each deep learning algorithm on the Guragigna Semantic Text Similarity (STS) model.

1.6. Scope of the study

This study's objective is to examine how similar basic sentences are semantically in Guragigna language in dialect of cheha with specifically focusing on sentence-level semantic similarity. The study employs approach of deep learning to investigate semantic similarity within the context of Guragigna language. The goal is to develop a model that can accurately measure semantic similarity in Guragigna language sentences. To facilitate the development and evaluation of the model, a dataset is prepared, consisting of annotated sentences in Guragigna language along with their corresponding similarity scores. The dataset covers diverse sentence pairs, representing various semantic relationships and degrees of similarity. The study aims to leverage deep learning techniques to advance the understanding and capabilities of semantic similarity analysis in the Guragigna language.

1.7. Limitations of the study

The analysis is focused on sentences meaning that the findings may not directly apply to more complex sentence structures or longer texts and the study is limited to the specific dialect of Cheha. Which could restrict its generalizability to other dialects or languages. The effectiveness of the model developed in this study heavily depends on the quality and representativeness of the dataset used. Any limitations or biases in the dataset may have an impact on the accuracy and reliability of the model's results. Lastly, this study primarily focuses on general semantic relationships in Guragigna language sentences and does not

address specific domain semantic similarity analysis. By considering these limitations, we can better interpret and contextualize the findings of the study.

1.8. Significance of the study

Semantic text similarity is an important and fundamental task in natural language processing (NLP). Being able to compare semantic similarities between sentences has many applications in various fields. Examples of areas where text similarity is used include plagiarism detection, search engines, and customer service. The development of STS has different benefits for both the Gurage language community and the research community.

- For the Gurage community: Semantic Textual Similarity (STS) holds great significance for the Gurage community by contributing to language preservation, technology development, education, information retrieval, and cultural representation. STS enables the accurate measurement of semantic similarity in Gurage language sentences help in the preservation and revitalization of the language and facilitating the development of language technologies specific to the community's needs. It supports educational applications and empowering learners to improve their language proficiency. Lastly, it promotes cultural representation and identity by conveying the unique aspects of the Gurage community's language and culture in various domains. Overall, STS empowers the Gurage community in communication, information access, and language preservation for future generations.
- For the research community: It contributes to researchers doing more advanced NLP applications of the Guragigna language as a preprocessing component.

1.9. Organization of the thesis

The rest of this thesis is organized as follows. In Chapter 2, we explain the different approaches used to develop Semantic Text Similarity (STS) and review related works on developing Semantic Text Similarity (STS) for Guragigna language. Chapter 3 focuses on the methodology employed in this study. Chapter 4 presents the design of STS and implementation of the proposed Semantic Text Similarity (STS) system for Guragigna language. In Chapter 5, we present the STS experimental results of the proposed system. Chapter 6 describes the results and discussion. Finally, in Chapter 7, we conclude the thesis by highlighting the research contribution and discussing future works.

CHAPTER TWO

2. LITERATURE REVIEW

2.1. Introduction

Semantic similarity is significant in Natural Language Processing (NLP), and it plays a crucial role in various NLP applications. One fundamental task in this field is Semantic Textual Similarity (STS) which involves assessing the similarity between different documents. To determine this similarity a metric is used to evaluate the direct and indirect relationships among the documents. By identifying semantic relations we can measure and recognize these relationships accurately [8][10].

The primary objective of the STS task is to establish a unified framework that include different independent semantic components. This assess the influence of these elements on different NLP tasks. Developing such a framework is a crucial research challenge with significant applications in NLP include retrieval of information (IR) and summarization of text in area [4], [11], as well as question answering [12],, relevance feedback [13], text classification [14], WSD, and summarization for extractive [15].

Semantic similarity is not only relevant for NLP applications but also plays a significant role in various semantic web applications include extraction, generation of ontology, and disambiguation. Semantic similarity is particularly valuable in search [50], where the performance to accurately with all entities measure the semantic relatedness is valuable in IR. One of the key problem is how semantically related documents or images retrieving to a user's query in a web search engine, including retrieving images based on their captions [11].

Text similarity has applications beyond NLP and the semantic web, extending into the field of databases as well. In database systems, text similarity can be leveraged for schema matching, addressing the challenge of semantic heterogeneity in data sharing systems, data integration systems, message passing systems, and peer-to-peer data management systems [16]. Additionally, text similarity is beneficial for relational join operations in databases, particularly when the join attributes exhibit textual similarity. The utility of text similarity spans various application domains, including the integration and querying of data from diverse resources, data cleansing, and data mining [17].

In NLP, STS) is connected to both Textual Entailment (TE) and paraphrasing, but have a differences between them. In TE, three directional relationships can be established between two text fragments. The task involves a two text considering as fragments of "text" (t) and the

"hypothesis" (h). In another case, paraphrasing identification aims to recognize text fragments that have approximately the same meaning within a specific context. Therefore, TE and paraphrasing focus on providing a yes/no decision, while STS goes a step further by evaluating the degree of equivalence between texts and assigning ratings with their semantic connection.

2.2. The Semantic Textual similarity Approach

2.2.1. String based similarity

The string-based Similarity methods evaluate the text from a lexical standpoint and only work with string sequences and characters. String based similarity is a way of evaluate the strings similarity. More used in NLP tasks to compare phrases, sentences and other text fragments. These measures may be taken to determine the level of semantic or syntactic relatedness between two strings.

2.2.1.1. Character-Wise Approach

LCS and N-grams are two of the most common approaches in a character level evaluation. Longest Common Substring (LCS) algorithm uses dynamic programming to consider the length of common substrings in both terms. N-gram algorithm considers a sub-sequence of n items of the term. Distance in N-Gram is computed by dividing the number of similar n-grams by the maximal number of n-grams available.

Longest Common Substring (LCS) algorithm is employed to identify the longest shared substring between two strings. It compares the two strings and determines their similarity by examining the longest sequence of characters they have in common [16]. The measurement can be computed as follows:

$$LCSubstr\ (S1, S2) = \max LCSuff(S1 \dots i, S2 \dots j), 1 \leq i \leq m, 1 \leq j \leq n$$

The measurement of the LCS algorithm can be computed using the following formula:

$$\text{LCS(S1, S2)} = \text{LCSuff(S1, S2, m, n)}$$

Here, m represents the length of the first string (S1), n represents the length of the second string (S2), and LCSuff is a function that finds the longest common suffixes of the possible prefixes of S1 and S2.

Damerau-Levenshtein distance, also known as the Damerau-Levenshtein is a metric of two strings to evaluate the difference between them. It the number of operations needed to transform one string into another to quantifies.[16].

Jaro: no similarity between the strings is explain in 0, and represents an exact match is explain in 1 called distance score is normalized is calculated as follows:

$$d_j = \begin{cases} 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \end{cases} \text{if m=0 otherwise}$$

Here, |s1| denotes the length of string s1, |s2| denotes the length of string s2, m is the "number of matching characters", and t is "transpositions of half the number".

$$\left[\frac{max(|S1|, |S2|)}{2}\right] - 1$$

Jaro–Winkler distance is accept two strings and evaluate similarity of semantic that is a classification of distance edit and the Jaro distance metric. This way is more implement on simple sentence.

$$d_w = d_j + (lp(1 - d_j)),$$

Here, dj represents the Jaro distance between the strings, and lp is scale of prefix can matching characters at the beginning of the strings that ratings assigns up to a length l of prefix. This approach included the Jaro distance with a prefix bonus to provide a more refined similarity measure [20].

Needleman-Wunsch algorithm is optimal matching algorithm and a global alignment technique for dynamic programming algorithm commonly take in sequences of bio-informatics for aligning. [18].

Algorithm of Smith-Waterman is algorithm that make sequence alignment in local take that to evaluate the strings similarity like nucleotide sequences. Unlike Needleman-Wunsch, Smith-Waterman focuses on optimizing segments of strings similarity. This algorithm not implement for long-scale problems [19].

Model of N–gram is a model of probabilistic language that implement to execute the sequence of next term (n - 1) terms or characters. The big advantages of the N-gram model are its simply implement and more scalability [20].

2.2.1.2. Term-Wise Approach

At the Term level, there are commonly used measures to evaluate similarity: cosine similarity and Jaccard similarity. Cosine similarity compares two vectors in a space and calculates. There are also other methods available, such as Damerau-Levenshtein, Jaro-Winkler, Needleman-Wunsch, and Smith-Waterman, which are not explained in detail here due to space constraints.

Block Distance, also known as the City Block of Distance, Snake Distance, Manhattan Distance, Manhattan Length, L1 Distance[21], is a metric used to measure the two points distance d1 with calculated vectors of p and vectors of q as follows:

$$d_1(p,q) = \|p - q\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

Cosine Similarity is a metric of similarity in an inner product space between two non-zero vectors to determines the cosine of the vectors angle. Similarity of Cosine is commonly used in data mining to assess the cohesion between vectors [22]. The cosine of two non-zero vectors can be computed using the Euclidean dot product.

$$a - b = \|a\|\|b\| \cos\theta$$

Vectors A and vectors B is calculated Similarity of Cosine cos(θ) as divided by vectors to the product of their magnitudes in dot product. Mathematically, it can be expressed as

$$\cos(\theta) = \frac{A - B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Here, A · B represents the dot product of vectors A and B, and ||A|| and ||B|| represent the magnitudes (or norms) of vectors A and B, respectively.

Similarity of Soft Cosine is takes into count the Vector Space Model similarity [23]. It is calculated using the following formula:

$$soft\_\cos\theta = \frac{\sum_{i,j=1}^{n} s_{ij} A_i B_j}{\sqrt{\sum_{i,j=1}^{n} s_{ij} A_i A_j}\sqrt{\sum_{i,j=1}^{n} s_{ij} B_i B_j}}$$

In this formula, sij represents the value from the similarity matrix between features i and j. It's important to note that if the similarity matrix is diagonal, meaning the features are only similar to themselves, then the soft cosine similarity becomes same to the old similarity of cosine. [24]

Sorensen–Dice index (Dice's Coefficient) is taken to measure quantify the similarity of samples [25]. It is commonly employed to determine the presence of data set or absence of data sets.

The formula for calculating the Sorensen-Dice Index is as follows:

$$QS = \frac{2|X \cap Y|}{|X| + |Y|}$$

Here, |X| (number of elements one set) and |Y| (number of elements one set) compared. The quotient of similarity represent by QS, and ranges of 0 to 1 is value.

Coefficient on bigrams of Strings S1 and S2 similarity calculated as follows:

$$sim = \frac{2n_t}{n_{s1} + n_{s2}}$$

In this formula, nt represents the count of shared bigrams between the strings, while n_s1 and n_s2 represent the total number of bigrams in S1 and S2.

Euclidean Distance is masseur the two points of straight-line distance. Distance of euclidean in two points, denoted as "d(s,t)" or "d(t,s)", can be calculated using the following formula:

$$d(s,t) = d(t,s) = \sqrt{\sum_{i=1}^{n} (t_i - p_i)^2}$$

In this formula, n represents the number of dimensions or features in the space, t_i and p_i represent the corresponding coordinates or values of the points in each dimension. The formula calculates the total root of square with squared differences in the values of the two points.

Jaccard Index (Jaccard similarity coefficient) is a statistical measure used to similarity with diversity determine in two sets of finite [26]. It is defined by the following formula:

$$J(A,B) = \frac{|A||B|}{|A \cup B|} = \frac{|A||B|}{|A| + |B| - |A \cap B|}$$

In this formula, A and B represent the two sets being compared. |A| and |B| denote the cardinality (number of elements) of sets A and B, respectively.

SMC is a statistical measure used to assess the similarity and diversity between two objects. It considers the objects as a collection of n binary attributes. The SMC between objects A and B can be calculated using the following formula:

$$SMC = \frac{Number\ of\ Matching\ Attributes}{Total\ No.of\ Attributes}$$
$$= \frac{a_{00} + a_{11}}{a_{00} + a_{01} + a_{10} + a_{11}}$$

SMC = (Number of Matching Attributes) / (Total Number of Attributes)

Where the total number of attributes is represented by a_00 + a_01 + a_10 + a_11, and the matching attributes that have the same value in both objects.

In the formula, a_00 represents the total number of attributes that are 0 in both A and B, a_10 represents the total number of attributes that are 1 in A and 0 in B, a_01 represents the total number of attributes that are 0 in A and 1 in B, and a_11 represents the total number of attributes that are 1 in both A and B. The SMC provides a evaluation of similarity of two objects By dividing the quantity of matching attributes by the total number of attributes, ranging from 0 to 1.

Overlap Coefficient the Overlap Coefficient, also known as the Szymkiewicz-Simpson Coefficient, is a similarity evaluation that is closely related to the Jaccard Index. It quantifies the overlap between two sets and is defined as follows:

$$overlap(A, B) = \frac{|A| \cap |B|}{\min(|A|, |B|)}$$

In this formula, |A| and |B| represent the cardinalities (number of elements) of sets A and B, respectively. The numerator calculates the size of the intersection of sets A and B, while the denominator represents the size of the smaller set between A and B.

The overlap coefficient takes values between 0 and 1, with 1 indicating that set A is a subset of set B. In other words, when the overlap coefficient is equal to 1, all elements of set A are also present in set B.

2.2.2. Corpus-Based Approaches

Corpus-based methods in language studies involve using real language samples from large collections of written or spoken texts to examine language structure, usage, and meaning. These approaches are commonly used to identify patterns in communication, create networks of word meanings, develop computer models for language and learning, measure differences between

dialects, and understand how language changes over time. They can also help us understand how we learn languages and provide information for tasks like NLP.

Important aspect of this approach is finding similarities between words by analyzing the data in the collection. This method requires a sizable collection of texts. Analyzing large collections provides valuable information, allowing us to identify common word occurrences and accurately estimate word similarities. Many of the methods proposed for measuring word similarity rely on analyzing large collections of texts. This type of similarity measurement is based on information gathered from a substantial collection of texts. A text collection used for language research is called a corpus, and it contains written or spoken sentences. To determine word similarities, we often look at how words appear together in the corpus. To obtain reliable statistics on word co-occurrence, we need a very large and balanced corpus [27].

2.2.2.1. Method of LSA (Latent Semantic Analysis)

One example of this type of analysis is Latent Semantic Analysis (LSA). In LSA, each word is represented as a vector based on statistical calculations. To create these vectors, a large text is analyzed, and a matrix of words is constructed. The words are represented as rows in the matrix, and the paragraphs or segments of text are represented as columns. Singular value decomposition (SVD) is then applied to reduce the dimensionality of the matrix. After dimensionality reduction, word similarity is computed using cosine similarity.

In this method, contextual information for words is extracted from a large text corpus [28]. The first step involves representing the text as a matrix, where rows represent unique words and columns represent segments of text. Each entry in the matrix represents the frequency count of a word appearing in a particular segment of text [29]. The cell frequencies are weighted based on two factors. The importance of words in the text and the degree to which parts of speech share information in the discourse context. This approach can be implemented in two ways as a similarity matrix with words and text segments implementation use, and as a computational model that represents the underlying knowledge acquisition and usage. To reduce the number of rows in matrix, singular value decomposition (SVD) is develop while preserving the similarities with columns. To measure similarity, the cosine angle between vectors of word made by any two rows.

LSA relies on the distributional hypothesis, which suggests that words appearing in similar contexts to have similar meanings [30]. Therefore, evidence of word similarity can be computed through statistical analysis of large collections of sentences. LSA is a mathematical

and statistical technique that extracts and assumes relationships based on the expected contextual usage of words in a discourse passage. It is not a traditional natural language or artificial intelligence processing program. Instead of relying on human-made dictionaries, knowledge bases, semantic webs, grammars, syntactic parsers, morphologies, etc., LSA takes raw text as input, treating it as sentences or paragraphs [29].

2.2.2.2. Method of Hyperspace Analogue to Language

Hyperspace Analogue to Language (HAL) is a method that constructs a word co-occurrence matrix where both rows and columns represent words in the vocabulary. The matrix elements are filled with association strength values. These association strength values are computed by applying a sliding "window" over the corpus, and the size of the window can be adjusted. The strength of association between words within the window decreases as their distance from each other increases. For example, in the sentence "This is a survey of various semantic similarity measures," the words "survey" and "variety" would have a higher association value compared to "survey" and "measures."

Word vectors are formed by considering both the row and column of a given word in the co-occurrence matrix. To reduce dimensionality, columns with low entropy values are eliminated. Finally, semantic similarity is calculated by measuring the Euclidean or Manhattan distance between the word vectors [31].

2.2.2.3. Method of Explicit Semantic Analysis (ESA)

ESA (Explicit Semantic Analysis) is a semantic similarity measurement method that relies on Wikipedia concepts. By utilizing Wikipedia, this approach can be applied to different domains and languages. The dynamic nature of Wikipedia ensures that the method remains adaptable to changes over time [32]..

In ESA, each concept present in Wikipedia is represented as an attribute vector comprising the words associated with it. An inverted index is then constructed, linking each word to the concepts it is associated with. To determine the strength of technique called TF-IDF applied, which assigns weights to the associations. Concepts that have weak associations with words are subsequently filtered out. As a result, the input text is represented by weighted vectors of concepts, known as "interpretation vectors" [32]..

To measure semantic similarity, the cosine similarity between these interpretation vectors is calculated. The cosine similarity provides a measure of how similar the vectors are in terms of their direction in the vector space [32].

2.2.2.4. Method of Word-alignment Models

Word-alignment models are used to determine the semantic similarity between sentences based on their alignment in a large corpus [32]. These models have shown success in SemEval tasks 2015, securing the second, third, and fifth positions. One unsupervised method that ranked fifth utilized the word-alignment technique based on the Paraphrase Database (PPDB) [32]. This system measures the semantic similarity between two sentences by considering the proportion of aligned context words shared between the sentences compared to the total number of words in both sentences.

The supervised methods that ranked second and third employed word2vec to establish word alignments. In the first supervised method, a sentence vector is created by computing the "component-wise average" of the word vectors. The cosine model similarity between these sentence vectors is then implement as a measure of STS. The second supervised method focuses only on words that exhibit contextual semantic similarity [32].

2.2.2.5. Method of Latent Dirichlet Allocation (LDA)

LDA (Latent Dirichlet Allocation) is a technique commonly used for topic modeling tasks. It represents a document's topic or general idea as a vector, rather than including every single word from the document. This approach offers the advantage of reduced dimensionality since the number of topics is typically much smaller than the number of words of the document [33].

To evaluate similarity of document, a new method involves using vector representations of documents. Each document is represented as a vector in a high-dimensional space, where each dimension corresponds to a specific topic. The cosine similarity between these document vectors is then calculated to measure the semantic documents similarity [34].The cosine similarity provides a measure of how similar the directions of the vectors space in document, indicating their semantic similarity.

2.2.2.6. Method of Normalized Google Distance (NGD)

Normalized Google Distance (NGD) is a measure of similarity between two terms based on the results obtained from querying them using the Google search engine. The underlying assumption is that if two words are more related, they will appear together more frequently in web pages [35].

To calculate the NGD between two terms, denoted as t1 and t2, the following formula is used:

$$NGD(x,y) = \max\{\log f(t1), \log f(t2)\} - \log f(t1,t2), \quad (9)\ \log G - \min\{\log f(t_1), \log f(t_2)\}$$

In this formula, f(x) and f(y) represent the number of hits in the Google search results for the respective terms, while f(x,y) represents the number of hits when the terms are searched together. The variable G represents the total number of pages in the overall Google search.
NGD is commonly used to measure semantic relatedness rather than semantic similarity. This is because related terms tend to appear together more frequently in web pages, even if they have opposite meanings.

2.2.2.7. Method of Dependency-based Models

Dependency-based approaches aim to determine the meaning of a given word or phrase by examining its neighboring words within a specified window. These approaches typically begin by parsing the corpus using Inductive Dependency Parsing [36]., which involves analyzing the distribution of words within the corpus.
For each word, a "syntactic context template" is constructed, taking into account the preceding and succeeding nodes in the parse tree. As an example, the phrase "thinks <term> delicious" could have a context template such as "pizza, burger, and food." A vector representation of a word is then created by aggregating the context templates in which the word appears as the root word. The frequency of these word windows occurring in the entire corpus is also considered.
Once the vector representation is formed, semantic similarity can be calculated using cosine similarity between these vectors. Levy et al. [36].introduced DEPS embedding as a word-embedding model based on a bag-of-words approach in dependency based. The model was evaluated using the WS353 dataset, which involved ranking similar words above related words. When comparing the recall-precision curves, the DEPS curve demonstrated a stronger affinity towards similarity rankings compared to the bag-of-words (BoW) methods.

2.2.2.8. Method of Word-attention Models

In many corpus-based methods, all components of the text are typically treated as equally significant. However, in human interpretation of similarity, the importance of specific keywords in a given context is often emphasized. Word attention models aim to capture the importance or relevance of words from the underlying corpus before calculating semantic similarity [37].
Word attention models employ various techniques to determine the attention weights of the words in the text being analyzed. These techniques may include factors such as word frequency, alignment, and word association. By assigning higher attention weights to main terms in the context, word attention models can capture the relative importance of specific words in

determining semantic similarity. This allows the models to focus on the most relevant information when calculating similarity measures.

2.2.2.9. Method of GLSA (Generalized Latent Semantic Analysis)

A technique for calculating semantically motivated phrase and document vectors is called Generalized Latent Semantic Analysis (GLSA). By emphasizing term vectors rather than the dual document-term representation, it expands on the LSA methodology. A dimensionality reduction technique and a measure of semantic association between concepts are needed for GLSA. Any appropriate technique for dimensionality reduction can be used with any type of similarity measure on the space of words using the GLSA approach. The final phase provides the weights in the linear combination of term vectors using the conventional term document matrix [10].

2.2.3. Knowledge Base STS Approaches

Newly developed state-of-the-art methods for determining similarity scores with in text sample pairs include knowledge-based linguistic variables. These techniques use lexical relations and word-level semantic networks to assess relevance at the text (sentence) level. Electronic resources, such as lexical resources and knowledge bases, serve as the primary sources of information for these methods. The semantic similarity between two simple sentences is quantified by evaluating a global measure based on pairwise comparisons of word similarity within these sentences. The construction of sentence-to-sentence semantic similarity relies on the aggregation of individual word semantic similarities [16].

One specific measure used for sentence-to-sentence similarity is the term set-to-term set measure, which represents an extreme case of this approach [38]. To compute this measure, two texts that are being compared are queried separately in the corpora to determine the number of documents containing each text. Additionally, the number of documents in which both words appear together is queried. These queries are performed using a Lucene index built on the corpora. The cardinality of a word refers to the number of corpora in which the word appears, while the cardinality of a word's conjunction represents the number of documents in which both words appear [8].

Based on the principles take to assess the semantic similarity between words, knowledge-based semantic similarity methods can be further categorized into edge-counting methods, feature-based methods, and information content-based methods. These different categories employ distinct techniques and measures to capture the semantic relatedness between words and extend them to sentence-level similarity assessments.

2.2.3.1. Edge-counting Methods

A simple approach to measure similarity between terms is to view the underlying ontology as a graph, where words are connected in a taxonomic manner. By counting the edges between two terms, we can gauge their similarity. The shorter the path between the terms, the more similar they are. This measure, known as "path," was proposed by Rada et al [39]. It determines similarity by considering the inverse of the shortest path length between two terms.

However, the edge-counting method does not take that words lowermost in the structure may have more specific meanings. These words could be more similar to each other, even if they have the same distance as two term indicate a more general concept. To address this, Wu and Palmer [39] proposed the "wup" measure, which considers the depth of words in the ontology as an important factor. The wup measure counts the number of edges between each term and their Least Common Subsumer (LCS). The LCS represents the common ancestor shared by both terms in the given ontology.

Let's denote two terms as t1 and t2, their LCS as tlcs, and the shortest path length between them as min_len(t1,t2). The path is then measured as follows:

$$sim_{path}(t_1, t_2) = \frac{1}{1 + \min\_len(t_1, t_2)}$$

and wup is measured as,

$$sim_{wup}(t_1, t_2) = \frac{2depth(t_{lcs})}{depth(t_1) + depth(t_2)}$$

Li et al. [40] proposed a measure that takes into account both the minimum path distance and depth. li is measured as,

$$\mathrm{sim_{li}} = \mathrm{e}^{-\mathrm{amin\_len}(t_1,t_2)}, \frac{\mathrm{e}^{\beta \mathrm{depth}(t_{lcs})} - \mathrm{e}^{-\beta \mathrm{depth}(t_{lcs})}}{\mathrm{e}^{\beta \mathrm{depth}(t_{lcs})} + \mathrm{e}^{-\beta \mathrm{depth}(t_{lcs})}}$$

However, the edge-counting methods ignore the fact that the edges in the ontologies need not be of equal length. To overcome this shortcoming of simple edge-counting methods, feature-based semantic similarity methods were proposed.

2.2.3.2. Feature-based Methods

The feature-based methods calculate similarity as a properties of the words function, such as gloss, neighboring concepts, and so on [12]. Gloss is defined as the meaning of a word in a dictionary; a collection of glosses is called a glossary. There are various semantic similarity methods proposed with gloss of words. Gloss-based semantic similarity measures exploit the

knowledge that words with the same meanings have more common words in their gloss. The semantic similarity is measured as the extent of overlap between the words glosses in consideration. The measure [41]] assigns a value of relatedness between two words based on the overlap of words in their gloss and the glosses of the concepts they are related to in an ontology like WordNet [42] [14]. Proposed a feature-based method where semantic similarity is measured using the glosses of concepts present in Wikipedia. Most feature-based methods take into account common and non-common features between two words/terms. The common features contribute to the increase of the similarity value and the non-common features decrease the similarity value. The major limitation of feature-based methods is its dependency on ontologies with semantic features, and most ontologies rarely incorporate any semantic features other than taxonomic relationships [12].

2.2.3.3. Information Content-based Methods

Information content (IC) of a concept is defined as the information derived from the concept when it appears in context [43]. A high IC value indicates that the word is more specific and clearly describes a concept with less ambiguity, while lower IC values indicate that the words are more abstract in meaning [44]. The specificity of the word is determined using Inverse Document Frequency (IDF), which relies on the principle that the more specific a word is, the less it occurs in a document. IC-based methods measure the similarity between terms using the IC value associated with them.

Resnik and Philip [45] proposed a semantic similarity measure called res that measures the similarity with on the idea that if two concepts share a common subsumer, then they share more information, since the IC value of LCS is higher. Considering IC represents the IC of the given term, res is measured as,

$$\text{simres (t1,t2)} = \text{ICtlcs} .$$

D. Lin [46] proposed an extension of the res measure consideration to taking the IC value of both the terms that attribute to the individual information or description of the terms and the IC value of their LCS that provides the shared commonality between the terms. lin is measured as,

$$sim_{lin}(t_1, t_2) = \frac{2IC_{tlcs}}{IC_{t2} + IC_{t2}}$$

Jiang and Conrath [47] calculate a distance measure with on the difference between the sum of the individual IC values of the terms and the value of IC their LCS using the below equation:

$$\text{disjcn}(t1,t2) = \text{IC}t1 + \text{IC}t2 - 2\text{IC}tlcs.$$

The distance measure replaces the shortest path length in Equation (1), and the similarity is inversely proportional to the above distance. Hence jcn is measured as,

$$sim_{jcn}(t_1, t_2) = \frac{1}{1 + \text{dis}_{jcn}(t_1, t_2)}$$

an underlying corpora measured by IC or from the intrinsic ontology structure itself [33] with on the assumption that the ontologies are structured in a meaningful way. Some of the terms may not be included in one ontology, which provides a scope to use multiple ontologies to calculate their relationship [13]. Based on whether the given terms are both present in a single ontology or not, IC-based methods can be classified as mono-ontological methods or multi-ontological methods. When multiple ontologies are involved, the IC of the Least Common Subsumer from both the ontologies are accessed to estimate the semantic similarity values. Jiang et al.[48] Proposed IC-based semantic similarity measures based on Wikipedia pages, concepts, and neighbors. Wikipedia was both used as a structured taxonomy likewise, a corpus to provide IC values.

Semantic Textual Similarity (STS) is a task in natural language processing that involves measuring how closely related pairs of text units are in terms of their meaning. To preprocess STS data, several steps are typically followed to enhance the accuracy of the similarity measurement. These steps include:

1. Tokenization: The text is divided into individual words or tokens to establish the basic units of analysis.
2. Stop word removal: Common words that do not carry much semantic meaning, such as "a," "the," or "of," are removed to reduce noise and focus on more meaningful content.
3. Stemming and lemmatization: Words are reduced to their base or root forms to handle variations of the same word. It advantageous in capturing the core meaning and avoids redundancy.
4. Part-of-speech (POS) tagging: Each word is assigned a syntactic category, such as noun, verb, adjective, etc., to understand the grammatical structure and potential relationships between words.
5. Dependency parsing: The relationships and dependencies between words are analyzed to determine which words depend on others in terms of syntax and meaning.

6. Named entity recognition: Entities such as names of people, organizations, locations, etc., are identified to handle their specific semantic significance.
7. Parsing trees: The syntax structure of the sentence is represented using parsing trees, which capture the hierarchical relationships between words.

By applying these preprocessing techniques, noise is reduced, and the semantic meaning of the sentence pair is captured more accurately. This, in turn, improves the performance and accuracy of STS systems in measuring the similarity between texts.

2.3. Deep learning techniques

A Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of Neural Network that addresses the requirement for sequential information processing. Unlike traditional neural networks, where inputs and outputs are treated independently, RNNs consider the previous output as input for the current step. This is particularly useful in tasks like predicting the next word in a sentence, where the context of previous words is necessary. To enable this sequential processing, RNNs introduce a Hidden Layer that plays a crucial role. This Hidden Layer, also known as the Hidden State or Memory State, retains information about the sequence's previous inputs. It make as a memory that helps the network remember and utilize past inputs. One of the key advantages of RNNs is their ability to share parameters across different inputs or hidden layers. This means that the same set of parameters is used for each input, performing the same operation on all of them. As a result, the complexity of parameters is reduced compared to other neural network architectures. RNNs are specialized neural networks that address the requirement for sequential information processing. They utilize a Hidden Layer or Hidden State to remember past inputs, enabling them to capture dependencies and context in sequential data. The parameter sharing property of RNNs contributes to their efficiency by reducing the complexity of parameters [49].

When modeling sequential data, where the context and order of the input items are important, RNNs perform especially well. They are capable of processing input sequences with varying lengths and identifying the connection between the sequence's components. Because of this, RNNs can be used for a variety of applications, including language modeling, machine translation, speech recognition, time series prediction, and sentiment analysis. Long-term dependencies and contextual information within a sequence are excellently captured by RNNs. An RNN's hidden state stores the details of the inputs it has already seen, enabling the network to retain context memory. Tasks requiring the word analysis or phrase meaning in the context

of the full sequence benefit from this contextual comprehension. Because RNNs can represent sequential data, they are frequently utilized in NLP tasks. Language generation, text categorization, named entity identification, sentiment analysis, question answering, and machine translation are just a fewer of the jobs they have successfully completed. RNN variations that have shown to be very successful in capturing long-range dependencies and reducing the vanishing gradient issue are LSTM and GRU. For the analysis and prediction of time series data values arranged according to a specific time interval RNNs are a good fit. They are helpful for applications like signal processing, anomaly detection, weather forecasting, and stock market prediction because they can extract temporal patterns and dependencies from the data. Transfer learning is made possible by pre-training RNN models on extensive language modeling tasks, such as training on a sizable corpus of text data. Then, using smaller labeled datasets for certain downstream tasks, the pre-trained RNN models can be improved. By utilizing the acquired language knowledge from the pre-training phase, this method enhances performance on the intended job. RNNs enable for the examination of the hidden states and their temporal evolution, which contributes to their interpretability to some degree. This could provide information into the attributes the model deems crucial for the task and aid in understanding how it makes decisions. RNNs are a popular option in learning of machine in case of their versatility in NLP and time series analytic tasks, as well as their capacity to handle sequential input, capture context and dependencies, and perform well in these areas. RNNs are a useful tool for different applications due to their adaptability and efficiency in modeling sequential information [49].

LSTM (Long Short-Term Memory) is a special kind of recurrent neural network (RNN) design that solves the issue of the vanishing gradient problem found in traditional RNNs. LSTM introduces a memory cell and three gates: input gate, forget gate, and output gate. These gates control the flow of information into and out of the memory cell, allowing LSTMs to selectively retain or discard information over long sequences. The memory cell enables LSTMs to capture long-range dependencies and remember relevant information from earlier parts of the sequence.

GRU (Gated Recurrent Unit) is another variant of the RNN architecture that addresses the vanishing gradient problem and has a simpler structure compared to LSTM. GRU also includes gating mechanisms, but it uses only two gates which is update gate and a reset gate. The update gate regulates how much of the previous hidden state should be maintained, while the reset gate determines the extent to which past information should be disregarded. GRU performs

similarly to LSTM in many sequence modeling tasks while being computationally more efficient due to its reduced number of gates [49].

In some cases, information from both past and future inputs is important for understanding the current input in a sequence. Bidirectional RNNs (Bi-RNNs) address this by processing the input sequence in two directions: one forward and one backward. This means that the hidden state of the network at each step is influenced by both the past and future input contexts. Bi-RNNs are particularly useful in tasks where context from both directions is crucial, such as part-of-speech tagging or named entity recognition. By capturing information from both directions, Bi-RNNs can provide improved context awareness and capture dependencies in the entire sequence [49].

Stacked RNN involve stacking multiple recurrent layers on top of each other. Each layer in the stack processes the input sequence sequentially, and its hidden state is passed as input to the next layer. Stacked RNNs allow for more complex representations and can capture hierarchical dependencies in the data input. The lower layers capture local dependencies, while the higher layers capture more abstract and global dependencies. The use of stacked RNNs can enhance the models the ability to understand intricate relationships and patterns in sequential data, making them beneficial for tasks that require a deeper understanding of the input sequence.

In general, RNNs are commonly utilized for Semantic Textual Similarity (STS) tasks due to their semantic similarity between sentences, which requires considering the order and context of words and phrases. RNNs, with their recurrent connections and hidden states, can effectively model the sequential nature of sentences and encode contextual information. They are appropriate for capturing the complex semantic linkages between phrases because they may record long-term interdependence. Furthermore, RNNs' flexibility in handling variable-length input and their capacity for transfer learning make them a valuable choice for STS, allowing them to leverage pre-trained language models and improve performance on the task.

2.4. Overview of Guragigna Language

Guragigna, also known as Gurage or Guragegna, is a Semitic language spoken by the Gurage people in Ethiopia. It belongs to the Afro-Asiatic language family and specifically falls under the South Ethiopian Semitic branch. Guragigna is primarily spoken in the Gurage Zone, which is located in the southern part of the country. Guragigna has several dialects, with variations in vocabulary, pronunciation, and grammar across different Gurage communities. These dialects include Ezha, Cheha, Soddo, Inor, Gumer, Gura, Meskane, Muher, and Gyeto. The language is

characterized by a rich oral tradition and has its own unique writing system which is the Guragigna script. However, the script is not widely used, and the majority of Guragigna speakers primarily use the Ethiopian script known as Fidel [50].

Guragigna exhibits a phonological system characterized by a diverse range of consonants and a set of five vowel phonemes. The language allows for complex syllable structures and permits consonant clusters in both initial and final positions. Stress typically falls on the penultimate syllable, while intonation plays a significant role in conveying meaning. Grammatically, Guragigna features noun and verb conjugation, adjective agreement with nouns, and a predominantly subject-verb-object word order. The language historically used the Ge'ez script, but in modern times, it is commonly written using the Ethiopian script, an abugida and now the Gurage zone Culture and Tourism Office prepare new Guragigna Script. Guragigna holds socio-cultural significance for the Gurage people, being intertwined with their traditions, folklore, and identity. Efforts are underway to preserve and promote Guragigna through educational initiatives and cultural events, contributing to the linguistic and cultural landscape of the Gurage Zone and Ethiopia as a whole [51][52][53].

Guragigna has influenced and been influenced by other Ethiopian languages, particularly Amharic, due to historical and geographical interactions. As outcome, there are similarities in vocabulary and grammar between Guragigna and Amharic. The Gurage people, who are the native speakers of Guragigna, have a diverse cultural heritage and are known for their agricultural practices, craftsmanship, and music. Guragigna plays a significant role in preserving and transmitting their cultural traditions and expressions [51][52][53].

While Guragigna is primarily spoken within the Gurage community, there have been efforts to promote the language and its cultural significance through educational initiatives and documentation projects. These endeavors aim to preserve and enhance the understanding and use of Guragigna among its speakers and to promote appreciation for its linguistic and cultural richness [51][52][53].

2.5. Related works

As mention in [6] the paper investigates several word embedding techniques (Word2Vec, GloVe, and FastText) to estimate the semantic similarity of “Bengali” sentences. Due to the unavailability of the standard dataset, this work developed a Bengali dataset containing 187031 text documents with 400824 unique words. Moreover, this work considers three semantic distance measures to compute the similarity between the word vectors using Cosine similarity

with no weight, term frequency weighting, and Part-of-Speech weighting. The performance of the proposed approach is evaluated on the developed dataset containing 50 pairs of Bengali sentences. The evaluation result shows that Fast Text with continuous bag-of-words with 100 vector sizes achieved the highest Pearson's correlation (ρ) score of 77.28% [6].

As mentioned in [8], offers three distinct methods for producing STS Arabic models that work well. The first one is based on a fine-tuning evaluation of automatic machine translation from English STS data to Arabic. The second strategy is based on integrating English data resources with Arabic models. Using a proposed translated dataset, the third strategy focuses on optimizing the knowledge distillation-based models to improve their performance in Arabic. Using a very small collection of resources a few hundred Arabic STS sentence pairs. Were able to obtain an 81% correlation score when using the regular STS 2017 Arabic assessment set. Additionally, it was possible to expand the Arabic models to process the two regional dialects, Saudi Arabian (SA) and Egyptian (EG) [13].

Determining how similar two sentences are in meaning is a crucial part of comprehending natural languages automatically. The problem of semantic similarity involves evaluating the closeness of sentence meanings. To address this problem, recurrent and recursive neural networks have been used and have shown significant improvements over basic models. These neural networks are designed to handle the structure of language. Recurrent neural networks (RNNs) are suitable for processing sentences and understanding the relationships between words. Recursive neural networks (RecNNs) take this further by considering the hierarchical structure of sentences. By utilizing recurrent and recursive neural networks, there have been notable enhancements in measuring semantic similarity, with reported improvements ranging from 16% to 70% compared to basic models. This highlights the effectiveness of these neural network approaches in evaluating the similarity of sentence meanings. These advancements contribute to better automated language understanding and have various applications in tasks like question answering, information retrieval, and language translation [54].

Semantic Textual Similarity (STS) forms the foundation for numerous applications in Natural Language Processing (NLP). To measure the semantic similarity of sentences, a system combines convolutional and recurrent neural networks. It utilizes a convolutional network to consider the nearby context of words and a Long Short-Term Memory (LSTM) network to account for the overall context of sentences. By combining these networks, the system retains important sentence information and enhances the calculation of sentence similarity. The model

has demonstrated favorable outcomes and is competitive with the leading state-of-the-art systems, as indicated by reference [7].

this study[9] was attempted to develop an Amharic-English CLSTSM system by utilizing a Statistically topic modeling-based semantic text similarity measurement approach It helps native speakers of Amharic gauge the amount of web content available in Amharic by utilizing a query in their own language. The publicly accessible Amharic and English text materials that make up the similar and non-comparable collected documents were utilized to test the system prototype. By projecting the two texts into an LDA topic space and utilizing three distinct techniques to measure the similarity of the two text documents, the LDA topic model methodology is used to turn text documents into vectors. In varying data sizes, the Jaccard algorithm outperforms other matching algorithms with accuracy rates of 70%, 79%, 92%, and 96%. Additionally, on non-comparable corpora, the Jaccard algorithm surpasses other algorithms with accuracy rates of 65%, 78%, 92%, and 95.6.

To measure the Semantic Textual Similarity (STS) is an important study area in NLP which plays a significant role in many applications such as question answering, document summarization, retrieval of information and information extraction. This paper evaluates Siamese recurrent architectures, a special type of neural networks, which are used here to measure STS. Several variants of the architecture are compared with existing methods [55].

Table 2-1: List of related works

Year	Topic	Method	Accuracy	Datasets	Algorithm	Evaluation matrix	Gaps/Feature Research work
2021 [6]	"Word Embedding-based Textual Semantic Similarity Measure in Bengali"	word embedding techniques and cosine similarity	77.28% Pearson Correlation	187031 text documents	No weight, term frequency weighting, and Part-of-Speech weighting	Pearson's correlation (ρ)	Ambiguity words cannot consider and not use any RNN or CNN models
2022 [8]	"Semantic textual similarity for modern standard and dialectal Arabic using transfer learning"	transfer learning with BERT embedding	81% Pearson Correlation	100 pair of sentence	transfer learning	Pearson Correlation	Large gap in accuracy compared to English STS Model and lacks lexical standard for Arabic
2022 [54]	"Deep learning based semantic similarity detection using text data"	Combine LSTM and CNN with word embedding	70% Accuracy	404290 question pairs	LSTM and CNN	Precision, Recall and F1	Insufficient experiment detail (not explain about actual score and model predictions) and compare the purpose model with not related works
2018 [7]	"Predicting the Semantic Textual Similarity with Siamese CNN and LSTM"	CNN and LSTM	0.79 Pearson	9,927 sentence pairs	LSTM and CNN	Pearson (r) and Spearman	Not provide information about pre-training embedding and lack of annotated corpora

			Correlation			(ρ) correlation coefficients, and Mean Squared Error	
2019 [55]	"Semantic Textual Similarity with Siamese Neural Networks"	Siamese neural networks with word embedding	0.81 Pearson Correlation	9927 sentence pairs	Siamese Neural Networks	Pearson Correlation	better performance on smaller datasets and only training data available
2021 [8]	"Cross-Language Semantic Text Similarity Measurement using Statistical Topic Model: The Case of Amharic-English Languages"	The LDA topic model and Jaccard algorithm	96%	1200 comparable and non-comparable text	Cosine, Jaccard and Hellinger	Precision, Recall and F1	It primarily relies on word co-occurrence statistics and fails to incorporate the semantic meaning of the words. model may not always align perfectly with human interpretation

Summary of Related Work

Research on STS has been conducted in a different ways with varying approaches to foreign languages. Nevertheless, no deep learning study on STS for Guragigna and local languages has been done. As we examine multiple research conducted on languages other than those heavily recommended by the STS, like deep learning algorithms, as difference to other conventional STS approaches. The purpose of this work is to use deep learning techniques to the development of semantic textual similarity for the Guragigna language. Depending on the chosen LSTM, BI-RNN, GRU, and Stacked RNN model, we conducted the experiments using those models since, generally speaking, as we have seen in the works linked above, the majority of recent research has been conducted for various languages. To assess the effectiveness of the model, Mean Squared Error (MSE) is used as the evaluation metric. Which compares system output to reference sentences that have been manually scored in order to assess score correctness. We employed preprocessing approaches and optimization strategies to improve performance using either an MSE or training speed using a deep learning STS model, hence reducing the complexity of the work.

Based on the gaps described above Table 2-1, this thesis done as in the list ways to bridge the gap in STS models' accuracy, the thesis could explore alternative approaches and using different embedding's technique. Could investigate specific-domain pre-training techniques or leverage annotated datasets in Guragigna to improvement of performance of semantic similarity models. In addition, the thesis includes complete set-ups, providing test details and comparisons thesis aim to include thorough setups, detail descriptions of model architectures, hyper-parameters, and evaluation metrics. It compare their proposed models with relevant approaches, highlighting the strengths and weaknesses of each models based on the results.

Moreover, by collaborating with linguists or using different data sourcing platforms to create an annotated corpus, these models allow for more accurate and reliable evaluation. These efforts contribute to increasing the quality of research on semantic similarity of Guragigna languages.

CHAPTER THREE

3. MATERIAL AND METHODS

3.1. Introduction

This chapter explain the approaches, methods, tools, and techniques of proposed models in detail used in studies for the developing Semantic Textual Similarity (STS) for the Guragigna language model. This covers the material, tools, and technique utilized in the thesis as well as the research approach employed in the show which reflects the suggested method of the STS model. In these studies to accomplish the overall and particular goals of this investigation. We adhered to the sequence and followed different steps. These are data collection, a literature review from different journals, methods, the preparation of a corpus, preprocessing, training and testing of the proposed framework. The details of the techniques and algorithms reviewed and followed to build the STS model also be reviewed. The deep learning approach utilizes a large collection of Guragigna text data to extract relevant features and compute similarity scores between text pairs. On the other hand, incorporates linguistic resources specific to the Guragigna language to derive semantic relationships and calculate similarity based on domain-specific knowledge.

3.2. Proposed Approach

The research methodology employed in this study consists of several distinct stages. To begin, a diverse range of sentence pairs that contribute to the advancement of STS in the Guragigna language is collected from multiple sources. Next, a comprehensive review of relevant literature pertaining to previous STS studies is conducted. Following this, various data preprocessing techniques, such as eliminating excess spaces, stop words, punctuation, and tokenization are applied to the gathered data. The processed data is subsequently converted into a vector representation. Then a model is designed and constructed to handle the input specifically tailored for STS in the target language. The prepared dataset is utilized to train the proposed model. The learning model is subsequently evaluated to assess its performance, and depending on the evaluation results. The most optimal model for developing Semantic Textual Similarity (STS) in the Guragigna language is selected. Finally, the research findings and outcomes are documented in the thesis report.

3.3. Material and Tools

3.3.1. Hardware Tools

We used the selected Hardware tools such as Dell Desktop with Processor: Intel(R) Core(TM)

i5-6200U CPU @ 2.30GHz 2.40 GHz, 4GB dual RAM, 500 GB dual hard disk, screen size: 15 inches, Flash disk 64 GB, Wi-Fi Device: 4G lite.

3.3.2. Software Tools

Software tools are used to implement the research because they can help researchers carry out their work more efficiently and accurately. The software tools used depend about the subject of study and the type of research being conducted. We considered some Standards for tools selecting that are helpful in selecting the good tools of software including their related package or libraries. The other Standards are to choose tools having access to sufficient educational resources, including free video tutorials, and prior expertise and the other is to select tools that must be used on machines with limited resources. We used programming language which is Python with its Package and libraries environment to implement the neural layers. These tools fulfill all the consideration criteria and they are used in Python.

Google Colab: is based in the cloud platform that offers a free environment for Jupiter notebooks for running Python code. Google has provided that users to run Python code using Google's computational resources including GPUs and TPUs without the required for any local hardware or software installation. And Integration with Google Drive: Google Colab and Google Drive are integrated enabling to access notebooks and data from anywhere. While Google Colab is popular platform for running python code online and provides basic python environment some commonly used libraries like Tensor flow, Pandas, Numpy etc.

Python: Python is a popular programming language for machine learning tasks including STS development. It provides a rich ecosystem of libraries and frameworks for data processing, feature extraction, model training, and evaluation.

Tensor-Flow is a powerful tool for deep learning that can be used effectively in tasks related to semantic textual similarity (STS). It provides a wide range of features and capabilities to process text create word representations build models for sentences, train deep learning models, and performance. It can be combined with other Python libraries such as NLTK or spacy to handle tasks like splitting text into smaller parts or removing common words. TensorFlow offers different methods to generate word representations including creating your own or using pre-existing ones like Word2Vec or GloVe. For sentence representation, TensorFlow allows you to construct models using recurrent neural networks (RNNs) architectures. These models can transform sentences into fixed-length representations that capture their meaning. TensorFlow provides user-friendly tools like Keras and TensorFlow Hub making it easy to access pre-trained models or build your own custom ones.

NLTK (Natural Language Toolkit): NLTK provides various NLP functionalities including tokenization, stemming, lemmatization, and syntactic parsing. It used to preprocess and analyze the Guragigna text data.

Table 3-1 Tools and materials

Tools and material	uses
Anaconda with integrated Python	To implement and run model code
HP PC: 14-inch, 2.50 GHz, Intel Corei5 CPU- 8G, 8GB RAM, 1 Tera HDD	To process multiple tasks of the research and as storage
External storage such as Flash disk	To store data/files
Microsoft office word	To write documentation and
Microsoft PowerPoint	Prepare presentation PowerPoint
Mendeley	For reference Citation
Notepad++	To prepare dataset and text processing
Pen and paper	To take a note and printing
Google Colab	To compile and execute Python code
Google Drive	To upload dataset

In addition, Viso, and the online diagrams tool used to draw diagrams of these studies and web browsers like Google Chrome and, Micro soft Edge also be used for searching source. we used different tools in order to build the proposed model.

3.4. Corpus

In the process of developing Semantic Textual Similarity (STS) for the Guragigna language a diverse and comprehensive corpus has been carefully edited. The collection of this corpus involved use various sources to gathering data and domains to ensure a representative sample of the language's semantic landscape. The corpus comprises a wide range of texts, including but not limited to bible, news articles, literary works, social media posts, Guragigna fiction books, history, and dictionary. By including texts from different domains, the corpus captures the semantic variations and contextual nuances prevalent in the Guragigna language. This diversity enables the STS system to handle semantic similarity across diverse topics and genres, enhancing its applicability in real-world scenarios. To ensure a substantial corpus data was sourced from reputable and authentic linguistic resources such as books, newspapers, websites, and language-specific repositories. These sources provide reliable language data reflective of natural language usage and contribute to the corpus's credibility. The corpus also incorporates

annotated data where human experts have provided similarity judgments at the sentence level. This annotated data serves as a benchmark for evaluating and training the STS models enabling the development of accurate and reliable algorithms.

Developing a Guragigna STS corpus in collaboration with the Gurage Zone Cheha Wereda Communication Office, the Culture and Tourism Office, the Cheha Wereda Betekihinet Office (ቸሃ ወረዳ ቤተክነት ጽ/ቤት), and the Education Office involves a systematic and collaborative approach. Upon contacting these offices a comprehensive explanation of the annotation process is provided to convey the purpose and potential benefits of the thesis. The staff from each office is then equipped with the necessary guidelines to ensure they are proficient in assessing the similarity between Guragigna sentences. Through annotation sessions the staff diligently evaluates sentence pairs and assigns similarity scores based to the established guidelines. A quality control process is implemented to verify the accuracy and consistency of the annotations including regular meetings for clarifications and calculating to ensure reliability. Finally, the annotated data is organized into a well-structured corpus respecting data privacy and ethical considerations throughout the entire process.

The collaboration with the Gurage Zone Cheha Wereda Communication Office, the Culture and Tourism Office, the Cheha Wereda Betekihinet Office, and the Education Office enables the development of a valuable resource for Guragigna STS research. By leveraging their expertise and local knowledge the corpus benefits from the staff's linguistic insights and understanding of the specific linguistic nuances present in the Gurage Zone Cheha Wereda region. This partnership fosters a collaborative environment where knowledge and expertise are shared resulting in a corpus that reflects the linguistic diversity of the Guragigna language. The process ensures that the Guragigna STS corpus is meticulously annotated. Providing a foundation for training and evaluating deep learning models in the domain. The efforts put into establishing this collaboration and following the outlined steps yield a linguistically informed and reliable resource that can be utilized for various applications in computational linguistics and NLP.

Furthermore, the corpus takes into account the linguistic phenomena present in the Guragigna language. It includes a balanced representation of morphology, syntax, semantics, and discourse allowing for a comprehensive analysis of semantic relationships and similarity at various linguistic levels. The combination of a carefully curated corpus, encompassing diverse sources, domains, and linguistic phenomena, forms the foundation for deep learning approaches in developing STS for the Guragigna language. With this rich linguistic resource

researchers can explore and model the semantic aspects of the language thereby advancing the understanding and application of semantic textual similarity in Guragigna.

In addition a valuable contribution has been made by collecting a dataset comprising more than 3000 pairs of sentences. These sentence pairs were meticulously evaluated by linguistic experts from the Cheha Werda. The dataset encompasses a diverse range of sentence pairs covering various semantic relationships and contexts within the Guragigna language. Each pair of sentences was carefully crafted to represent different levels of similarity. The linguistic experts from the Cheha Wereda played a crucial role in evaluating the semantic similarity of the sentence pairs. Their expertise and deep understanding of the Guragigna language allowed for accurate and reliable judgments regarding the similarity degree between the sentences. Through their linguistic expertise the experts provided insightful annotations and similarity scores for each sentence pair. These annotations serve as valuable ground truth data for training and evaluating STS models enabling the development of robust and accurate algorithms specifically tailored to the Guragigna language. The involvement of linguistic experts from the Cheha Wereda in evaluating the dataset ensures the cultural and linguistic authenticity of the annotations. Their expertise and native understanding of the language contribute to the overall dataset quality and dataset reliability improve the effectiveness of STS models in capturing semantic similarity in the Guragigna language. The dataset evaluated by linguistic experts from the Cheha Wereda forms a significant resource for the development of STS in Guragigna. It provides a solid foundation for training and evaluating STS models allowing for the advancement of natural language understanding and applications in the Guragigna-speaking community.

The labels provided by the Gurage Zone Cheha Wereda assessors namely the Communication Office, the Culture and Tourism Office, the Cheha Wereda Betekihinet Office, and the Education Office. Serve as a way to express their subjective judgments regarding the similarity between the sentence pairs. These labels encompass a wide range of perspectives reflecting varying degrees of agreement or disagreement with the similarity observed. Labeling from "Very strongly disagree" up to "Very strongly agree" the labels offer a nuanced and graded assessment of semantic similarity. The assessor’s expertise in communication, culture and tourism local administration, and education brings valuable insights into the evaluation process. To combine the diverse viewpoints of these assessors a comprehensive understanding of the semantic similarity between the text pairs can be achieved. Enabling a robust comparison with the similarity scores provided by the model.

The assessors who are experts in the field assigned similarity scores between 0 and 1 for each sentence pair depended on the Response Label. These scores were used to evaluate the model's ability to measure semantic similarity. To facilitate a comparison between the model's performance and human judgments the scores were normalized between 0 and 1 using an average labeling score approach. This normalization process allows for a standardized assessment of the model's performance in relation to the assessors' judgments of similarity.

Label 0.0: Indicates a deep disagreement regarding the similarity between text pairs.

Label 0.1: Reflects a strong disagreement with the similarity between text pairs.

Label 0.2: Represents a disagreement with the similarity between text pairs.

Label 0.3: Implies a direction to disagree with the similarity between text pairs.

Label 0.4: Suggests a partial disagreement with the similarity between text pairs.

Label 0.5: Indicates a neutral stance agreement with the similarity between text pairs.

Label 0.6: Suggests a partial agreement with the similarity between text pairs.

Label 0.7: Implies a tendency to agree with the similarity between text pairs.

Label 0.8: Represents an agreement with the similarity between text pairs.

Label 0.9: Reflects a strong agreement with the similarity between text pairs.

Label 1.0: Indicates a profound agreement with the similarity between text pairs.

Once the assessors finished their annotations, we conducted an analysis of the variance in their similarity scores. Text pairs that exhibited a high degree of variance were deemed perplexing and subsequently excluded from further analysis.

3.5. Preprocessing

3.5.1. Removing extra spaces

In process of develop STS for Guragigna removing extra spaces from the corpus is a crucial preprocessing step. Extra spaces, such as leading, trailing, and consecutive spaces within sentences can introduce noise and impact the accuracy of similarity measurements. To address this, a systematic way is used to remove extra spaces throughout the corpus. This involves applying string manipulation techniques or regular expressions to identify and replace consecutive spaces with a single space. Leading and trailing spaces are also eliminated to maintain consistency. By employing this preprocessing technique the collected corpus is

effectively cleaned, ensuring accurate and reliable data for subsequent stages of feature extraction, modeling, and evaluation. This enhances the quality of the corpus and contributes to the development of robust and accurate STS models for Guragigna. Here's the Algorithms to remove extra spaces from a Guragigna string [56].

Algorithm 3-1 Algorithms for remove extra spaces from dataset

Input: pair of Guragigna sentences
Remove extra spaces from sentence 1
clean_sentence1 = ' '.join(sentence1.split())
Remove extra spaces from sentence 2
clean_sentence2 = ' '.join(sentence2.split())
Output: pair of sentences without extra spaces

In the above Algorithms, apply the same process to each sentence individually. Split each Guragigna sentence using the split method which splits the string on whitespace characters (including extra spaces) and returns a list of words. Then we use the join method to join the words back together using a single space as the separator effectively removing any extra spaces between the words. Finally print the resulting clean sentences [56].

3.5.2. Removal of stop-words

In the development of Semantic Textual Similarity (STS) models for Guragigna, removing stop words from the corpus is an important preprocessing step. Stop words are commonly used words that do not carry significant semantic meaning. By excluding these words the focus is shifted to content words that convey more substantial information resulting in more accurate similarity measurements. To achieve this a custom list of Guragigna stop words (Appendix C) is prepared including articles, prepositions, pronouns, and other function words. During preprocessing each sentence is tokenized into words, and each word is compared against the stop words list. If a token matches a stop word it is removed otherwise, it is retained. This process eliminates noise and irrelevant words ensuring that the subsequent analysis captures the true semantic similarities. The cleaned corpus free from stop words forms the basis for reliable and insightful STS modeling in Guragigna [57].

Algorithm 3-2 Algorithms for remove stop words from dataset

Custom list of Guragigna stop words
Input: pair of Guragigna sentences
Tokenize sentence 1
tokens1 = sentence1.split()
Tokenize sentence 2
tokens2 = sentence2.split()
Remove stop words from sentence 1
filtered_tokens1
If token1 not in guragigna_stop_words]
Remove stop words from sentence 2
If token2 not in guragigna_stop_words]
Output: lists of tokens without stop words

In the above Algorithms, define a custom list of Guragigna stop words in the Guragigna stop words variable. Then, tokenize each sentence by splitting them into individual words. After that use list comprehension to filter out any stop words from the tokenized sentences based on the custom stop words list.

3.5.3. Removing punctuation

Removing punctuation marks is a common preprocessing step in natural language processing. In Guragigna language processing it is main activities to remove punctuation for accurate analysis. By eliminating punctuation marks like periods, commas, and question marks we can focus on the essential words and structures. To remove punctuation we can define a list of Guragigna punctuation marks (Appendix B) and use string manipulation techniques. By iterating through the text and removing identified punctuation marks. We obtain cleaned text ready for further processing. Customizing the list of punctuation marks ensures precise removal while preserving the integrity of the Guragigna text.

Algorithm 3-3 Algorithms for remove punctuation from dataset

Input: pair of Guragigna sentences
Custom list of Guragigna punctuation marks
Remove punctuation from sentence 1
If char in sentence1 not in guragigna_punctuation
Remove punctuation from sentence 2
If char in sentence2 not in guragigna_punctuation

In the above Algorithms, the custom list guragigna_punctuation contains the Guragigna punctuation marks provided. Effectively removing those punctuation marks from the sentences Print (sentence1_cleaned and sentence2_cleaned) [58].

3.5.4. Tokenization

Tokenization plays a fundamental role in the implementation of Semantic Textual Similarity (STS) models for the Guragigna language. By breaking down the text into tokens, such as words or sub-words tokenization enables the analysis and comparison of semantic meaning between pairs of sentences. In the context of Guragigna STS the tokenization process involves several important considerations. Firstly, preprocessing steps are applied to the text including diacritic removal, and handling language-specific cases unique to Guragigna. These steps ensure consistency and accuracy in subsequent tokenization.

Next, sentence segmentation is performed to separate the text into individual sentences. In Guragigna sentences are typically delimited by punctuation marks like periods, question marks, or exclamation marks. Sentence segmentation enables the comparison of semantic similarity at the sentence level. Subsequently, word tokenization is applied to each sentence splitting them into individual tokens. In Guragigna words are often separated by spaces or specific characters. Language-specific rules and patterns are employed to correctly identify word boundaries and ensure accurate tokenization. Guragigna language exhibits the use of compound words where multiple words fuse together to convey a single concept. Handling compounds during tokenization is crucial. Depending on the specific STS approach compounds can be either split into constituent parts or treated as single tokens. Careful consideration is required to capture the intended semantic meaning. This helps to ensure that similar words are represented consistently contributing to more accurate STS modeling.

The resulting tokenized representations provide a foundation for STS modeling in Guragigna. Semantic similarity scores can be calculated between pairs of tokenized sentences facilitating the measurement of their similarity or relatedness. Fine-tuning the tokenization process based on linguistic expertise, available resources, and the specific requirements of the STS task is crucial for achieving reliable and meaningful results in Guragigna STS development. To tokenize a pair of Guragigna sentences can use the word_tokenize function from the nltk.tokenize module. Here's the Algorithms to tokenize a pair of Guragigna sentences:

Algorithm 3-4 Algorithms for tokenization a dataset

Input: pair of Guragigna sentences
Tokenize sentence 1
tokens1 = word_tokenize(sentence1)
Tokenize sentence 2
tokens2 = word_tokenize(sentence2)
Output: lists of tokens

In the above Algorithms, import the necessary modules from NLTK. Then define the input sentences as Guragigna strings. Next, use the word_tokenize function to tokenize each sentence into words. The word_tokenize function splits the sentence into individual words and returns a list of tokens. Finally print the lists of tokens for each sentence.

3.6. Universal Sentence Encoder (USE)

The approach involves creating an encoder that condenses any sentence into a 512-dimensional representation called a sentence embedding. This sentence embedding is used for different tasks, and by learning from the mistakes made on those tasks the sentence embedding is updated. Because the same embedding is applied to various tasks it focuses on the most important features and disregards irrelevant details. The idea is that this process leads to a versatile embedding that can be used effectively across different NLP tasks including STS as mentioned in reference [59].

USE (Universal Sentence Encoder) is a deep learning model. It is designed to encode sentences or short texts into fixed-length numerical vectors. Which can then be utilize for a various natural language processing (NLP) tasks. The key idea behind USE is to capture the semantic meaning and contextual information of a sentence in a compact vector representation. This allows sentences with similar meanings to have similar vector representations even if the words and structures used in sentences are different [59].

The USE model is trained on a different amount of diverse text data including books, and other sources to learn universal sentence representations. It uses a deep neural network architecture that leverages the power of transfer learning to generate meaningful and semantically rich vector representations of sentences. The model architecture consists of Encoder that takes a

sentence as input and transforms it into a fixed-length vector representation. It uses a RNNs capture both the sequential and contextual information of the input sentence and the transformer component further refines the encoded representations by attending to different input parts of sentence. It uses self-attention mechanisms to capture the relationships between words and phrases in the sentence allowing for a more comprehensive understanding of the sentence semantics. The USE model is particularly useful in scenarios where the semantic similarity or relatedness between sentences require to be evaluated. It can be used for tasks such as sentence similarity calculation and more. It's important to note that while the Universal Sentence Encoder is a powerful tool for many NLP tasks. It is not capable of understanding the meaning of sentences in the same way humans do. It relies on patterns learned from the training data and may not capture certain nuances or context-specific information [59].

Algorithm 3-5 Algorithms for Universal Sentence Encoder (USE)

```
Encoded_dataset
For pair in dataset:
Unpack the pair
Tokenize sentence 1
Tokenize sentence 2
Combine tokens into sentences (if needed)
Encode sentence 1
Encode sentence 2
Append the encoded pair to the result
Return encoded_dataset
```

3.7. Word embedding

Word embedding’s are a type of word representation that helps machines understand language in a way that is similar to how humans do. They are learned representations of text in a multi-dimensional space, where words with similar meanings are represented by similar vectors. This means that words that are alike in meaning are placed close to each other in this vector space. Word embedding’s play a crucial role in solving many natural language processing problems as they provide a way for machines to capture the meaning and relationships between words which is essential for various language-related tasks.

3.7.1. Global Vectors for Word Representation (GloVe)

This technique helps reduce the computational burden of training the model by utilizing a simpler least square cost or error function. This simplified approach leads to the creation of different and enhanced word embedding. The technique combines local context window methods, such as the skip-gram model proposed by Mikolov. With global matrix factorization methods to generate word representations in lower dimensions. By employing these methods the technique aims to produce efficient and effective word embedding [60].

GloVe (Global Vectors for Word Representation) is an alternative method for creating word embedding. It utilizes matrix factorization techniques on the word-context matrix. To construct this matrix a large corpus is analyzed to determine the co-occurrence information between words. Each word is counted based on how frequently it appears in a specific context. The context is defined by a window size that includes words before and after the target word with less emphasis placed on more distant words. This process is performed for all terms in the corpus. Since the number of contexts is extensive and combinatorial in size the matrix becomes large. To overcome this, the matrix is factorized to obtain a lower-dimensional representation. Each row in the resulting matrix represents a vector that serves as a representation for a word. Typically, this factorization is achieved by minimizing a "reconstruction loss" that aims to find lower-dimensional representations capable of explaining most of the variance in the high-dimensional data [60]. GloVe employs a weighted least squares objective function denoted as J, which aims to minimize the difference between the dot product of two word vectors and the logarithm of their co-occurrence count. The objective function is defined as follows:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(\omega_i^T \grave{\omega}_j + b_i + \grave{b}_j - \log X_{ij})^2$$

In this equation, ωi represents the word vector and bi represents the bias term associated with word i. Similarly, ωj represents the context word vector, and bj represents the bias term associated with word j. Xij denotes the number of times word i occurs in the context of word j. The function f assigns weights to the co-occurrence counts, giving lower weights to rare and frequent co-occurrences [60]. By minimizing this objective function, GloVe optimizes the word vectors and biases to capture meaningful relationships between words based on their co-occurrence information. Reference [59] provides further details on the weighting function and the specific formulation of the objective function.

GloVe embedding's is use for Semantic Textual Similarity (STS) to Pre-trained Word Embedding's on large corpora these embedding's capture global statistical properties of word co-occurrences by making them suitable for a wide level of NLP tasks including STS. By leveraging pre-trained embedding's can benefit from the knowledge and semantic connection learned from extensive text data and GloVe embedding's provide encode semantic connection with words. Words with the same meanings vector representations. This property allows GloVe embedding's to capture the semantic similarity between words which is crucial in STS tasks where the goal is to determine the similarity or relatedness between sentences. Another use of GloVe embedding's is to provide fixed-length representations of vector for words. This allows for efficient computations and comparisons between sentences. By averaging or aggregating the word embedding's in a sentence can obtain a fixed-length representation for the entire sentence which can then be used for similarity calculations. GloVe embedding's have good coverage for in common words language since GloVe is trained on large corpora. It tends to have embedding's for a wide range of frequently occurring words. This ensures that common words in STS tasks are likely to have meaningful representations. Finally using pre-trained GloVe embedding's can leverage transfer learning to improve STS performance. The embedding's capture general semantic relationships that are transferable across various NLP tasks. By fine-tuning STS model on specific data can adapt the pre-trained GloVe embedding's to specific STS domain and achieve better results. While GloVe embedding's have their advantages, it's important to note that they have limitations. They may not fully capture the compositional and contextual aspects of sentences as they are context-independent word embedding's. This can affect their performance on certain STS tasks that require a deeper understanding of sentence-level semantics. In such cases, more advanced contextual embedding's or models may be more suitable [60].

Algorithm 3-6 Algorithms for Global Vectors for Word Representation (GloVe)

Sentence pair
Load pre-trained GloVe embeddings
Tokenize and preprocess the sentences
Get word embeddings for each token
Calculate the sentence embeddings by averaging word embeddings
Calculate similarity between the sentence embeddings
Return similarity

3.7.2. Word2Vec (Word to Vector)

Word2Vec, a powerful algorithm for learning word embedding's can be effectively applied to Semantic Textual Similarity (STS) tasks in the Guragigna language. To begin a diverse corpus of Guragigna text should be collected, covering various topics and domains to ensure a representative dataset. This corpus is then preprocessed by removing punctuation, special characters, and irrelevant information, and tokenizing the text into individual words or morphemes. The Word2Vec model is trained on this preprocessed corpus model based on Guragigna specific characteristics [6].

The trained model generates word embedding's which represent words as dense fixed-length vectors capturing their semantic meaning in context. When comparing Guragigna texts for similarity to use these word embedding's that convert the texts into vector representations. Similarity scores between texts can be calculated using metrics. The model's performance is evaluated using appropriate evaluation datasets or human judgments, and fine-tuning can be done to adjust parameters and incorporate domain-specific data. The Word2Vec-based STS model can then be applied to various Guragigna NLP works. It is important to consider the availability of a representative training corpus, linguistic complexities, and the need for fine-tuning and achieve optimal performance in the Guragigna language [6].

Algorithm 3-7 Algorithms for Word2Vec (Word to Vector)

Data Preparation of Guragigna
Tokenize the text corpus
Vocabulary Construction
Word Context Extraction
Model Training

3.8. Optimization Algorithms in STS using deep learning

When training Semantic Textual Similarity (STS) deep learning models optimization algorithms are used to update the model's parameters and minimize a loss function that quantifies the discrepancy between predicted and target similarity scores. RMSprop (Root Mean Square Propagation) is an optimization algorithm that also adapts the learning rate for each parameter. It uses exponentially decaying averages of past squared gradients to normalize the learning rate. RMSprop helps to alleviate the diminishing learning rate problem and has been effective in training deep learning models for STS.

3.9. Performance Measurement Methods

3.10. Accuracy

Performance measures the efficiency and reliability of proposed model in a given context of use. The overall performance of the STS system should be evaluated to what extent matching scores are correctly or incorrectly matched based on semantic.

$$\text{Accuracy} = \frac{\text{Correctly matched}}{\text{total number of texts}} * 100$$

Correctly matched is the quantity of similar sentence which is match correctly

3.11. Evaluation Metrics

3.11.1. Mean Squared of Error (MSE)

Mean Squared of Error (MSE) is a commonly used evaluation metric for regression tasks, including Semantic Textual Similarity (STS) tasks. It quantifies the average squared variance of the predicted similarity scores and the true similarity scores.

The formula for calculating MSE is as follows:

$$\text{MSE} = \frac{1}{n} \times \sum((y\_i - \hat{y}\_i))^2$$

Where n is the total number of instances or samples, y_i represents the true similarity score for the i-th instance, ŷ_i represents the predicted similarity score for the i-th instance, and Σ denotes summation over all instances,

Here's an explanation of the steps involved in calculating MSE:

1. For each instance in dataset, calculate the variance of the true similarity score (y_i) and the predicted similarity score (ŷ_i).
2. Square each difference to ensure that they are positive and to give more weight to larger differences.
3. The squared variance of Sum of all.
4. to find the average squared difference, Divide the sum by the total number of instances (n)

MSE provides a measure of the average magnitude of the errors made on model. A lower MSE indicates that the model's predictions are closer to the true similarity scores on average indicating better performance. However, it is important to consider the scale and context of the STS task to interpret the MSE appropriately.

When using MSE as an evaluation metric, it is often helpful to compare it with other metrics such as Spearman's rank correlation coefficient or Pearson correlation coefficient to gain a more comprehensive understanding of the model's performance in STS tasks [61].

3.11.2. Process of adapting a pre-trained model

In the context of Semantic Textual Similarity (STS) overlay may be occur when the model becomes too specific to the training data and performs poorly on new and unseen data. To address this issue, regularization techniques can be applied. Regularization adjusts the weights and biases of the model during training to minimize the loss function and improve accuracy. It is widely used to train neural networks efficiently.

In STS tasks, the rate of learning plays a crucial role in determining how much the model's weights are updated during training. It controls the step size and influences the speed of convergence and loss function is sensitivity to changes. Learning rate selecting is important as different learning rates can significantly impact the model's performance.

Another consideration in STS Process of adapting a pre-trained is the batch size which determines the quantity of examples processed in each iteration before updating the model's weights. The batch size affects the training dynamics, memory requirements, and convergence speed. Smaller batch sizes can introduce more stochastic updates, while larger batch sizes provide more stable gradients but may require more memory. The quantity of epochs is also an important aspect in STS Process of adapting a pre-trained. It refers to the quantity of model iterates times over the data training too small epochs may result in under fitting. Where the model unable to identify the underlying patterns overfitting is happened too many epochs where the model becomes too specific data training. Finding the right balance of epochs is crucial for achieving optimal performance in STS tasks.

Process of adapting a pre-trained model for STS tasks it is important to address overfitting by applying regularization techniques. Additionally, selecting an appropriate learning rate, batch size, and number of epochs can significantly impact the model's performance and help achieve better results.

CHAPTER FOUR

4. RESEARCH DESIGN

4.1. Corpus Preparation

In developing Semantic Textual Similarity (STS) for the Guragigna language collecting text pairs from the Holy Bible and primary textbooks is a valuable approach. These sources are often considered authoritative and can provide diverse and representative language usage. To begin select a representative subset of text pairs from the collected sources. Ensure that the selected pairs cover various topics, genres, and styles to capture the diversity of language usage in different contexts. Next, align the sentences in the text pairs to create corresponding sentence pairs. This step ensures that each sentence in one text corresponds to its appropriate counterpart in the other Sentence alignment is crucial for training STS models as it establishes the ground truth for similarity judgments. Develop annotation guidelines or instructions for annotators who assess the semantic similarity of the sentence pairs. Clearly define the criteria and scale for similarity judgments. Provide examples and clarify any language-specific nuances or challenges that may arise during the annotation process. Employ human annotators who are proficient in Guragigna to evaluate the similarity with sentence pairs based on the provided guidelines. Annotators should assign similarity scores or labels to each pair indicating the degree of semantic similarity. Ensure that the similarity distribution in the annotated corpus is balanced aim for an equal representation of various similarity levels in sentence pairs. This balance helps in training a model that can perform effectively differentiate in each similarity levels. Apply the preprocessing steps discussed earlier to the sentence pairs. Additionally, format the data into a suitable format, such as xslx or txt, with each pair accompanied by its similarity label. Split the annotated corpus into training and test sets following a suitable ratio based on the size of dataset. Ensure that the distribution of similarity levels is maintained in both training and test sets to avoid any biases during model evaluation. By following these steps can create a well-prepared and annotated corpus from the Holy Bible and primary textbooks for training Guragigna STS model. This corpus provide a valuable resource for developing and evaluating the model's performance in capturing semantic similarity in the Guragigna language.

4.2. Architecture of Developing STS for Guragigna Language

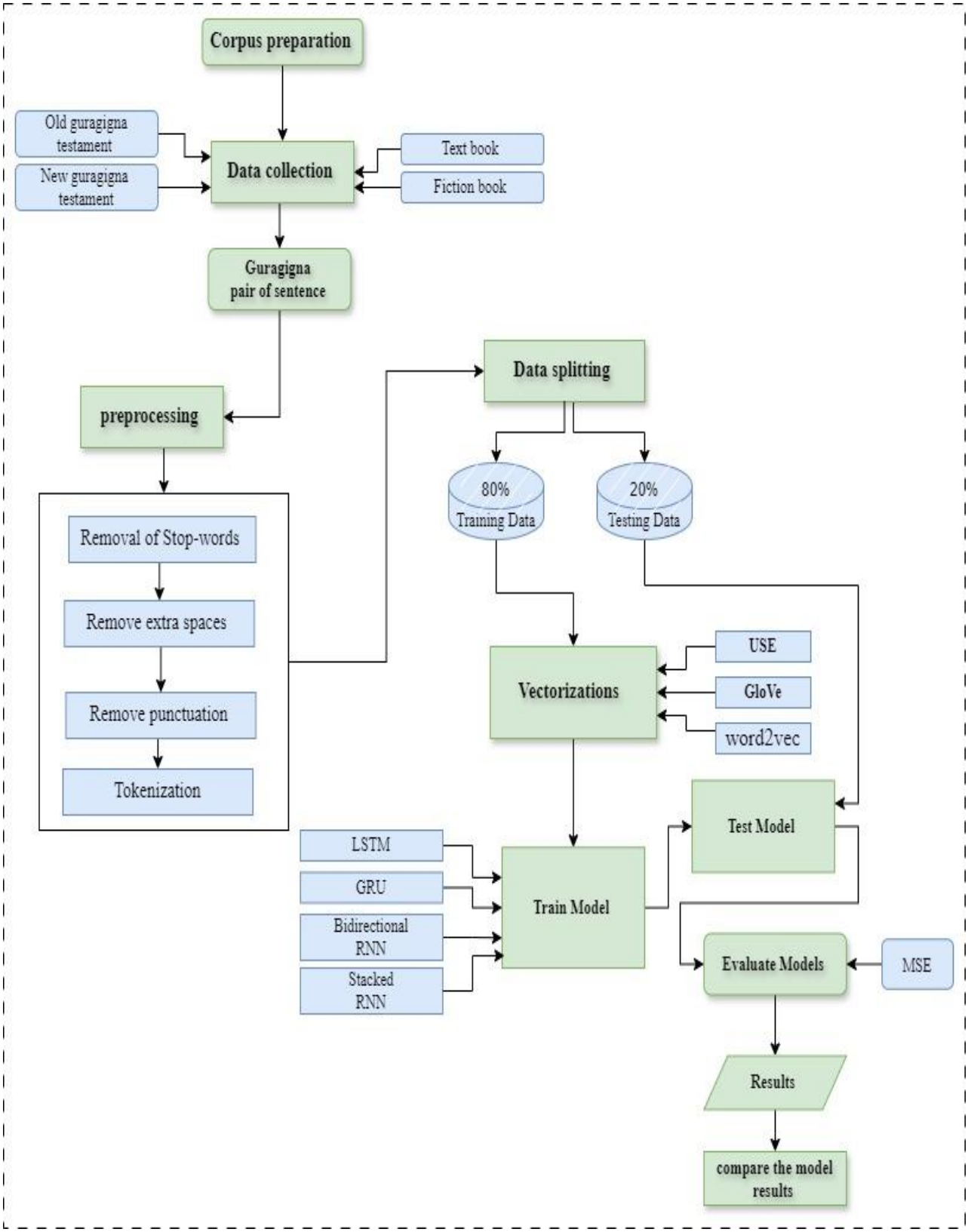


Figure 4-1 Architecture of Developing STS of Guragigna Language

4.3. Pre-processing

Pre-processing is a critical step in developing Semantic Textual Similarity (STS) models for the Guragigna language. To prepare the text data for analysis and modeling several key pre-processing steps are typically employed. The first step is avoid non-significant words for semantic meaning. The stop words are prepositions, articles, and conjunctions. By avoid these words we can minimize noise and focus on the more meaningful content words. Next, removing extra spaces is important to ensure consistency and proper tokenization. In Guragigna, where words are often written without spaces between them removing extra spaces helps to segment the text into individual tokens accurately. Another crucial pre-processing step is removing punctuation marks. Punctuation, such as commas, periods, and quotation marks, serves grammatical purposes but does not contribute directly to semantic similarity. By eliminating punctuation we can simplify the text and facilitate the subsequent analysis.

Lastly, tokenization is performed to split the sentence into single words which is tokens. Tokenization establishes the basic units of analysis and allows the model to process and compare words effectively. By following these pre-processing steps avoid stop words, removing extra spaces, removing punctuation, and tokenization. We can prepare the Guragigna text data for more analysis and modeling. These steps help to clean and structure the text, enabling the STS model to capture meaningful semantic relationships and similarities between sentences accurately.

4.4. Data Splitting

For our proposed study, we utilized a 3,000 sentence pairs. To prepare the data for analysis we divided them into two sets a training set and a testing set. Before splitting the dataset we applied a shuffling process to ensure randomness. This step was crucial as it helps minimize bias and overfitting by randomizing the data order of points in the rearranged dataset. We partitioned the dataset into an 80% training set and a 20% testing set. This split ensures that 80% of the data is used for training the model while 20% is reserved for evaluating its performance.

4.5. Vectorization

In the implementation of Semantic Textual Similarity (STS) for the Guragigna language, the significance of vectorization is crucial particularly in the context of Semantic Textual Similarity (STS). Vectorization involves representing words or sentences as processed by machine learning models in numerical vectors. This transformation enables the models to effectively analyze and understand the semantic relationships within the text.

There are several common approaches to vectorization that have been implement in this thesis. Widely used approach for vectorization is word embedding. In a continuous vector space Word embedding express words as dense vectors. This method take the semantic connection and conceptual information between words. Notable word embedding models, such as word2vec, GloVe, and USE have been commonly employed in this thesis for vectoring Guragigna language. Word embedding models are trained on sentence corpor facilitate them to learn the distributed words representations with in their co-occurrence patterns. These learned embedding's enable the models to capture semantic similarities and relationships between words facilitating more accurate analysis and the sentence understanding.

By implement vectorization methods like word embedding the STS system for the Guragigna language can effectively represent words or sentences as numerical vectors. These vectors capture the semantic information enabling models of deep learning to compare and measure the semantic similarity between texts accurately. Vectorization make a main role in developing Semantic Textual Similarity (STS) for the Guragigna language particularly in the context of STS. Approaches word embedding including word2vec, GloVe, and USE, are commonly used for vectorization. Word embedding in particular allows for the representation of words capturing the semantic relationships between words. This enables accurate analysis and measurement of semantic similarity between Guragigna texts.

4.6. Model Selection

4.6.1. Long Short-term Memory Model

The architecture for Semantic Textual Similarity (STS) is called Long Short-Term Memory (LSTM) that is a main class of RNN and in context of STS mainly used for measuring the degree of similarity. RNN is a kind of neural network whose input is sequence data (x1... xT) and connections between nodes form a directed graph along a sequence. It is widely used in NLP, speech recognition, and generating image descriptions recently. The input of the hidden layer includes the output of the input layer and the output of the hidden layer at the previous moment. The equation is $St = f(U \times Xt + W \times St - 1)$, in which Xt is the input at time t, U is the matrix of weight from current hidden state input, W is the weight matrix from previous hidden state to current hidden state and f is activation function. However, RNN has It gradually disappears problems in that small over long sequences in back propagated process to make gradient-based training learning algorithms difficult for long distance learning. To solve gradient vanishing, long short term memory [62] comes into being. LSTM is a variant of RNN that can learn long-term dependencies. Hidden unit is only a single neural network layer unit

in RNN. However, LSTM hidden unit is collection of a memory cell, an input gate, an output gate and a forget gate. Memory cell is used for storing a value or multiple values, input gate decides how many values enter the unit, output gate decides how many values output from unit, forget gate decides whether value remains in the unit. LSTM encoder is an unsupervised artificial learning neural network. It also has one input, output layer, and more hidden layers. The output of output layer fit the input of input layer as much as possible. It essentially compresses features and then decompresses them. It is mainly used for data dimensionality reduction and extraction of feature. It is now also used in generative models and made up of encoder and decoder. In general, the output of encoder is take in the data of feature. Figure 4.2 is Schematic diagram of LSTM encoder. To leverage semantic information and sequence information of short text we propose a algorithm based LSTM encoder to calculate short text similarity. It doesn't require labeling short text. Unlabeled short text can directly feed into our algorithm. Our algorithm includes preprocessing, training, and evaluation stage[62].

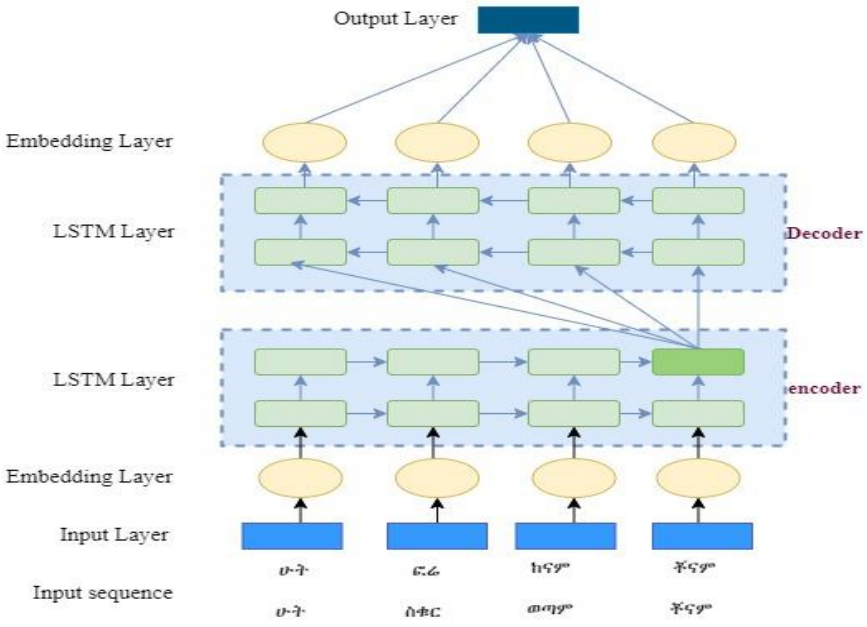


Figure 4-2 LSTM Encoder Decoder Architecture

Given a source sentence X = (x*1*; x*2*… x**l**) and the target (simplified) sentence Y = (y*1*; y*2*... y**l**0), where both vocabulary xi and vocabulary y**i** are the same, l and l*0* are the length of each sentence. Our goal is to build a neural network to model the conditional probability p(Y *j*X), then train the model to maximize the probability.

We show our LSTM Encoder-Decoder model in Figure 4.2 This model uses to words sequence representation in the input layer, and converts it to a 300 dimensional vector in the following

embedding layer. We find that adding an embedding layer can significantly improve performance when the vocabulary becomes large.

Then we feed word embedding through two LSTM layers, and get a vector representation of the input sequence after finishing reading listed words. Finally, we decode this vector to output sequence through two LSTM layers and one output embedding layer. Let us take the input sentence “ሁት ስቁር ወጣም ቾናም” and “ሁት ፎሬ ከናም ቾናም” as a difficult sentence represent the pair of sentences as a pair of word indices

We only apply sorting, reversing, and replacement to the indices to simplify a sentence, where sorting and reversing could be highly related to changing the structure of a sentence or simplifying a grammar replacement could be highly related to lexical simplification or removing redundant words. Motivated by this observation we conduct experiments to show the LSTM Encoder Decoder able to learn these three rules automatically, and thus can potentially perform text simplification.

4.6.2. Bi-directional RNN

BRNN is an architecture that designed for sequential data process and is particularly useful for developing Semantic Textual Similarity (STS) models for Guragigna. Unlike conventional recurrent neural networks BRNNs process input sequences in both the forward and backward directions allowing them to utilize information with future and past contexts in their predictions. A BRNN use of two unique hidden layers recurrent. First forward of input sequence processes then it is processed backward by the other. these hidden layers from outputs are then collected and fed into a final prediction-making layer[63].

In the forward direction, the Bidirectional Recurrent Neural Network (BRNN) works similarly to traditional recurrent neural networks. It updates the hidden state at each time step based on the current input and the previous hidden state. However, the backward hidden layer analyzes the input sequence in the opposite way. It updates the hidden state using the current input and the hidden state of the next time step. This bidirectional approach unlike unidirectional recurrent neural networks improves accuracy by considering information from both past and future contexts. The two hidden layers complement each other providing more information to the final prediction layer and acting as a form of model regularization.

During training, the model parameters are updated using a technique called backpropagation through time. This technique calculates gradients for both the forward and backward passes. At inference time the input sequence is processed by the BRNN in a single forward pass, and predictions are made based on the combined outputs of the two hidden layers. This process

helps the BRNN make accurate predictions by leveraging information from both directions [63]..

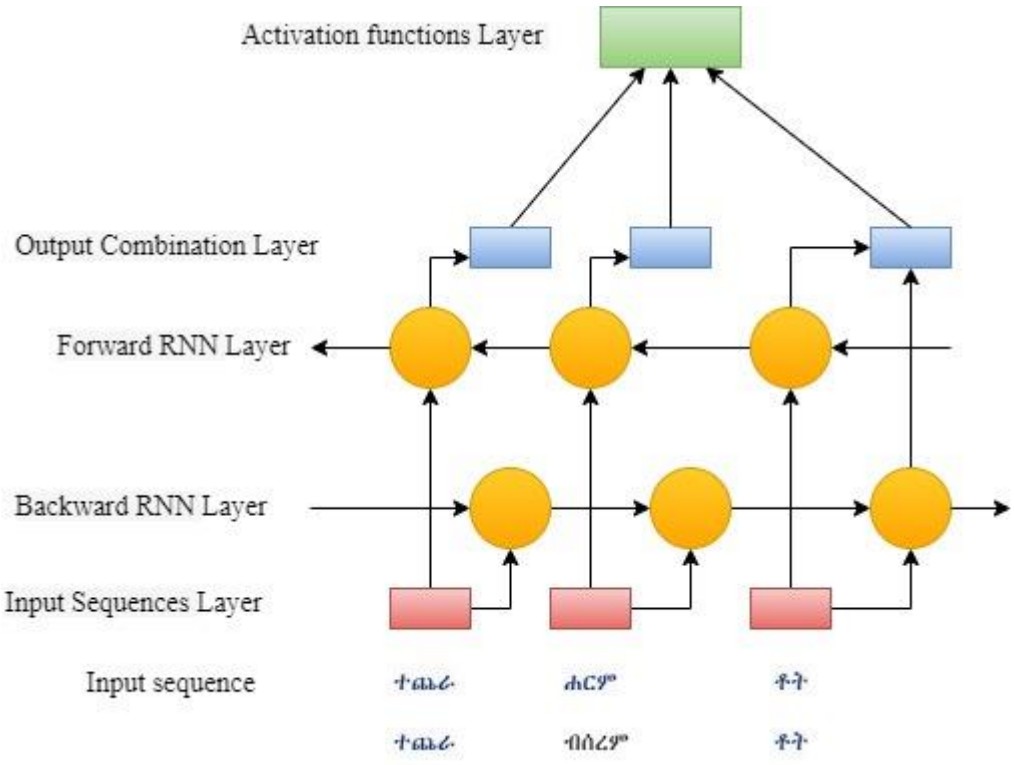


Figure 4-3 Architecture of Bi-directional RNN Model

The BRNN processes a sequence of data points where each point is represented as a vector of the same size. The sequence can have varying lengths. The BRNN utilizes both the forward and backward directions to handle the data. In the forward direction the hidden state at each time step is determined based on the input at that step and the hidden state from the previous step. Conversely, in the backward direction the hidden state at each step is calculated using the input at that step and the hidden state from the next step.

To calculate the hidden state at each step a non-linear activation function is applied to the weighted sum of the input and the previous hidden state. This mechanism allows the network to remember information from earlier steps in the process creating a memory component. Additionally, a non-linear activation function is used to determine the output at each step by applying it to the weighted sum of the hidden state and a set of output weights. This output can serve as the final output or be used as input for another layer in the network.

During training, the network employs supervised learning. The objective is to minimize the difference between the predicted output and the actual output. The network adjusts the weights

in the connections between the input and hidden layers, as well as between the hidden and output layers using a technique called backpropagation. This process ensures that the network learns to make accurate predictions by iteratively updating its weights based on the training data [63].

4.6.3. Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a modification to the hidden layer of recurrent neural networks that enhances their ability to capture long-range connections. The GRU model employed to effectively capture the semantic relationships between Guragigna textual inputs. To address the problem in networks of recurrent neural the GRU model computes activations at each time step by applying an activation is previous time steps of activations function and the current input. This allows the GRU to capture the contextual data and depend within Guragigna sentences[64].

In the decoder phase, using the input context an output sequence generates by GRU model and the last hidden state. This process begins with a special token that marks the beginning of output production and is appended to the end of the input. The Guragigna sentence is passed through the GRU layers sequentially with each layer updating its hidden state with the current input and the previous hidden state. The output of the final layer is then passed through a SoftMax activation to generate the first output word. This generation process is repeated with each generated word being first layer fed back reached until the output required duration or length sentence [64].

The decoder outputs are utilized for word prediction generating a new hidden decoder state and a new output word prediction. The GRU model effectively manages the information flow between the past and present inputs through its reset and update gates allowing it to capture the semantic similarities in Guragigna textual data. By incorporating the GRU model into the development of Semantic Textual Similarity for Guragigna. It becomes possible to accurately model and measure the semantic relationships between Guragigna sentences. This can be valuable for various applications including machine translation, sentiment analysis, and information retrieval in Guragigna language.

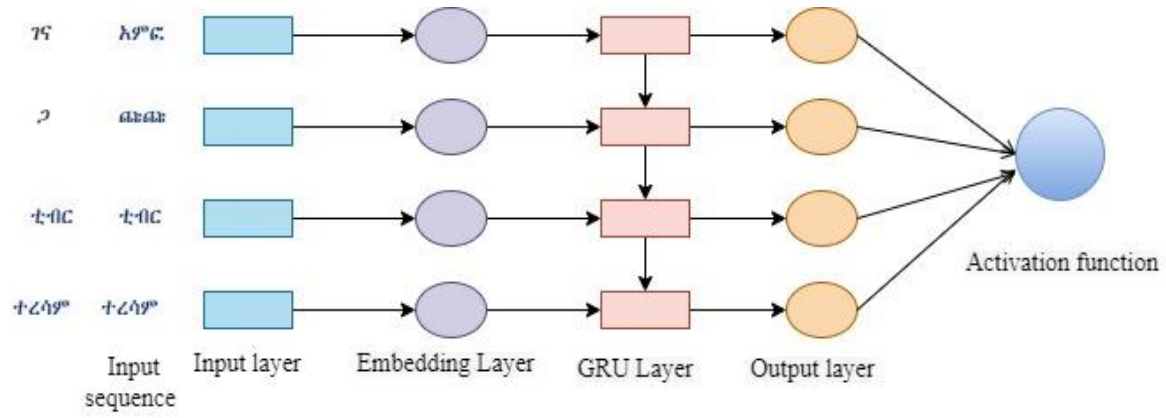


Figure 4-4 Gated Recurrent Unit (GRU) Model Architecture

The Gated Recurrent Unit (GRU) is a network of recurrent neural (RNN) architecture that can be used for Semantic Textual Similarity (STS) tasks. GRU is develop to take dependencies in sequential data of long-term while addressing some of the limitations of traditional RNNs is vanishing gradient problem[64].

The input textual data, such as sentences or phrases is tokenized into words units. Each word is then encoded into a dense vector representation using word embedding's. The encoded word representations are fed into one or multiple GRU layers. Each GRU layer consists of GRU cells that process the input sequence sequentially. The GRU cells update their hidden state at each time step with the present input and the hidden state in previous. The result of the GRU layer(s) captures the contextual information from the input sentences. To obtain a fixed-length representation for each sentence various pooling techniques can be applied, such as max-pooling or mean-pooling to summarize the GRU outputs. The fixed-length representations used to estimate the input sentence STS between them. This can be done by calculating a similarity score using various methods. With the suitable loss function model is trained that compares the predicted similarity scores with the ground truth similarity labels. To update the model parameters and minimize the loss Optimization techniques are used. By utilizing the GRU architecture which incorporates gating mechanisms. The model can capture relevant information from the input sentences and effectively model their semantic similarity. GRU has demonstrated good performance in various NLP tasks including STS due to its capacity to handle sequential dependencies and mitigate the vanishing gradient problem.

4.6.4. Stacked RNN

A stacked RNN is a RNN that use of different hidden layers contains multiple cells of memory in each layer. The result of one layer is fed as input to the next layer forming a deep or

hierarchical structure[65]. Stack RNNs have shown promise in developing Semantic Textual Similarity (STS) models for Guragigna a language limited resources available. By leveraging the hierarchical nature of Stack RNNs. These models can capture complex semantic patterns and dependencies within Guragigna sentence pairs. The architecture involves stacking multiple recurrent layers allowing information to flow through the network at different levels of abstraction. With an initial embedding layer that converts Guragigna sentences into dense vector representations followed by Simple RNN layers the model can learn and encode the semantic nuances language. The final dense layer with a sigmoid activation function enables the model to predict the similarity score between sentence pairs. Through training and optimization the Stack RNN-based STS model can effectively estimate the semantic similarity of Guragigna text [65].

The development of a Stack RNN-based STS model for Guragigna entails several critical steps. Firstly, a suitable dataset comprising sentence pairs with similarity labels in Guragigna is collected and preprocessed. Word embedding's specifically trained on Guragigna text are then utilized to map the vocabulary into dense vector representations. The Stack RNN architecture consisting of embedding layers, Simple RNN layers, and a dense layer is designed to capture the hierarchical patterns and dependencies within the language. The model is trained using appropriate loss functions and optimizers with performance monitored on a validation set. Finally, the trained model is evaluated on a separate test set using standard STS evaluation metrics. This approach allows for the development of robust STS models for Guragigna contributing to a better understanding of semantic similarity in this language. The figure below shows a stacked RNN with two hidden layers [66].

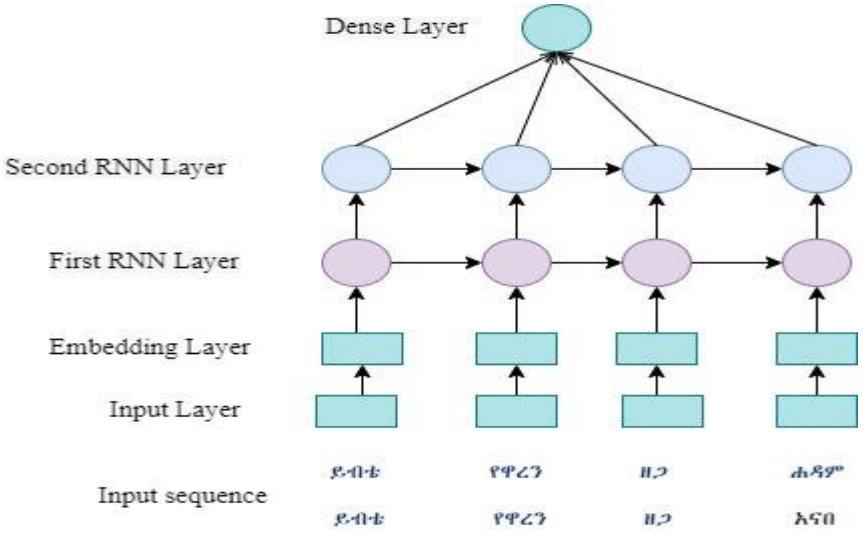


Figure 4-5 Architecture of Stacked RNN

The first layer is an Embedding layer that converts integer-encoded input sequences into dense vectors. The input_dim parameter represents the size of the vocabulary and output_dim is the dimensionality of the dense embedding, and input_length specifies input range of the sequences.

First RNN Layer and the second layer with 50 units. It uses activation function which is relu (Rectified Linear Unit) with return_sequences is true which means it outputs the hidden state for step in the input sequence of individual time.

The third layer is also a Simple RNN layer with 50 units and the activation function which is Rectified Linear Unit. Unlike the previous layer it doesn't have return_sequences is true so it only outputs the final hidden state. Dense Layer is fourth layer with 1 unit and a linear activation function. It produces a single output value.

CHAPTER FIVE

5. EXPERIMENTATION

5.1. Introduction

The previous chapter explored the intended framework's structure. Consequently, this section offers an elaborate account of the corpus source, experimental implementation, and a thorough explanation way of the dataset was preprocessed and utilized. It also encompasses an examination of the various model experiments conducted the process of parameter selection, and suggests how preprocessing impacts training the chosen model.

5.2. Data Collection and Preparation

In Table 5-1, we have gathered a sample of 3000 sentence pairs for the purpose of utilizing the STS (Semantic Textual Similarity) method in the Guragigna language. The corpus we have compiled consists of texts extracted from religious books such as Genesis (ኦሪት ዘፍጥረት), Exodus (ኦሪት ዘጸአት), Leviticus (ኦሪት ዘሌላውያን), the Gospel of Matthew (የማቴዎስ ወንጌል), Wudase Mariyam (ውዳሴ ማርያም), a fiction book titled "Yechamute Shika" (የጫሙት ሽካ ልበወለድ መጽሐፍ), various Guragi poems (የተለያዩ የጉራጊኛ ግጥሞች), then excerpts from primary grade textbooks. The specific dataset utilized from each source is outlined in the accompanying table.

Table 5-1 Source of Data Collection

Data source	Number of pair Sentence
Genesis (ኦሪት ዘፍጥረት)	700
Exodus (ኦሪት ዘጸአት)	280
Leviticus (ኦሪት ዘሌላውያን)	70
Gospel of Matthew (የማቴዎስ ወንጌል)	250
Wudase Mariyam (ውዳሴ ማርያም)	400
Yechamute Shika"(የጫሙት ሽካ)	300
Guragi poems (የጉራጊኛ ግጥሞች)	200
primary grade textbooks	800
Total	**3000**

5.3. Environment of implementation

To implement our proposed model for the study necessary to select a python programming and arrange the basic environment. Python as one of the programming languages for developing the STS. Python offers a wide range of freely available libraries that support our work. Specifically we utilized Python 3.8 along for coding purposes. To demonstrate of the model, we relied on the library of Keras, TensorFlow 2.12 library, and the numpy library, all of which are freely accessible. To facilitate our work we selected Google Colab as the primary integrated development environment (IDE). Google Colab offers GPUs with 32GB RAM and provides cloud resources making it an ideal choice for significantly accelerating training time compared to using a CPU. Additionally, we used a desktop computer and a Dell Laptop for corpus preparation, preprocessing, and writing the research report. To begin our experimental tasks we first stored the dataset and saved it as xlsx files were then uploaded to Google Drive. In the Google Colab notebook we mounted the Google Drive allowing us to access the dataset. After mounting the drive. We proceeded with each preprocessing step which be discussed in detail below.

Draw.io is a popular tool used for architecture design and diagramming. It provides a user-friendly interface for creating various types of diagrams. With Draw.io can easily drag and drop shapes, icons, and connectors to design and visualize architecture. The tool offers a wide range of pre-built shapes and symbols that are commonly used in architecture design, such as Architecture of STS for Guragigna language Model, LSTM, GRU, BIRNN, and Stacked RNN. It also allows to customize the appearance of diagrams by adjusting colors, sizes, and styles to match specific requirements. It also offers features for exporting diagrams in various formats, including PDF, PNG, and JPG and making it easy to share and present architecture designs with others.

5.3.1. Removing extra spaces

The provided code demonstrates a process for reading the material of an xlsx file removing spaces from the content, and saving the modified content to an Excel file. It begins by opening the xlsx file using the 'latin-1' encoding to handle non-UTF-8 characters. The content is then read and cleaned by removing spaces. Next, a new Excel workbook is created using the ‘openpyxl’ library, and the modified content is written to cell A1 of the active sheet. Finally, the workbook is saved to an Excel file specified by ‘output_file_path’, and a confirmation message is printed. This code can be useful for converting text data into a structured Excel format while handling non-standard characters and preserving the original formatting.

Additionally, the code showcases the usage of the ‘openpyxl’ library which provides functionality for creating, manipulating, and saving Excel files by leveraging this library. The code offers a straightforward approach to transforming text data into an Excel format. The 'latin-1' encoding is take to handle potential character decoding issues ensuring the successful reading and processing of the xlsx file. The resulting Excel file serves as a structured representation of the modified content, facilitating further analysis, sharing, or integration with other tools that support Excel files. This code snippet exemplifies a practical solution for converting textual data into a more structured and accessible format for various purposes. The code provided looks correct. It reads the file material located at file_path, removes spaces from the content, creates a new workbook using openpyxl.Workbook(), gets the active sheet, writes the modified content to cell A1, and saves the workbook to an Excel file located at output_file_path. Finally, it prints a confirmation message indicating the location of the saved file.

```
file_path = '/content/drive/MyDrive/all in one/result.txt'
output_file_path = '/content/drive/MyDrive/all in one/result.xlsx'
# Read the content of the file
with open(file_path, 'r') as file:
    content = file.read()
# Remove spaces and newlines
content = content.replace(" ", "").replace("\n", "")
# Create a new workbook
workbook = openpyxl.Workbook()
# Get the active sheet
sheet = workbook.active
# Write the modified content to cell A1
sheet['A1'] = content
# Save the workbook to an Excel file
workbook.save(output_file_path)
# Confirm that the file was saved
print("Output saved to", output_file_path)
```

Figure 5-1 Sample code of removing spaces and newlines

5.3.2. Removal of stop-words

The provided code demonstrates a practical approach to removing stopwords from a corpus of text. Stopwords are common words that often do not carry significant meaning and can be safely excluded from text analysis tasks. The code begins by reading a list of custom stopwords

from a specified file. This allows flexibility in customizing the stopwords based on specific requirements or domain-specific language.

Next, the code loads the corpus from a separate file. The corpus consist a collection of sentence, where each sentence is indicate as a separate line in the file. The code then tokenizes each sentence into individual words using the 'word_tokenize' function from the NLTK library. Using the custom stopwords list the code iterates over each word in the documents and removes any words that are present in the stopwords list. This process effectively filters out the stopwords from the corpus and creates a new filtered corpus. Finally, the code saves the filtered corpus to a new file where each document is written on a separate line. This allows for easy retrieval and further to process filtered text data.

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
# Read custom stopwords from file
stopwords_file_path = '/content/drive/MyDrive/custom_stopwords'
with open(stopwords_file_path, 'r') as file:
    custom_stopwords = [word.strip() for word in file.readlines()]
# Load the corpus from file
corpus_file_path = '/content/drive/MyDrive/all in one/result.txt'
with open(corpus_file_path, 'r') as file:
    corpus = file.readlines()
# Remove stopwords from the corpus
filtered_corpus = []
for document in corpus:
    # Tokenize the document into words
    words = word_tokenize(document)
    # Remove the custom stopwords from the list of words
    filtered_words = [word for word in words if word.lower()
    not in custom_stopwords]
```

```
    # Reconstruct the document without stopwords
    filtered_document = ' '.join(filtered_words)
    # Add the filtered document to the filtered corpus
    filtered_corpus.append(filtered_document)
# Define the output file path
output_file_path = '/content/drive/MyDrive/all in one/filtered_result2.txt'
# Write the filtered corpus to the output file
with open(output_file_path, 'w') as file:
    for document in filtered_corpus:
        file.write(document + '\n')
```

Figure 5-2 Sample Code of Removing Stopwords

5.3.3. Removing punctuation

The provided code is a Python script that removes punctuation from a given corpus file and saves the modified corpus without punctuation to a new file. The code starts by defining a function called 'remove_punctuation' which takes a string as input. Uses regular representation inside this function to define a pattern that matches any character that is not a word character or whitespace. The 're.sub()' function is then used to replace all matches of this pattern with an empty string effectively removing the punctuation. Finally, the function returns the modified text without punctuation.

Another function called 'remove_punctuation_from_corpus' is defined which takes two arguments: 'corpus_file' and 'output_file'. This function opens the specified corpus file reads its contents, and stores them in a string variable. It then calls the 'remove_punctuation' function to remove the punctuation from the corpus text. After that, it opens the output file in write mode and writes the modified corpus text without punctuation to it.

```
import re
def remove_punctuation(text):
    punctuation_pattern = re.compile(r'[^\w\s]')
    text_without_punctuation = re.sub(punctuation_pattern, '', text)
    return text_without_punctuation
def remove_punctuation_from_corpus(corpus_file, output_file):
    with open(corpus_file, 'r') as file:
        corpus_text = file.read()
    modified_text = remove_punctuation(corpus_text)
    with open(output_file, 'w') as file:
        file.write(modified_text)
# Example usage
input_corpus_file = '/content/drive/MyDrive/all in one/filtered_result2.txt'
output_corpus_file = '/content/drive/MyDrive/all in one/without_punctuation.txt'
remove_punctuation_from_corpus(input_corpus_file, output_corpus_file)
```

Figure 5-3 Sample Code of Removes Punctuation

5.3.4. Tokenization

The provided code demonstrates a workflow for processing textual data from an Excel file using Python and the Keras library. The code is designed to read sentences and associated similarity scores from the Excel file, tokenize the sentences, and perform padding to ensure uniform sequence lengths. The resulting tokenized and padded sequences are then used for further analysis or to train machine learning models.

The code imports the necessary libraries: 'pandas' for working with data frames and `Tokenizer` and 'pad_sequences' from Keras for text preprocessing. It then reads the data from

an Excel file using the ‘pd.read_excel ()’ function specifying the file path. The sentences from the "Sentence1" and "Sentence2" columns are extracted and converted to lists. While the similarity scores are stored in a separate variable. This initial step sets the foundation for subsequent text processing focuses on the text preprocessing steps. A ‘Tokenizer’ object is created, and the sentences from both columns are combined to fit the tokenizer and build the vocabulary. The ‘Tokenizer’ class is responsible for converting text into sequences of integers based on word frequency. Additionally, the maximum sequence length for padding is set to a predefined value in this case 20. The sentences are then tokenized and padded using the ‘texts_to_sequences()’ and ‘pad_sequences()’ methods respectively. This ensures that all sequences have the same length which is essential for inputting the data into certain machine learning models. The code prints the results of the text preprocessing. It outputs the total number of unique words in the vocabulary which provides insights into the richness of the textual data. Additionally, it displays the length of the longest sentence after padding which is crucial for determining the appropriate input size for models that require fixed-length inputs. These results give users an understanding of the characteristics of the dataset and can guide subsequent steps in data analysis, feature engineering, or model development.

```
import pandas as pd
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
# Read the data from an Excel file
df = pd.read_excel('/content/drive/MyDrive/all with score and none similar.xlsx')
# Extract the sentences and scores from the DataFrame
sentences1 = df['Sentence1'].tolist()
sentences2 = df['Sentence2'].tolist()
scores = df['SimilarityScore'].values
# Tokenize and pad the sentences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences1 + sentences2)
max_seq_length = 20  # Adjust based on your data
X1 = pad_sequences(tokenizer.texts_to_sequences(sentences1), maxlen=max_seq_length)
X2 = pad_sequences(tokenizer.texts_to_sequences(sentences2), maxlen=max_seq_length)
y = scores
# Print the results
print("Total unique words:", len(tokenizer.word_index))
print("Length of the longest sentence:", max_seq_length)
```

Figure 5-4 Sample Code of Tokenization

5.4. Embedding Process

The provided code demonstrates how to load pre-trained word embedding's and create an embedding matrix for take in a neural network model. The code initializes an empty dictionary called 'embedding's_index' to store the word embedding's. It then opens the pre-trained embedding's file specified by 'embedding_path' and reads it line by line. Each line is split into order of values where the first element represents the word and the remaining elements represent the embedding vector values. The word and embedding vector are extracted from each line and added to the 'embedding's_index' dictionary. This dictionary function as a lookup table for retrieving the embedding vector of a given word.

The code creates an 'embedding_matrix' by iterating over each word in the 'tokenizer.word_index' dictionary. For each word, it retrieves the corresponding embedding vector from the 'embedding's_index'. If the word is found in the 'embedding's_index' the embedding vector is assigned to the corresponding row of the 'embedding_matrix'. The 'embedding_layer' is then defined using the 'Embedding' class. Specifying the input dimension as the vocabulary size the output dimension as the embedding dimension. The embedding initializer as the pre-trained embedding matrix the input length as the maximum sequence length, and trainable as 'False' to keep the pre-trained embedding's fixed. This embedding layer can be taken as the input layer in a neural network model. Allowing model to benefit from the semantic information captured by the pre-trained word embedding's. By incorporating pre-trained word embedding's into a model the code facilitates the transfer of knowledge from a large corpus to the specific task at hand. This approach is particularly useful in scenarios where the available training data is limited. The code efficiently loads the pre-trained embedding's creates an embedding matrix that aligns with the tokenizer's vocabulary, and sets up an embedding layer that can be integrated into a neural network architecture. This enables the model to leverage the semantic relationships captured by the pre-trained embedding's enhancing its ability to understand and generalize from the input text.

```
# Load pre-trained GloVe embeddings
embeddings_index = {}
with open(embedding_path, 'r', encoding='utf-8') as f:
    for line in f:  values = line.split()    word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
# Create embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items(): embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:embedding_matrix[i] = embedding_vector
# Define embedding layer
embedding_layer = Embedding(
    input_dim=vocab_size,
    output_dim=embedding_dim,
    embeddings_initializer=Constant(embedding_matrix),
    input_length=max_seq_length,
    trainable=False  # Set to True if you want to fine-tune the embeddings
)
```

Figure 5-5 Sample Code of Embedding

5.5. Parameter Selection

The chosen hyper-parameters for the STS task provide a solid foundation for training RNN models. The batch size set to either 64 or 32 helps balance computational efficiency and model convergence. By processing multiple samples in parallel. The model's weight updates can benefit from the statistical properties of the batch. An embedding dimension of 100 allows the model to capture and represent the semantic information of sentences in a lower-dimensional space. This dimensionality strikes a balance between capturing meaningful features and avoiding overfitting.

The hidden dimension also set to 100 determines the capacity of the RNN models to capture complex dependencies within the sentences. This dimensionality choice ensures that the models have sufficient capacity to learn intricate patterns while avoiding excessive complexity. A learning rate of 0.001 combined with the Adam optimizer facilitates effective optimization by adaptively adjusting the learning rate with the gradients magnitudes. This combination helps the model converge to a good solution without being excessively sensitive to the learning rate's

initial value. The choice the loss function is MSE allows the model to minimize the discrepancy between predicted and true similarity scores aligning with the typical objective of STS tasks. Finally, applying a dropout rate of 0.1 helps regularize the models reducing overfitting and improving generalization by randomly dropping out a fraction of the connections during training.

Table 5-2 list of Hype-Parameters

No	Hype- parameters	Models			
		GRU	LSTM	Bi-RNN	Stack RNN
1	Batch size	64, 32	64, 32	32, 64	32, 64
2	Embedding dimension	100	100	100	100
3	Hidden dimension	100	100	100	100
4	Learning rate	0.001	0.001	0.001	0.001
5	Optimization	Adam optimizer	Adam optimizer	Adam optimizer	Adam optimizer
6	Loss function	Mean squared error (MSE)	Mean squared error (MSE)	Mean squared error (MSE)	Mean squared error (MSE)
7	Epoch	10,30,50,100 and 300	10,30,50,100 and 300	10,30,50,100 and 300	10,30,50,100 and 300
8	Dropout rate	0.1	0.1	0.1	0.1

As we can see from the table 5-2, we employed different criteria and strategies for selecting each parameter during the hyper parameter tuning process. Firstly, we identified the hyper parameters that had fixed values across all model architectures, such as the embedding dimension, hidden dimension, learning rate, optimization algorithm, loss function, and dropout rate. These hyper parameters were considered fixed and were not subject to search.

For hyper parameters with a small and discrete search space like the batch size we utilized grid search. We defined a grid of values (64, 32) for the batch size and systematically evaluated the model performance for each combination of batch sizes.

On the other hand for hyper parameters with a larger or continuous search space such as the epoch parameter we employed random search. We randomly sampled from the given options (10, 30, 50, 100, and 300) for the epoch and assessed the model performance for each sampled value by combining these approaches. We were able to effectively explore the hyper parameter

space. We systematically explored the batch size options using grid search and randomly sampled epoch values using random search. The other hyper parameters with fixed values provided a baseline configuration for comparison. Using this combination of grid search, random search, and fixed hyper parameters we were able to efficiently explore and optimize the hyper parameter space leading to improved performance of the models.

CHAPTER SIX

6. RESULT AND DISCUSSION

6.1. Introduction

In the following chapter presents the experimental setup including the details of the dataset, the architectural choices, and the training procedures. Then present the results obtained from the evaluation of different models on the STS task in Guragigna. We explore the STS of the Guragigna language using deep learning models with Word2Vec, Glove, and Universal Sentence Encoder (USE) embedding's combined with LSTM, GRU, Bi RNN, and Stacked RNN architectures.

To initiate our research, we construct a specific dataset for STS in Guragigna. This dataset comprises pairs of sentences each annotated with similarity scores by human evaluators. We leverage to train and evaluate the dataset by deep learning models incorporating Word2Vec, Glove, and USE embedding's in combination with LSTM, GRU, Bi RNN, and Stacked RNN architectures. Word2Vec and Glove are popular word embedding techniques that map words to continuous vector representations to capturing semantic relationships between words based on their co-occurrence patterns. These embedding's provide a foundation for our models understand the contextual meaning of words within the Guragigna language. In addition to word-level embedding's we incorporate Universal Sentence Encoder (USE) embedding's into our models. USE embedding's capture the semantic meaning of entire sentences that enabling a more comprehensive understanding of the contextual relationship between sentences in STS tasks.

For the LSTM and GRU architectural which network of recurrent neural (RNN) variants. Their ability to model sequential data effectively. Bi-RNN (Bidirectional RNN) and Stacked RNN architectures are also utilized to further enhance the models' ability to capture dependencies and context from both past and future information within the sentences. To train our models we utilize the annotated Guragigna STS dataset combined with appropriate loss functions and optimization techniques. We experiment with different combinations of embedding's and architectures. Fine-tuning hyper parameters to achieve optimal performance on the STS task in Guragigna.

In the subsequent sections we present the experimental setup including details of the dataset, preprocessing steps, embedding techniques, and model configurations. We then present and analyze the results obtained from evaluating the different models on the Guragigna STS task.

Furthermore, we discuss the implications of our findings, highlighting the strengths and limitations of the models, and propose potential avenues for future research.

6.2. Experimental Result

Performed experiments to evaluate the effectiveness of various hyper-parameters in training the selected models for the task. The training process was conducted using Colab. Web-based platform developed by Google that enables users to write and run Python code. Colab is widely utilized for tasks such as data analysis and machine learning. Our experimentation involved four models of deep learning namely LSTM, Bi-RNN, GRU, and Stacked RNN were combined with Word2Vec, Glove, and Universal Sentence Encoder (USE) embedding's.

During the experiments, explored the impact of different hyper-parameters on the performance of the selected models. This involved adjusting parameters such as learning rate, batch size, and number of hidden layers to find the optimal configuration. By systematically varying these hyper-parameters we aimed to identify the settings that yielded the most efficient and accurate results for Guragigna language STS task. The use of Colab provided us with a convenient and powerful platform for training our models, enabling us to leverage its computational resources and streamline the experimentation process.

Experiment 1: implementation on LSTM using Word2Vec, Glove, and Universal Sentence Encoder (USE) embedding's

During the preprocessing step, clean and tokenize the Guragigna text data splitting it into individual words or tokens. Split the dataset into training and testing sets and convert the sentences into numerical representations suitable for Word2Vec, Glove, or Universal Sentence Encoder (USE) embedding's. Train a Word2Vec model to learn word embedding's that capture semantic relationships. For Glove embedding's download pre-trained specific word vectors and map each word in the sentences to its corresponding Glove embedding vector. For USE embedding's pass sentences through the USE pre-trained model to obtain sentence-level embedding's. Set up an LSTM model initialize it with the appropriate embedding layer with the chosen embedding's, and train it using suitable loss functions and optimizers. Finally, evaluate the LSTM model's performance using evaluation metrics such as Mean Squared Error (MSE) on test order, and optimize hyper-parameters through techniques like grid search or random search for improved performance in the Guragigna semantic textual similarity tasks.

In process of training, the LSTM model was trained for a range of 10 to 300 epochs with a batch size of 64 and 32. The embedding dimension is 50, and the 0.001 rate of learning was

set. The optimization was performed using the Adam optimizer, and the loss function used was mean squared error. When using Word2Vec embedding's the LSTM model yielded a test loss of 0.1369 and a test accuracy of 0.8631. This indicates that the model achieved a reasonably low loss and a decent level of accuracy in capturing the semantic similarity of the Guragigna language using Word2Vec embedding's. When USE utilizing embedding's, the LSTM model achieved a loss of 0.0685 and an accuracy of 0.9157. These results suggest that the USE embedding's provided better performance compared to Word2Vec and Glove with a higher accuracy and a slightly higher loss. Using Glove embedding's the LSTM model obtained a test loss of 0.1416 and a test accuracy of 0.8584. This indicates that the model performed similarly to Word2Vec in terms of loss and accuracy suggesting that Glove embedding's also captured the semantic information effectively for the Guragigna language. Overall, these results demonstrate that the LSTM model. when combined with different types of embedding's can effectively capture the semantic textual similarity in the Guragigna language with Word2Vec, USE, and Glove embedding's all providing reasonably good performance. The focus of the experiment was to assess the model's performance by evaluating metrics such as Mean Squared Error (MSE), accuracy, and loss.

The executed code meticulously evaluates the LSTM model's performance in a sentence similarity task utilizing three distinctive embedding's Word2Vec, USE, and GloVe. The training histories visually depicted through the plot_training_history function offer a detailed insight into the model's learning. The progressive reduction in training loss and mean squared error (MSE) across epochs. Specifically the LSTM model is trained consecutively with Word2Vec, USE, and GloVe embedding's each embedding capturing semantic nuances in diverse manners. The quantitative evaluation metrics reveal notable performance distinctions. LSTM Mean Squared Error for Word2Vec is 0.3344, for USE it is 0. 0973, and for GloVe, it is 0. 3484. Correspondingly, these precise values enable a nuanced comparison of the LSTM's adaptability and effectiveness with respect to the different embedding techniques.

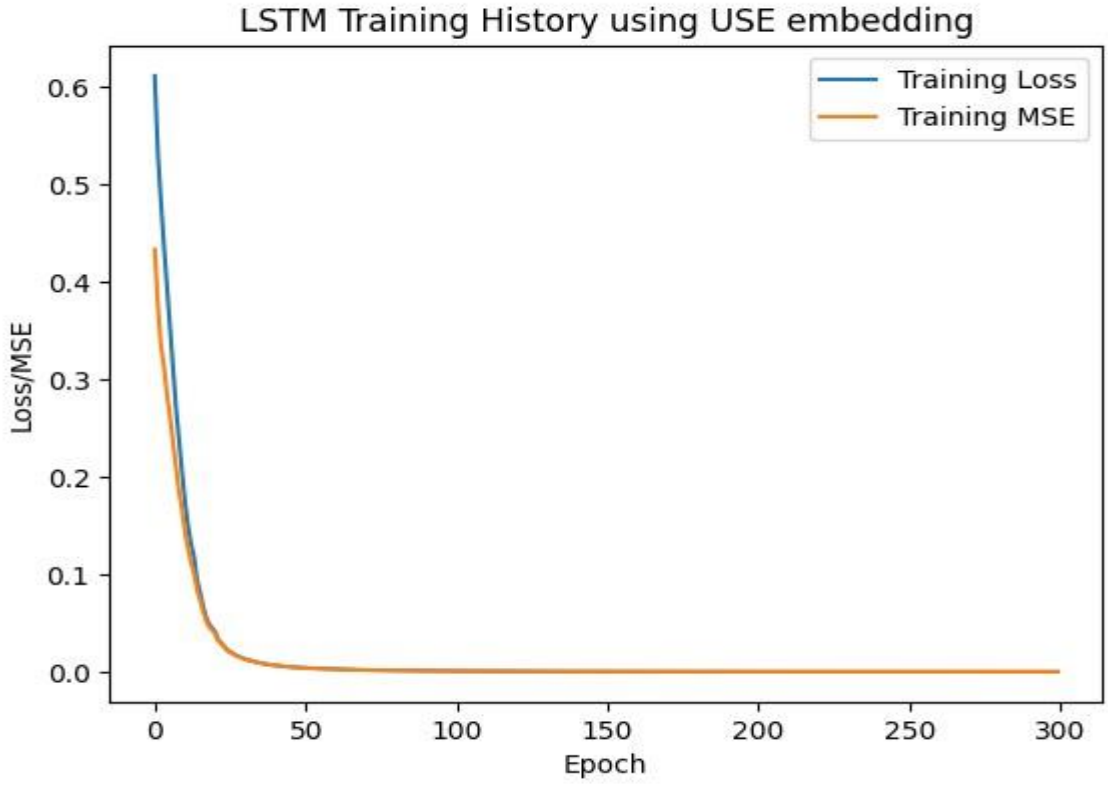


Figure 6-1 LSTM Training Using USE Embedding

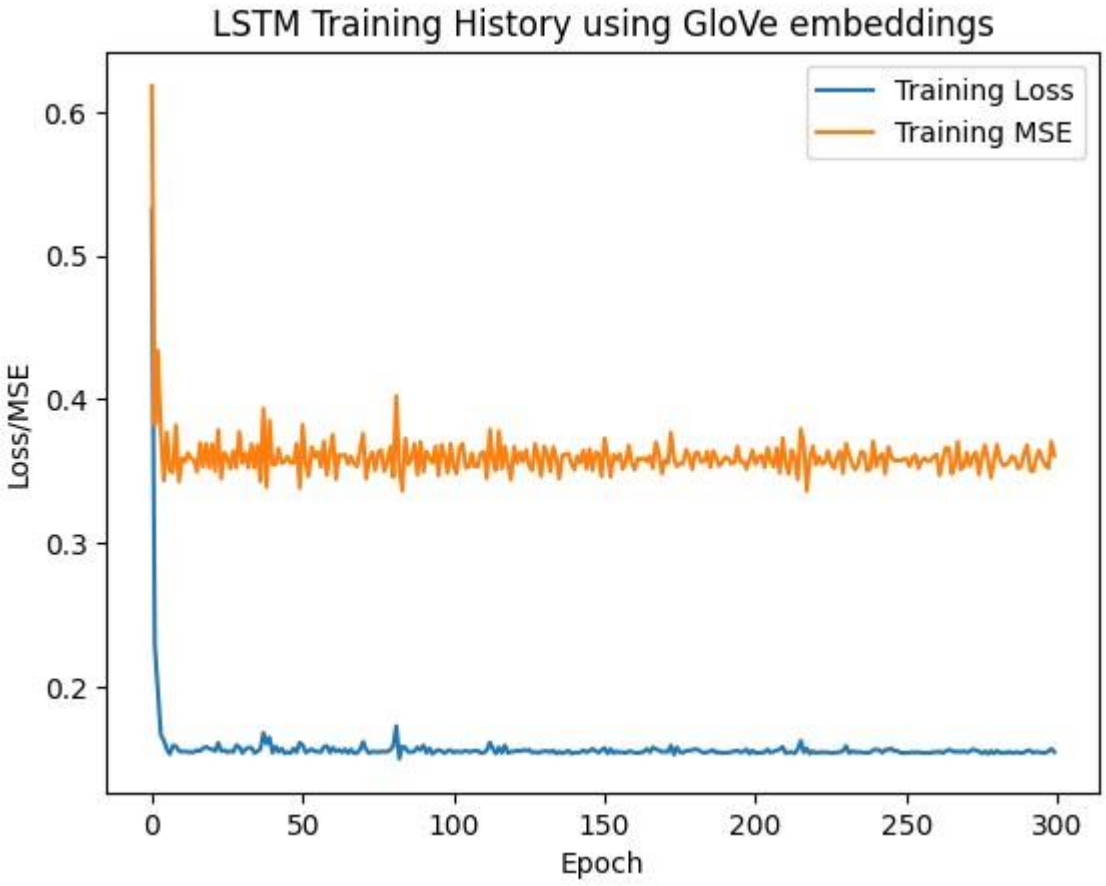


Figure 6-2 LSTM Training Using Glove Embedding

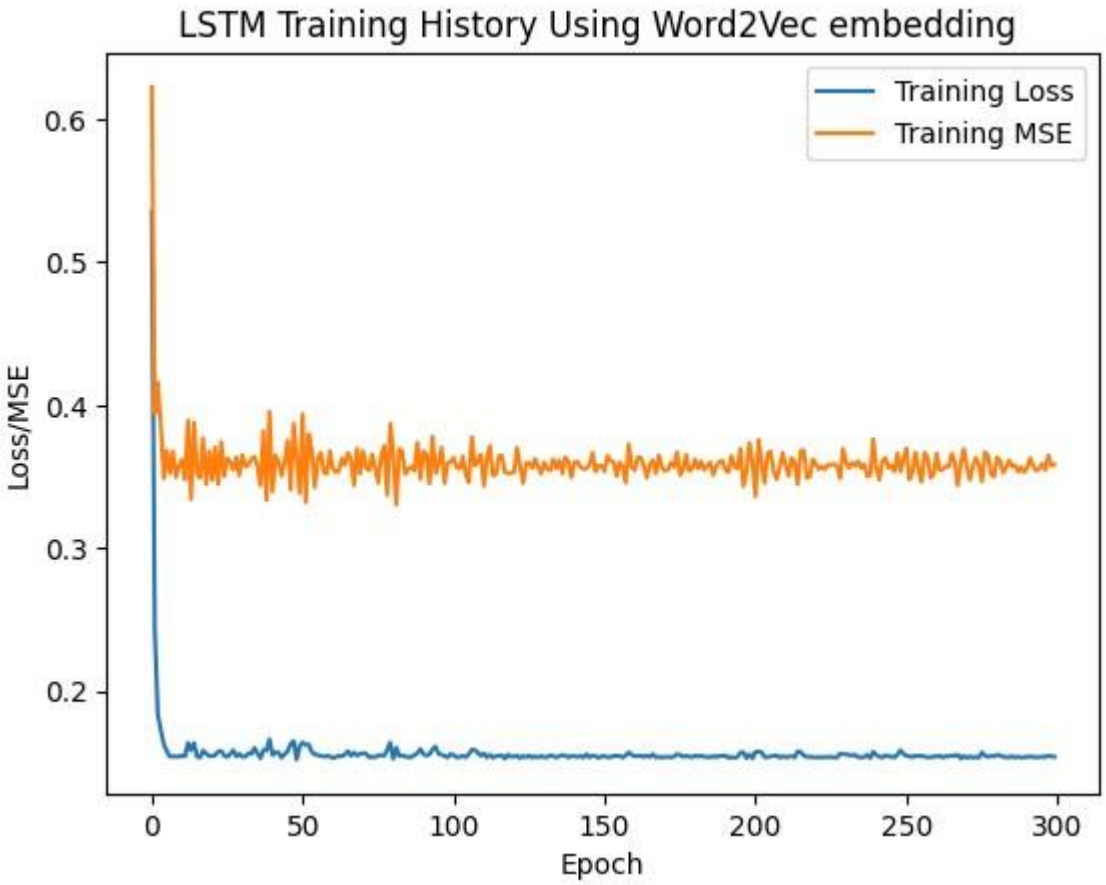


Figure 6-3 LSTM Training Using Word2vec Embedding

The visualization of LSTM models unfolds a comprehensive comparison between actual and predicted similarity scores. Explain the model's performance in capturing semantic relationships within sentence pairs. Utilizing Word2Vec, USE, and GloVe embedding's, the red bars signify the predicted similarity scores generated by the LSTM model. While the blue bars represent the actual similarity scores for the corresponding sentence pairs. LSTMs, with their inherent ability to capture long-range dependencies and mitigate vanishing gradient issues, showcase their prowess in encoding and predicting textual similarities across diverse embedding representations. The plot provides a nuanced understanding of how LSTMs leverage memory cell mechanisms to discern intricate semantic nuances offering valuable insights into their effectiveness across varied embedding contexts.

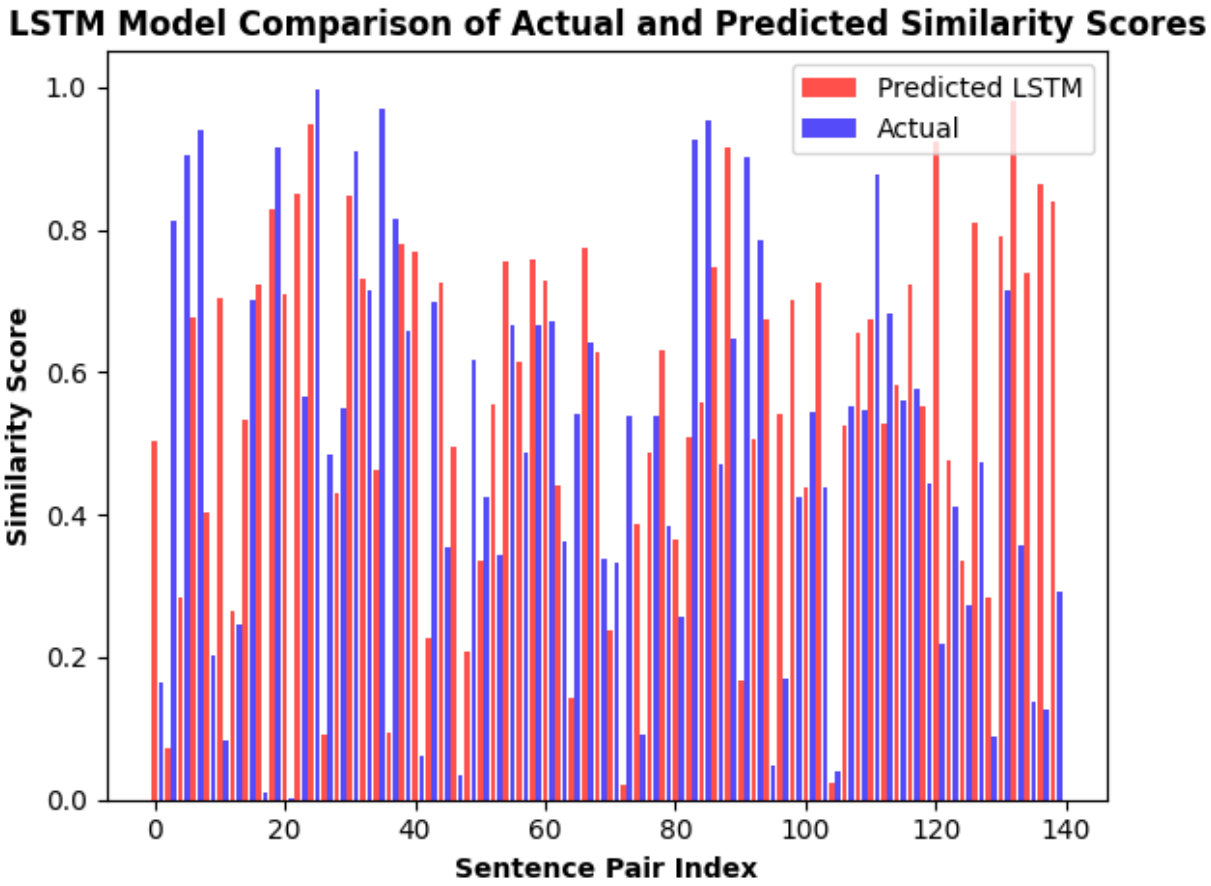


Figure 6-4 LSTM Model Comparison of Actual and Predicted Similarity Scores

Experiment 2: implementation on GRU using Word2Vec, Glove, and USE embedding's

In process of training, the GRU model was trained for a range of 10 to 300 epochs with a batch size of 64 and 32. The embedding dimension is 50, and the 0.001 rate of learning was set. The optimization was performed using the Adam optimizer, and the loss function used was mean squared error. When using Word2Vec embedding's, the GRU model yielded a test loss of 0.1401 and a test accuracy of 0.8599. When USE utilizing embedding's, the GRU model achieved a loss of 0.0700 and an accuracy of 0.9157. These results suggest that the Word2Vec embedding's provided better performance compared to USE and Glove with a higher accuracy and a slightly higher loss. Using Glove embedding's, the GRU model obtained a test loss of 0.1396 and a test accuracy of 0.8604.

The implemented code meticulously evaluates the GRU model's performance in a sentence similarity task, employing three distinctive embedding's Word2Vec, USE, and GloVe. The training histories visually depicted through the plot_training_history function, provide a detailed insight into the model's learning trajectory, showcasing the progressive reduction in training loss and mean squared error (MSE) across epochs. The GRU model is consecutively

trained with Word2Vec, USE, and GloVe embedding's each capturing semantic nuances in diverse ways. The quantitative evaluation metrics reveal specific performance values GRU Mean squared Error for Word2Vec is 0.3225 for USE it is 0.0938, and for GloVe, it is 0.3498. Correspondingly, accuracy values offer additional granularity, these precise metrics facilitate a nuanced comparison of the GRU model's adaptability and effectiveness with respect to the different embedding techniques.

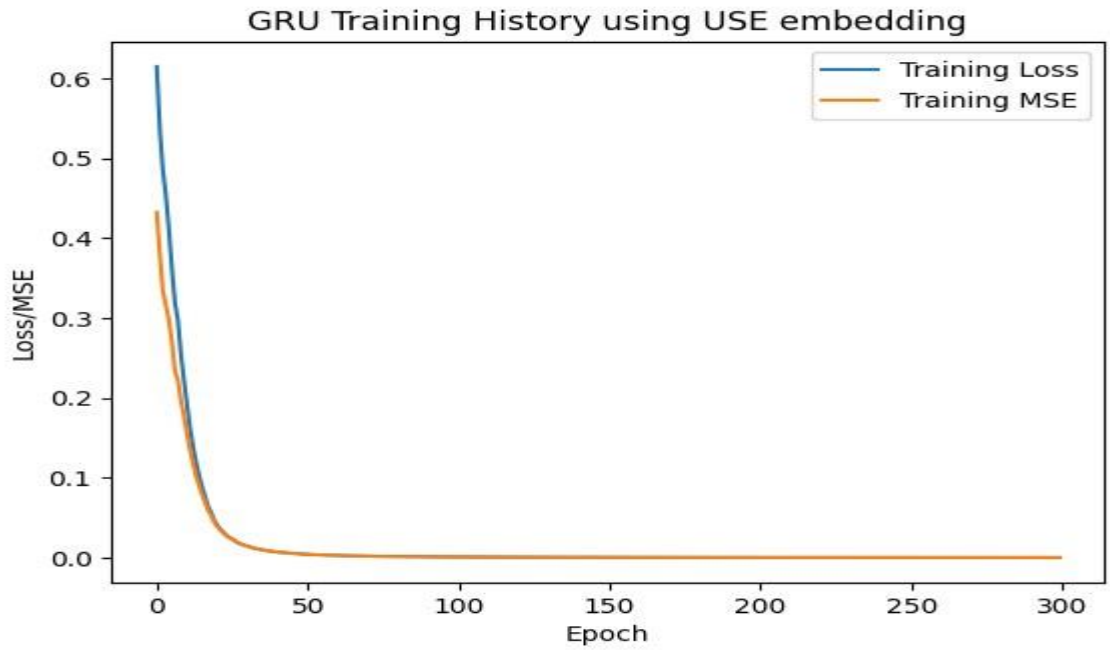


Figure 6-5 GRU Training History Using USE Embedding

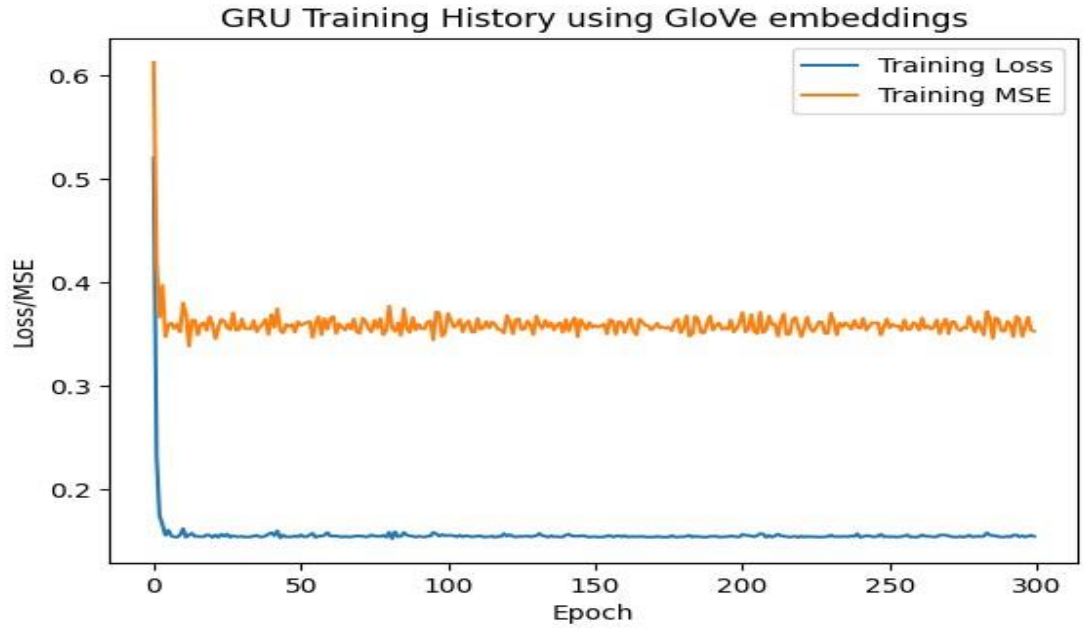


Figure 6-6 GRU Training History Using Glove Embedding

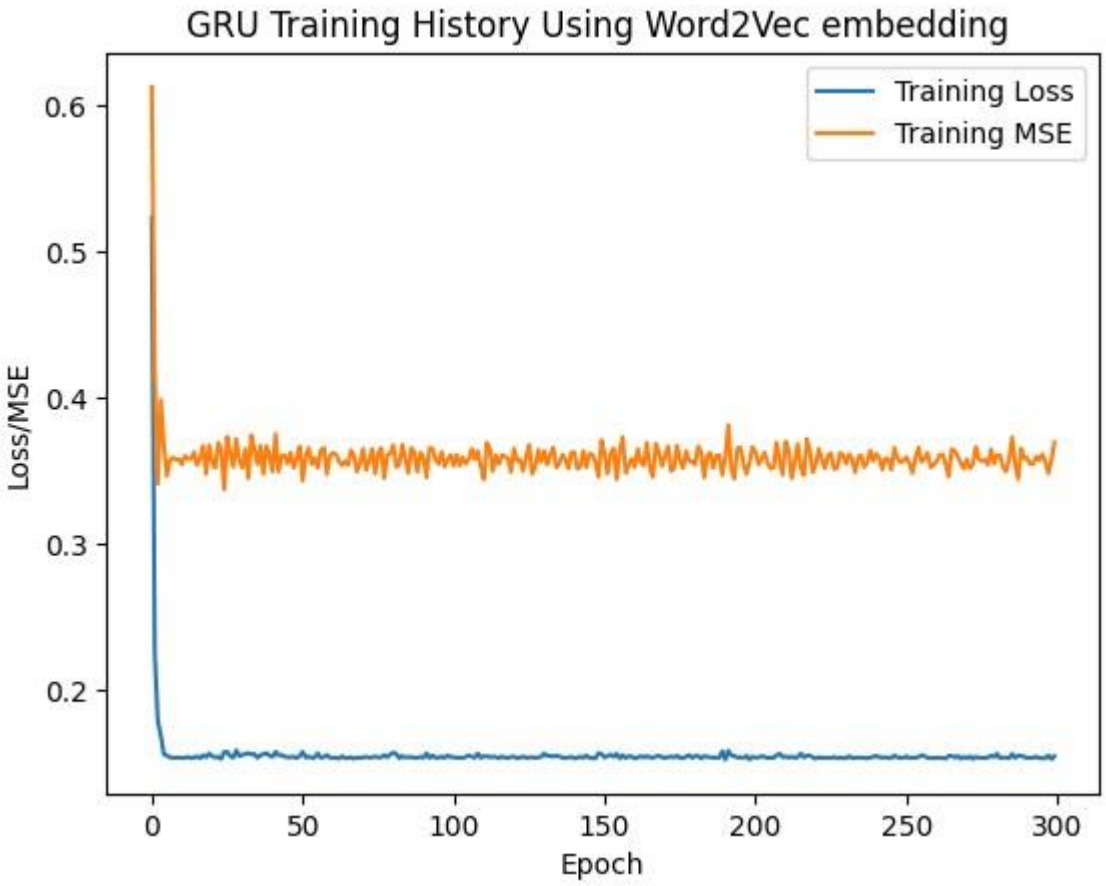


Figure 6-7 GRU Training History Using Word2vec Embedding

The GRU (Gated Recurrent Unit) model's actual and predicted similarity scores are visualized in the comparison plot showcasing its performance in capturing semantic relationships between sentence pairs using Word2Vec, USE, and GloVe embedding's. The red bars represent the predicted similarity scores generated by the GRU model while the blue bars depict the actual similarity scores for the corresponding sentence pairs. The plot provides a concise overview of how well the GRU model aligns with the ground truth similarity values offering insights into the model's ability to discern the underlying semantic structures within the given textual data. The comparison facilitates a nuanced understanding of the GRU model's effectiveness in capturing and predicting the similarity between pairs of sentences with the Word2Vec, Universal Sentence Encoder (USE), and GloVe embedding's can extracted features.

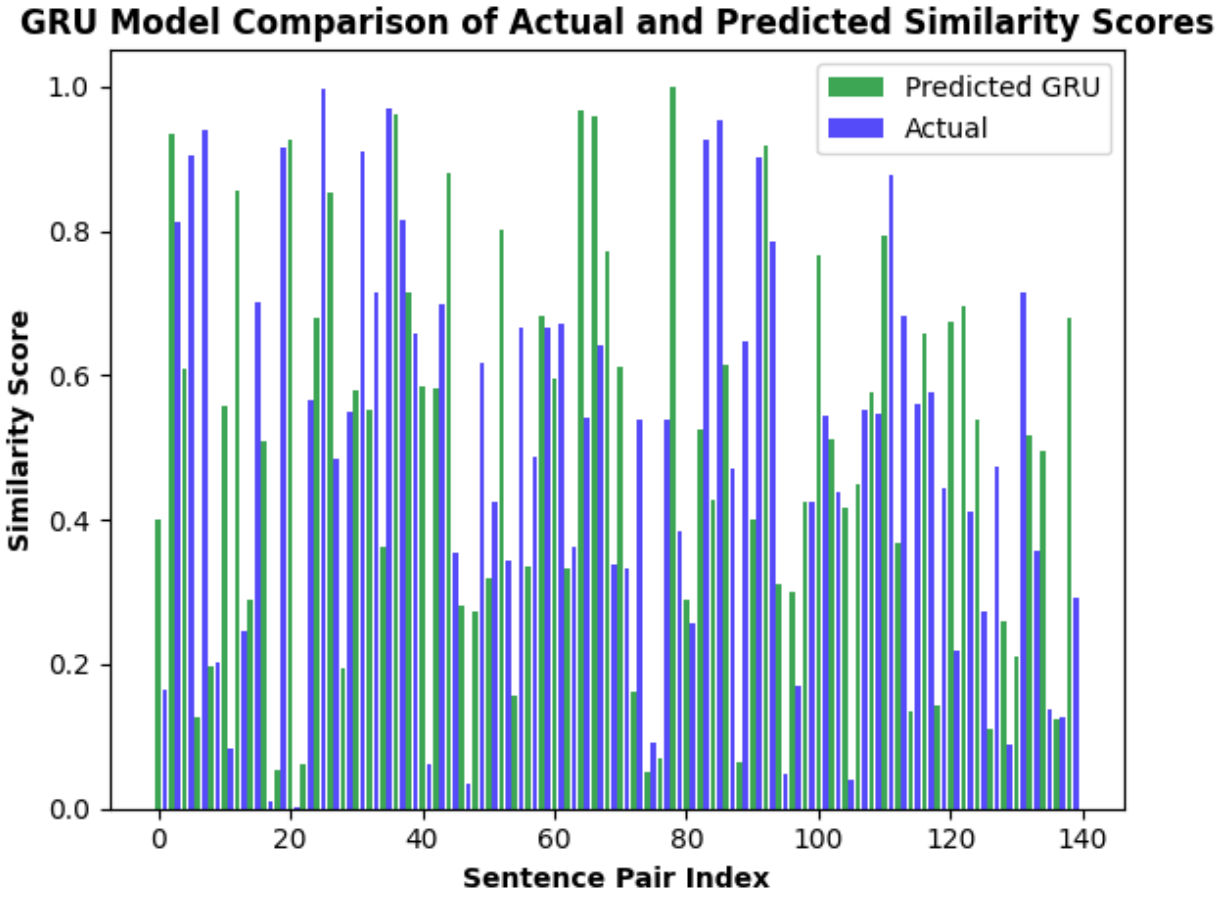


Figure 6-8 GRU Model Comparison of Actual and Predicted Similarity Scores

Experiment 3: implementation on Bidirectional RNN using Word2Vec, Glove, and USE embedding's

In process of training, the Bidirectional RNN model was trained for a range of 10 to 300 epochs with a batch size of 64 and 32. The embedding dimension is 50, and the 0.001 rate of learning was set. The optimization was performed using the Adam optimizer, and the loss function used was mean squared error. When using Word2Vec embedding's the Bidirectional RNN model yielded a test loss of 0.1370 and a test accuracy of 0.8630. When USE utilizing embedding's, the Bidirectional RNN model achieved a loss of 0.0662 and an accuracy of 0.9244. These results suggest that the USE embedding's provided better performance compared to Word2Vec and Glove with a higher accuracy and a slightly higher loss. Using Glove embedding's, the Bidirectional RNN model obtained a test loss of 0.1455 and a test accuracy of 0.8545.

The implemented code thoroughly evaluates the Bidirectional RNN model's performance in a sentence similarity task, leveraging three distinctive embedding's Word2Vec, USE, and GloVe. The training histories, visually depicted through the plot_training_history function offer a

detailed insight into the model's learning trajectory showcasing the progressive reduction in training loss and mean squared error (MSE) across epochs. The Bidirectional RNN model is consecutively trained with Word2Vec, USE, and GloVe embedding's each capturing semantic nuances in diverse ways. The quantitative evaluation metrics reveal specific performance values of Bidirectional RNN Mean squared Error for Word2Vec is 0.3740, for USE it is 0.0950, and for GloVe, it is 0.3493. These precise metrics enable a nuanced comparison of the Bidirectional RNN model's adaptability and effectiveness with respect to the different embedding techniques.

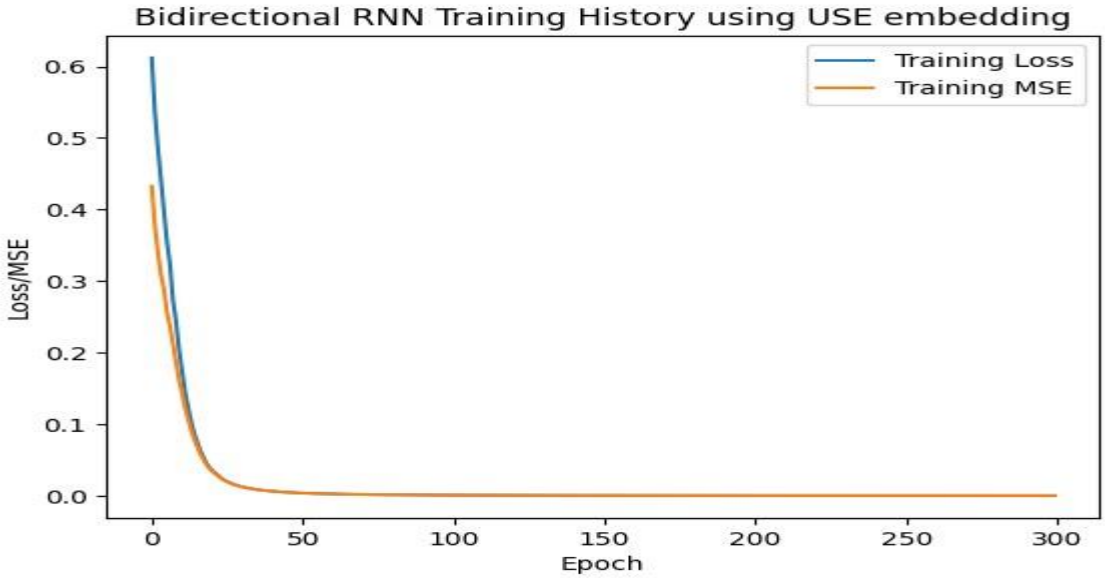


Figure 6-9 Bidirectional RNN Training History Using USE Embedding

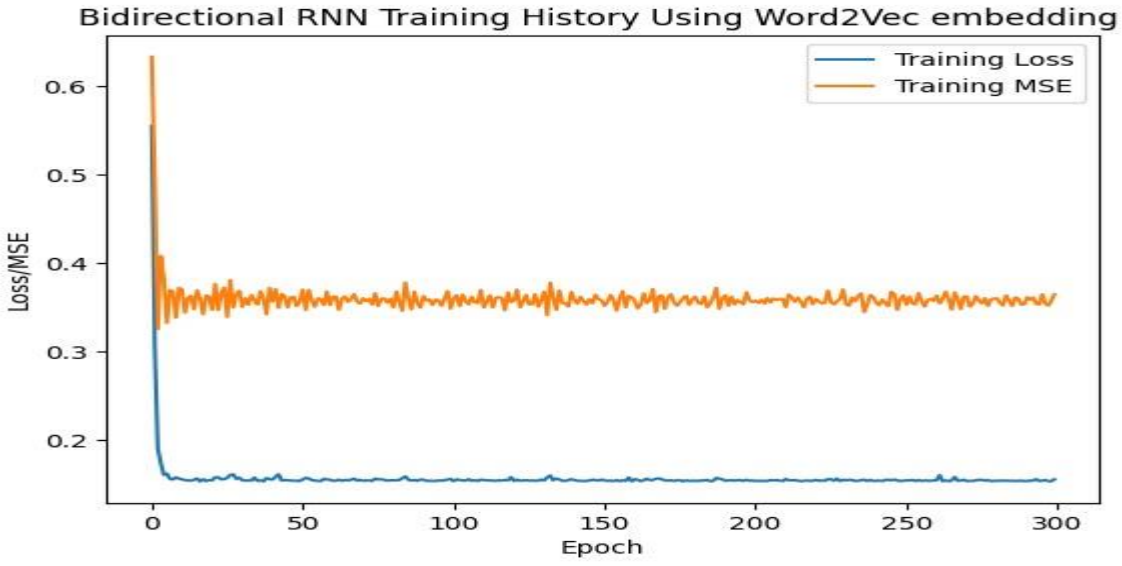


Figure 6-10 Bidirectional RNN Training History Using Word2vec Embedding

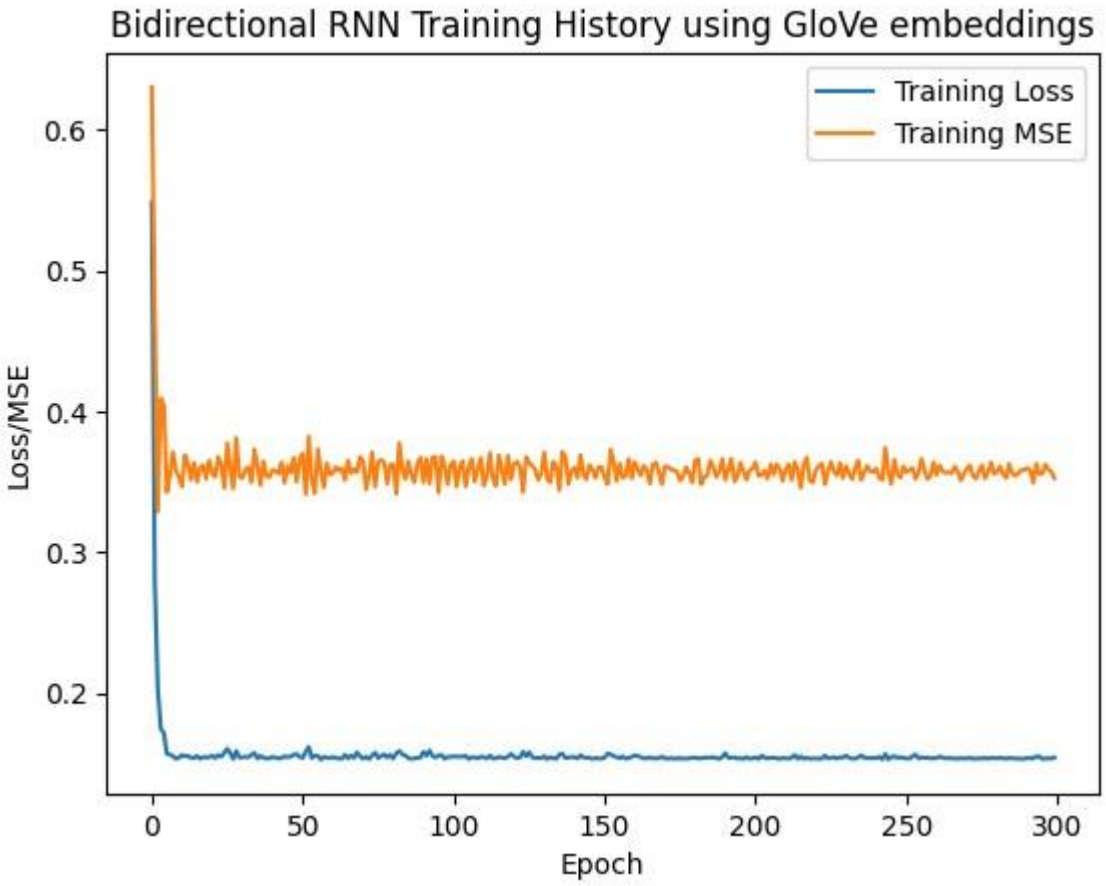


Figure 6-11 Bidirectional RNN Training History Using Glove Embedding

The comparison plot for Bidirectional Recurrent Neural Network (RNN) models unveils the actual and predicted similarity scores the model's competency in capturing semantic relationships within sentence pairs using three distinct embedding's Word2Vec, USE, and GloVe. The orange bars denote the predicted similarity scores by the Bidirectional RNN model while the blue bars represent the ground truth similarity scores. This visual analysis offers valuable insights into how Bidirectional RNNs leverage bidirectional information flow to enhance their understanding of contextual nuances and dependencies within the text. By examining the alignment between predicted and actual similarity scores across different embedding's the plot provides a nuanced understanding of the Bidirectional RNN model's effectiveness in capturing and predicting textual similarity across a diverse range of embedding representations.

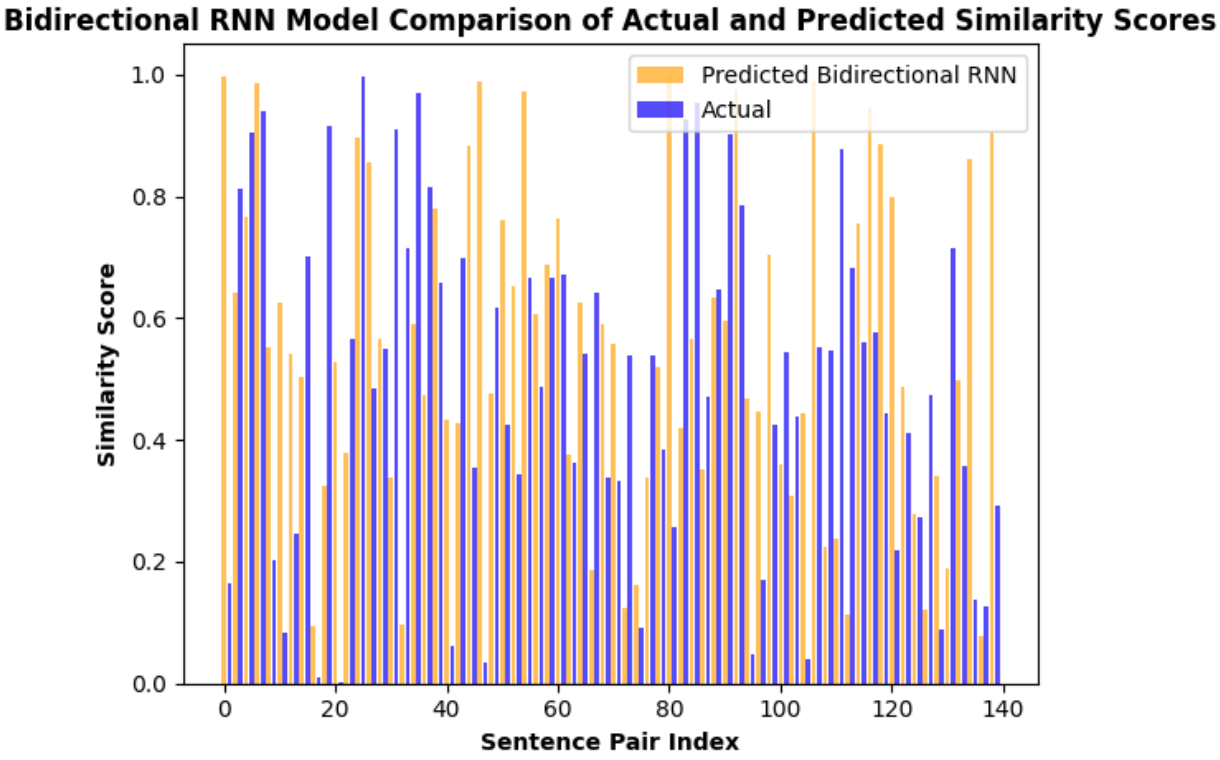


Figure 6-12 Bid-RNN Model of Comparison Actual and Predicted Similarity Scores

Experiment 4: implementation on Stacked RNN using Word2Vec, Glove, and USE embedding's

In process of training, the Stacked RNN model was trained for a range of 10 to 300 epochs with a batch size of 64 and 32. The embedding dimension is 50, and the 0.001 rate of learning was set. The optimization was performed using the Adam optimizer, and the loss function used was mean squared error. When using Word2Vec embedding's the Stacked RNN model yielded a test loss of 0.1427 and a test accuracy of 0.8573. When USE utilizing embedding's the Stacked RNN model achieved a loss of 0.0655 and an accuracy of 0.9215. These results suggest that the USE embedding's provided better performance compared to Word2Vec and Glove, with a higher accuracy and a slightly higher loss. Using Glove embedding's of Stacked RNN model obtained a test loss of 0.1404 and a test accuracy of 0.8569.

The implemented code meticulously assesses how to perform the Stacked RNN model in a sentence similarity task utilizing three distinctive embedding's Word2Vec, USE, and GloVe. The training histories visually depicted through the plot_training_history function offer a detailed insight into the model's learning trajectory showcasing the progressive reduction in training loss and mean squared error (MSE) across epochs. The Stacked RNN model is consecutively trained with Word2Vec, USE, and GloVe embedding's each capturing semantic

nuances in diverse ways. The quantitative evaluation metrics reveal specific performance values of Stacked RNN Mean squared Error for Word2Vec is 0.3532, for USE it is 0.1165, and for GloVe, it is 0.3423. These precise metrics facilitate a nuanced comparison of the Stacked RNN model's adaptability and effectiveness with respect to the different embedding techniques.

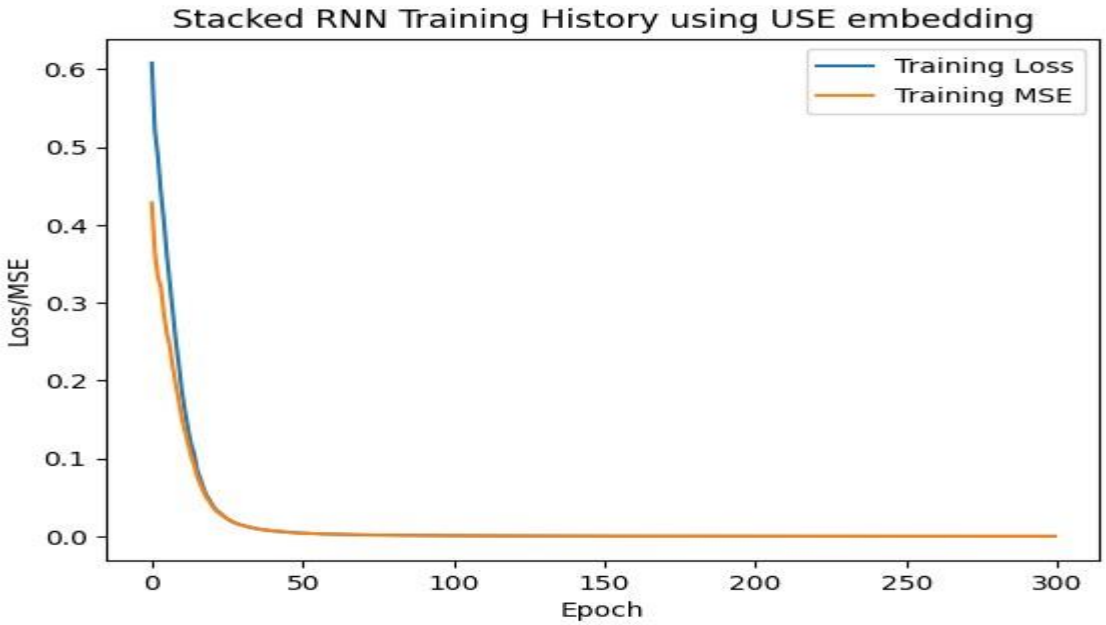


Figure 6-13 Stacked RNN Training History Using USE Embedding

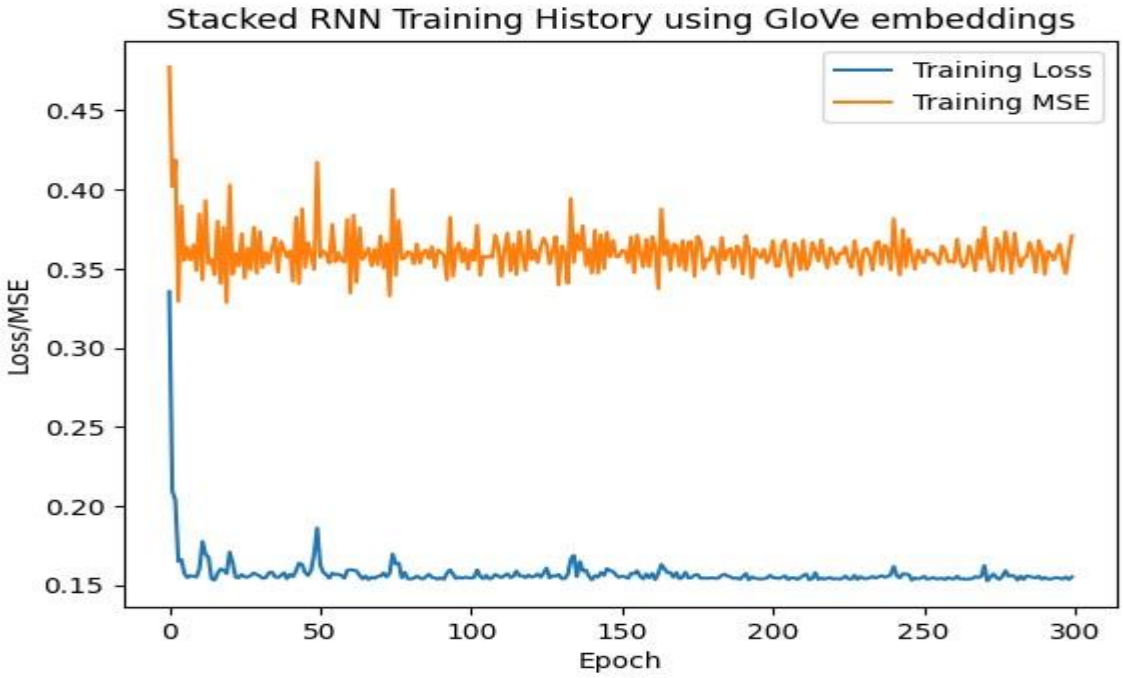


Figure 6-14 Stacked RNN Training History Using Glove Embedding

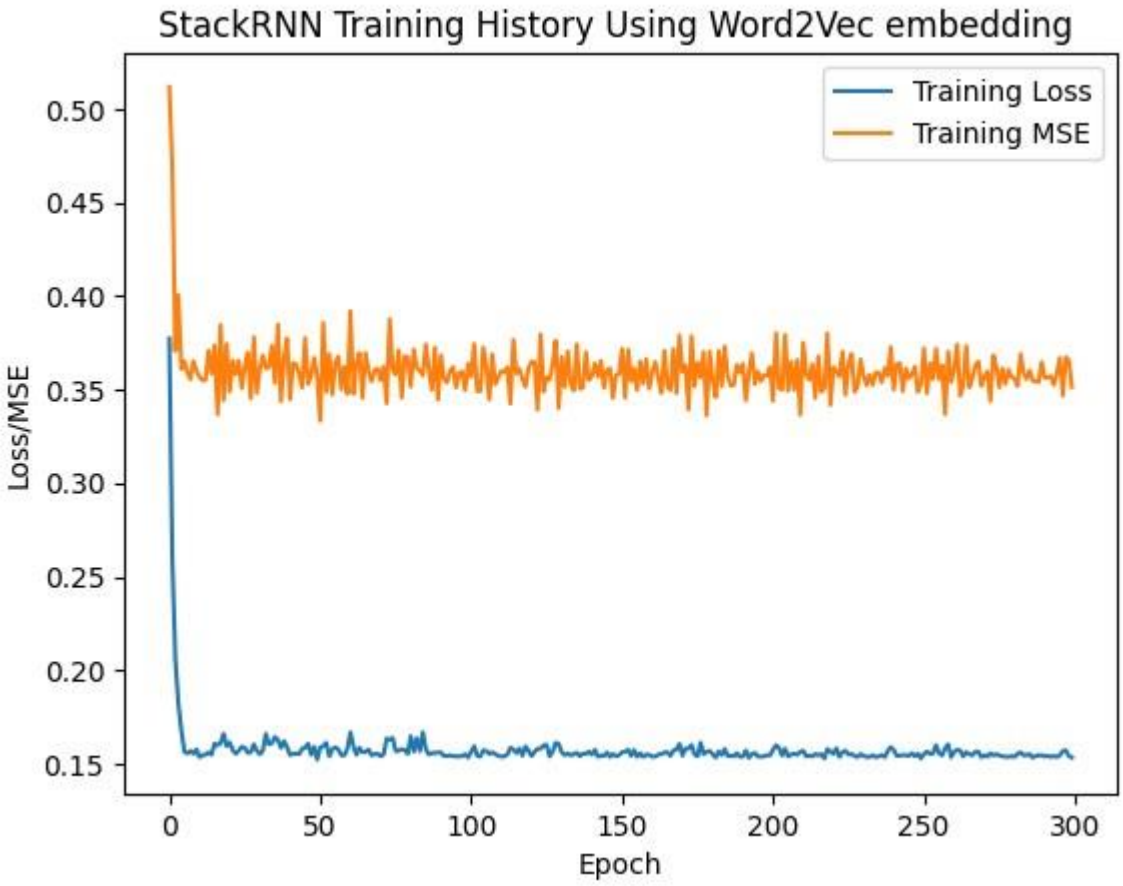


Figure 6-15 Stacked RNN Training History Using Word2vec Embedding

The visual representation of Stacked Recurrent Neural Network (RNN) models unveils the comparison between actual and predicted similarity scores shedding light on the model's performance in capturing semantic relationships within sentence pairs. Employing Word2Vec, USE, and GloVe embedding's the purple bars depict the predicted similarity scores generated by the Stacked RNN model. While the blue bars represent the actual similarity scores for the corresponding sentence pairs. Stacked RNNs leverage the hierarchical nature of multiple recurrent layers allowing them to capture intricate dependencies and nuanced patterns in the text. The plot provides a comprehensive assessment of the Stacked RNN model's efficacy in predicting similarity offering insights into its ability to exploit the hierarchical information encoded by different embedding's of nuanced understanding of textual relationships.

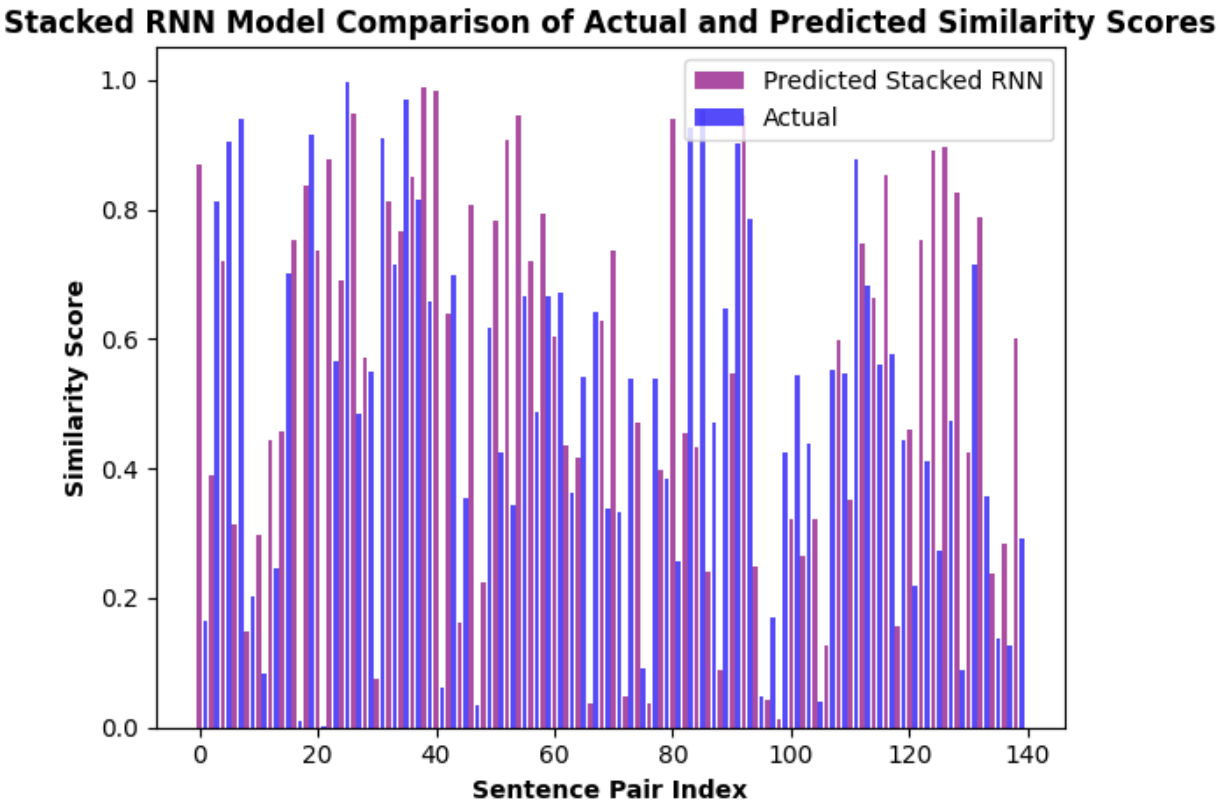


Figure 6-16 Stacked RNN Model Comparison of Actual and Predicted Similarity Scores

6.3. Discussion on the Result

In the table 6-1, the performance of various recurrent neural network (RNN) models is evaluated across different embedding's Word2Vec, Glove, and USE. For the LSTM model. It is observed that the USE embedding achieves the lowest Mean squared Error (MSE) of 0.0973, indicating accurate predictions of similarity scores. Additionally, the USE embedding yields 0.9157 best is accuracy emphasizing its effectiveness in capturing semantic relationships between sentences. The Word2Vec embedding also performs well with a slightly higher MSE of 0.3344 and accuracy of 0.8631. Glove on the other hand exhibits slightly higher MSE and lower accuracy compared to USE and Word2Vec with LSTM model.

Moving to the GRU model similar trends are observed. The USE embedding stands out with the lowest MSE (0.0938) and the highest accuracy (0.9157). Glove performs competitively MSE (0.3498) and accuracy of 0.8604. While Word2Vec shows slightly higher MSE and lower accuracy.

For the Bidirectional RNN (Bi RNN) model the USE embedding again demonstrates superior performance with the lowest MSE (0.0950) and the highest accuracy (0.9244). Word2Vec

closely follows showing a comparable MSE (0.3740) and accuracy (0.8630). In contrast, Glove lags slightly behind in terms of MSE and accuracy.

Finally, in the Stacked RNN model the USE embedding maintains its impressive performance with the lowest MSE (0.1165) and the highest accuracy (0.9215). Glove also performs well with a slightly lower MSE (0.3423) and high accuracy (0.8596). Word2Vec exhibits a marginally higher MSE and lower accuracy depend on the trends observed in the other models. Overall, the results highlight the best perform of the USE embedding in enhancing the predictive capabilities of RNN models for sentence similarity tasks.

Table 6-1 Experimental result of the proposed model

Model	Embedding	Loss	Accuracy	MSE
LSTM	USE	0.0685	0.9157	0.0973
	Wor2vec	0.1369	0.8631	0.3344
	Glove	0.1416	0.8584	0.3484
GRU	USE	0.0700	0.9157	0.0938
	Glove	0.1396	0.8604	0.3498
	Wor2vec	0.1401	0.8599	0.3225
Bidirectional RNN	USE	0.0662	0.9244	0.0950
	Wor2vec	0.1370	0.8630	0.3740
	Glove	0.1455	0.8545	0.3493
Stacked RNN	USE	0.0655	0.9215	0.1165
	Glove	0.1404	0.8596	0.3423
	Wor2vec	0.1427	0.8573	0.3532

The study's findings offer insightful information on how various recurrent neural network (RNN) models perform when applied to sentence similarity tasks using various embedding's namely Word2Vec, Glove, and Universal Sentence Encoder (USE). The importance findings of choosing an appropriate embedding technique as models' ability to capture semantic relationships between sentences is significantly impacts.

One notable observation is the consistently superior the USE embedding performance across all RNN models (LSTM, GRU, Bidirectional RNN, and Stacked RNN). The USE embedding consistently achieves the MSE is lowest and the highest accuracy indicating its robust representation of semantic content in sentences. This aligns with the Universal Sentence Encoder's design which aims to get a universal and context-aware representation of text.

Glove embedding's also perform competitively demonstrating strong performance in capturing sentence similarities. While Word2Vec performs reasonably well it consistently lags behind Glove and USE regarding accuracy of model and MSE of model. This focus the value of choosing embedding's that consider the context and semantic nuances present in sentences.

The study's findings highlight the trade-offs and strengths associated with different embedding's providing practitioners with valuable guidance when selecting models for sentence similarity tasks. Additionally, the study emphasizes the versatility of the Universal Sentence Encoder which emerges as a robust choice across diverse RNN architectures. Further exploration and fine-tuning of hyper-parameters could offer additional insights and potential improvements in performance. Overall, the study contributes to recognize embedding choices in RNN models for sentence similarity tasks offering valuable considerations for researchers and practitioners in natural language processing.

6.4. Confusion matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. It summarizes the predictions made by the model on a set of test data and compares them to the actual ground truth labels. The confusion matrix provides a detailed breakdown of the model's performance by showing the counts of true positive, true negative, false positive, and false negative predictions. It is particularly useful in assessing the accuracy of a classification model and understanding the types of errors it makes.

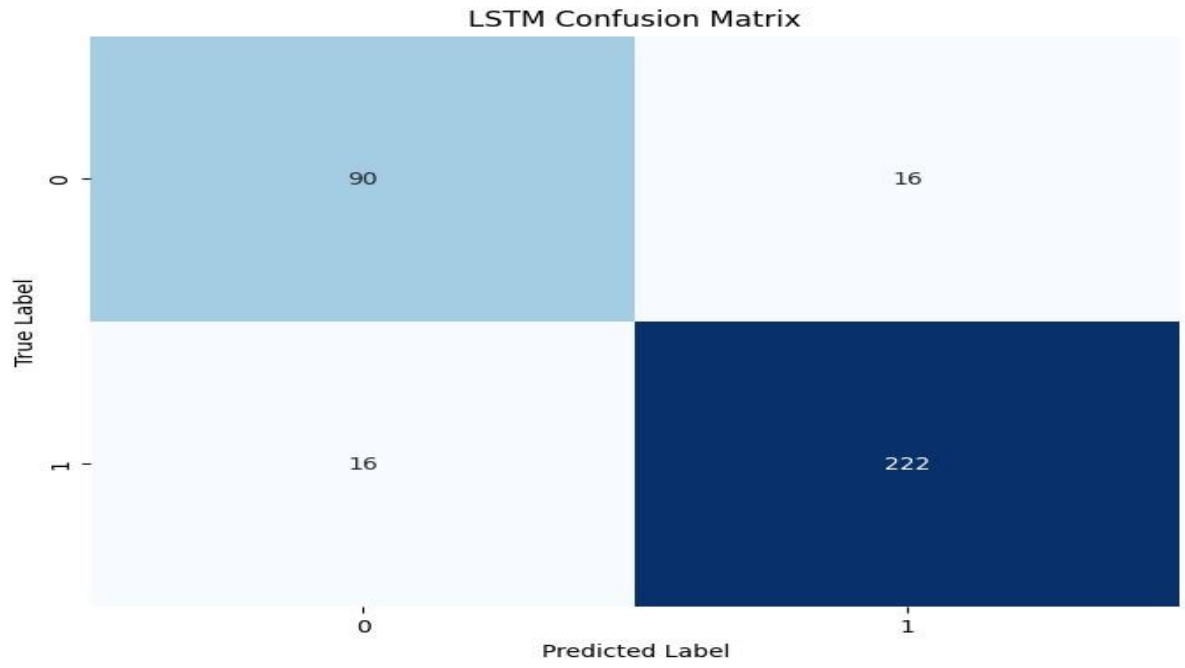


Figure 6-17 LSTM confusion matrix

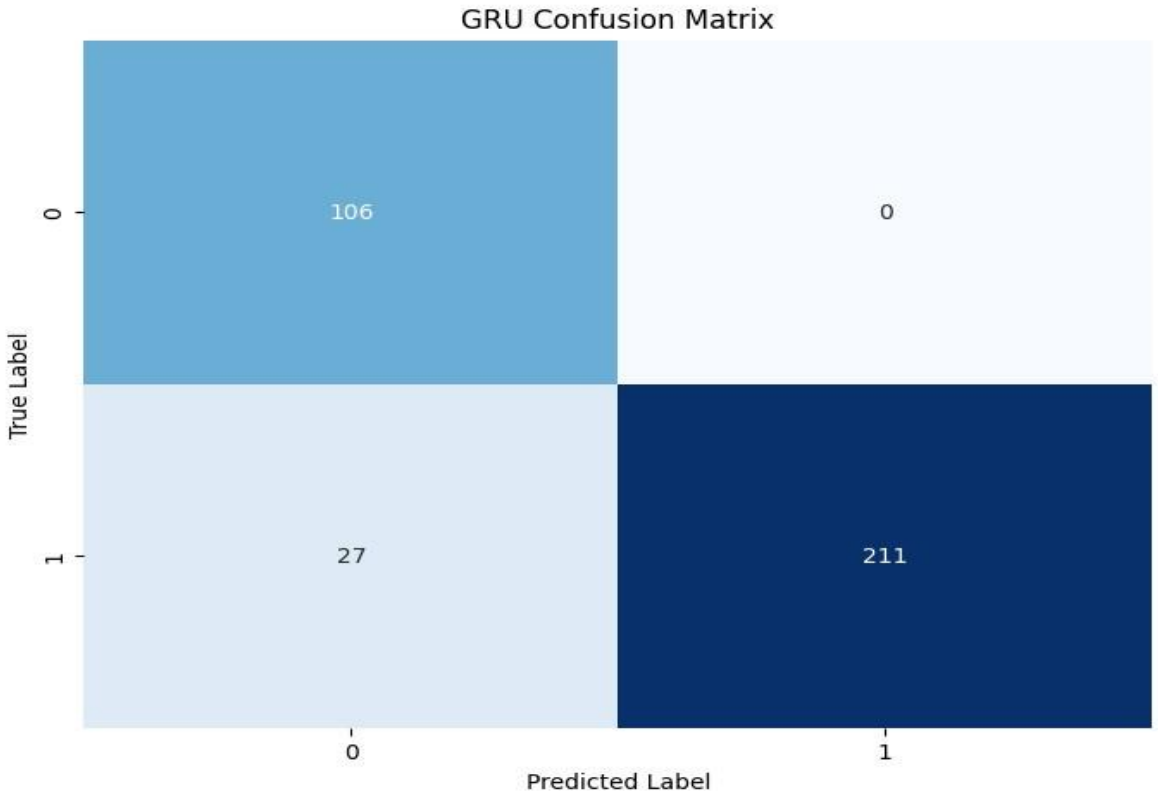


Figure 6-18 GRU confusion matrix

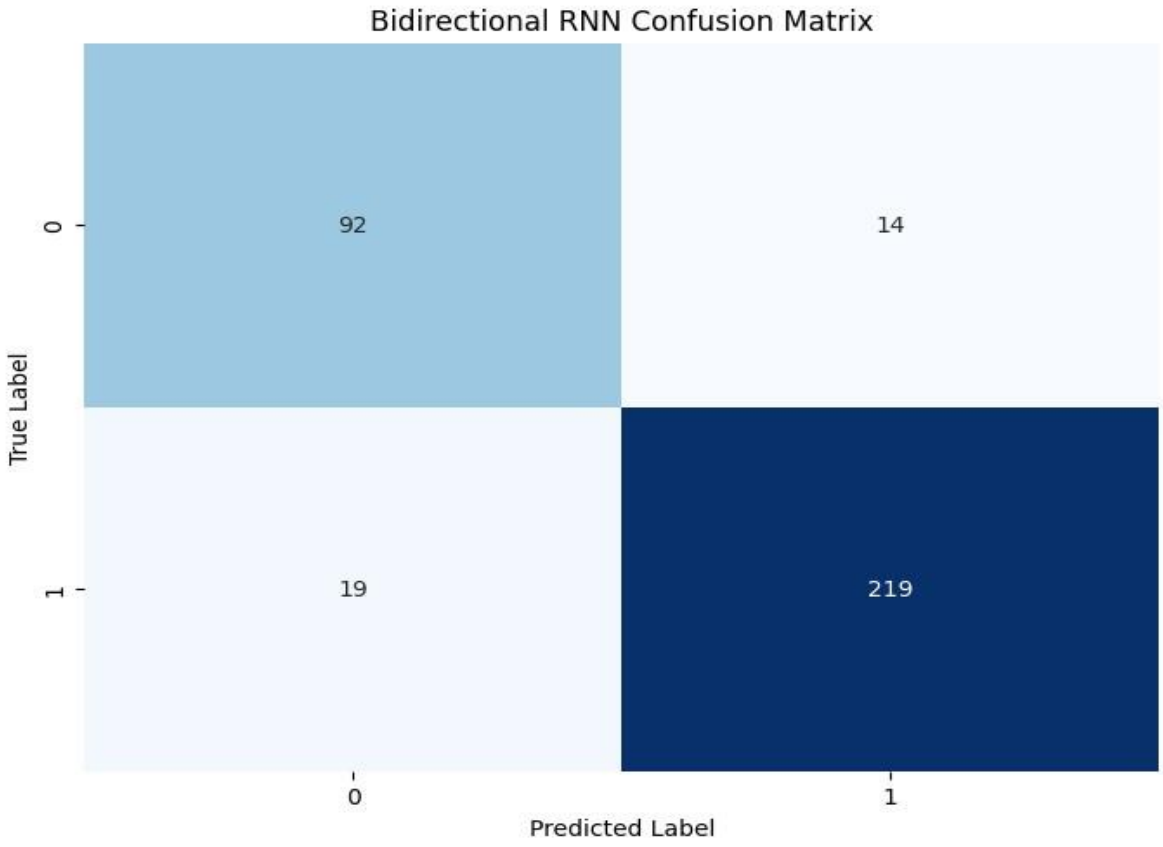


Figure 6-19 Bidirectional RNN confusion matrix

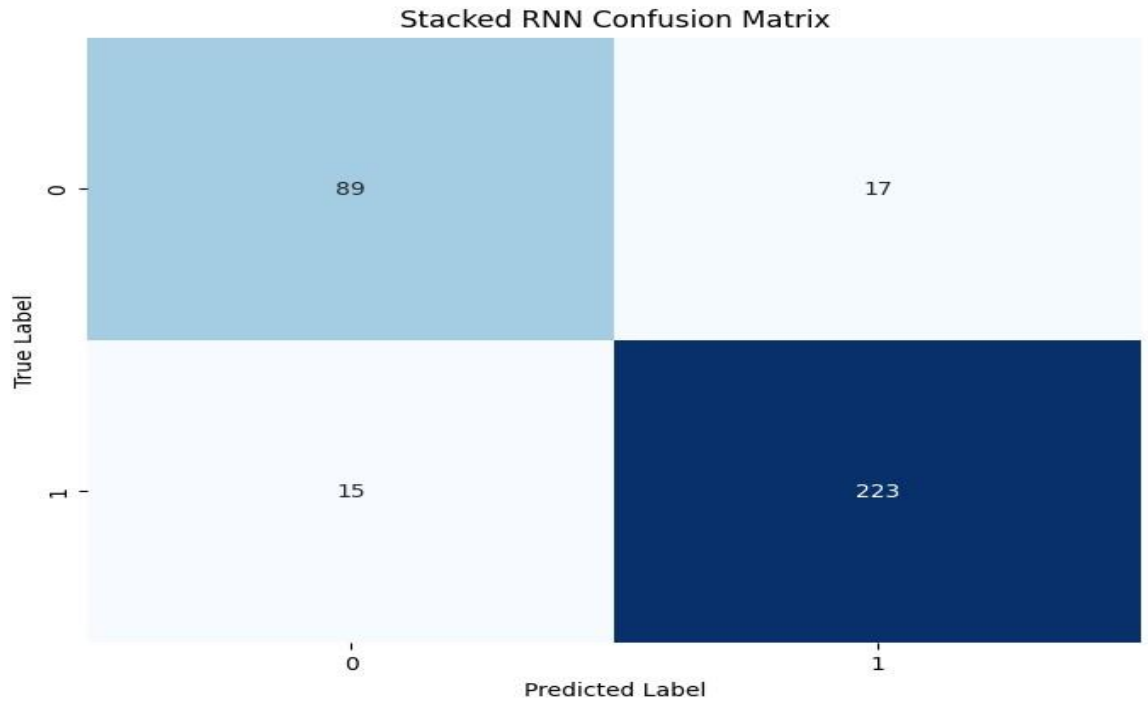


Figure 6-20 Stacked RNN confusion matrix

The provided code snippet is used to visualize the confusion matrices for various deep learning models. The first step is to import the necessary library, ‘matplotlib.pyplot’ which is commonly used for creating and customizing plots. The core of the code is the ‘plot_confusion_matrix’ function, which takes the confusion matrix ‘cm’ and the model name ‘model_name’ as two arguments.

Inside the ‘plot_confusion_matrix’ function, the code creates a heatmap representation of the confusion matrix using the ‘plt.imshow’ function. The ‘interpolation = nearest’ argument ensures that the heatmap is displayed without any interpolation between the cells, and the ‘cmap = plt.cm.Blues’ argument sets the color map to shades of blue. Which is a common choice for visualizing confusion matrices. The function also adds a title, a colorbar, and labels for the x and y axes. Finally, the ‘plt.show ()’ function is called to display the plot. The second part of the code calls the ‘plot_confusion_matrix’ function for each of the models (‘lstm_cm’, ‘gru_cm’, ‘bidirectional_cm’, and ‘stacked_cm’), passing the corresponding confusion matrix and model name as arguments. This allows the user to visualize the confusion matrices for different deep learning models which can be helpful in evaluating their performance and understanding the distribution of true positives, true negatives, false positives, and false negatives.

6.5. Summary

The study systematically evaluated the capacity of various recurrent neural network (RNN) models LSTM, GRU, Bidirectional RNN, and Stacked RNN applied to sentence similarity tasks. The investigation incorporated three different embedding's like Word2Vec, Glove, and Universal Sentence Encoder (USE). The key findings revealed significant variations in the models' predictive capabilities with the chosen embedding's.

Universal Sentence Encoder consistently emerged as the top-performing embedding across all RNN architectures displaying the MSE in lowest and the highest accuracy. This underscores the effectiveness of USE in capturing nuanced semantic connection between sentences aligning with its design as a versatile and context-aware embedding.

Glove embedding's also demonstrated competitive performance consistently outperforming Word2Vec regarding accuracy and MSE. Word2Vec while still yielding reasonable results consistently lagged behind USE and Glove emphasizing the significance of considering contextual embedding's for accurate sentence similarity predictions.

The study's comprehensive evaluation not only highlighted the advantage of different embedding's but also provided valuable guidance for researchers and practitioners in natural language processing. The findings emphasize the significance of embedding choices in enhancing the capacity of RNN models for sentence similarity tasks. Further research and fine-tuning of hyper-parameters may offer opportunities for future even more refined models. Overall, the study contributes valuable insights to the area aiding decision-making in the selection of embedding's for specific NLP applications.

6.1. Evaluation by Linguists experts

We conducted a thorough evaluation of our training model's performance in measuring Semantic Textual Similarity (STS) by utilizing a score evaluation metric. This metric allowed us to quantitatively assess the effectiveness of our model in capturing semantic connection between pair of sentences but we accept the significance of validating our model's results against actual scores to ensure the reliability and acceptability of our proposed approach. By comparing the predicted similarity scores generated by our model with the actual similarity scores. We could gain valuable insights into the model's accuracy and its ability to align with human judgment.

To comprehensive evaluation we carefully handpicked three experts in the Guragigna language each with a unique background. One expert hailed from the culture and tourism office another

from the Cheha Wereda Betekihinet office (ቸሃ ወረዳ ቤተክህነት ጽ/ቤት), and the third from the education office. The selection of these experts was with their deep understanding and extensive knowledge of Guragigna language concepts. Moreover their familiarity with the specific nuances and contextual usage of the language. Their expertise and perspectives were crucial in evaluating the STS generated by our proposed model accurately. In the provided table 6-2, we present a sample of sentence pairs along with the predicted similarity scores from each model in addition to the corresponding actual similarity scores.

Table 6-2 Comparing predicted similarity scores with the actual similarity scores

Sentence1	Sentence 2	Actual score	LSTM Prediction	GRU Prediction	Stacked-RNN Prediction	Bi-RNN Prediction
ዜማንቸ አሬ ይቅረዎ	ወጅመነ አሬ ይቅረዎ	1	0.867	0.856	0.983	0.985
ዋጋ ዝፐረም	ዋጋ አቸነም	1	0.856	0.874	0.935	0.941
ይማትእማት ዝረክዎም	የሚርሚሬ ዝረክዎም	1	0.832	0.821	0.925	0.933
የበኽር ትክ ባን	ይንምንትቅ ትክ ባን	0.8	0.652	0.651	0.714	0.702
የዘበርየ ባሸ	የገዝየ ባሸ	0.8	0.652	0.651	0.731	0.724
የዙርየ ኣትሕር	ጀግጀግተነ ኣትሕር	0.8	0.650	0.652	0.784	0.795
የደመደ ሰበ ባን	የትስበሰበ ሰበ ባን	0.8	0.651	0.653	0.7	0.7
ሜና ያሚ ባነ	ሜና ይቾት ባነ	0.8	0.658	0.657	0.794	0.798
አትፋጥርም ሰናም	ጀፐረም ሰናም	0.8	0.634	0.633	0.744	0.752
ሐልቅ ተትግሬ ገነ ቸነም	እስያም ያነቦ ሰብ	0.3	0.112	0.102	0.203	0.213
ጉራ ቲውሪ መደር ቸነም	ባሮም ይትክራክሮ	0.1	0.051	0.057	0.178	0.184
ተሓረር ቸነም	ቲቶፕያ እስያ ወሮም	0.2	0.101	0.136	0.241	0.228
ተጉራም ሐረ ተሓረር ቸነም	የትዦፐሮ ሰብ ነረ	0.1	0.067	0.069	0.81	0.94

ሕኖ ይፍት ወረር ተደቡብ ቸነም	ይሕርሽ ይብሮ	0.1	0.214	0.321	0.687	0.954
የቸነቦ ሐማ ዩጂ	ሕም ሐረ ዝሕ	0.1	0.361	0.354	0.841	0.862

6.2. Answering Research Questions

RQ1. Which word embedding techniques can be used for Model development that can determine the effectiveness and robustness of Semantic Text Similarity (STS)?

The study employs three embedding techniques Such as Word2Vec, Glove, and Universal Sentence Encoder (USE). These techniques map words to continuous vector representations capturing semantic relationships based on co-occurrence patterns. The study's results highlight the effectiveness of these embedding techniques particularly emphasizing the robust of the Universal Sentence Encoder (USE) performance across different deep learning models.

Various embedding techniques contribute to make effective and robust models for Semantic Text Similarity (STS) in Guragigna. Word embedding's such as Word2Vec, Glove, and Universal Sentence Encoder (USE) offer semantic representations for words and sentences capturing contextual relationships. These embedding's are fundamental for encoding the semantic content of Guragigna text. Word2Vec, Glove, and USE can be taken as embedding of word techniques for model development to determine the effectiveness and robustness of Semantic Text Similarity (STS). Among them, USE tends to provide the best performance with the above table.

RQ2. Which deep learning model is the most effective in performing Semantic Text Similarity (STS) analyses for the Guragigna language?

With the experiment comparing the predicted similarity scores by different models with the actual similarity scores we can evaluate the effectiveness of each deep learning model in performing Semantic Text Similarity (STS) analyses for the Guragigna language. It appears that the Bidirectional RNN model consistently performs the best among the four models (LSTM, GRU, Stacked-RNN, and Bi-RNN) regarding aligning with the actual similarity scores. The predicted similarity scores generated by the Bidirectional RNN model are closer to the scores of actual compared to the other models for most of the sentence pairs. As example, in the first sentence pair ("ዜማንች አሬ ይቅረዎ" vs. "ወጅመነ አሬ ይቅረዎ") the Bidirectional RNN

model predicts a similarity score of 0.985 which is very close to the actual score of 1. Similar observations can be made for other sentence pairs as well where the predicted scores by the Bidirectional RNN model are consistently closer to the scores of actual compared with each models. Therefore, in last results of evaluation we can conclude that the Bidirectional RNN model is the most effective deep learning model for performing Semantic Text Similarity (STS) analyses for the Guragigna language. It demonstrates a better capacity to take the semantic connection between pair of sentences and produces similarity scores that align well with the actual scores.

CHAPTER SEVEN

7. CONCLUSION AND RECOMMENDATION

7.1. Overview

This chapter serves as the end of the entire study providing a comprehensive conclusion and recommendation regarding the contributions and research findings. It present the study overview the key outcomes and suggests potential areas for future research that can be pursued by upcoming researchers.

7.2. Conclusion

This study focused on developing a deep learning based approach for measuring semantic textual similarity in the Guragigna language. Through extensive research and experimentation, several key findings and contributions have been made.

Firstly, the study proposed approach that leverages deep learning techniques which is recurrent neural networks (RNNs) models including LSTM, GRU, and BI-RNN and stacked RNN to accurately measure semantic textual similarity. This approach outperformed traditional string-based and corpus-based methods highlighting its effectiveness in capturing the semantic meaning of text in the Guragigna language.

Furthermore, the study contributed to the area by creating a specific Guragigna language corpus and implementing a preprocessing pipeline tailored to this language. This dataset and preprocessing techniques serve as valuable resources for future research on Guragigna language processing tasks. The evaluation of various models and embedding techniques for measuring semantic textual similarity in the Guragigna language yielded significant findings. The LSTM, GRU, Bi-RNN, and Stacked RNN models demonstrated commendable performance in capturing the semantic information inherent to the Guragigna language and delivering accurate similarity measurements. Notably, the selection of embedding technique had a discernible impact on the model's performance. The USE (Universal Sentence Encoder) consistently outperformed Word2Vec and Glove embedding’s exhibiting MSE is lower values and higher accuracy scores across all models. This suggests that the USE embedding technique effectively captured the Guragigna language's semantic nuances. The LSTM model emerged as a standout performer showcasing consistently low MSE values and high accuracy scores regardless of the embedding technique employed. The GRU model also displayed competitive performance albeit with slightly lower scores of accuracy compared to LSTM. Similarly the Bi-RNN model demonstrated comparable results to LSTM and GRU highlighting the potential

of bidirectional recurrent neural networks in capturing semantic information for similarity measurement. The Stacked RNN model presented slightly higher MSE values but remained capable of measuring the accuracy of semantic textual similarity. The findings emphasize the efficacy of LSTM, GRU, Bi RNN, and Stacked RNN models in measuring semantic textual similarity in the Guragigna language with the USE embedding technique consistently delivering superior results. These insights provide a solid foundation for further model optimization and the organization of practical applications in semantic textual similarity measurement for the Guragigna language.

The research results demonstrate the effectiveness of LSTM, GRU, Bi RNN, and Stacked RNN models in measuring semantic textual similarity in the Guragigna language. The selected embedding technique such as Word2Vec, Glove, or USE influenced the capacity of the models with USE consistently showing the best results. These findings get valuable understanding for further model optimization and the development of practical applications in semantic textual similarity measurement in the Guragigna language.

Significant advancements in the field of natural language processing have been accomplished by this study and semantic textual similarity measurement in under-resourced languages. The findings and recommendations presented here provide a foundation for further research and contribute to advancing natural language understanding in the Guragigna language and beyond.

7.3. Contribution and challenges

Significant research output resulted from the study of quantifying semantic textual similarity in the Guragigna language. It involved to implement the LSTM, GRU, Bi RNN, and Stacked RNN models specifically designed for this purpose. These models incorporated deep learning architectures to effectively capture the semantic meaning of text in Guragigna. The research also examined different embedding techniques such as Word2Vec, Glove, and USE (Universal Sentence Encoder) to represent text in a context-aware manner. Evaluation metrics is Mean Squared Error (MSE), loss, and accuracy were used to assess the models' performance. The results of the study provided detailed performance analysis of each model and embedding technique highlighting the strengths of each approach and the weaknesses of each approach. Additionally, recommendations were made for more study including exploring multilingual and cross-lingual similarity measurements and collaborating with domain experts to enhance the models further. The creation of a Guragigna language corpus and implementation of a preprocessing pipeline specific to this language were also important participation of the study. Overall, the research output advances the field of natural language processing by providing

valuable understanding and resources for measuring semantic textual similarity in the Guragigna language.

One of the primary challenges in developing semantic textual similarity (STS) for the Guragigna language is the lack of sufficient data. Guragigna, being a less commonly studied language, suffers from insufficient or non-existent digital linguistic resources. The scarcity of annotated datasets, corpora, and other linguistic resources makes it difficult to train and evaluate deep learning models effectively. Moreover, the data that is available is often unstructured and not organized in a format suitable for STS tasks. This lack of structured data Obstacle the ability to preprocess and analyze text efficiently, further complicating the development of accurate and reliable STS models. Another significant challenge is the expertise gap. Developing STS models for any language requires a solid understanding of data analysis and statistical methods, which is compounded when dealing with the unique linguistic characteristics of the Guragigna language and the absence of essential preprocessing tools, such as stemmers and lemmatizes, presents another formidable challenge.

7.4. Future work

Future work in measuring semantic textual similarity in the Guragigna language can build upon the existing research and explore the following areas:

- This work focus to implement semantic textual similarity in the Guragigna language only for sentence similarity. We suggest that additional studies conduct paragraph similarity and document similarity.
- Explore multimodal approaches that incorporate both textual and visual information for measuring semantic similarity. This can involve incorporating visual features from images or videos along with text to capture a broader understanding of meaning and accuracy improvement of the similarity measurements.
- Invest in the creation of more linguistic resources for the Guragigna language such as annotated corpora or lexicons which can aid in training and evaluating similarity models and other language processing tasks and Seek feedback from users and domain experts to understand the practical utility of the similarity measurements and continuously improve the models based on real-world requirements and use cases.

7.5. Recommendation

With this work findings, several recommendations can be made to further improve and explore the measurement of semantic textual similarity in the Guragigna language. Firstly, investigating ensemble approaches by combining the predictions of multiple models can enhance the overall accuracy and robustness of similarity measurement. Techniques like model averaging, stacking, or boosting can be explored to leverage the diverse strengths of different models. Secondly, adapting the models to specific domains within the Guragigna language, such as medical or legal texts can be beneficial. By models training on domain-specific datasets higher accuracy and relevance can be achieved in measuring similarity within those domains. Thirdly, exploring cross-lingual similarity measurement by models training on multilingual data can facilitate communication and understanding across different languages, enabling retrieval of information and cross-lingual applications. Additionally, expanding the size and diversity of the Guragigna language dataset used for training and evaluation is crucial. A larger dataset can enhance the models' generalization capabilities and performance across a wider range of texts and topics. Lastly, applying the developed models to real-world applications like question answering systems, machine translation, retrieval of information and text summarization in the Guragigna language can provide practical use cases and further the effectiveness of the models in measuring semantic textual similarity. These recommendations contribute to advancing the field and enhancing the applicability of semantic textual similarity measurement in the Guragigna language.

REFERENCES

[1] D. Chopra, I. Mathur, and N. Joshi, *Mastering natural language processing with Python : maximize your NLP capabilities while creating amazing NLP projects in Python*. 2016.

[2] M. Shajalal and M. Aono, "Semantic textual similarity between sentences using bilingual word semantics," *Prog. Artif. Intell.*, vol. 8, no. 2, pp. 263–272, Jun. 2019, doi: 10.1007/s13748-019-00180-4.

[3] B. A. Kipfer, "Population, Statistical," *Encycl. Dict. Archaeol.*, pp. 1083–1083, 2021, doi: 10.1007/978-3-030-58292-0_160816.

[4] Fitsum Gizachew, "Developing Part of Speech Tagger for Guragigna Language," *Master's thesis*, no. November, 2020, doi: 10.13140/RG.2.2.30209.84325.

[5] T. Haile, "Devloping Automatic Character Recoginition System for Guragigna Printed Real-Life Document Addis Ababa, Ethiopia," no. July, 2020.

[6] M. A. Iqbal, O. Sharif, M. M. Hoque, and I. H. Sarkar, "Word Embedding based Textual Semantic Similarity Measure in Bengali," in *Procedia Computer Science*, Elsevier B.V., 2021, pp. 92–101. doi: 10.1016/j.procs.2021.10.010.

[7] E. L. Pontes, S. Huet, A. C. Linhares, and J.-M. Torres-Moreno, "Predicting the Semantic Textual Similarity with Siamese CNN and LSTM," Oct. 2018, [Online]. Available: http://arxiv.org/abs/1810.10641

[8] M. Al Sulaiman, A. M. Moussa, S. Abdou, H. Elgibreen, M. Faisal, and M. Rashwan, "Semantic textual similarity for modern standard and dialectal Arabic using transfer learning," *PLoS One*, vol. 17, no. 8 August, Aug. 2022, doi: 10.1371/journal.pone.0272991.

[9] "Cross-Language Semantic Text Similarity Measurement using Statistical Topic Model: The Case of Amharic-English Languages".

[10] I. Matveeva, G. A. Levow, A. Farahat, and C. Royer, "Term representation with generalized latent semantic analysis," *Int. Conf. Recent Adv. Nat. Lang. Process. RANLP*, vol. 2005-Janua, pp. 308–315, 2005, doi: 10.1075/cilt.292.08mat.

[11] J. Steinberger and K. Ježek, "Using Latent Semantic Analysis in Text Summarization," *Proc. ISIM 2004*, no. May, pp. 93--100, 2004.

[12] D. Sánchez, M. Batet, D. Isern, and A. Valls, “Ontology-based semantic similarity: A new feature-based approach,” *Expert Syst. Appl.*, vol. 39, no. 9, pp. 7718–7728, 2012, doi: 10.1016/j.eswa.2012.01.082.

[13] M. A. Rodríguez and M. J. Egenhofer, “Determining Semantic Similarity Among Entity Classes from Different Ontologies M. Andrea Rodríguez and Max J. Egenhofer IEEE Transactions on Knowledge and Data Engineering,” *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 2, pp. 3–3, 2003, [Online]. Available: file:///Files/C6/C622BF5E-92D5-49B8-A8B0-2371162F6AB1.pdf

[14] Y. Jiang, X. Zhang, Y. Tang, and R. Nie, “Feature-based approaches to semantic similarity assessment of concepts using Wikipedia,” *Inf. Process. Manag.*, vol. 51, no. 3, pp. 215–234, 2015, doi: 10.1016/j.ipm.2015.01.001.

[15] G. Salton, A. Singhal, M. Mitra, and C. Buckley, “Automatic text structuring and summarization,” *Inf. Process. Manag.*, vol. 33, no. 2, pp. 193–207, 1997, doi: 10.1016/S0306-4573(96)00062-3.

[16] G. V. Bard, “Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric,” *Conf. Res. Pract. Inf. Technol. Ser.*, vol. 68, pp. 117–124, 2007.

[17] E. Schallehn, K. U. Sattler, and G. Saake, “Efficient similarity-based operations for data integration,” *Data Knowl. Eng.*, vol. 48, no. 3, pp. 361–387, 2004, doi: 10.1016/j.datak.2003.08.004.

[18] W. E. Winkler, “Overview of record linkage and current research directions,” *Current*, no. 2006–2, pp. 1–28, 2006, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.1519

[19] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970, doi: 10.1016/0022-2836(70)90057-4.

[20] A. Barrón-Cedeño, P. Rosso, E. Agirre, and G. Labaka, “Plagiarism detection across distant language pairs,” *Coling 2010 - 23rd Int. Conf. Comput. Linguist. Proc. Conf.*, vol. 2, no. August, pp. 37–45, 2010.

[21] K. L. Poore, “DigitalCommons @ University of Nebraska - Lincoln Taxicab Geometry Taxicab Geometry Expository Paper Kyle Lannin Poore,” 2006.

[22] B. de Ville, *Introduction to Data Mining*. 2001. doi: 10.1016/b978-155558242-5/50003-6.

[23] G. Sidorov, A. Gelbukh, H. Gómez-Adorno, and D. Pinto, "Soft similarity and soft cosine measure: Similarity of features in vector space model," *Comput. y Sist.*, vol. 18, no. 3, pp. 491–504, 2014, doi: 10.13053/CyS-18-3-2043.

[24] G. Sidorov, H. Gomez-Adorno, I. Markov, D. Pinto, and N. Loya, "Computing text similarity using Tree Edit Distance," *Annu. Conf. North Am. Fuzzy Inf. Process. Soc. - NAFIPS*, vol. 2015-Septe, no. October, 2015, doi: 10.1109/NAFIPS-WConSC.2015.7284129.

[25] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species Author (s): Lee R . Dice Published by : Ecological Society of America Stable URL : http://www.jstor.org/stable/1932409," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[26] P. A. Zeitschrift, S. N. Band, P. Link, and E. Dienst, "Étude comparative de la distribution florale dans une portion des Alpes et du Jura," no. June, 2013, doi: 10.5169/seals-266450.

[27] L. Han, A. Kashyap, T. Finin, J. Mayfield, and J. Weese, "UMBC EBIQUITY-CORE: Semantic Textual Similarity Systems."

[28] T. K. Landauer and S. T. Dumais, "Psychological Review A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge," 1997.

[29] P. W. Laham, "Introduction to Latent Semantic Analysis," 1998.

[30] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A Comparison of String Distance Metrics for Name-Matching Tasks." [Online]. Available: www.aaai.org

[31] D. Chandrasekaran and V. Mago, "Evolution of Semantic Similarity-A Survey," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–35, 2021, doi: 10.1145/3440755.

[32] J. A. Hansen, E. K. Ringger, and K. D. Seppi, "Probabilistic explicit topic modeling using Wikipedia," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8105 LNAI, pp. 69–82, 2013, doi: 10.1007/978-3-642-40722-2_7.

[33] D. Sánchez, M. Batet, and D. Isern, "Ontology-based information content computation,"

Knowledge-Based Syst., vol. 24, no. 2, pp. 297–303, 2011, doi: 10.1016/j.knosys.2010.10.001.

[34] R. A. Sinoara, J. Camacho-Collados, R. G. Rossi, R. Navigli, and S. O. Rezende, "Knowledge-enhanced document embeddings for text classification," *Knowledge-Based Syst.*, vol. 163, pp. 955–971, 2019, doi: 10.1016/j.knosys.2018.10.026.

[35] R. L. Cilibrasi and P. M. B. Vitányi, "The Google similarity distance," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 3, pp. 370–383, 2007, doi: 10.1109/TKDE.2007.48.

[36] J. Du, B. Wang, Y. Liu, G. chun Yao, Z. Fang, and P. Hu, "Study on the Bubble Behavior and Anodic Overvoltage of NiFe2O4 Ceramic Based Inert Anodes," *Light Met. 2015*, no. June, pp. 1193–1197, 2015, doi: 10.1002/9781119093435.ch200.

[37] Y. Le, Z. J. Wang, Z. Quan, J. He, and B. Yao, "ACV-tree: A new method for Sentence similarity modeling," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2018-July, pp. 4137–4143, 2018, doi: 10.24963/ijcai.2018/575.

[38] K. L. Wudineh, "ADDIS ABABA UNIVERSITY SCHOOL OF GRADUATE STUDIES SCHOOL OF INFORMATION STUDIES FOR AFRICA DESIGN AND DEVELOPMENT OF AUTOMATIC MORPHOLOGICAL SYNTHESIZER FOR AMHARIC PERFECTIVE VERB FORMS A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE," 2002.

[39] R. Rada, H. Mili, E. Bicknell, and M. Blettner, "Development and Application of a Metric on Semantic Nets," *IEEE Trans. Syst. Man Cybern.*, vol. 19, no. 1, pp. 17–30, 1989, doi: 10.1109/21.24528.

[40] Y. Li, Z. A. Bandar, and D. McLean, "An approach for measuring semantic similarity between words using multiple information sources," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 4, pp. 871–882, 2003, doi: 10.1109/TKDE.2003.1209005.

[41] S. Banerjee and T. Pedersen, "Extended gloss overlaps as a measure of semantic relatedness," *IJCAI Int. Jt. Conf. Artif. Intell.*, pp. 805–810, 2003.

[42] J. J. Lastra-Díaz, J. Goikoetxea, M. A. Hadj Taieb, A. García-Serrano, M. Ben Aouicha, and E. Agirre, "A reproducible survey on word embeddings and ontology-based methods for word similarity: Linear combinations outperform the state of the art," *Eng. Appl. Artif. Intell.*, vol. 85, no. February, pp. 645–665, 2019, doi: 10.1016/j.engappai.2019.07.010.

[43] D. Sánchez and M. Batet, “A semantic similarity method based on information content exploiting multiple ontologies,” *Expert Syst. Appl.*, vol. 40, no. 4, pp. 1393–1399, 2013, doi: 10.1016/j.eswa.2012.08.049.

[44] G. Zhu and C. A. Iglesias, “of Concepts in Knowledge Graphs,” *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 72–85, 2017.

[45] P. Resnik, “Using Information Content to Evaluate Semantic Similarity in a Taxonomy,” vol. 1, 1995, [Online]. Available: http://arxiv.org/abs/cmp-lg/9511007

[46] Dekang Lin, “An information-theoretic definition of similarity,” *Icml*, vol. 98, no. 1998, pp. 296–304, 1998.

[47] J. D. W. C. J.Jiang, “Semantic similarity based on corpus statistics and lexical taxonomy.” 1997.

[48] Y. Jiang, W. Bai, X. Zhang, and J. Hu, “Wikipedia-based information content and semantic similarity computation,” *Inf. Process. Manag.*, vol. 53, no. 1, pp. 248–265, 2017, doi: 10.1016/j.ipm.2016.09.001.

[49] Z. H. Amur, Y. Kwang Hooi, H. Bhanbhro, K. Dahri, and G. M. Soomro, “Short-Text Semantic Similarity (STSS): Techniques, Challenges and Future Perspectives,” *Appl. Sci.*, vol. 13, no. 6, 2023, doi: 10.3390/app13063911.

[50] T. L. Feleke, “Determinants of language change in the Gurage area of Ethiopia,” *J. World Lang.*, vol. 9, no. 2, pp. 253–288, 2023, doi: 10.1515/jwl-2022-0024.

[51] C. M. Ford, “Notes on the phonology and grammar of Chaha Gurage,” *Journal of Afroasiatic languages*, vol. 2, no. 3. pp. 231–296, 1991.

[52] M. Bulakh, “Endalew Assefa, Descriptive grammar of Ezha, a Gurage language of Ethiopia (Ethio-Semitic),” *Linguist. langues africaines*, vol. 9, no. 9(1), pp. 0–4, 2023, doi: 10.4000/lla.4775.

[53] H. Foster, “Towards a grammar of emergency,” *New Left Rev.*, no. 68, pp. 105–118, 2011.

[54] M. Mansoor, Z. Ur Rehman, M. Shaheen, M. A. Khan, and M. Habib, “Deep learning based semantic similarity detection using text data,” *Inf. Technol. Control*, vol. 49, no. 4, pp. 495–510, 2020, doi: 10.5755/j01.itc.49.4.27118.

[55] T. Ranasinghe, C. Orˇ, and R. Mitkov, “Semantic Textual Similarity with Siamese Neural Networks.” [Online]. Available: https://github.com/

[56] D. Dasari and P. S. Varma, “Data Cleaning Techniques Using Python,” vol. 1, pp. 11–21.

[57] J. Kaur, “Stopwords Removal and Its Algorithms Based on Different Methods,” *Int. J. Adv. Res. Comput. Sci.*, vol. 9, no. 5, pp. 81–88, 2018, doi: 10.26483/ijarcs.v9i5.6301.

[58] A. Nigam, A. Dhruv, and F. J. J. S, “TEXT PRE-PROCESSING AND FEATURE EXTRACTION USING NLP,” no. 06, pp. 1550–1554, 2021.

[59] D. Cer *et al.*, “Universal sentence encoder for English,” *EMNLP 2018 - Conf. Empir. Methods Nat. Lang. Process. Syst. Demonstr. Proc.*, pp. 169–174, 2018, doi: 10.18653/v1/d18-2029.

[60] HANSON ER, “GloVe: Global Vectors for Word Representation,” *AES J. Audio Eng. Soc.*, vol. 19, no. 5, pp. 417–425, 1971.

[61] S. Devasahayam, “Deep learning models in Python for predicting hydrogen production: A comparative study,” *Energy*, vol. 280, no. February, p. 128088, 2023, doi: 10.1016/j.energy.2023.128088.

[62] L. Yao, Z. Pan, and H. Ning, “Unlabeled Short Text Similarity with LSTM Encoder,” *IEEE Access*, vol. 7, no. c, pp. 3430–3437, 2019, doi: 10.1109/ACCESS.2018.2885698.

[63] A. Anand, T. Chakraborty, and N. Park, “We used neural networks to detect clickbaits: You won’t believe what happened next!,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10193 LNCS, pp. 541–547, 2017, doi: 10.1007/978-3-319-56608-5_46.

[64] M. Zulqarnain, R. Ghazali, M. G. Ghouse, and M. F. Mushtaq, “Efficient processing of GRU based on word embedding for text classification,” *Int. J. Informatics Vis.*, vol. 3, no. 4, pp. 377–383, 2019, doi: 10.30630/joiv.3.4.289.

[65] P. S. Mashhadi, S. Nowaczyk, and S. Pashami, “Stacked ensemble of recurrent neural networks for predicting turbocharger remaining useful life,” *Appl. Sci.*, vol. 10, no. 1, 2020, doi: 10.3390/app10010069.

[66] J. Austen, “recurrent neural network (RNN) and long short-term - (LSTM) memory,” 2024.

APPENDICES

Appendix A

1. Sample of Corpus

Sentence1	Sentence2	Similarity Score
ተጨራ ሐርም ቶት	ተጨራ ብሰርም ቶት	1
ይብቴ የዋረን ዘጋ ሐዳም	ይብቴ በዋረን ዘጋ ቶዘከበም	1
ሕምተሕም ቸነተም	ሕምጊ ቸነተም	1
ለህና ቐጠመ	አርዋና ቐጠመ	1
ግሬድ ቄ ላሌ ይትፍቐሮ ባነ	ግሬድ ቄ ፍቐረ ይትፍቐሮ ባነ	1
ሌባ ጠውጪም አጐጂም	ፈንገያ ጠውጪም አጐጂም	1
መስቀር ሰናም	ወኽመያ ስናም	1
ዋጋ መኬም	ዋጋ አገኔም	1
የሰር ቤት መኽረም	የሰር ቤት ተከሰም	1
ገረድቄ ጣባ□ታ ሼመችም	ገረድቄ እንቁስባረችም	0.8
ሚራሐ ጥብጥ	ሚራሐ ድምድ	0.8
ጨቐሰም ነከበም	ተሳረም ነከበም	0.8
ቦሔም ጨነችም	ቦሔም ተነፈችም	0.9
ጨንቅራ ጠበጥችም ወረችም	የሰገራትከ ጠበጥችም ወረችም	0.9
ጉራ ቲውሪ መደር ቸነም	ቸነም ባሮም ይትክራከሮ	0.1
ተሓረር ወረም ቸነም	ቲቶፕያ እስያ ወሮም	0.2
ሕኖ ይፍት ወረር ተደቡብ ቸነም	ሕኖ ይሕርሽ ይብሮ	0.1
የቸነቦ ሐማ ዩጂ	ሕም ዩጂ ሐረ	0.1
የጉራጌ ሐልቅ ተትግሬ ገነ ቸነም	የጉራጌ ገነ እስያም ያነቦ ሰብ	0.3

Appendix B

2. Punctuation marks commonly used in Guragigna Language

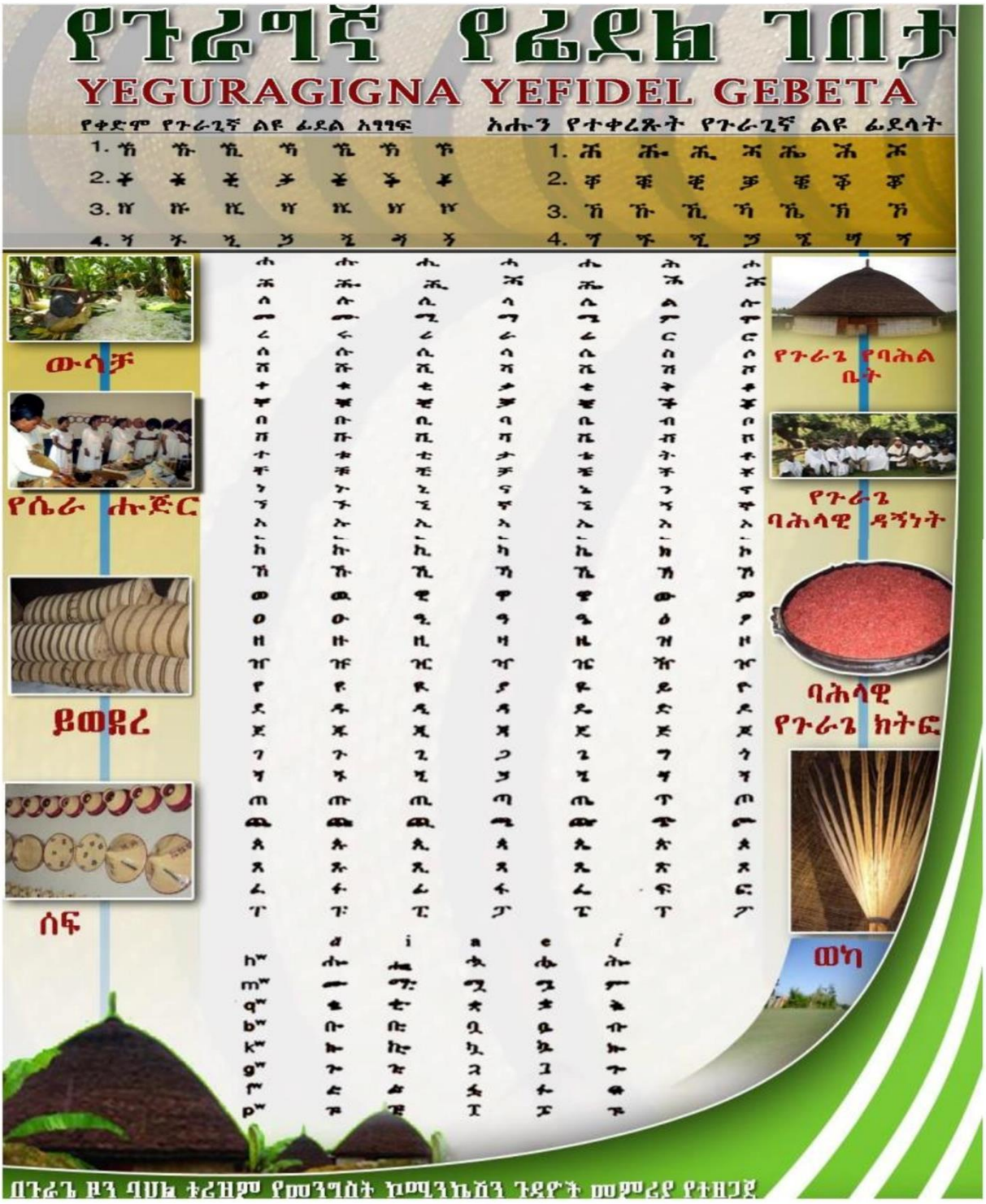
Full Stop	።።	Question Mark	?
colon	፥	hyphenation	-
Comma	፣	Double Quotation Mark	« »
Semicolon	፤	Single Quotation Mark	‹›
Preface Colon	፡-	slash	/

Appendix C

3. Sample List of Stop Word of Guragigna

No.	Word	No.	Word	No.	Word
1.	መደር	2.	የሐረ	3.	እክ
4.	ም	5.	ኤሐር	6.	ቤ
7.	ምስ	8.	ነረ	9.	ኧኀ
10.	ምር	11.	እክም	12.	ሐት
13.	ምሽት	14.	የሚኒመታ	15.	ብር
16.	ሰብ	17.	ሕ	18.	ተዝበፎር
19.	ሽም	20.	አተነ	21.	በሐት
22.	ቃር	23.	ቄም	24.	ባሐረ
25.	በር	26.	ተዝ	27.	ቤማ
28.	ባረም	29.	በሕ	30.	ሕምኀ
31.	ባረችም	32.	በሕም	33.	እያ
34.	ባሬም	35.	ይረዝ	36.	ኸማ
37.	ቤት	38.	ባነ	39.	ቄም
40.	ታነ	41.	ስን	42.	ቄሸ
43.	ናማጋ	44.	ሓሐና	45.	ዘንጋ
46.	አርብ	47.	አቂ	48.	ዛሀ
49.	እኳ	50.	ሕሐታ	51.	ዝርጋት
52.	የምር	53.	ዳር	54.	
55.	ሞጭን	56.	ጋመ	57.	

Appendix D



1. Alphabete of Guragigna

Appendix E

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
import gensim.downloader as api
import tensorflow as tf
from tensorflow.keras.layers import GRU
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, SimpleRNN, Dense, Concatenate, Bidirectional
from tensorflow.keras.initializers import Constant
```

2. Required Python Libraries

Appendix F

```
Sample Train Data:
Sentence1: ዴኝ ቁየ ይትዋዶ, Sentence2: ዴኝ ቁየ ይጮዶ, Actual Similarity Score: 0.8
Sentence1: ቦሔም ጨነችም , Sentence2: ቦሔም ተነፈችም, Actual Similarity Score: 0.9
Sentence1: ዱየ ቄ ያትሸንፍ, Sentence2: ጠነማ ቄ ያትሸንፍ, Actual Similarity Score: 0.8
Sentence1: አርዳ ያሕር, Sentence2: ንቅየ ያሕር, Actual Similarity Score: 1.0
Sentence1: ፊርዝ ነኽመም , Sentence2: ወናድ ነኽመም, Actual Similarity Score: 0.9
Sentence1: ነደደም በኽም, Sentence2: ጋረም በኽም  , Actual Similarity Score: 1.0
Sentence1: ገረድ ኽጣ ያ, Sentence2: ገረድ ቆጣ ያ, Actual Similarity Score: 1.0
Sentence1: ግብር አቸፐረም, Sentence2: ግብር አበም, Actual Similarity Score: 1.0
Sentence1: እነ ሸኸተም, Sentence2: የጌንዞኣጨ ሸኸተም, Actual Similarity Score: 1.0
Sentence1: ይሸር ቄ, Sentence2: ይፈዝ ቄ, Actual Similarity Score: 0.8
Sentence1: ቸግ ባነ, Sentence2: ፍጥምጥም ባነ, Actual Similarity Score: 1.0
Sentence1: ኰቤ ሐናም, Sentence2: ጫማ ሐናም, Actual Similarity Score: 1.0
Sentence1: እማዘር አገዜም, Sentence2: እማዘር ተዳፐረሂም, Actual Similarity Score: 1.0
Sentence1: ቦራ አደገም, Sentence2: ቦራ ቃሸም, Actual Similarity Score: 1.0
Sentence1: ተሸፓፐረም አገሦም, Sentence2: ገረም አገሦም  , Actual Similarity Score: 1.0
Sentence1: ጠንየ ጠመጠንም, Sentence2: ጋጀ ጠመጠንም, Actual Similarity Score: 0.8
Sentence1: ተደመዶም ይጮዶ, Sentence2: ተራከቦም ይጮዶ, Actual Similarity Score: 1.0
Sentence1: ኩራዝቄ ደፍባረም, Sentence2: ኩራዝቄ ጠፉም, Actual Similarity Score: 0.8
Sentence1: ተገተረም ባነ, Sentence2: ይነ ባነ, Actual Similarity Score: 1.0
Sentence1: ገፓት ስን አቴረም, Sentence2: ገፓት ስን አከስም , Actual Similarity Score: 1.0
Sentence1: አማዘር ፍረሀኒ ባረም ጨቆሰም, Sentence2: አማዘር አፎበራ ባረም ጨቆሰም , Actual Similarity Score: 0.9
Sentence1: የሜና ዳማዳ ነረኖ, Sentence2: የሜና ጌዝ ነረኖ, Actual Similarity Score: 0.8
Sentence1: ሸረትቄ አትየሌም , Sentence2: ሸረትቄ አዝመኔም, Actual Similarity Score: 1.0
Sentence1: ስርም ኤረሕብ, Sentence2: አቸም ኤረሕብ, Actual Similarity Score: 1.0
Sentence1: አመጨ ተሳ, Sentence2: ዘርማ ተሳ, Actual Similarity Score: 1.0
```

3. Sample Train Data

Appendix G

```
========================================
Sentence1: ዋጋ ዘሙለ
Sentence2: ዋጋ አውን
Actual Similarity Score: 1.0
Predicted Similarity Score: 0.9729533195495605
========================================
Sentence1: መጨቁንም ወርም
Sentence2: ነጠቁንም ወርም
Actual Similarity Score: 1.0
Predicted Similarity Score: 0.801904559135437
========================================
Sentence1: ጨቁረረ ተረሳም በናም
Sentence2: ዘፍተሙ ተረሳም በናም
Actual Similarity Score: 0.8
Predicted Similarity Score: 0.9731764793395996
========================================
Sentence1: ይሸን ግብር አነከወን
Sentence2: ይሸን ግብር በኮረንም
Actual Similarity Score: 1.0
Predicted Similarity Score: 0.8442500829696655
========================================
Sentence1: ይሸን ዘንጋ ጨሙተንም
Sentence2: ይሸን ዘንጋ ነከወንም
Actual Similarity Score: 0.8
Predicted Similarity Score: 0.797271192073822
========================================
```

4. Sample output of model predict

Appendix H

```
Epoch 292/300
4/4 [==============================] - 0s 70ms/step - loss: 0.1559 - mse: 0.1559 - val_loss: 0.1649 - val_mse: 0.1649
Epoch 293/300
4/4 [==============================] - 0s 61ms/step - loss: 0.1537 - mse: 0.1537 - val_loss: 0.1659 - val_mse: 0.1659
Epoch 294/300
4/4 [==============================] - 0s 62ms/step - loss: 0.1544 - mse: 0.1544 - val_loss: 0.1649 - val_mse: 0.1649
Epoch 295/300
4/4 [==============================] - 0s 64ms/step - loss: 0.1544 - mse: 0.1544 - val_loss: 0.1672 - val_mse: 0.1672
Epoch 296/300
4/4 [==============================] - 0s 73ms/step - loss: 0.1541 - mse: 0.1541 - val_loss: 0.1649 - val_mse: 0.1649
Epoch 297/300
4/4 [==============================] - 0s 61ms/step - loss: 0.1546 - mse: 0.1546 - val_loss: 0.1669 - val_mse: 0.1669
Epoch 298/300
4/4 [==============================] - 0s 61ms/step - loss: 0.1558 - mse: 0.1558 - val_loss: 0.1650 - val_mse: 0.1650
Epoch 299/300
4/4 [==============================] - 0s 63ms/step - loss: 0.1564 - mse: 0.1564 - val_loss: 0.1684 - val_mse: 0.1684
Epoch 300/300
4/4 [==============================] - 0s 64ms/step - loss: 0.1542 - mse: 0.1542 - val_loss: 0.1653 - val_mse: 0.1653
3/3 [==============================] - 0s 15ms/step - loss: 0.1435 - mse: 0.1435
Stacked RNN - Test Loss: 0.14350512623786926,mse:0.14350512623786926, Test Accuracy: 0.8564948737621307
LSTM - Test Loss: 0.13888239860534668, Test Accuracy: 0.8611176013946533
GRU - Test Loss: 0.1389165222644806, Test Accuracy: 0.8610834777355194
Bidirectional RNN - Test Loss: 0.1422329843044281, Test Accuracy: 0.8577670156955719
Stacked RNN - Test Loss: 0.14350512623786926, Test Accuracy: 0.8564948737621307
```

5. Sample output of model Loss and Accuracy

Appendix I

Sentence1	Sentence2	Similarity Score		
		exp1	exp2	exp3
የሚን አሳብ የሐረ ሐማ አንቂሪ	ተደቡብ አረብ ኢቶፕያንየ ቸነም	0.1	0.1	0.2
የሰር ቤት መኽረም	የሰር ቤት ተከሰም	1	1	1
የባሕለታ ቃር ንድብር	ጉራጌ ይረብሮ ባነ	0.1	0.1	0.1
የባሕል ይወረፆ	የጉራጌ ሐልቅ ይብሮ	0.1	0	0.2
የቤት አርሸ	በጉራጌ ገነ ባነም	0.1	0.1	0.2
የታሪክም ጣፊ	አታት የአምፍ ተታሪክ ተምራማሪ	0.1	0.1	0
የትዝፐሮ ሰብ	ተጉራም ሐረ ተሓረር ቸነም	0.1	0.1	0
የጉራጌ ሐልቅ እንም በሜና ንቡ	ዝሕ ታረብ ገነ ቸነቦም	0.1	0.1	0
የሕትሜ ተቀኔብትሐ አድን	ብያ ይፍቴ ባጢር እንቾትሐ	0.3	0.3	0
ያማሪና ቋንቋ ሐማም	በመጣፍ ጣፍሁም	0.2	0.1	0.1
ያማሪና ቋንቋ አምፍ	በሳዊ የሻኽሬ ይችል	0.2	0.2	0.1
ይሕርሸ ይብሮ	ሕኖ ይፍት ወረር ተደቡብ ቸነም	0.1	0.1	0
ይረብርወ መደር ገፈረም ኔጌብ ወረም	በቃዲስም በሹርም ግብት ቾናም	0.3	0.3	0.2
ይብቴ የዋረን ዘጋ ሐዳም	ይብቴ በዋረን ዘጋ ቶዘከበም	1	1	1
ይቾተታ በኘዙ	ይውርዮ ሰብም ባነዎ	0.1	0.1	0.1
ይንም ጉራጌ ባሕል	ኢቶፕያ ሐልቅ ሐማ ይረብሮ ባነ	0.1	0.1	0.1
ይጠረዮ ሰብ ሜትየው ቢሪም	አብርሐም በመምሬ የረፑረን	0.2	0.1	0.2
ይጠሬዮ ሰብ ሜትየው ቢሪም	እርስመሕታ አርች ነችም	0.3	0.3	0
ይፍት ወረርሎ	የጉራጌ አትነት ትያትየሶ	0.1	0.1	0.2
ዱየ ቄ ያትሸንፍ	ጠነማ ቄ ያትሸንፍ	0.8	0.8	0.9
ጉራም ነፈሰም ከነ	ሓበሻት ቢርም ይጠረዮ ባነ	0.1	0.2	0.1
ጉራጌ የርቅ ዘበር ታሪክ ያነን ሐልቁ	ዝሕ በዝም ታነ	0.1	0.1	0.2
ግሬድ ቄ ላሌ ይትፍቾሮ ባነ	ግሬድ ቄ ፍቾረ ይትፍቾሮ ባነ	1	1	1
ግብርውቄ ባት ሰስብኖ	ግብርውቄ ባት ድምድኖ	1	1	1
ግሙኟሐ የደንድር	ግሙኟሐ የፍቀር	0.8	0.9	0.7
ጨነችምታነ ኘመታ ሞአብ ቢረቾንም	ሕትም አኳ የሞአብ ሰብ ቢሪም	0.2	0.2	0.3
ጨነችምታነ ኘመታ ቤንአሚ ቢረቾንም	ሕትም አኳ የአሞን ሰብ ቢሪም	0.1	0.1	0.1
ሹትም ጐፔየናው ባረቼም	እያ ዝሕ ዬፐሕ ቦሔነትም ባነ	0.3	0.3	0
ሔት አርዋ ሐረማም	ንቀሹታ አርች ባነ	0.1	0.1	0.1

Expert one (Exp1)
Name Al Asgawi
Signature
Date 06/05/16

Expert Two (Exp2)
Name
Signature
Date 6/5/16

Expert Three (Exp3)
Name ABiyHom
Signature
Date 6/5/16

SNNP/PRS Gurage Zone Office