DIRE DAWA UNIVERSITY

INSTITUTE OF TECHNOLOGY

DESIGN AND DEVELOP SPELLING CHECKER AND CORRECTOR FOR AFAAN OROMOO USING DEEP LEARNING APPROACH

BY

HAYU BEKELE

JUNE 27, 2022

DIRE DAWA, ETHIOPIA

DIRE DAWA UNIVERSITY
INSTITUTE OF TECHNOLOGY

Director of Post Graduate Studies
School of Computing
Master of Science in Computer Science

Design and Develop Spelling Checker and Corrector for Afaan Oromoo Using Deep Learning Approach

By

HAYU BEKELE

Advisor:

TESSFU GETEYE (PhD)

June 27, 2022

Dire Dawa, Ethiopia

Design and Develop Spelling Checker and Corrector for Afaan Oromoo Using Deep Learning Approach

By

HAYU BEKELE

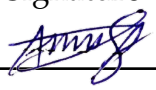
A thesis submitted to Dire Dawa University Institute of Technology in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science in the School of Computing

Dire Dawa University
Institute of Technology
Dire Dawa, Ethiopia
June, 2022

Dire Dawa University
Institute of Technology
School of Computing

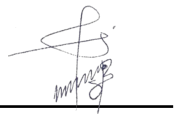
This is to certify that the thesis prepared by HAYU BEKELE, entitled "Design and Develop Spelling Checker and Corrector for Afaan Oromoo Using Deep Learning Approach" and submitted in partial fulfillment of requirements for the degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Approved by the Examining Board

Tessfu Geteye (PhD)		27/06/2022
Thesis advisor	Signature	Date
Alemebante Mulu (PhD)		27/06/2022
External Examiner	Signature	Date
Internal Examiner	Signature	Date
Chairperson	Signature	Date
School Postgraduate coordinator	Signature	Date
School Dean	Signature	Date
Postgraduate Director	Signature	Date

Declaration

I hereby declare that the work which is being presented in this thesis entitled "Design and Develop Spelling Checker and Corrector for Afaan Oromoo Using Deep Learning Approach" is original work of my own, has not been presented for a degree of any other university and all the resources used for the thesis have been duly acknowledged. I understand that non-adherence to the principles of academic honesty and integrity, misrepresentation/fabrication of any idea, data, fact and source will constitute sufficient ground for disciplinary action by the university and can also evoke penal action from the sources which have been properly cited or acknowledged.



Hayu Bekele

(Candidate)

Signature

Quotation

I can do all things through Christ which strengtheneth me (Philippians 4:13)

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntellegence
API	**A**pplication **P**rogrammingf **I**nterface
BBC	**B**ritish **B**roadcasting **C**orporate
BiRNN	**Bi** **R**ecurrent **N**eural **N**etwork
CEC	**C**onstant **E**rror **C**arousel
FBC	**F**ana **B**rodicasting **C**orporate
GRU	**G**ated **R**ecurrent **U**nit
LSTM	**L**ong **S**hort-**T**erm **M**emory
NLP	**N**atural **L**anguage **P**rocessing
OBN	**O**romia **B**roadcasting **C**orporate
OBN	**O**romia **M**edia **N**etwork
OLF	**O**romo **L**iberation **F**ront
RNN	**R**ecurrent **N**eural **N**etwork
SCRNN	**S**emir **C**haracter **R**ecurrent **N**eural **N**etwork
SLR	**S**ystematic **L**iterature **R**eview
VoA	**V**oice **o**f **A**merica

Abstract

In this digital era, texts are provided to many text editing and processing tools for copious tasks. Search engines, emailing and word processing are top word entry and editing environments in different languages where the faultlessness is a keynote for the intent of that text being processed. For the support of this goal, many languages used a spelling checker and corrector in their text processing tools. A spelling checker is an application, program or simply a software that arbitrate the correctness of the spelling for a given word based on the languages' spelling rule. With the fact of its meritorious, different languages are being used spelling checkers in their document processing tasks. Afaan Oromoo, spoken by the largest ethnolinguistic group, constituting more than one-third of Ethiopian population, is one of federal official language now a days. To its digital advancement, Afaan Oromo needs such a computer based support of text editing and processing tool. Hence, in this thesis we have designed and developed spelling checker and corrector for Afaan Oromoo using deep learning algorithms.

We gathered and prepared dataset of 37311 Afaan Oromo sentences which consists 609272 words. These data are collected from media news of Afaan Oromo program Facebook pages like Oromia Broadcasting Corporate (OBN), Oromia Media Network (OMN), Fana Broadcasting Corporate (FBC) and Voice of America (VoA) scraped using python script. In addition to these sources, we also collected Afaan Oromo corpus from several political, academic, scientific, and fiction books and as well as new testament Afaan Oromo holy bible book. These data finally, preprocessed and spited in to train and test with 80:20 ratio respectively.

In this thesis work, we used deep learning algorithms Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) models of Recurrent Neural Network (RNN) with their bidirectionalities. We developed four model using LSTM, GRU, BiLSTM, BiGRU models for Afaan Oromo spelling checker and corrector. Precision, recall, F-scoreand, Support evaluation metrics of scikit learn are used to compare their performance with each other on the same test data. Finally, we preferred BiGRU model with 92.325%, 83.5% and 91.008% of precision, recall and F1 score respectively.

Key Words: Afaan Oromo Spelling Checker and Corrector, Deep Learning Spelling Checker, LSTMSpell, GRUSpell, Artificial Injection.

Chapter 1

Introduction

1.1 Background

Natural Language Processing(NLP) is one area of computer science and Artificial Intelligence(AI) which deals with the interaction between computers and humans' natural language. Since the ultimate goal of NLP is to enable computers to read, decipher, understand, and make sense of the human natural language as well as human beings do, nowadays, natural language processing use cases like virtual assistants, speech recognition, sentiment analysis, automatic text generation, machine translation, sentiment analysis, social media analysis and so many others made the area the most attractive researchable area. Moreover, spelling checker is a back bone for these applications of natural language processing. The applicability of natural language processing as support of linguistic like grammar and spelling checker and corrector also abundantly seen as a main contribution to the researchers in the area. As a usage of natural language processing, many researches have been done on spelling checker for many languages all over the world and for some of languages in Ethiopia.

Spell checker is a software program or simply program that is used to check whether the word is spelled correctly or not in a particular text editing tool and in case notify the user for the misspelled word with regard to the languages' rule. Depending on the feature of the spell checker tool, spelling checkers can either automatically correct the word (auto correct) or allow the user to select suggested word/words to be correct from alternative words. Hence, spell checker can provide two basic functionalities based on the particular language it built for: word spell error detection- verify the correctness of the word spelled and word spell error correction- suggest an alternative word/word instead based on previously developed algorithm.

Several spelling checker and corrector approaches were done for different languages where English, Arabic and Chinese are the top leading languages of plenty spell checker users. Some researches are also done for certain languages in Ethiopia like Amharic, Afan oromo, Tiggrigna and Wolayta. However, approaches used in developing this checker for those languages are traditional based like dictionary look up, morphological analysis, rule based, statistical and many several. In this research, we want to apply one feature of natural language processing along with deep learning method for spelling and Oromo writing system.

Afan Oromo also written as "Afaan Oromoo" or "Oromoo" is one language of Cushitic family of Afroasiatic phylum spoken by Oromo people in Ethiopia and some neighbouring countries in the horn of Africa like Kenya, Sudan, Somalia and some other countries beyond horn of Africa like South Africa, Libya and Egypt. Afan Oromo is the most widely spoken language in Ethiopia with 35% (about 37 million) of the total population which is followed by 29.33% Amharic speakers in the country. It is also the 4th widely spoken language in Africa[1]. Afan Oromo uses "Qubee Afan Oromoo" as its writing system since 1840 to translate section of bible in to Afan Oromo[2]. Despite its densely spoken, Afan Oromo lacks many technology aided services like machine translation, speech recognition, grammar checker, spelling checker and so many others. Hence, in this work we want to contribute the solution to the spelling checker challenge for the language. As discussed in [3], in Afaan Oromoo word spelling errors can occur in many different ways like omission, addition, analogy, substitution, transposition, spacing, inconsistency in spelling and some others may occur with no category. Spelling errors that occur in omission can be either due to vowel omission or consonant omission where vowel omission is the most common one. In case of addition spelling error type, an extra consonant or vowel can be added to the word where addition of vowel is most frequent. These word errors also mainly categorized in two main groups real word error and non word error.

In this thesis work, we used a deep learning approach to implement spelling checker and corrector for Afaan Oromo language as to make the language an advantageous to the natural language processing and machine learning.

1.2 Statement of the Problem

These days we need to type a text on many text editors for different purposes. Search engines, emailing and word processing are top word entry and editing environments in different languages and for different purposes. Texts typed on the editor in the word level are crucial for the goal of that process, the correct word/sequence of words typed is the most message or goal accomplished. For the search engine for example, we found the most likely word/sentence the word we entered. Most developed languages like English have a spelling checker and corrector for text editing tools as plugins. However, this type of machine support for language is not used for our country languages even if they need all activities of word processing.

In our country Ethiopian, for example Amharic is a federal official language that scripted digitally and used for many purposes of official work, but such spelling checker support for the language is not built for the language either as stand alone software or as a plugin. Even though there are more than 88 languages spoken in Ethiopia, Afan oromo is the leading spoken language with 35% speakers from the total population of the country. It is the official working language, an instructing medium of primary and secondary schools and official languages for other religious institutions for the regional state of Oromia. In addition to these, Afan Oromo is used for many media languages in Ethiopia nowadays, like FBC, OBN, OMN, BBC Afaan Oromoo, VOA Afan Oromo program, and EBC Afan Oromo program. All these institutions, and nany other organizations use Afan Oromo word processing for the particular work goal. However, the support of machines for word spelling accuracy and speed is not built for this language. Even if many native speakers of the language used the writing system of Afan oromo many of the users cannot spell Afan Oromo words correctly as prescribed rule of the language. There fore, the general spelling rule of the language uniquely required for the language uniformity and understandably. This can be done by this era feature of computer vision to support natural language.

It is a pleasure if Afan Oromo has built in spelling checker support for text editing tools either as plugins or stand alone setups like other languages. This can help the users in

many angles as the language is a medium for many official uses at regional level and currently claiming to be a federal official language. For example, the one who cannot type Afan Oromo words correctly can use the checker in case. Not only to spell correctly, it can also help them to type fast. Finally, message, quality and content of the processed documents be accurate with the support of spell checker. This means lack of such technical support may lead to time spending and inaccurate document processing [4].

1.3 Research Questions

At the end of the day, this research is expected to addressed the following research questions:

- How to make easy of text processing environment for Afaan Oromoo?
- Which deep learning approach is best to develop a model for Afaan Oromoo spelling checker and corrector?
- How to improve the performance of the model?

1.4 Motivation

Nowadays, almost all information processing requires a text editor and to process and get desirably processed information (document), meaningfully spelled word is mandatory. Hence, the first insight made us to work on this problem domain is to enable the users of Afan Oromo language and its writing system capable of editing their official work with the support of machine aids even if some or majority of the users cannot spell correctly. To complement this initiation, the astonishing features and correlation of machine learning and natural language processing investigated during class sessions encouraged us more to decide and take an action over the idea.

1.5 Objectives

1.5.1 General Objective

The general objective of this thesis is to design and implement a spelling checker and corrector for Afan Oromo writing system using a deep learning approach.

1.5.2 Specific Objectives

To develop a prototype based on the best performed model, we have planned to:

- Review different researches held on deep learning, natural language processing, spelling checker and correction, background of Afan Orom and Afan Oromo writing system and any other related to the topic.
- Review spell checker and corrector techniques and approaches previously implemented for the language and other related languages.
- Collect Afan Oromo text corpora from different sources and prepare it for the model.
- Design and implement spelling checker and corrector model for Afaan Oromoo.
- Train and evaluate the performance of the developed model using desired evaluation metrics.
- State experimental result and conclusions; finally, suggest some recommendations for future work

1.6 Scope and Limitation of the Study

Even if the problem of ambiguity happens in different levels in Afan Oromo, like morphology, phonology, syntax, semantic, phrase level and grammar level this study focuses only on type of error occurring at word level. At the end of the research the model may not be used as a plugin for high commercial text editing tools like Microsoft offices because of long chain bureaucracies. The model, on evaluation, can only detect whether the word provided

to the text editor is correctly spelled Afaan Oromo word or not; only none word error type of mistake is corrected. Word sense ambiguity like contextual meaning is beyond our research. Also fully-featured, completely tested application may be beyond the scope of this paper for a time being because of limited time, dataset and computational resources. We hope to enlarge and continue work on this in the future, if all goes well.

1.7 Methodology

As methodology is the systematic and theoretical analysis of the methods applied to an area of study that offers the theoretical underpinning for understanding that which method, set of methods or best practices can be applied to a specific case, here in this section methods regarding review of related works, data collection and preparation, design and implementation method, tools to be used and evaluation metrics are covered.

1.7.1 Literature Review Method

For the purpose of this study, we conducted systematic literature review (SLR) even though there are other several methods. The reason why we preferred this literature review method is that a SLR proposes a fair assessment of the research topic as it uses a rigorous and reliable review methodology, together with auditing tasks to reduce the researcher bias [5]. It is also stated as "a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest" so that we can apply all these techniques on works around Afaan Oromoo, spelling correction approaches and natural language processing applications to the specific area we preferred.

1.7.2 Data Gathering Method

The data we used for the training, testing and validating the model are from different sources like facebook pages of Afaan Oromoo news medias, Oromia regional state constitution, Bible (in Afan Oromo) and general books (fictions, educational and scientific, political and social). News data are collected from their Facebook pages using a python code we developed for this particular task. From all these sources, we have collected 30,000 Afaan

Oromoo sentences. Then, we continued to the preparation of these data to be used for training and testing the model.

The corpus then separated to training, validating and testing partitions with 80:10:10 respectively. Once data are prepared, to develop a spelling checker and corrector, both misspelled and correctly spelled words are needed [6]. To get misspelled words, we used an artificial injection to the correctly spelled words that have been prepared.

1.7.3 Sytem Design and Development Method

Once statement of a problem is identified, system design is a phase that bridges the gap between problem domain and the proposed solution basically focuses on the solution domain, to answer the question 'how to implement?' We used the architectural design type of system design, even though there are several types of system-design like logical, physical, conceptual, detailed and others. Since this type of system design focuses on the design of system architecture and describes the structure and behavior of the system, it is helpful to use in high level programming languages like python[7]. We explicitly did and explained this in chapter four(4).

For this model development, we used machine learning algorithms alongside with NLP features. For the model development, we used both sequence to sequence LSTM and GRU cells with their bidirectionalities [6][8][9]. This is as a result of that both these architectures are special kind of RNN, capable of learning long-term dependencies since they are designed to avoid the long-term dependency problem[10]. The analysis and interpretation of the generated result was followed with the comparison of each algorithm. The best performing model is selected and finally completed by comparison of the result obtained with deep learning.

1.7.4 Tools

We used Google Collaboratory for implementation environment which is a free Jupyter notebook environment that runs totally in the cloud. It provides more than 12 GB RAM and 38 GB hard drive virtually for 12 per day and we used it's GPU accelerator. For the implementation programming language, we used Python 3.8 version.We used a powerful python libraries we required from our corpus preparation up to model development and

testing its performance. some of top libraries we used are: re, os, datetime, random, unidecode, numpy, keras and sklearn. We described these and others too in chapter four in detail.

1.7.5 Testing and Evaluation Metrics

During and after best model choosing (for the final model testing), we evaluated and tested the model on 20% our dataset 10% for evaluating and 10% for testing. Throughout all types of evaluations and testing for all candidate models, an scikit learn evaluation metrics, Precision_Recall_Fscore_Support are used with ..... averaging label. These evaluation metrics are the most applicable metrics especially for multi class classifications and/or for unsupervised datasets where the model can learn from the dataset pattern itself[11]. Several scholars in their research work and article reviews, recommended evaluation metrics for spelling checker to get more accurate evaluation by justifying linguistic of several languages[12][13].

1.7.6 Organization of the Paper

The organization of this thesis paper is presented in six chapters. In the fist chapter chapter one, background of the study, problem specification, iconic research questions, motivations to the research goal and methodologies to be used are declared. In the second chapter chapter two, which is the literature review, the two soul of the chapter is raised; spelling checker and corrector techniques and related works. The third chapter addressed an overview of Afaan Oromoo writing system commonly known as "Qubee Afaan Oromoo". The most important part of the language we focused on was the morphology of the language and that is why we deal on it. In the fourth chapter, the system design and methodology is explicitly discussed. The fifth chapter covers the experimental result and discussion of the thesis outcome. In the last chapter, chapter six, the over all conclusion and some future work and recommendation is stated.

Chapter 2

Literature Review

2.1 Spelling Checker and Corrector

Spelling checker and corrector development can be seen mainly in two basic approaches: traditional way and the modern artificial intelligent based and deep learning. The traditional approaches include dictionary lookup, statistical based, N-gram, rule based, etc. whereas the modern approaches are the recent methods through deep learning like Recurrent Neural Network (RNN) especially with its extensions Long Short-Term Memory (LSTM) and GRU with their bidirectionalities and attention base that most of the time uses sequence to Sequence (seq2seq) for time series. This approach can also use combination of two or more of these methods. We will try to see each of these approaches in spelling checker and corrector in detail in this section. However, regardless of its different approach, spelling checker and corrector have a general architecture shown in Figure 2.1 below.

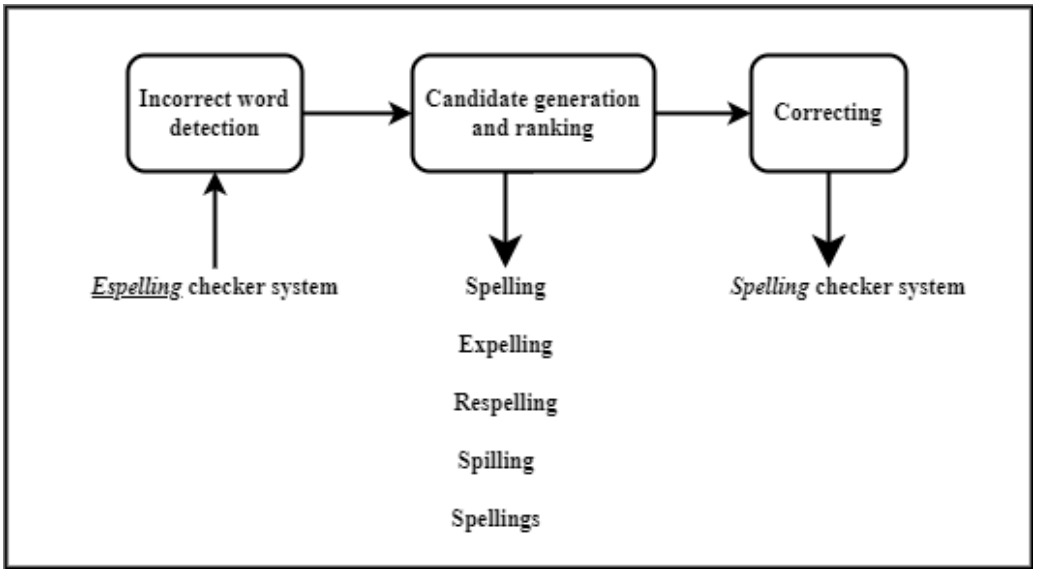


FIGURE 2.1: General Structure of Spelling Checker and Corrector System

In the first phase, the target misspelled word is detected based on the technique built. It is where the the spelling checker flags the word as incorrect word using the algorithm built for it. From the example given in Figure 2.1 the word ***'Spelling'*** is typed wrongly as ***'Espelling'*** so that the director should flag it as a misspelled word. Once the the incorrect word is detected, most probably correct word or words generated in the second phase in their priority order based the candidate generator algorithm built for it. As we can see from the example in the Figure 2.1, all ***'Spelling'***, ***'Expelling'***, ***'Respelling'***, ***'Spilling'*** and ***'Spellings'*** are candidates generated for the targeted word ***'Espelling'*** ordered by their most probably correct word. Finally, in the last phase misspelled word is corrected to the top ranked word either by user option or automatically if the spelling checker and corrector is set to auto correct.

An accuracy of a word in any language is checked by spell checker with the help of lexicon/dictionary of the language [15]. The manner of spell checker is straight forward at all that if a text or word is provided as an input, spelling detector identifies the off-base word in the document and then the spelling corrector tries to supplant the off-base with the most likely word crosschecking a dictionary/corpus of accurate words. In most systems, before making any corrections, a recognition process is performed on the input sentence to extract words that may be incorrect. However, approaches of doing this may differ according to different scholars.

Despite of the general concept of spell checker architecture as seen in the Figure 2.1, it can be applied and appeared differently according to method of development. In the next section we will try to take a look at different spelling checker approaches done for different languages by different researchers.

2.1.1 Spelling Checker and Corrector Approaches

The two most popular techniques of detecting misspelled word held for several languages including Afaan Oromoo are dictionary lookup and N-Gram analysis. These approaches are mainly considered as traditional approaches.

2.1.1.1 Dictionary Lookup and Morphological Analysis

Dictionary lookup employs efficient dictionary lookup algorithms and/or pattern matching algorithms. A dictionary (Wordnet) is a lexical source that contains list of correct words a particular language. Most of the time it works alongside with morphological analysis. The non-word errors can be easily detected by checking each word against a dictionary. The drawbacks of this method are difficulties in keeping such a dictionary up to date, and sufficiently extensive to cover all the words in a text. These resources are used for preparing, processing and managing linguistic information and knowledge needed for the computational processing of natural language [16]. An example of such large-scale lexical resources is given by linguistic ontology that covers many words of a language and has a hierarchical structure based on the relationship between concepts. It covers nouns, verbs, adjectives and adverbs. Gaddisa Olani and Dida Midekso[17] stated that morphology-based spell checker has advantages such as its ability to reduce the dictionary size drastically and the ability to recognize new words that are not included in the dictionary. Morphological rules also address word categories and their possible inflections, derivation and compounding. The following diagram shows a morphological analysis along with dictionary lookup architecture proposed for Afaan Oromoo spelling checker and corrector.

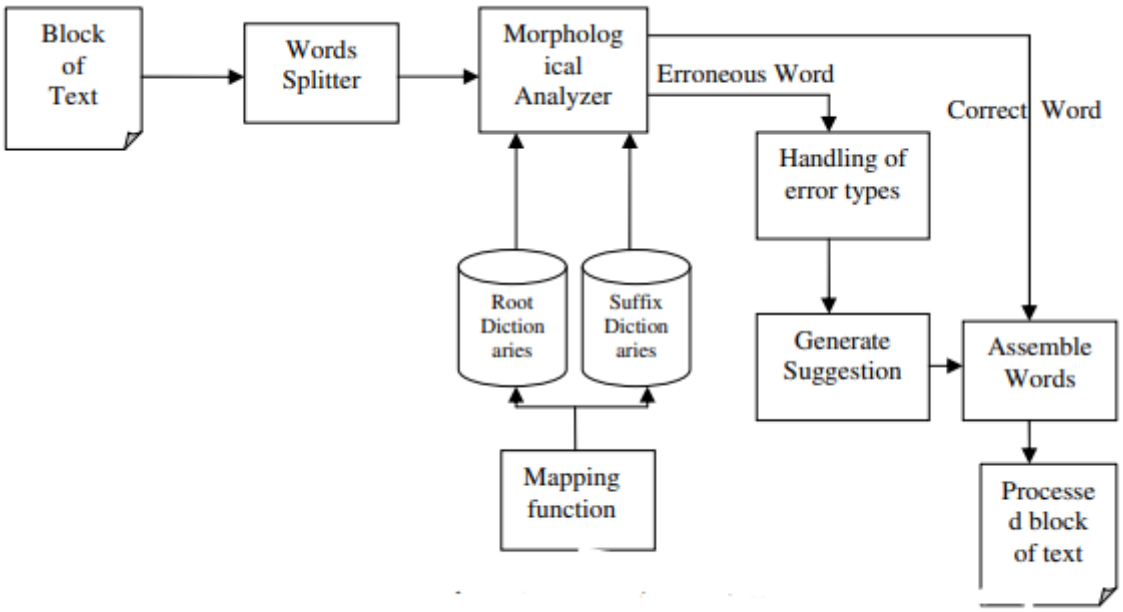


FIGURE 2.2: Dictionary Lookup and Morphological Analysis Design [17]

2.1.1.2 N-Gram Analysis

An n-gram is an n-letter subsequences of a sequence, either of n- letters or words, where n is an element of {1,2,3...}. If n = 1; 2; 3, reference is made to a unigram, bigram, or trigram, respectively[14]. In other words, n-grams are substrings of length n. R.K Sharma[4] suggested in his studies that N-gram analysis is a method to find incorrectly spelled words in a mass of text. Instead of comparing each entire word in a text to a dictionary, just n-grams are checked. A check is done by using an n-dimensional matrix where real n-gram frequencies are stored. If a non-existent or rare n-gram is found the word is flagged as a misspelling, otherwise not. An n-gram is a set of consecutive characters taken from a string with a length of whatever n is set to. This method is language independent as it requires no knowledge of the language for which it is used. In this algorithm, each string that is involved in the comparison process is split up into sets of adjacent n-grams. The similarity between two strings is achieved by discovering the number of unique n-grams that they share and then

calculating a similarity coefficient, i.e., the number of the n-grams in common (intersection), divided by the total number of n-grams in the two words (union). The n-gram probability can be calculated by dividing the number of times a particular string is observed by the frequency of the context as follow.

$$P\left(c_i|c_{i-n+1}^{i-1}\right) = \frac{count\left(c_{i-n+1}^{i}\right)}{count\left(c_{i-n+1}^{i-1}\right)} \tag{2.1}$$

In an n-gram approach of spelling error detection whether each n-gram result in an input string representing some text is likely to be valid in the language. There are two approaches for n-gram spelling error checker: binary value approach and probabilities approach. In a binary value approach, a two-dimensional bigram array which contains combination of binary value from letters or words is compiled in a table of one and zero (possible and impossible n-gram respectively) determined by n-gram and these are presented upon chosen lexicon. Probabilities approach is where probabilistic values are divided from frequency count of n-gram resulted in large corpora. In this case n-gram of input string is compared to that of the n-gram statistics table.

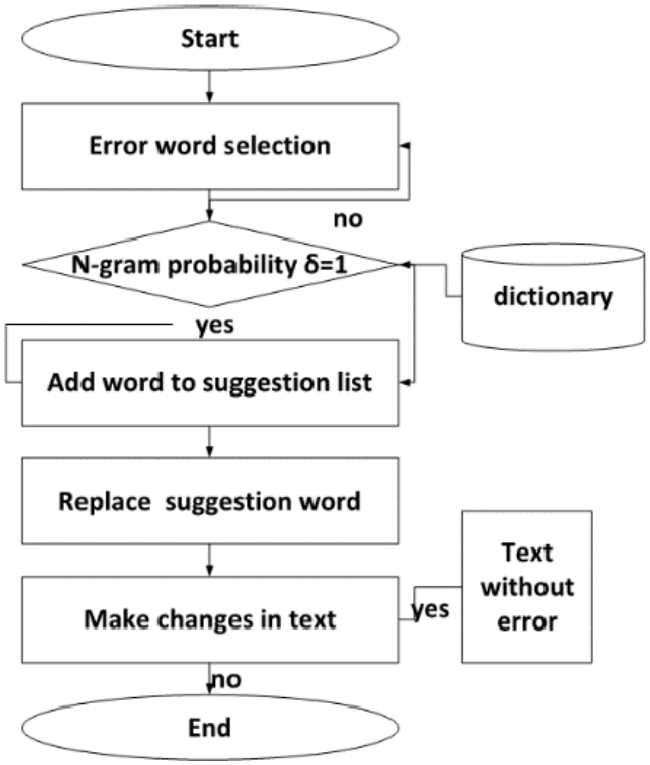


FIGURE 2.3: N-Gram Analysis Technique for Spelling Checker [18]

After the process of spelling checker detected a spelling error, then the next step is to correct it according to the mechanism undertaken. It can be using either of the following methods.

2.1.1.3 Minimum Edit Distance

The minimum edit distance (also simply known as edit distance) method is by far the most studied and widely used spell checking method. The minimum edit distance, as the name suggests, is a method that requires the least number of editing operations (insertion, deletion, replacement, and transposition) required to convert one string to another. The idea of using processing distance to check spelling was first proposed by Wagner [19]. Levenshtein[20] proposed an edit distance algorithm for performing correction operations called

insertion, deletion, and replacement.

$$[H]lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{a_i \neq b_j} \end{cases} & \text{otherwise.} \end{cases} \tag{2.2}$$

Damerau Editing Distance [18] allows the above three types of editing operations and transposition between two adjacent characters. These four editors account for more than 80% of all human spelling errors. The minimum edit distance between two spellings, w1 and w2, refers to the minimum number of editing operations required to convert w1 to w2. The editing operations are mentioned here: insert, delete, replace and transpose. Usually, the minimum edit distance between the misspelled word in the text and the word in the dictionary is guaranteed. Wagner introduced the use of dynamic programming techniques (such as the minimum edit method) to correct spelling to reduce execution time. According to [19] and [21], dynamic-programming techniques make use of overlapping subproblems, optimal structure, and memorization which was first invented by Bellman[22] in order to reduce the runtime of algorithms as discussed below.

2.1.1.4 Similarity Key Techniques

The essence of similarity key method is to convert each word into a key. The mapping Assigned so that words with the same spelling have the same keywords. When calculating keywords for misspelled words, it will provide pointers to all spelled words in the dictionary, and these dictionary entries will be returned as possible corrections. That is, all words in the dictionary that are similar to the key value of the current misspelled word are returned as possible correct words. Since there is no need to compare misspelled words with each dictionary entry, similarity key is fast. [23] discussed that those key techniques are based on transforming words into similarity keys that reflect the relations between the characters of the words, such as positional similarity, material similarity, and ordinal similarity

and these similarities are either of Positional similarity, Material similarity or Ordinal similarity. Various algorithms propose different approaches to design key groups for characters of a language.

In general, the similarity is measured based on the position and the order of the characters in the words. Anagrams and the phonology of language are suitable factors to construct corresponding keys for each character. Figure 2.4 shows spelling checker based on similarity key technique.

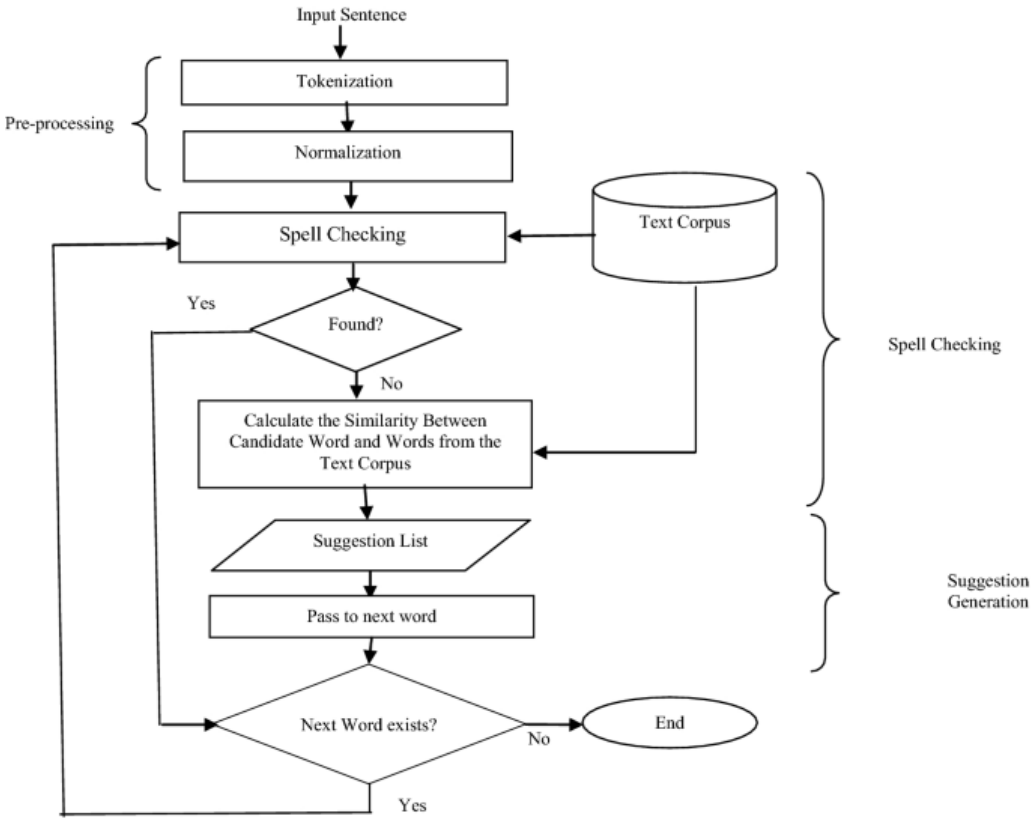


FIGURE 2.4: Similarity Key Method Structure [23]

2.1.1.5 Rule-based Techniques

Rule-based techniques include algorithms that try to represent knowledge of common spelling patterns in order to convert misspelled words into correct words. Knowledge is presented

in the form of rules and these rules can include general morphological information (for example, the rules for converting verbs into adjectives by adding "ing" at the end of verbs), the length of misspelled words, etc., misspelled words, and store all valid words in the resulting dictionary. Based on a predetermined estimate of the probability of errors corrected by a specific rule. It is independent of any grammar or parsing formulation and due to this it can be a mere lexical lookup routine. Most recently, an evidence of rule-based techniques being incorporated in (context independent) spelling correction system can be found in the phonetic module of AURA, an algorithm designed to represent phonetic spelling error patterns [24].

The rule-based spell-checking method is used to maintain and control the correct spelling of Afaan Oromoo's words according to certain written rules. It directly focuses on the rules provided for error correction, and uses less space and time complexity compared to morphology-based and dictionary lookup spell checker. Figure 2.5 shows a design of rule based Afaan Oromoo spelling checker system proposed in [25].

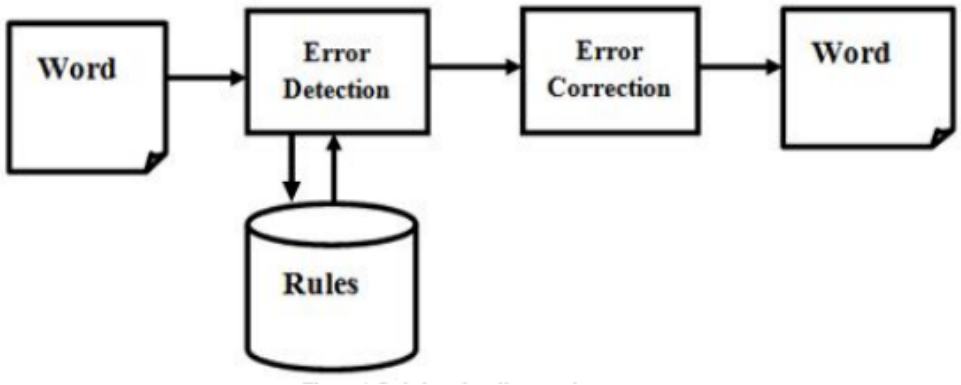


FIGURE 2.5: Rule-based Spelling Corrector Method Structure. [25]

Other scholars, Asanilta Fahda and Ayu Purwarianti [26] proposed a prototype of an Indonesian spelling and grammar checker that uses a combination of statistical and rules based methods. The rule comparison module uses 38 rules to identify, correct, and interpret common punctuation marks, word choices, and spelling errors. The spell checker uses a dictionary process to check each word to find spelling errors and Damerau-Levenshtein distance neighbors as correction candidates. It also adds some morphological analysis of

word forms. Hidden binary/coincidental Markov models are used to rank and select candidates. The grammar checker uses a ternary language model of tags, POS tags, or sentence fragments to identify invalid sentences.

2.1.1.6 Deep Learning Approach

The first deep learning approach for text error correction was proposed by Ghosh and Kristensson (a correction model for English language text) [30]. Later on, Keisuke Sakaguchi further investigated how the semi character recurrent neural network (SCRNN) can be used in word recognition and spelling correction. In their experiment, they demonstrated that the SCRNN outperforms than many other available spelling correctors. In the following section we will try to take a look at some deep learning approaches done for different spelling languages. Gated Recurrent Unit(GRU) and Long Short Term Memory (LSTM) are specialized versions of RNN that are created to address the problem of Vanishing gradient. Let us discuss these two flavors of RNN in detail since we used both of them throughout this work.

Long Short-Term Memory (LSTM): Long Short-Term Memory (LSTM) is a newly emerging approach that transformed both machine learning and neurocomputing fields. Recently, it is also a hot preferable architecture for natural language processing tasks like text summarization, word prediction, language translation, grammar checkers, spelling checkers and even in speech recognition. The first spelling checker using LSTM was first introduced by Hochreiter S. Schmidhuber[27]. This is due to the fact that LSTM uses a constant error carousel (CEC) to prevent the problem, as it maintains the error signal within each unit's cell. It consists of three different gates: forget gate, input gate and output gate.

- *forget gate*
- *input gate*
- *Output gate*

The following figure (Figure 2.6) illustrates the general architecture of an LSTM.

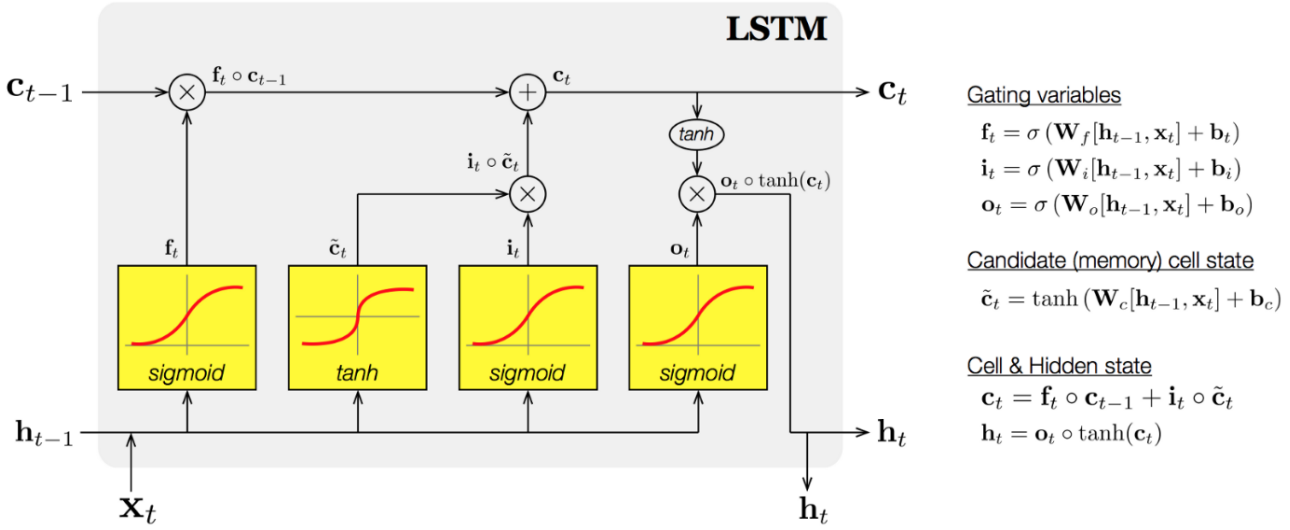


FIGURE 2.6: LSTM Architecture

An LSTM architecture have been used for several Spelling checkers of different languages. Among these, Hindi[31], Indonesian[32], Malayalam[33], Chinese[34], Turkish[35], English[36], Italian[37], and others also used it in combination with other deep learning. The following figure shows the general structure of an LSTM in a spelling checker and corrector.

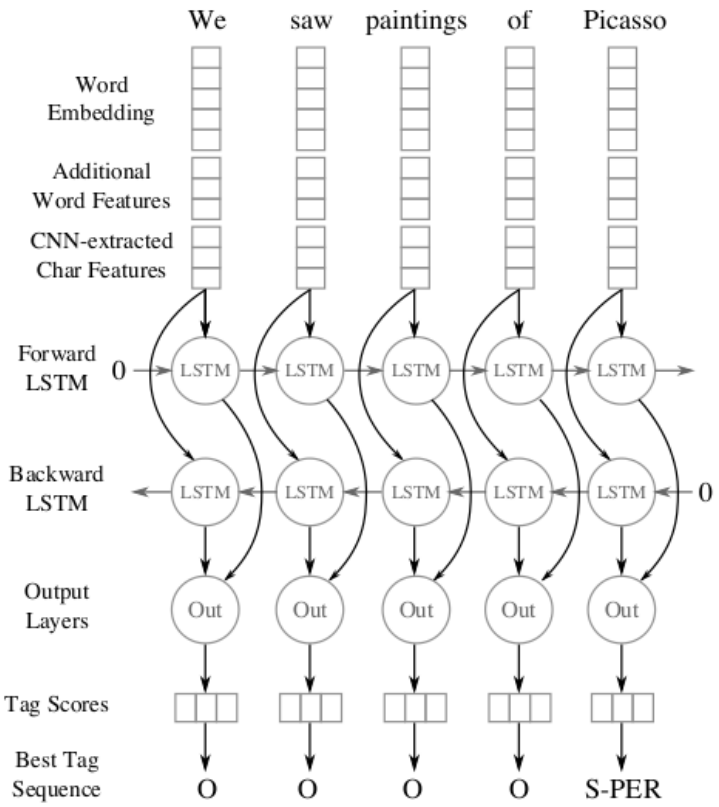


FIGURE 2.7: LSTM in Spelling Checker and Corrector [38]

In addition to long-term dependence, another problem with language models with neural networks is variable-length output spaces, such as words and sentences. Neural sequence-to-sequence models or encoder-decoder networks have shown that this problem can be solved by variable input and output lengths. It is a generative neural network model. Given an input chain, it can generate an output chain of random length. The sequence-to-sequence model consists of two processes: encoding and decoding. During the encoding process, the encoder (essentially an RNN model) is provided with the input sequence x = (x1, x2, ..., xT). A simple RNN, considering the output of each state unit, only retains the hidden unit of the last ht layer. This vector is usually called a sentence embedding or c-context vector, and is intended to contain the representation of the input sentence.

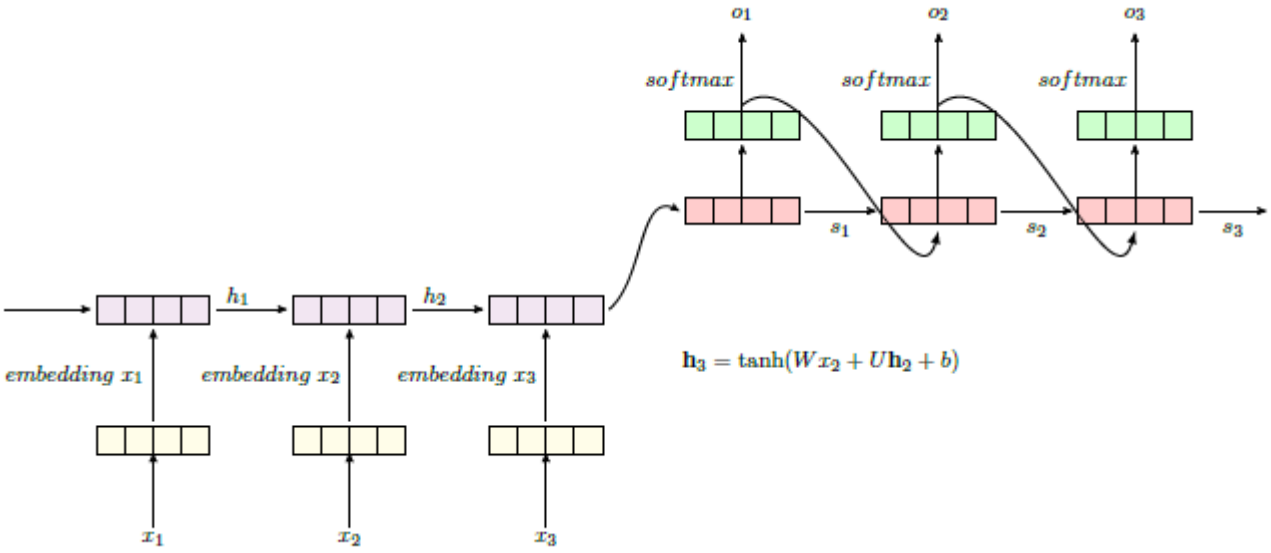


FIGURE 2.8: Seq2Seq Architecture in Spelling Checker and corrector [39]

Gated Recurrent Unit (GRU): The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over long short term memory (LSTM). GRU uses less memory and is faster than LSTM, however, LSTM is more accurate when using datasets with longer sequences. Figure 2.7 below shoows an architecture of GRU.

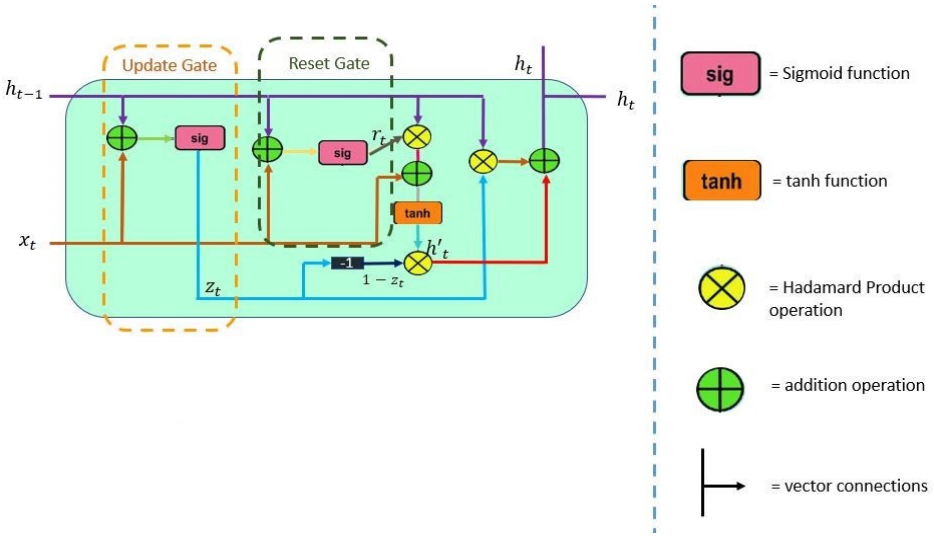


FIGURE 2.9: GRU Architecture [29]

Unlike LSTM, GRU does not have an output gate and combines the input and the forget

gate into a single **update gate** alongside with **reset gate**.

Shashank Singh and Shailendra Singh [31], used the attention-based encoder–decoder bidirectional recurrent neural network (BiRNN) along with long short-term memory cells for Hind languages. Spelling checker and corrector have been done using different types of deep learning approaches for several languages.

2.2 Related Works

Spelling checker and corrector have been conducted and has being conducted for most languages across the globe using several approaches to different level. Some languages are at high pick of having spelling checker and corrector for all digital document data processing. English, Chinese and Arabic are the top leading languages of the world having spelling checker and corrector. These days it is also a hot practice of several researchers to sophisticate it through deep learning approaches. In our country, Ethiopia spelling checker and corrector has been proposed for some languages including Afaan Oromoo using different traditional approaches. However, it is not attempted using deep learning for any of Ethiopian languages yet. The following are some of spelling checkers and correctors proposed for some of Ethiopian languages: Afaan Oromoo, Amharic, Tigrigna and Somali.

2.2.1 Spelling Checker and Corrector for Amharic

Getnet Assefa, [34] condacted automatic Amharic spelling error detection and correction using hybrid (Metaphone algorithm and edit distance algorithm) approach. He used 125,000 words used for dictionary preparation and 500 of it for test data. The edit distance algorithm was used for determining the likelihood of a given word to the correct words in the dictionary and for ranking possible correct words and the metaphone algorithm is applied for spelling error detection. for Microsoft Visual Studio 2010 is used for implementation and for the prototype development, python and VB.NET programming languages are used in the research. The over process that the prototype pass is: it selects possibly correct word/s with the help of Levantine edit distance Algorithm, ranking these selected words based on their distance and finally, after all possible words are listed the system enables the user to select

it or ignore accordingly. The system is integrated to a Microsoft office 2010. As discussed in the papers' overall work, the performance testing of the integrated system succeeded more than 95%.

Melaku Tilahun[40] conducted automatic spelling checker for Amharic language that works on inflection of Amharic words. The researcher designed the whole system in to five components; input component, normalization component, error detection component, morphological analyzer component, and spelling error correction and suggestion component. After conducting five major experiments, the researcher got 97.27% overall performance.

Andargachew Mekonnen, Andreas Nürnberger and Binyam[41] proposed an automatic spelling corrector for Amharic using a corpus-driven approach with the noisy channel. They used a modified version of the System for Ethiopic Representation in ASCII for transliteration in the like manner as most Amharic keyboard input methods do to over come the problem of Amharic letters syllabus. They used manually annotated spelling error test corpora for evaluation. The spelling error detection performance was evaluated by precision, recall, and F1 measure and they got 89.4%, 80.6% and 84.8% respectively. Their model evaluation was on Amharic and English test data and they stated that their model has scored better performance result than the baseline systems like GNU Aspell and Hunspell.

Mariawit Shimelis [37] developed morphology based Amharic spelling error detection and correction system. The research was mainly focused on non-word errors. The researcher employed design science methodology. She designed and evaluated a prototype that employs an algorithm to detect and correct Amharic spelling errors. The researcher used python programming language for the development of the system. The test data was manually gathered from an online available Amharic documents and texts and Amharic bible as discussed by the researcher. These data contain combination of words spelled correctly and incorrectly. Finally, the system performed an average precision and recall of 96% and 99% respectively after tested on different types of styles including: disaggregated by derivation style and confusion matrix.

2.2.2 Spelling Checker and Corrector for Tigrigna

Berihun Hadis[42] conducted spell checker for Tigrigna language using rule based morphological analyzer and unsupervised approach. The research is focused on the real word error to detect and automatically correct if any misspelled word is inputted. The researcher employed two approaches: unsupervised Morphological based approach using Morfessor tool and rule-based dictionary look up and morphological analysis technique and he got that rule-based approach out performs well than the unsupervised approach. The main resource of data that the researcher used is extracted from web document, books, and newspapers having more than half million unique words from world wide web. For the prototype implementation purpose, the researcher used different tools for different purposes including Hun spell for spell checker and morphological analyzer library and Python as programming language. the system is integrated in to Openoffice2.2 and it tries to handle inflectional problems, word compounding error problems. pluralization, hybridization, acronyms and abbreviation of commonly used Tigrigna language. 787 randomly taken data, the proposed system reached 89%, 87% and 88% accuracy of recall, precision and F-measure respectively which is averagely 80% overall performance.

2.2.3 Spelling Checker and Corrector for Somali

Ali Olow, Wan Mohd Nazmee and Lul Farah[43] developed spell Checker for Somali Language Using Knuth-Morris-Pratt String match Algorithm. The system development approach is composed of three components that the researchers used namely; preprocessing, processing and post processing. 30,000 Somali words from online dictionary are used and this is claimed that these data from online dictionary are up to date.

2.2.4 Spelling Checker and Corrector for Afaan Oromoo

Wubetu Barud Demilie[9] has did spelling checker for five selected Ethiopian languages (Amharic, Afaan Oromoo, Tigrigna, Hadiyyisa and Awngi) using dictionary-based approach. The model that the researcher developed provides correction and suggestion by selecting the most suitable from a list of corrective suggestions based on lexical resources and dictionary-based statistics and it depends on the lexicon of these selected languages. He has prepared

a dictionary of 993072, 866328, 966328, 987176 and 678534 words for Amharic, Afaan Oromoo, Tigrigna, Hadiyyisa and Awngi languages respectively and used manually prepared spelling error test corpora for evaluation of the performance. The models' performance was evaluated by Precision, recall and F Measure gaining between 82.8% - 86.6% precision, 81.6% - 84.7% recall and 82.2% - 85.65% F measure for all languages.

Workineh Tesema and Duresa Tamirat[44] have developed a computer software and file editing tool as a plugin to Microsoft office using unsupervised machine learning which was trained on unlabeled corpus from government media, cultural, historical, sport news, political, and economical documents of Afaan Oromoo. Researchers in this article tried to use unsupervised machine learning method alongside with n-gram method. They used Java, NetBeans IDE 8.02 as implementation tool.

Another researcher Yehuwalashet[45] developed the prototype for word sense disambiguation that offer the related meaning of the ambiguous word based on the underlying contexts. The researcher used a combination of unsupervised with rule-based approach and the fact behind this approach was to overcome the problem of a bottleneck for the machine learning approaches, while hybrid method can improve the accuracy and suitable when there is scarcity of training data. Even though word sense ambiguous is not directly a problem of the word misspelling, the impact of the problem is the same as that of spelling error for the message of a sentence. Java NetBeans 8.0.2 (for preprocessing activities like tokenization, stop word removal and normalization purpose to make sense example sentences ready for experimentation) and weka 3.7.9 (to cluster, based on the hierarchical clustering, EM and K-means algorithms built into the package) ware preferred for the implementation. He used precision and recall as evaluation metrics and got 90.35 % accuracy performance.

Authors in[17] used morphological analysis and dictionary lookup based to develop Afaan Oromoo spelling checker. They stated that using this approach A morphology-based spell checker has advantages such as its ability to reduce the dictionary size drastically and the ability to recognize new words that are not included in the dictionary. In addition to this, morphological rules address word categories and their possible inflections, derivation and compounding and even the approach can be drawn upon in building grammar checkers,

as the researchers discussed. The architecture they designed for their approach (morphological analysis and dictionary look up) has eight components to pass as a phase. These components are: tokenizer, knowledge base, error detection, morphological analyzer, error correction, morphological generator, suggestion ranker and word assembler. They used Microsoft Visual C# 2010 for their prototype implementation and they have integrated their system into Apache OpenOffice. As evaluation metrics they used lexical recall, error recall, and precision for the developed system after reviewing and comparing several spelling checker system evaluation methods. Finally, they got 88.62% lexical recall, 100 % error recall and 28.62% precision.

Chapter 3

Overview of Afaan Oromoo Writing System (Qubee Afaan Oromoo)

3.1 Introduction

Qubee became the official writing system of Oromo in 1991. In 1993 Tilahun Gamta[46] participated in a meeting to adopt the Latin alphabet and proposed alternative scripts for the Oromo writing system. He said that Afaan Oromoo speakers met formally at the center of Oromia for the first time to discuss and decide on their language. He claimed that this historic meeting was convened by Oromoo Liberation Front (OLF) on November 3, 1991 and it was composed of several Oromoo scholars. As Tilahun Gamta (1993: 17) explained: "The purpose of the meeting was to adopt the Latin alphabet used by OLF". After several hours of discussion and reflection, in with more than one thousand men and women attended on the meeting unanimously, he pointed out and decided that the Latin alphabet should be the alphabet that Oromo should use. An official Oromo language, a writing borrowed from the Latin alphabet system, which uses writing principles through the correspondence between sound fragments and characters or symbols in the alphabet.

Twenty-five letters of the Latin alphabet and the apostrophe (') were adopted without changing their shape or size (such as diacritics) to represent Afaan Oromoo sounds. The letter "v" is the only character in the Latin alphabet that is not used in the Qubee system. Most Oromo phonemes, except for the exclusives, such as "C", "Q" and "X", have corresponding symbols in the Latin alphabet and are not difficult to match. However, matching the ejectives and expressing other aspects (such as gemination and vowel length ware proved to be a challenge, especially in terms of consistency and simplicity of the writing system. How to represent

gemination and vowel lengths in the qubee system, and what possible alternatives can be used to show Afaan Oromoo ejectives, geminates, and vowel lengths [47].

Afaan Oromoo (Oromoo) writing system commonly known as "Qubee Afaan Oromoo" which uses Latin based alphabet was come to official writing system for Oromo people since 1991. Even though some materials were written during early 1970's and 1980's using different representation (writing system) like Roman and Sabean, "Qubee Afaan Oromoo" was decided to be designed due to different reasons. In some scholars it is stated that Qubee Afaan Oromoo uses thirty-three (33) letters, five (5) vowels and twenty-eight (28) consonants [47] [1]. Table 3.1, 3.2 and 3.3 shows the three classifications of letters in Afaan Oromoo as consonants, vowels and diagraphs (qubee dachaa) respectively. However, these diagraphs reduced to five excluding TS and ZY later on.

TABLE 3.1: Afaan Oromoo consonant letters

B b	C c	D d	f F	G g	H h	J j
K k	L l	M m	N n	P p	Q q	R r
S s	T t	V v	W w	X x	Y y	Z z

TABLE 3.2: Afaan Oromoo Vowel letters

A a	E e	I i	O o	U u

Among these twenty-eight consonants seven (7) of them are diagraphs (paired) letters to be considered as a single consonant letter which is known as "Qubee Dachaa". These paired letters are 'CH', 'DH', 'NY' 'PH', 'SH', 'TS' and 'ZY' which can be also written in lower case in the same way.

TABLE 3.3: Afaan Oromoo paired (diagraph) letters

CH ch	DH dh	NY ny	PH ph	SH sh	TZ ts	YY zy

However, basic Qubee Afaan Oromoo does not contains some of Latin letters ('P', 'V' 'Z', 'TS' and 'ZY') since there is no native word in Afaan Oromoo to be formed from these letters,

but used to indicate other language's word other than Afaan Oromoo words like "poolisii" meant to be 'Polis'. As a result of this, in Afaan Oromoo these letters are said to be borrowed letters (Qubee Ergisaa). As Bijiga Teferi stated, in regional state of Oromia, there are five types of dialects in Afan Oromo spoken in different parts Oromia; Harar (eastern part), Macha (western part), Raya (northern part), Borana (southern part) and Tulama (central part). Regardless of having multiple dialects, Oromo peoples across the region can communicate with each other. Similarly, the way (spelling) they use for a particular word in all dialects is similar except they may use different words which have the same meaning. For example 'Eessa' is used in western part of Oromia, 'Eeysa' in eastern part and 'Eecha' is used in central part having the same meaning "where".

3.2 Morphology of Afaan Oromoo

Morphology is the study of words, how they are formed, and their relationship to other words in a particular language. It analyzes the structure of words and parts of words such as stems, root words, prefixes, and suffixes. It also looks at parts of speech, intonation and stress, and the ways context can change a word's pronunciation and meaning. The rules understood by a speaker reflect specific patterns or regularities in the way words are formed from smaller units in the language they are using, and how those smaller units interact in speech. In this way, morphology is the branch of linguistics that studies patterns of word formation within and across languages and attempts to formulate rules that model the knowledge of the speakers of those languages. Phonological and orthographic modifications between a base word and its origin may be partial to literacy skills. investigations have indicated that the presence of modification in phonology and orthography makes morphologically complex words harder to understand and that the absence of modification between a base word and its origin makes morphologically complex words easier to understand. Morphologically complex words are easier to comprehend when they include a base word. In general definition, the discipline that deals specifically with the sound changes occurring within morphemes is morphophonology [48].

There are two kinds of morphology: inflectional which is concerned with the inflectional changes in words where word stems are combined with grammatical markers for things

like person, gender, number, tense, case and mode (do not result in changes of parts of speech) and derivational which deals with those changes that result in changing classes of words (changes in the part of speech). For example, noun or an adjective may be derived from a verb.

Basically, morphemes can be categorized as free and bound morphemes (Schiffman, 1999) and in Afaan Oromo root words are bound as they cannot occur on their own without affix and also the same is true for the affix. For instance, the root word "barn-" (education) cannot appear by it self without the affix "-oota" and the same is true for the affix. Even though there are a wide range of word formation processes in Afaan Oromo, the most common according to several scholars, is the morphological analysis of the language which is categorized in to six group. These categories are: nouns, pronouns and determinants, case and relational concepts, functional words, verb and adverbs. Since Afaan Oromo is morphologically very rich, derivation, reduplication and compounding are also common in the language [49].

1. **Noun:**

The noun morpheme of Afaan Oromoo can be further categorized under four sub categories.
Gender: two affixes *(-ssa and -ttii)* are used to identify a gender in Afaan Oromoo. For example, *duuressa* (for male) and *duurettii* (for female) to mean "rich".
Number: to form plural of a noun, there are a number of suffixes especially concerning several Afaan Oromoo dialects. The most common suffixes in Afaan Oromoo to for a plural form of a noun are: *-(o)ota, -lee,-wwan,-een,-olii,-olee and -a(n)* (Mewis, 2001).
For example,

Sangoota (-oota) – 'Oxen'
Lammiilee (-lee) - 'Citizens'
Bakkeewwan (-wwan) – 'Places'
Haadholii (-olii) – 'Mothers'
Naannoolee (-olee) – 'Regions'

Gaarreen (-een) – 'Mountains'

Derived noun forms: Afaan Oromoo is morphologically rich and the most factor of this are the use of different derivational suffixes and formation of compounds. This type of noun morpheme also have two sub categories: derivational suffix and compound words.

2. **Pronouns and Determiners**

Adjectives, numerals and other quantifiers and pronouns in Afaan Oromoo are categorized in this type of morpheme. Adjectives in Afaan Oromoo by themselves are subcategorized as gender, number and definiteness indicators. Pronouns also subcategorized as personal, demonstrative, possessive, reflexive, reciprocal and interrogative pronouns.

3. **Case and Relational Concepts**

In Afaan Oromoo nouns, pronouns and adjectives are subject to base form (example: from *mana* (house) *manicha* (the house), subject form (example: *harka* (hand) *harki* (hand), possessive (which can be genitive construction or names of person), dative (example: *barataa* (student) *barataaf* or *barataadhaaf* (for the student), instrumental (example: *ija*(eye) *ijaan* (by eye), ablative (example: *Itiyoophiyaa* (Ethiopa) *Itiyoophiyaadhaa* (from Ethiopia) and locative (example: *Finfinneetti* (in Finfine).

Chapter 4

System Design and Methodology

4.1 Introduction

In this chapter we are going to discuss the over all architecture of the system starting from data set description to the model development. Figure 4.1 below illustrates the proposed model development design. Let us take a look at each components and discuss what they represents, how they interconnected and how they work with support of each other based on the implementation.

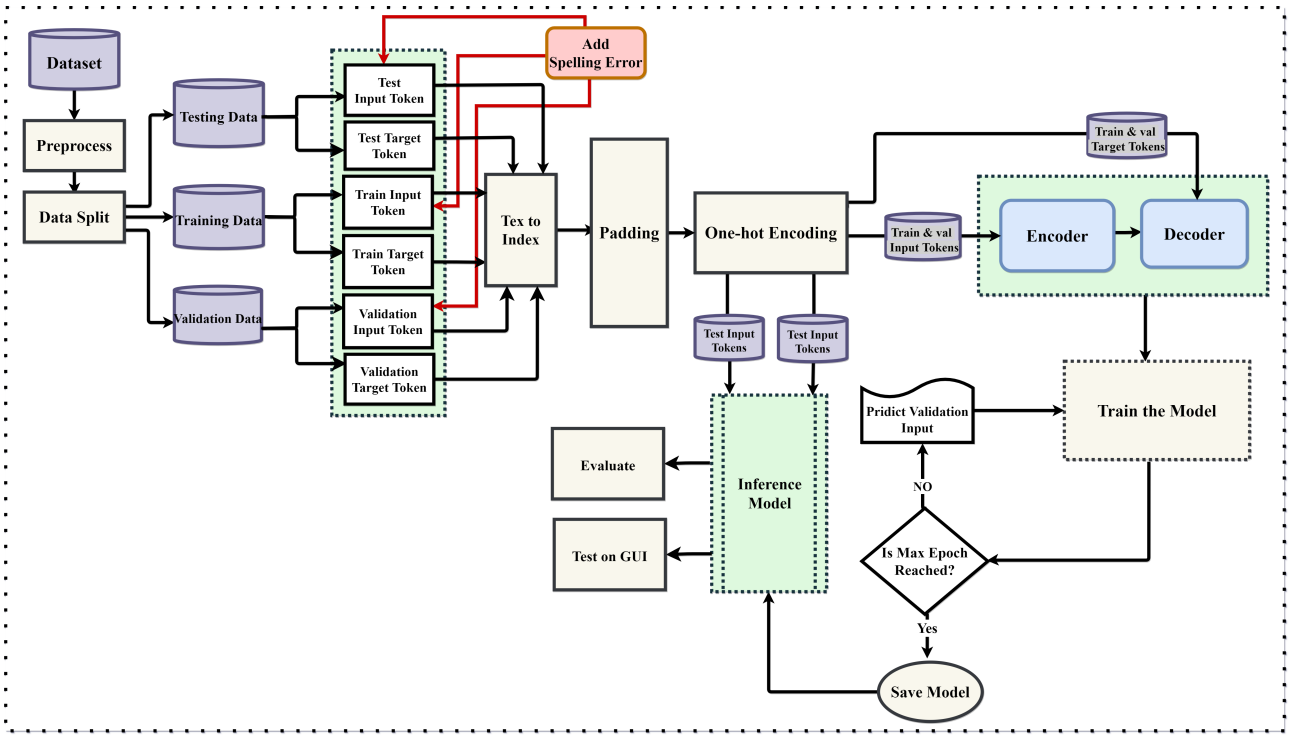


FIGURE 4.1: General Architecture of the System

4.2 Dataset Source

Dataset needed for Afaan Oromo spelling checker development is gathered from several sources. Several techniques and cares are taken under consideration for these data collection as of collection of sentences within that data source matters for the performance of the model being developed. Afaan Oromo written materials (as a data), especially with word consistent (well spelled) are rare to get easily. However, we have tried our best to get suitable data. We used different Afaan Oromoo books (consisting of fictions, academic, social, political and religious varieties), Afaan Oromo news data (Fana broadcasting corporate, Oromia broadcasting corporate, Oromia media network and Ethiopian broadcasting corporate Afaan Oromoo) that are available online, especially from their Facebook pages, new testament Holy bible in Afaan Oromo and Oromia regional state constitution. We got most of the books from telegram channel. For the data from media news (Afaan Oromo program), we have developed a python script to scrape data from their Facebook pages. 37311 Afaan Oromo sentences consisting 609311words are collected and prepared to be loaded for the preprocessing phase.

Collected Dataset	Vocabulary	Train and Validation	Train	Validation	Test
37311 **sentences (*609311* words)**	**66385**	**5310 (80%Vocab)**	**42486 (80%Train and Validation)**	**10622 (20%Train and Validation)**	**13277 (20% Vocab)**

- After dataset is cleaned and words are collected, the vocabulary of these words have identified
- We have **66385** vocabularies(in word) in our dataset (which are the total dataset we *used*)
- 80:20 ratio(for training and testing respectively) and 20% of training data for validation

4.3 Data Loading and Preprocessing

Once data is ready to the required amount, the next step is loading it to memory for desired preprocessing which are further goes for splitting phase. Thus, as indicated on the diagram data load is a stage where collected data is loaded to the working memory. At the preprocessing stage, these loaded data are cleaned by removing special characters which are not needed for the model development and pure sentences are tokenized into tokens of words. There are two basic functions in this activity, removing unwanted characters (re.sub()) and tokenizing each sentence into tokens of words (tokenize()). This is a step where all characters other than all upper and lower cases of A to Z has to be removed.

4.4 Data Splitting

After preprocess, these data are to be spited into training, validating and testing data at this phase. We used 80:20 ratio for training and testing respectively using train_test_split library from sklearn. The validation data then, split from the training data which is 80% of the whole dataset. We used the same 80:20 ratio to use 20% of the training data for validation. Once vocabulary of words within these dataset category is identified, tokens in each of them then sub-categorized as input and target words. Input token is a target token with some error addition.

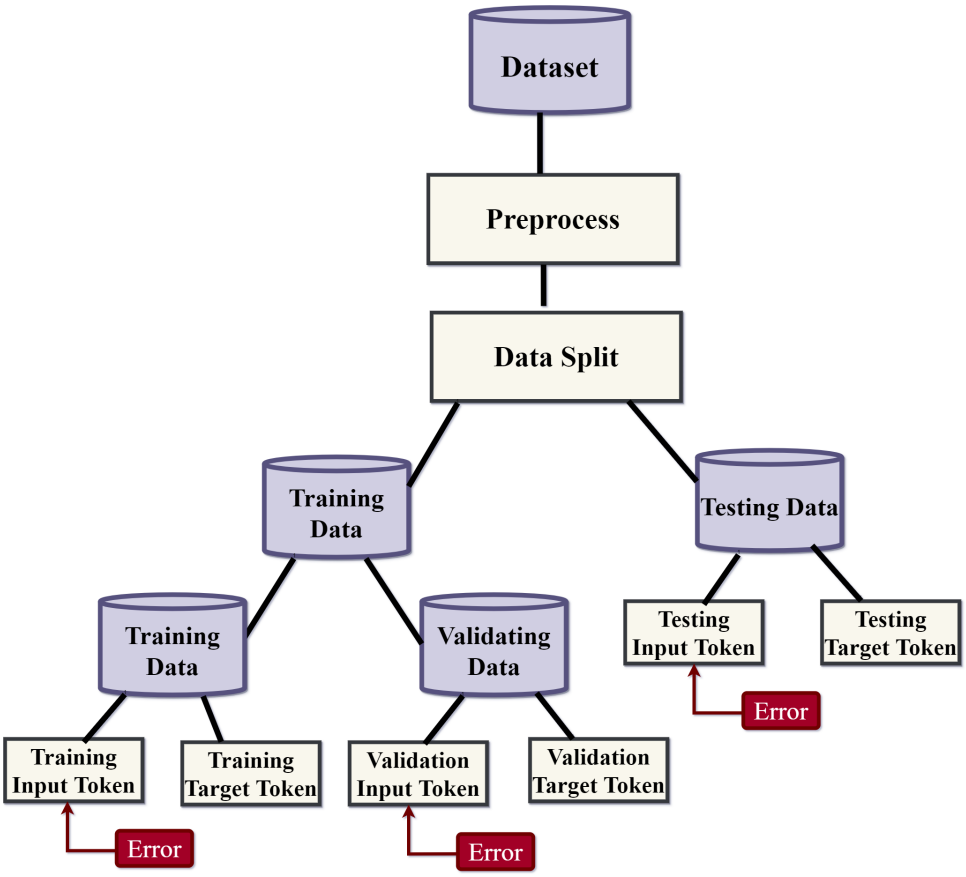


FIGURE 4.2: Data Splitting

4.5 Spelling error Addition(*Artificial Injection*)

As indicated in Figure 4.1 and 4.2, artificial spelling errors, literally made by either of character addition, deletion, transpose or replace added to the input. We defined a function that add spelling mistakes to a word (*"add_spelling_erors()"*). In this function there are four(4) main keys used to make word spelling error. The first is ***a random number*** between 0 and 1 generated using a random function from numpy (np.random.rand()). The second one is error rate which a number manually set between 0 and 1 which we used for probability calculation. Thirdly, probability which decides which type of spelling error will be applied (addition, deletion, replacement or transposition) for each word (token). It is calculated as follow:

$$Probability = \frac{Errorrate}{4.0}$$

Where:

Error rate: is a number between 0.0 and 1.0 and **4.0** represents a number of error types

The fourth key is number of characters to be added, deleted, replaced or transposed to/from a the input word. We used one, two and three characters variety to add spelling error. The reason why we used up to three number of characters to be applied is that at least to perceive words with more than three characters.

Once probability is calculated, the error type to be added can be selected using the following algorithm (pseudocode).

Given ***word*** and ***error rate:***
if length of ***word*** *< 3:*
    return ***word***
generate random number rand, 0.0 <= ***rand*** *<= 1.0*
compute probability, prop = error rate/4.0
if rand < prop:
    replace *a character with a random character*
*else if prob < rand < prob * 2:*
    delete *a character/s*
*else if prob < rand < prob * 3:*
    add *a random character*
*else if prob < rand < prob * 4:*
    transpose *characters*
else:
    pass (*no word error*)
return ***word***

This method is done for one character, two characters and three characters replace, deletion, addition and transposition. An error, to misspell the word, is added to the target word by applying one of the spelling error type based on the algorithm discussed above.

This function uses a set of A to Z and a to z (abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) characters to add a spelling error to the input data. figure 4.2 above shows the steps taken starting from data load phase up to error addition. The following example shows an input data with different spelling error applied on a single character and target data which will be the correction of these misspelled words (input data).

Example 1:

Input data: [*'Qajeechituu', 'sinra', 'bZarreeffama', 'Asaan', 'Oromoo'*]

Target data: [*'Qajeelchituu, 'sirna', 'barreeffama', 'Afaan', 'Oromoo'*]

As it is seen in the input data list, a character **'l'** is deleted from the target word **"Qajeelchituu"**, characters **'r'** and **'n'** are transposed from the target word **"sirna"**, a character **'Z'** is added to the target word **"barreeffama"**, character **'s'** is replaced by character **'f'** in the target word **"Afaan"** and the word **"Oromoo"** is left as it is as some times it is important to train the model with the word correctly spelled too. Once this process is done for all the three data splits, these input and target data pairs of training data, validation data and testing data are passed to word to integer conversion step (text2index).

Example 2:

Input data: [*'Qajechituu', 'isnra', 'bZarreeffamma', 'Asaam', 'Oromoo'*]

Target data: [*'Qajeelchituu, 'sirna', 'barreeffama', 'Afaan', 'Oromoo'*]

In this example, a character **'e'** and **'l'** are deleted from the target word **"Qajeelchituu"**, characters **('s', 'i')** and **('r', 'n')** are transposed from the target word **"sirna"**, a character **'Z'** and **'m'** is added to the target word **"barreeffama"**, character **'f'** and **n** are replaced by characters **'s'** and **'m'** of the target word **"Afaan"** and the word **Oromoo** is left as it is. This algorithm is done same way for three character operations.

4.6 One-hot Encoding

In a general scientific usage, word embedding used to capture the semantic, syntactic context or a word(term) to understand how similar or dissimilar it is to other terms(words) in an

article, blog, etc. It is a language modeling and feature extraction-based method that map a word to vectors of real numbers. Some of the major word embedding techniques are[50]:

- Binary Encoding
- TF Encoding.
- TF-IDF Encoding.
- Latent Semantic Analysis Encoding.
- Word2Vec Embedding.

In our case, we used one-hot encoding method to feed the data with its shape to the architectures cell. As the name indicates, one-hot encoding is an essential process of converting the categorical data variables to be provided to machine and deep learning algorithms which in turn improve predictions as well as classification accuracy of a model[51].

Given **n** numbers of words (tokens), in our context for example, from either of input or target data, it prepares **m** numbers of unique words (vocabulary).
For example, let us consider the following two sentences.

1. ***Gurbaan tapha kubbaa jaallata***
2. ***Inni tapha kubbaa daawwachuu jaallata***

There are a total of nine words (4 in the first sentence and 5 in the second) (**'Gurbaan', 'tapha', 'kubbaa', 'jaallata', 'Inni', 'tapha', 'kubbaa', 'daawwachuu', 'jaallata'**) So, when word2vec which is one-hot encoding (Skip-Gram) is applied to these sentences as input data, for example, a vocabulary (unique words) from the two sentences is prepared. There are six unique words (vocabularies) in our example (**'Gurbaan', 'tapha', 'kubbaa', 'jaallata', 'Inni', 'daawwachuu'**). So, after one-hot encoding representation, the given sentence as an input data looks like the following two-dimensional vector.

TABLE 4.1: Sentence Level One-hot Encoding Example

	Gurbaan	tapha	kubbaa	jaallata	Inni	daawwachuu
Gurbaan	**1**	0	0	0	0	0
tapha	0	0	**1**	0	0	0
kubbaa	0	0	1	0	0	0
jaallata	0	**1**	0	0	0	0
Inni	0	1	0	0	0	0
tapha	0	0	**1**	0	0	0
kubbaa	0	0	0	**1**	0	0
daawwachuu	0	0	0	0	**1**	0
jaallata	0	0	0	0	**1**	0

In our case, once vocabulary of words are specified the one-hot encoding takes a sequence of characters instead of words. However, the process of conversion is the same other than the maximum number of vocabulary is the higher number of characters included in the corpus after preprocess. This is obviously fixed for our data as we only need a collection of [a-z A-Z ‘]. One word from the data set for example, can be represented by at least two dimensions of a vector representation. Let us consider the following sort of sentence randomly taken from corpus :

"***Abbaa biyyaati*** "

This sentence consists of two words each words shall represented by a vector of positions as shown in the following table.

TABLE 4.2: Word Level One-hot Encoding Example

	A	a	b	i	y	t
A	**1**	0	0	0	0	0
b	0	0	**1**	0	0	0
b	0	0	**1**	0	0	0
a	0	**1**	0	0	0	0
a	0	**1**	0	0	0	0
b	0	0	**1**	0	0	0
i	0	0	0	**1**	0	0
y	0	0	0	0	**1**	0
y	0	0	0	0	**1**	0
a	0	**1**	0	0	0	0
a	0	**1**	0	0	0	0
t	0	0	0	0	0	**1**
i	0	0	0	**1**	0	0

As we can see from Table 4.2, the word 'Abbaa' for instance, can be represented as:

$$[[100000], [001000], [001000], [010000], [010000]]$$

After all of our input and target data are converted to a vector representation using one-hot encoding based on the maximum number of vocabularies, they provided to input the model. Train input and train target data are given to the encoder and decoder model to train the model, the validation input and validation target data are given to the inference model to validate how high performed is the model in a particular iteration (epoch) and the test input and test target are given to the saved mode to check how high the final model is performed predicting the misspelled words from the test data.

The one-hot vector we used as input is used to pick out the corresponding row in the matrix. This means that the hidden layers are only used as a lookup table. And the output of the

hidden layer is just the word vector for the input word.

4.7 Model Development

A sequence-to-sequence model is neural network that works in a way that it maps a fixed-length input with a fixed-length output however, the length of the input and output may differ. The model consists of three components as shown in the following diagram: encoder, internal states (intermediate) vector and decoder.

We used both LSTM and GRU model as encoder and decoder. The one-hot encoded input data is feed to the encoder model as shown in Figure 4.3 with regardless of either the encoder model is LSTM or GRU. Both encoder and the decoder represent both of our LSTM models and GRU models. We used a SoftMax dense as an activation function for the encoder model states. We first added an Input Layer, with the only parameter to consider is 'shape', which is the maximum length of the training input (words with spelling error). Here the parameters to take into account are 'hidden_size' and 'output_dimention'. This layer will convert any of the input tokens into a vector of the shape of the output dimension. The concept behind this is to extract the meaning of the word in a form of a spatial representation where each dimension will be a characteristic defining the word. For example, the world 'lafa' will be converted into a vector of shape 128. The higher the output dimension the more semantic meaning we can extract from each word, but also the higher the calculations required and the processing time. Figure 4.3 following illustrates the way encoder and decoder models of our architecture does.

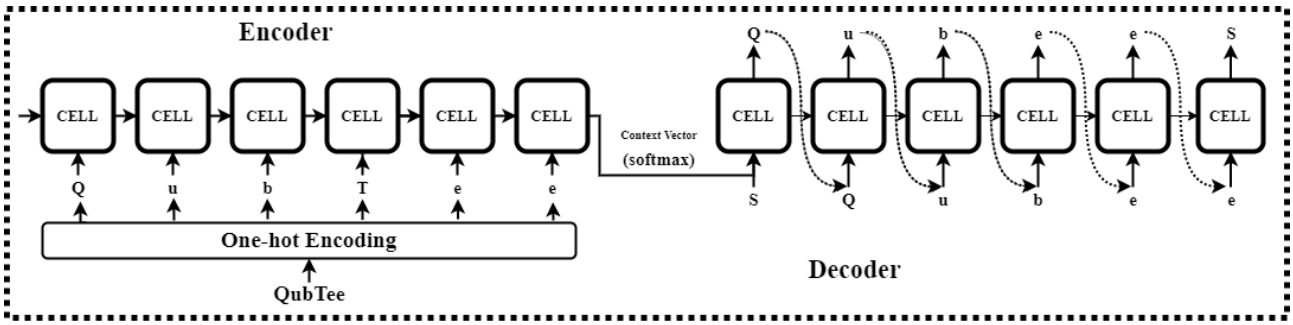


FIGURE 4.3: Seq2seq Encoder and Decoder Model

4.7.1 Encoder

The encoder model reads the input sequence from a vector of arrays provided using one-hot encoding method. These context vectors in case of GRU is hidden state and hidden state (h) and cell state (c) for an LSTM. We only preserve the internal states and discard the outputs of the encoder in case of an LSTM to encapsulate the information for all input elements so that it can help the decoder make accurate predictions.

4.7.2 Decoder

The decoder is also either of an LSTM model or GRU model whose initial states are initialized to the final states of the encoder which means, the context vector of the encoder's final cell is input to the first cell of the decoder network. Using these initial states, the decoder starts generating the output sequence, and these outputs are also taken into consideration for future outputs. We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which will help us determine the final output (as shown in Figure 4.5).

The most important point is that the initial states of the decoder are set from the final states of the encoder. This intuitively means that the decoder is trained to start generating the output sequence depending on the information encoded by the encoder. Finally, the loss and accuracy of training is calculated on the predicted outputs from each time step and the errors are backpropagated through time in order to update the parameters of the network.

4.7.3 Inference Model and Evaluation

After all hyper-parameters are set alongside with training input data and training target data, the model trained for desired number of iterations (epoch). The trained model then saved in .HDF5 file format after maximum number of epoch set is reached. Using the inference model on test data set, we evaluate the performance of the model. Testing input data, testing target data and the saved model is given to the inference model and testing evaluation is computed. We also prepared a graphical user interface so that we can manually provide a test data on and the model predict it.

Chapter 5

Experimental Result and Discussion

5.1 Introduction

In this chapter, we will discuss in detail about our experiment and analysis, the model development environment we used for the desired architectures, hyperparameters and their tunning, the evaluation metrics preferred to use and why they are suitable with our work, comparisons of models' performance within our work and with others developed previously, overall analysis and prototype.

5.2 Environment, Programming Language and Libraries

We used Google Collaboratory also typed as 'Colab' for short, which is a free Jupyter notebook environment that runs totally in the cloud. It provides more than 12 GB RAM and 38 GB hard drive virtually for 12 per day and we used it's GPU accelerator. The following shows the specification of the GPU environment we used.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   50C    P8    29W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

FIGURE 5.1: The GPU Spesification on Colab

We used python 3.8 throughout all our models development.

5.2.1 Libraries

Regular expression (re): A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. re can be used to check if a string contains the specified search pattern. We used this module of python during data preprocessing specifically during cleaning unwanted characters and tokenization by matching them as a regular expression.

Operating system (os): Tsis is a python module which provides a portable way to use operating system dependent functionality. We used this module to read and write a file as open() and read() to manipulate paths as os.path.

Random: This module of python is used to generate a pseudo number between 0 and 1 by default and a random number between given two intervals if provided by the user. The function rand.random() for example, returns an arbitrary number between 0 and 1 including 0 and 1 whereas rand.random(4, 8) returns a random number between 4 and 8 including 4 and 8. We used this function in artificial word error making. generated and the dataset split category respect to that number interval.

Unidecode: this module of python has helped us to represent our data that we have in Unicode in ASCII.

numpy: NumPy stands for Numerical Python and it is a library consisting of multidimensional array objects and a collection of routines for processing those arrays which is created in 2005 by Travis Oliphant [52]. We can use this module to do mathematical and logical operations on arrays. As we have discussed in chapter four (section 4.6), we used a word embedding mechanism word2vec, which is a two-dimensional array representation of our data per batch size. We used this python library to perform an operation on these data.

Tensorflow: TensorFlow is an open-source library for numerical computation and large-scale machine learning which is Created by the Google Brain team. It bundles both a slew of machine learning and deep learning. With the help of Python, it provides a convenient front-end API for applications which are executed in high-performance C++. Since the emerging of recent deep learning and machine learning TensorFlow made an easy task of training and running deep neural networks for several applications like image recognition, word embeddings machine translation and many others. It helps developers to create dataflow graphs which is a structure that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection nodes is a multidimensional data array, basically called tensor. The most advantage of TensorFlow is that of its abstraction.

Keras: Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow and it is developed with a focus of enabling fast experimentation. Using TensorFlow as a backend framework We used its API keras, to import layers, Input, Dense, Dropout, GRU and LSTM models alongside with their Bidirectionality, Concatenate, optimizers and metrics.

Sklearn: This library which is also downloaded as scikit learn but imported as sklearn is the most useful library for deep and machine learning in Python with many tools. We used this library to import our evaluation metrics, precision_recall_fscore_support.

5.3 Model Development Architectures

5.4 Hyperparameters

Hidden Layers: A neural network that is intended to mimic a human brain, consists finely balanced mathematical expresions, programming novels and designing scince of applications. Neural network basically constructed up on neurons which are organized in layer fasion. Most of the time deep learning consists of atleast three types of layers: an input layer, hidden layer and an output layer[53]. There fore, we used **number of hidden layer** hyperparameter to be tuned based on our dataset, model performance and also coopration with other hyperparameters.

Number of nuerons: As Obviously known, number of neurons are a critical component of any deep learning model. They are nodes in which data and mathematical computations flows in either of input or hidden layers to that of output layers. Taking signals from these layers, they perform particular calculations and finally send signals to the corresponding deeper neuron. Thus, we used this crucial hyperparameter to be setted to its better number in tuning section.

Batch Size: Is among the critical hyperparameters in current deep Learning. This hyperparameter defines the number of samples to work through before updating the internal model parameters. Since, this hyperparameter is crucial step to make sure that our models hit peak performance we used and tuned it to its better in all of our models.

5.5 Evaluation Metrics

We used precision, recall and fscore evaluation metrics from scikit learn (sklearn) to evaluate our model performance both for hyperparameters tunning and final model testing. Precision_recall_fscore of scikit learn metrics are more preferable evaluation metrics for unsupervised machine learning. Since our model is trained on unlabeled data set, we preferred this evaluation metrics as it is suitable for our work. The idea of precision, recall, and F1-Score is based on the concepts of True Positive, True Negative, False Positive, and False Negative

which is commonly known as confusion matrix. The following table illustrates the confusion matrix of these binary classification.

		Predictions	
		Positive	Negative
Target	Positive	True Positive	False Negative
	Negative	False Posative	True Negative

Positive: the model predict the given input as what the actual class expected. In our case for example, if the word **'Megra'** is provided and the model predict it as **'Marga'** which is the actual target.

Negative: The model predicts the non-target input as not the expected. For our case as an example, if we provide the model to predict the correct word for **'Oddaa'** which is not the actual word (**'Odaa'**), and if the model get it as not a correct Afaan Oromoo word. Now let us cosider the following table as an example from our data and model to discuss what true positive, false negative, false positive and true negative is in detail.

TABLE 5.1: Confusion Matrix example from Dataset and the Model

Actual Value	Prediction	Type	Description
Garaagarummaa	Garaagarummaa	True Positive	The model predicted as a correct word and it was correct.
Gaaraagarummaa	Garagarummaa	True Negative	Checked as not correct word and predicted the correct word
Gaaraagarummaa	Gaaraagarummaa	False Positive	predicted as a correct word but not correct
Garaagarummaa	Gaaraagarummaa	False Negative	Checked as not correct word and predicted another but it was a correct word

Hence so, given a test dataset our evaluation mertics precision, recall and fscore are calculated from the models' performance using the general formula of these metrics.

Precision- is a measure of how many of the positive predictions made are actually positive (true positives). In simple term it is the fraction of positives predictions that are actually positive.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Recall- which is also known as sensitivity, is a measure of how many of the positive predictions the model correctly predicted, over all the actual positives in the data.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

F1-Score- is a measurent combination of both precision and recall. It is also known as the harmonic mean of precision and recall. Harmonic mean is just another way to calculate an average of values, basically more suitable for ratios (such as precision and recall) than the traditional arithmetic mean.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Finally, we used this measurement mechanism for hyperparameter tunning and the final model performance evaluation in the next sections.

5.6 Hyperparameters Tunning

In this section, we have discussed the tuning process and finding of best performed hyperparameters for all our proposed models namely, GRU, BiGRU, LSTM and BiLSTM.

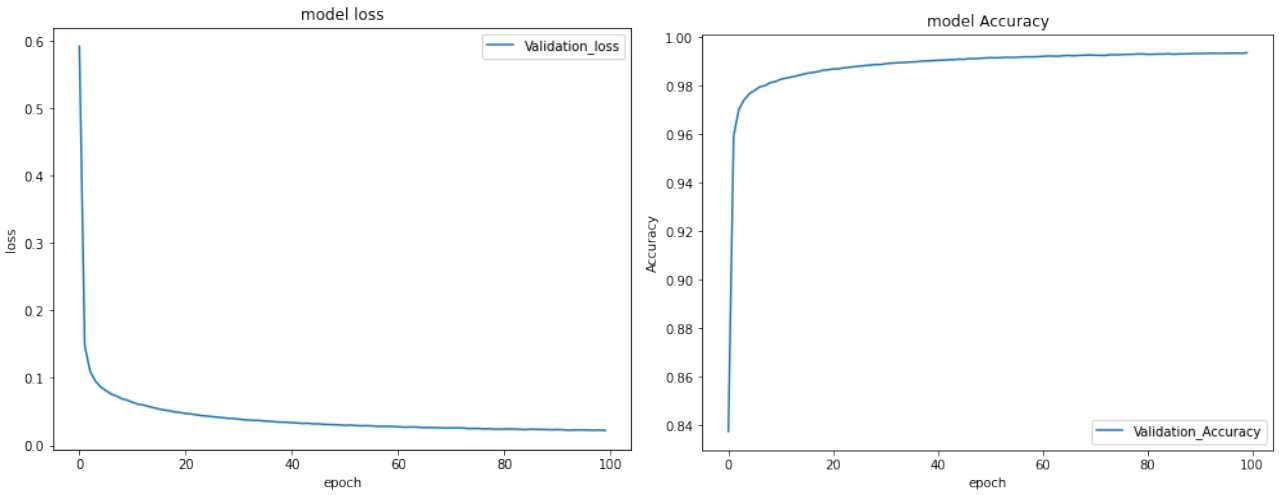
5.6.1 Hyperparameter Tunning for GRU Model

Firstly, we set the maximum number of epoch to 100 to decide the optimum number of epoch with randomly but most probably best hyper-parameters during model evaluation using categorical crossentropy loss and accuracy metrics at each epoch. The following are

randomly preferred hyperparameter setup initially taken to decide the most probably maximum number of epoch to make it fixed and then tune the rest of other hyperparameters.

Number of Layers = 2	Hidden cell = 512
Batch size = 128	Learning rate = 0.01
Activation function = Softmax	Optimizer = Adam
Sample mode = argmax	

As we can see from the Figure 5.2, both validation loss and validation accuracy did not show a significant change as the number of epoch became larger and larger. The higher number of epoch, the non-significant performance change is. As the epoch passes more than $\frac{1}{3}$ of the maximum number of epoch set (about 75), the models' performance get very smaller and smaller.



(A) Validation loss (B) Validation accuracy

FIGURE 5.2: Validation Loss and Accuracy with 100 Number of Epoch

Based on this fact, we decided to set number of epoch to 75 and proceeded to the tuning of other parameters. We still keep number of hidden layers to two hopping that the number of hidden layers are better to select based on the best selected parameters after tuning. Hence, we will proceed to tune for the optimum number of hidden layers after tuning the rest hyper-parameters. We followed the same steps with the same initial setups for all the rest models (BiGRU, LSTM and BiLSTM).

TABLE 5.2: Hyperparameters tuning for GRU model

Model	Fixed Hyperparameters	Variable Hyperparameters						Test Accuracy		
		Trials	Neuron	Batch Size	Learning rate	Error Rate	Activation Function	Precision	Recall	Fscore
GRU	Epoch=75	1	256	32	0.01	0.4	SoftMax	0.92522	0.74522	0.82552
		2	256	32	0.01	0.4	Sigmoid	0.9001	0.73588	0.80974
		3	256	32	0.01	0.6	SoftMax	0.92654	0.76882	0.83255
		4	256	32	0.01	0.8	SoftMax	0.90554	0.75251	0.82196
		5	256	32	0.001	0.6	SoftMax	0.94361	0.79025	0.86015
		6	256	32	0.0001	0.6	SoftMax	0.94628	0.78001	0.85514
		7	256	64	0.001	0.6	SoftMax	0.95547	0.80022	0.87098
		8	**256**	**128**	**0.001**	**0.6**	**SoftMax**	**0.95751**	**0.81725**	**0.899435**
		9	256	256	0.001	0.6	SoftMax	0.93245	0.77362	0.87236
		10	512	128	0.001	0.6	SoftMax	0.92891	0.79454	0.88071
		11	1024	128	0.001	0.6	SoftMax	0.90758	0.78491	0.85654

As shown in the Table 5.2 above, we made 11 trials to get the best combination of hyper-parameters by changing all possible values for each hyper-parameter. Finally the GRU model best performed on the 8^{th} trial(highlighted and bold row) with the following parameter values.

Number of neurons: **512** Error rate: **0.6 and**

Batch size: **128** Activation function: **Softmax**

Learning rate: **0.001**

The following Figure 5.3 illustrates the tuning trials and the pick at 8^{th} indicates the best collection of hyper-parameters the model performed well with.

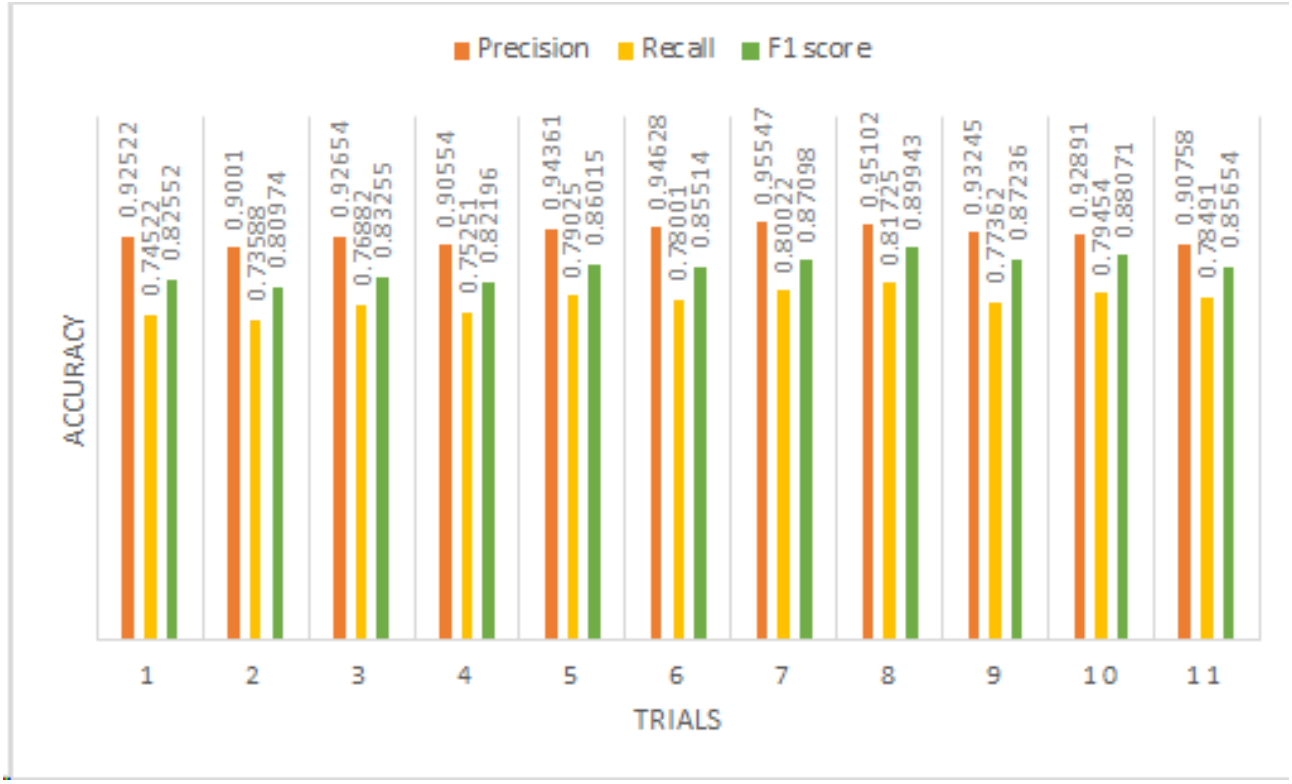


FIGURE 5.3: Hyperparameters Tuning Trials for GRU Model

Next we proceeded to tune number of hidden layers using these selected hyper-parameters. We used four trials providing number of hidden layer one up to five.

TABLE 5.3: Tuning of number of hidden layer with selected hyperparameters for GRU

Model	Fixed Hyperparameters	Variable Hyperparameters		Test Accuracy		
		Trials	Number of Layers	Precision	Recall	F-score
GRU	Epoch=75	1	1	0.84315	0.76650	0.85368
		2	2	0.95102	0.81725	0.8980
		3	**3**	**0.89624**	**0.82350**	**0.90320**
		4	4	0.9350	0.80451	0.88158
		5	5	0.92505	0.77051	0.84890

As shown in Table 5.3, on the first trial with preferred hyper-parameters from table 5.2 and a single hidden layer the model performed ***0.84315, 0.76650*** and ***0.85368*** of *precision, recall*

and *F1 score* respectively. On the second trial, assigning two hidden layers we got ***0.89102, 0.81725*** and ***0.8980*** performance on *precision, recall* and *F1 score* respectively. On the third trial, the model scored ***0.95624, 0.82350*** and ***0.90320*** of *precision, recall* and *F1 score* with selected hyper-parameters and three number of hidden layers. On the forth trial with four hidden layers the models' performance unfortunately decreased to ***0.9350, 0.80451*** and ***0.88158*** of *precision, recall and F1 score*. At the last on the fifth trial the models accuracy down performed to ***0.92505, 0.77051 and 0.84890*** *precision, recall* and *F1 score* respectively. Here, we stopped tuning and preferred the third trial which showed best accuracy.

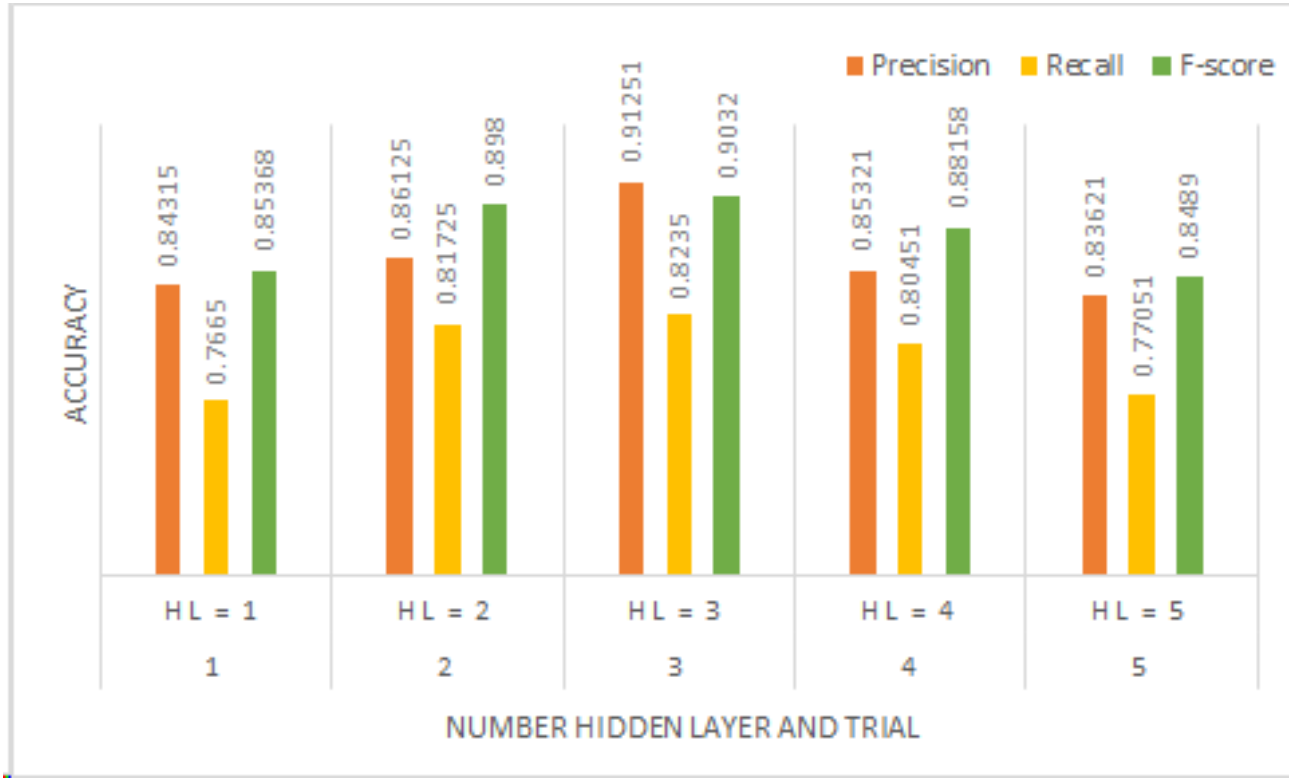


FIGURE 5.4: Number of Hidden Layer Tuning for GRU Model

Hence, the models best hyper-parameters to its optimum performance is:

TABLE 5.4: Final best hyper-parameters of GRU model

Epoch	Hidden layers	Neurons	Batch size	Learning rate	Error rate	Activation function	Precision	Recall	F-score
75	3	512	128	0.6	0.001	Softmax	89.624%	82.350%	90.320%

5.6.2 Hyper-parameter Tuning for BiGRU Model

We followed the same steps using initially selected hyper-parameters as we did in section 5.6.1 for GRU model and got the following experimental observation (Table 5.5).

TABLE 5.5: Hyper-parameters tuning for BiGRU model

Model	Fixed Hyperparameters	Variable Hyperparameters						Test Accuracy		
		Trials	Neuron	Batch Size	Learning rate	Error Rate	Activation Function	Precision	Recall	F1 score
BiGRU	Epoch=75	1	256	32	0.01	0.4	SoftMax	0.86570	0.80547	0.83450
		2	256	32	0.01	0.4	Sigmoid	0.82429	0.78952	0.80325
		3	256	32	0.01	0.6	SoftMax	0.91312	0.80512	0.85652
		4	256	32	0.01	0.8	SoftMax	0.88882	0.75694	0.81852
		5	256	32	0.001	0.6	SoftMax	0.87019	0.78591	0.85652
		6	256	32	0.0001	0.6	SoftMax	0.89042	0.76223	0.82014
		7	256	64	0.001	0.6	SoftMax	0.89125	0.81655	0.88502
		8	256	128	0.001	0.6	SoftMax	0.90947	0.80725	0.88506
		9	**256**	**64**	**0.001**	**0.6**	**SoftMax**	**0.91145**	**0.82440**	**0.90374**
		10	512	256	0.001	0.6	SoftMax	0.86321	0.77842	0.87541
		11	1024	128	0.001	0.6	SoftMax	0.87485	0.78030	0.87022

As shown in Table 5.5 above, we made 11 trials as usual to get the best combination of hyper-parameters by changing all possible values for each hyper-parameter. Finally, the Bi-GRU model best performed on the 9^{th} trial(highlighted and bold row) with the following parameter values.

Number of neurons: **256** Error rate: **0.6 and**

Batch size: **64** Activation function: **Softmax**

Learning rate: **0.001**

Despite that of general GRU that got high performance with 512 number of neurons and 128 batch size, BiGRU (evaluated by precision, recall and F1-score), performed ***0.91145, 0.82440 and 0.90374*** *precision, recall and F1 score* respectively with 256 number of neurons and 64 batch size. Perhaps, the rest of best hyper-parameters combined for BiGRU are the same as that of GRU.

Figure 5.5 shows the tuning trials and the pick at 9^{th} is where the best collection of hyper-parameters are combined together that the BiGRU performed better.

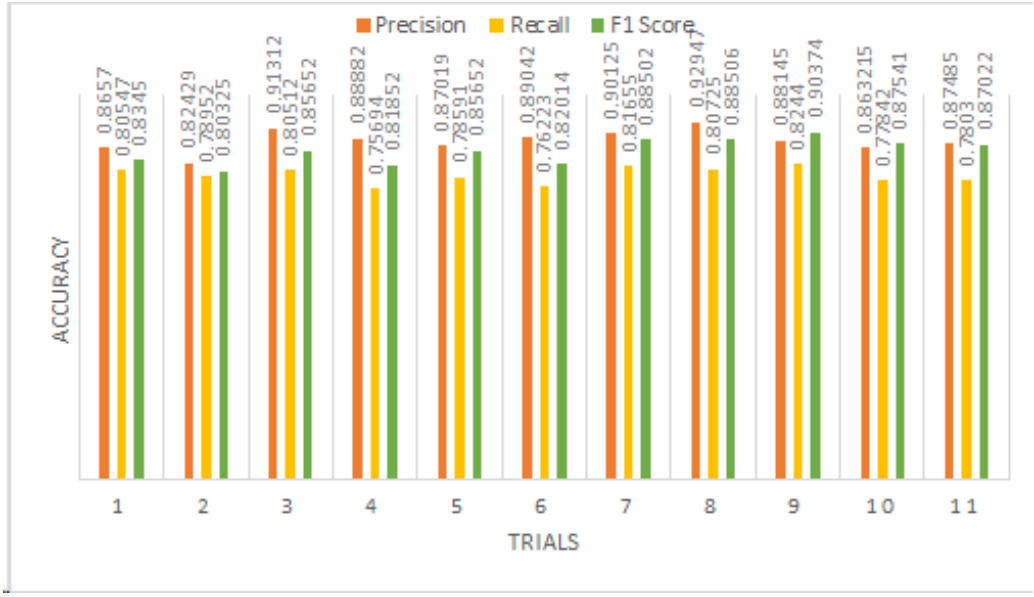


FIGURE 5.5: Hyperparameters Tuning Trials for BiGRU Model

The same way we did in the previous tuning in GRU for the optimum number of hidden layers, here also we proceeded to the tuning of number of hidden layers using these optimum hyper-parameters gained in table 5.5. We used four trials providing number of hidden layer from one up to five as shown in table 5.6.

TABLE 5.6: Tuning of number of hidden layer for BiGRU

Model	Fixed Hyperparameters	Variable Hyperparameters		Test Accuracy		
		Trials	Number of Layers	Precision	Recall	F-score
BiGRU	Epoch=75	1	1	0.78214	0.80599	0.89258
		2	2	0.84254	0.82440	0.90374
		3	**3**	**0.92325**	**0.83500**	**0.91008**
		4	4	0.85124	0.75952	0.86333
		5	5	0.84325	0.74002	0.83665

Table 5.6 show that, on the first trial with preferred hyper-parameters from table 5.5 and using one(1) hidden layer the models scored ***0.7821, 0.80599 and 0.89258*** of *precision, recall and*

F1 score respectively. On the second trial, providing two(2) hidden layers we got ***0.84254, 0.83249 and 0.90859*** accuracy on *precision, recall and F-score* respectively which is slightly the same as in the 9^{th} trial from table 5.5. On the third trial, the model performed ***0.92325, 0.83500 and 0.91008*** for *precision, recall and F1 score* respectively with other hyper-parameters tuned in table 5.5 and three(3) number of hidden layers. At the fourth trial with four(4) hidden layers the models' accuracy decreased to ***0.85124, 0.75952 and 0.86333*** of *precision, recall and F1 score* respectively. On the last trial (fifth trial) the models' accuracy under performed to ***0.84325*** *precision* ***0.74002*** *recall* and ***0.83665*** *F1 score*. Here, we then stopped searching for more better number of hidden layers and selected the third trial which showed **best** accuracy with three number of hidden layers. The Figure 5.6 blow illustrates summary of this observation.

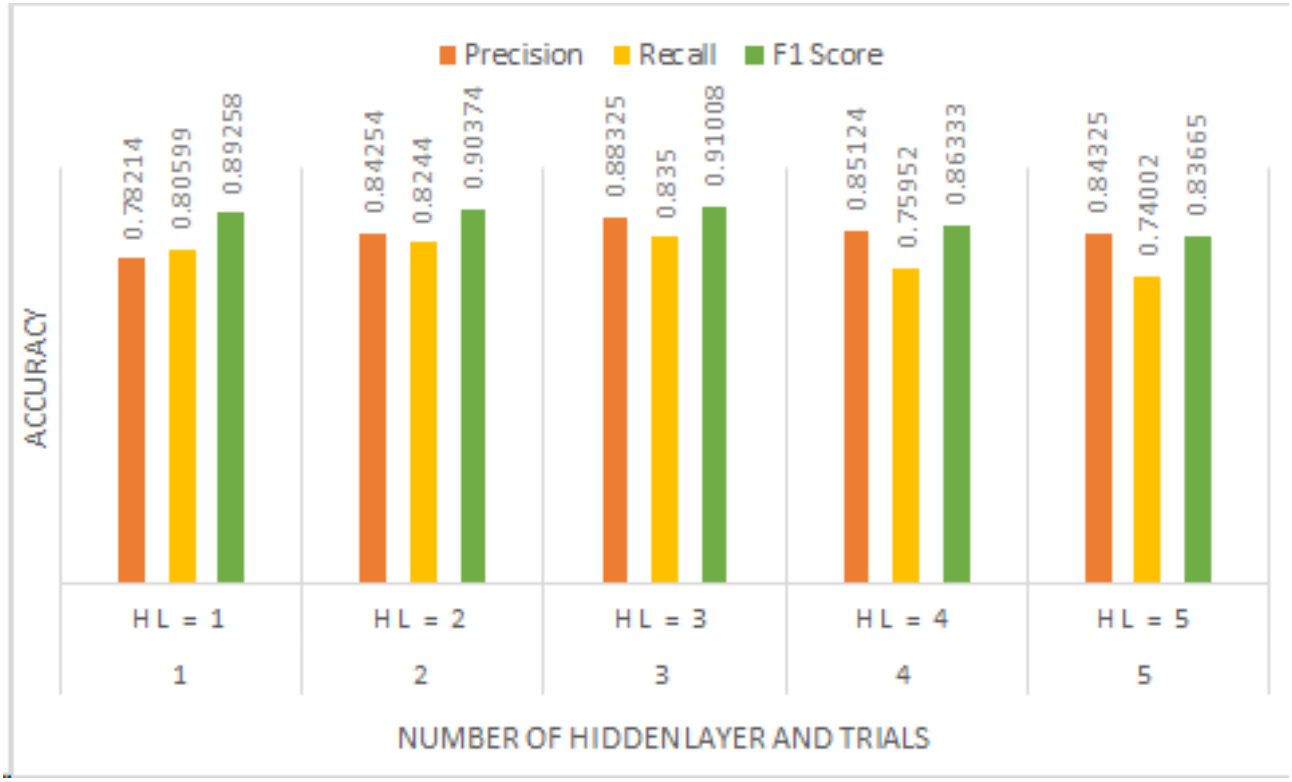


FIGURE 5.6: Number of Hidden Layer Tuning for BiGRU Model

There fore, the BiGRU's best hyper-parameters to its optimum accuracy are summarized as presented in Table 5.7

TABLE 5.7: Final best hyper-parameters of BiGRU model

Epoch	Hidden layers	Neurons	Batch size	Learning rate	Error rate	Activation function	Precision	Recall	F1 score
75	3	256	64	0.6	0.001	Softmax	92.325%	83.500%	91.008%

5.6.3 Hyper-parameter Tuning for LSTM Model

Even though the result may differs, we tuned the best hyper-parameters manually, all procedures we took is slightly almost the same for all cases. Here for LSTM and BiLSTM models, we also followed the same steps as in case of GRU and BiGRU. Using initially selected hyper-parameters as we did in previous sections, we got the following investigations summarized in Table 5.8 for LSTM model.

TABLE 5.8: Hyper-parameters tuning for LSTM model

Model	Fixed Hyperparameters	Variable Hyperparameters						Test Accuracy		
		Trials	Neuron	Batch Size	Learning rate	Error Rate	Activation Function	Precision	Recall	F-score
LSTM	Epoch=75	1	256	32	0.01	0.4	SoftMax	0.82654	0.71624	0.81839
		2	256	32	0.01	0.4	Sigmoid	0.80251	0.70751	0.82870
		3	256	32	0.01	0.6	SoftMax	0.84201	0.72560	0.84098
		4	256	32	0.01	0.8	SoftMax	0.86321	0.71201	0.83178
		5	256	32	0.001	0.6	SoftMax	0.85532	0.74504	0.85389
		6	256	32	0.0001	0.6	SoftMax	0.86326	0.73888	0.83968
		7	256	64	0.001	0.6	SoftMax	0.87019	0.76602	0.86078
		8	256	128	0.001	0.6	SoftMax	0.89014	0.77173	0.87115
		9	256	256	0.001	0.6	SoftMax	0.88652	0.75757	0.86207
		10	**512**	**128**	**0.001**	**0.6**	**SoftMax**	**0.90255**	**0.78029**	**0.87658**
		11	1024	128	0.001	0.6	SoftMax	0.87254	0.77396	0.87257
		12	1024	256	0.001	0.6	SoftMax	0.84962	0.75369	0.85954

As we can observe from Table 5.8, we made 12 trials to get the best collection of hyper-parameters for LSTM model by changing possible values of each hyper-parameter. At the end of our trials, the LSTM model scored best accuracy of ***0.90255, 0.78029*** and ***0.87658*** *precision, recall* and *F1 score* respectively on the 10^{th} trial as shown with the following hyper-parameter values:

Number of neurons: **512** Error rate: **0.6 and**

Batch size: **128** Activation function: **Softmax**

Learning rate: **0.001**

Figure 5.7 following is a graphical summary of hyper-parameter tuning for LSTM model.

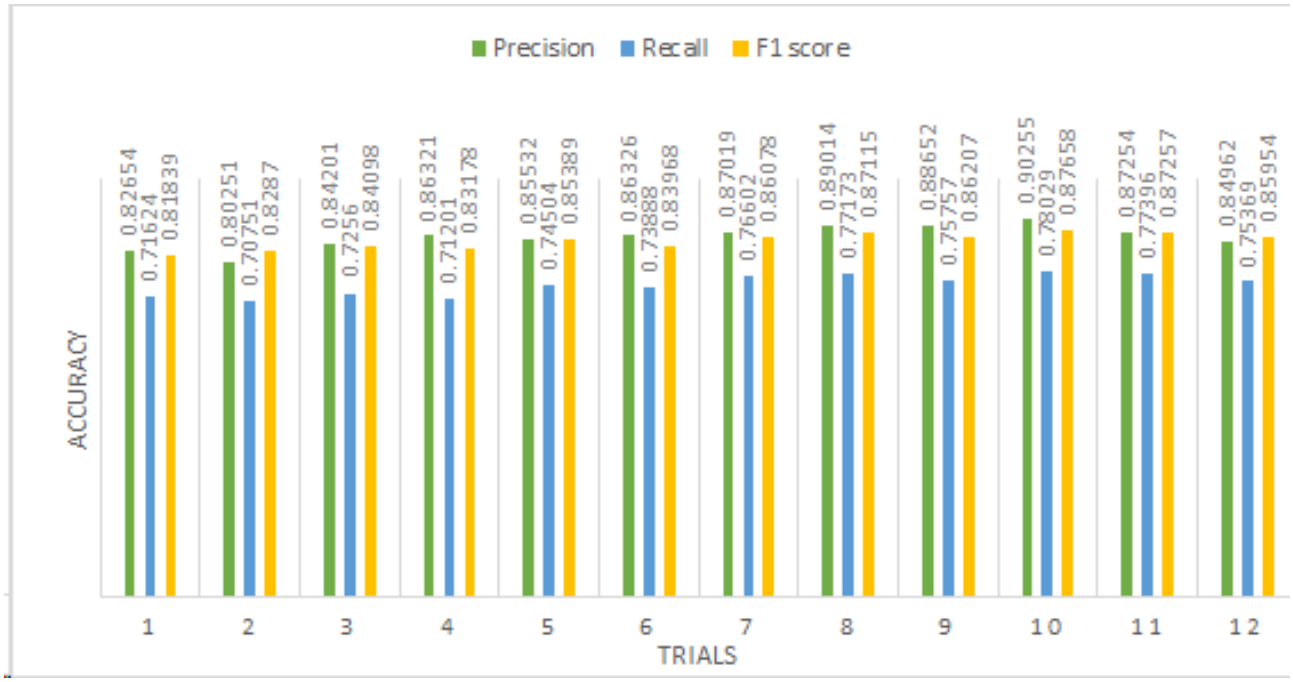


FIGURE 5.7: Hyperparameters Tuning Trials for LSTM Model

The same way, using these combination of best hyper-parameters gained from Table 5.8, we then take a search for optimum number of hidden layers for the LSTM model. Table 5.9 shows the procedure we used providing number of hidden layer from one up to five.

TABLE 5.9: Number of hidden layers Tuning for LSTM model

Model	Fixed Hyperparameters	Variable Hyperparameters		Test Accuracy		
		Trials	Number of Layers	Precision	Recall	F-score
LSTM	Epoch=75	1	1	0.79541	0.76275	0.80214
		2	2	0.80125	0.78121	0.8681
		3	**3**	**0.84215**	**0.79259**	**0.88429**
		4	4	0.82154	0.78121	0.83251
		5	5	0.79897	0.77514	0.78124

As summarized in Table 5.9, an LSTM model scored ***0.84215, 0.82420*** and ***0.90363*** *precision, recall and F1 score* on the third trial with three hidden layers. With cooperation of best hyper-parameters gained from Table 5.8, an LSTM model scored optimum accuracy at the third trail which is when three number of hidden layer is used.

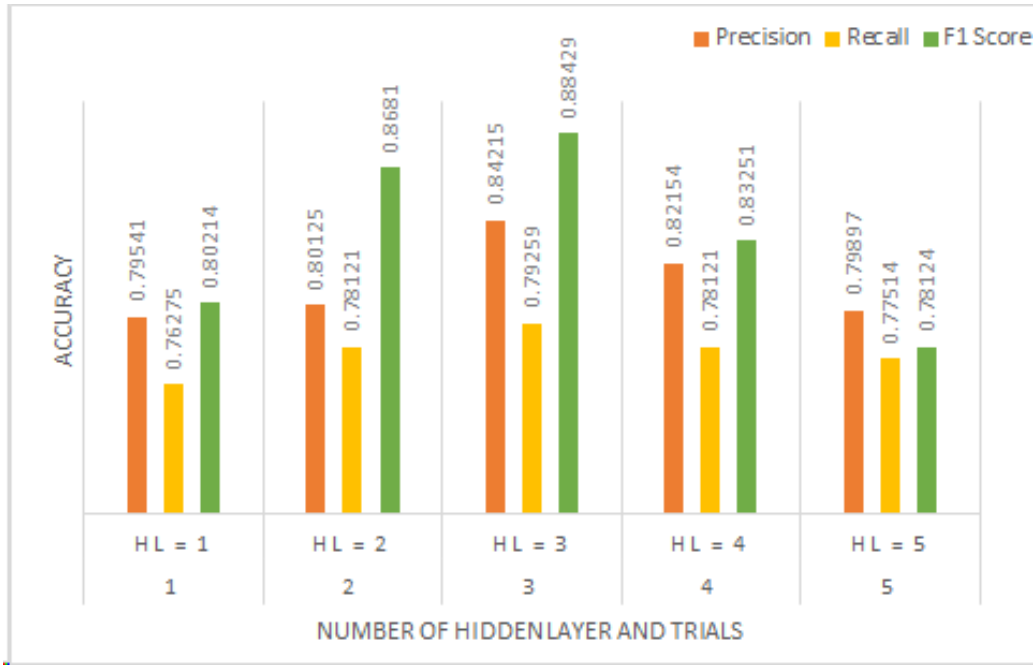


FIGURE 5.8: Number of hidden layer tuning for LSTM model

From both tuning trials, the models'(LSTM) best hyper-parameters is summarized as shown in Table 5.10 below.

TABLE 5.10: Final best hyper-parameters of LSTM model

Epoch	Hidden layers	Neurons	Batch size	Learning rate	Error rate	Activation function	Precision	Recall	F-score
75	3	512	128	0.6	0.001	Softmax	82.215%	79.259%	88.429%

5.6.4 Hyper-parameter Tuning for BiLSTM Model

Table 5.11 following shows 12 trials we took with different values of our hyper-parameters to get a best combination of them for BiLSTM model. Here also, we took the same steps as in all our previous hyper-parameter tuning we did manually. The default initial number of epoch is 75 and number of hidden layer are two with the fact we discussed in section 5.6.1.

TABLE 5.11: Hyper-parameters tuning for BiLSTM model

Model	Fixed Hyperparameters	Variable Hyperparameters						Test Accuracy		
		Trials	Neuron	Batch Size	Learning rate	Error Rate	Activation Function	Precision	Recall	F-score
BiLSTM	Epoch=75	1	256	32	0.01	0.4	SoftMax	0.78325	0.74670	0.81621
		2	256	32	0.01	0.4	Sigmoid	0.76491	0.73800	0.81098
		3	256	32	0.01	0.6	SoftMax	0.80621	0.76362	0.86596
		4	256	32	0.01	0.8	SoftMax	0.79656	0.74210	0.85196
		5	256	32	0.001	0.6	SoftMax	0.84565	0.77801	0.87514
		6	256	32	0.0001	0.6	SoftMax	0.83251	0.76500	0.86685
		7	256	64	0.001	0.6	SoftMax	0.86321	0.79213	0.88401
		8	**256**	**128**	**0.001**	**0.6**	**SoftMax**	**0.91521**	**0.81259**	**0.89661**
		9	256	256	0.001	0.6	SoftMax	0.82621	0.79303	0.88456
		10	512	128	0.001	0.6	SoftMax	0.80325	0.80206	0.89015
		11	1024	128	0.001	0.6	SoftMax	0.79252	0.78780	0.88130
		12	1024	256	0.001	0.6	SoftMax	0.73625	0.80101	0.88951

As we can see from Table 5.11, we made 12 trials to get the best collection of hyper-parameters for this model(BiLSTM) by changing most probably possible values of each hyper-parameter. At the end of our trials the model scored best accuracy of ***0.91521, 0.81259*** and ***0.89661*** *precision, recall* and *F1 score* on the 8^{th} trial with the following hyper-parameter values:

Number of neurons: **256** Error rate: **0.6 and**

Batch size: **128** Activation function: **Softmax**

Learning rate: **0.001**

The following graph(figure 5.5) shows the tuning trials and the pick at 9^{th} is where the best collection of hyper-parameters are combined together that the BiGRU performed better.

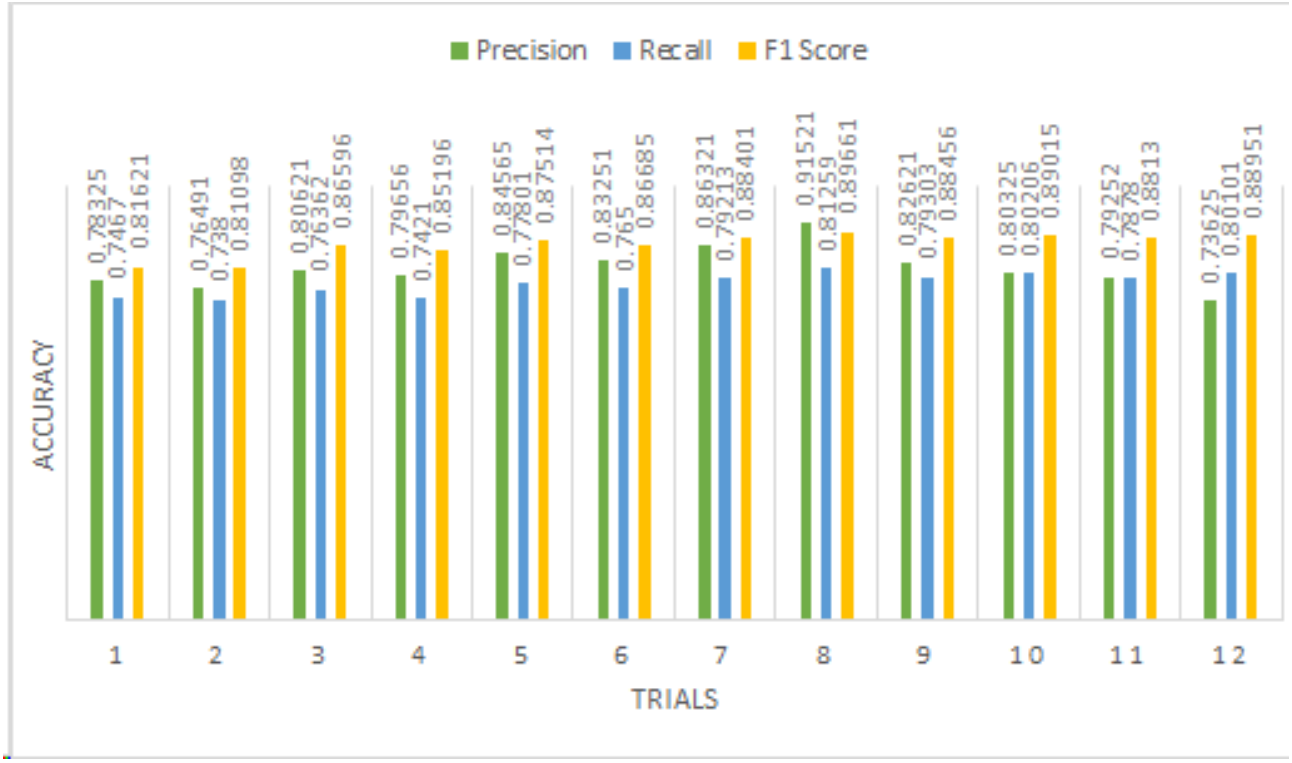


FIGURE 5.9: Hyperparameters tuning trials for BiLSTM model

The same way we did for our previous models, using these combination of best hiyperparameters gained from Table 5.11, we then tuned for optimum number of hidden layers for our BiLSTM model as well. Table 5.12 indicates activities we used four trials given number of hidden layers from one up to five.

TABLE 5.12: Tuning of number of hidden layers for BiLSTM model

Model	Fixed Hyperparameters	Variable Hyperparameters		Test Accuracy		
		Trials	Number of Layers	Precision	Recall	F-score
BiLSTM	Epoch=75	1	1	0.87254	0.70401	0.77926
		2	2	0.88121	0.81259	0.89661
		3	**3**	**0.92054**	**0.82420**	**0.90363**
		4	4	0.91654	0.80329	0.85618
		5	5	0.84621	0.78268	0.81320

Experiment in Table 5.12 illustrates, BiLSTM model permed ***0.92054, 0.82420*** and ***0.90363*** *precision, recall and F1 score* on the third trial with three hidden layers. With cooperation of

best hyper-parameters gained from Table 5.11, BiLSTM model scored optimum accuracy at the third trail.

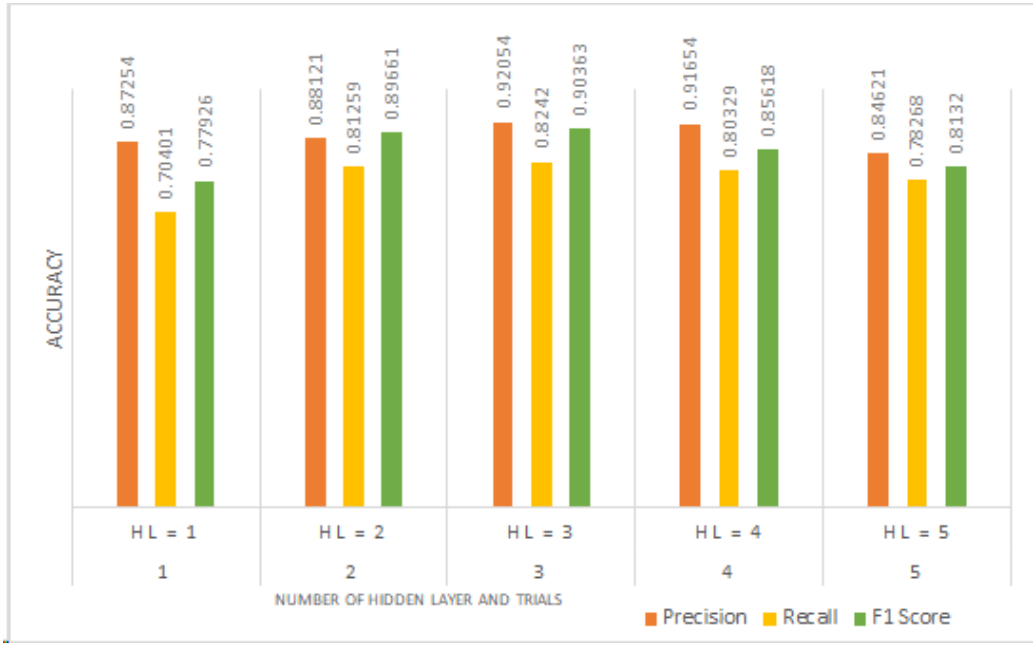


FIGURE 5.10: Number of hidden layer tuning for LSTM model

We now summarized the best hyper-parameters combination in Table 5.13 below, that a BiLSTM model performed best accuracy at the third trial with three numbers of hidden layers, with the rest of hyper-parameters obtained from Table 5.11.

TABLE 5.13: Final best hyper-parameters of BiLSTM model

Epoch	Hidden layers	Neurons	Batch size	Learning rate	Error rate	Activation function	Precision	Recall	F-score
75	3	256	128	0.6	0.001	Softmax	92.054%	82.420%	90.363%

5.7 Expermental Discussion and Conclusion

In this experiment we used to tune best hyper-parameters for four(4) different models namely GRU, BiGRU, LSTM and BiLSTM. We used manual trial best hyper-parameter selection and throughout our procedural steps, we got different values (weights) for all of our models testing them using Precision, Recall and F-score metrics. we initially set number of training epoch to 100 and number of hidden layers to two(2). We then randomly took (but most

probably best values) of other hyper-parameters like, number of neurons, batch size, learning rate, error rate and activation functions. We made eleven to twelve trials for each of our model to get best combination parameters. One combination of best performed hyper-parameters are chosen for each models and once it is done, we then used them to tune for best number of hidden layers in each model. Each of our models then had their own best hyper-parameters which we then compare them with each other to generalize and decide the best model for our thesis to work with. In the next section we visualized a comparison of these models in both tabular form and graphically.

5.7.1 Comparison of candidate Models

Table 5.14 summarizes the performance of all of our models measured by Precision, Recall and F-score.

TABLE 5.14: Comparisons of Our Models performance

Model	Precision	Recall	Fscore
GRU	89.624%	82.350%	90.320%
BiGRU	**92.325%**	**83.500%**	**91.008%**
LSTM	82.215%	79.259%	88.429%
BiLSTM	92.054%	82.420%	90.363%

As it is seen from Table 5.14, the BiGRU model best performed over the others scoring ***0.92325, 0.83500 and 0.91008*** for *precision, recall and F1 score* score respectively within the same test dataset used in all cases. We also illustrate this graphically in Figure 5.7.

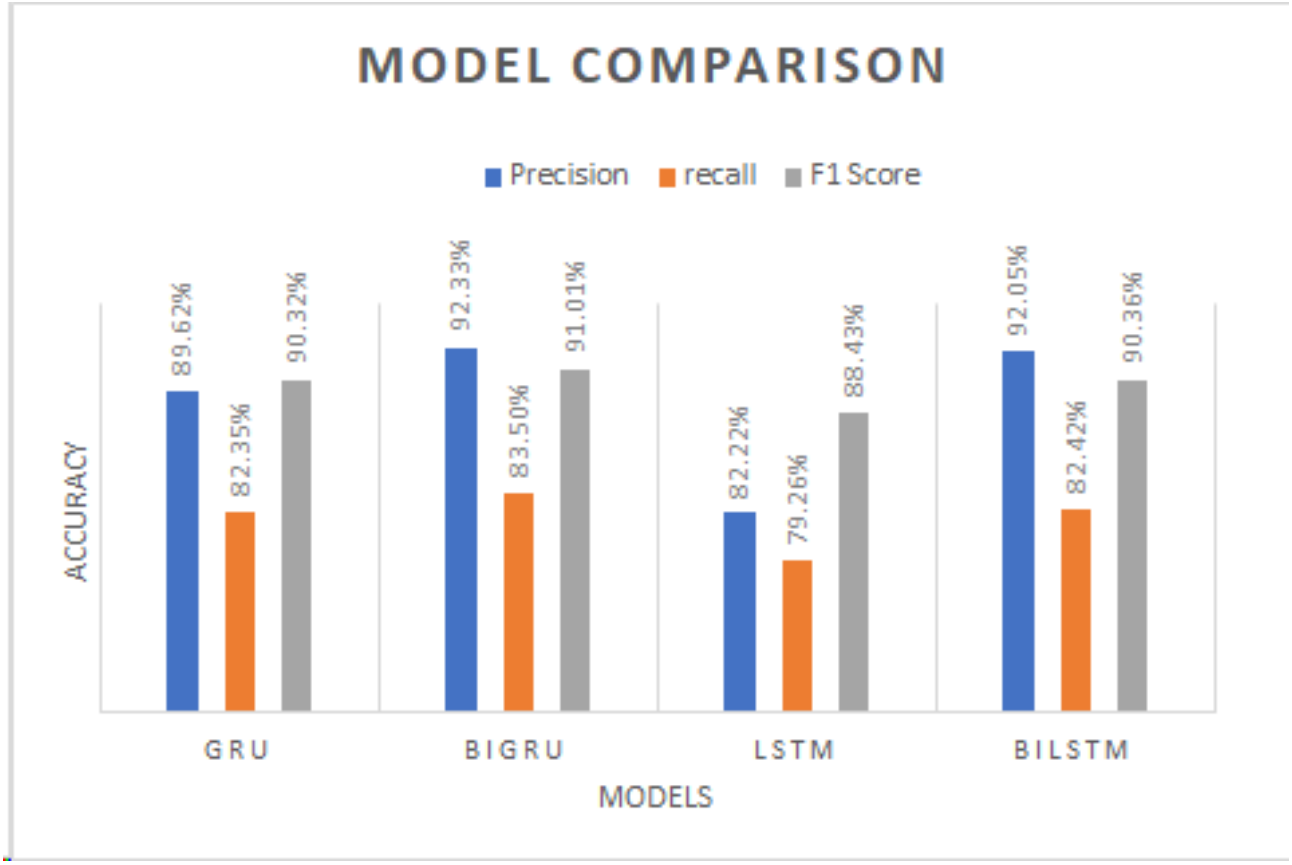


FIGURE 5.11: Comparisons of Models

Hence, we preferred BiGRU to the best of our thesis work with 92.325%, 83.5% and 91.008% precision, recall and F1 score respectively.

5.7.1.1 BiGRU with one, two and three characters operations

In addition to one character operation(deletion, addition, replacement and transposition) during artificial word error making, we also compromised two and three character operations for the preferred model (BiGRU). The model then scored the following accuracy summarized in table 5.15 below.

TABLE 5.15: Different number of character operation and accuracy gained

#of character operation (to BiGRU)	Precision	Recall	Fscore
One	**92.325%**	**83.500%**	**91.008%**
Two	**76.501%**	**72.602%**	**73.4332%**
Three	**72.005%**	**64.787%**	**68.206%**

Figure 5.12 shows the summary features of the preferred model(BiGRU) printed from the model.summary of model implementation.

```
Model: "model_27"
_________________________________________________________________________________________________
 Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
 encoder_data (InputLayer)    [(None, None, 55)]        0           []

 bidirectional_30 (Bidirectiona  [(None, None, 512),  480768     ['encoder_data[0][0]']
 l)                           (None, 256),
                              (None, 256)]

 bidirectional_31 (Bidirectiona  [(None, None, 512),  1182720    ['bidirectional_30[0][0]',
 l)                           (None, 256),                        'bidirectional_30[0][1]',
                              (None, 256)]                        'bidirectional_30[0][2]']

 bidirectional_32 (Bidirectiona  [(None, None, 512),  1182720    ['bidirectional_31[0][0]',
 l)                           (None, 256),                        'bidirectional_31[0][1]',
                              (None, 256)]                        'bidirectional_31[0][2]']

 decoder_data (InputLayer)    [(None, None, 55)]        0           []

 concatenate_9 (Concatenate)  (None, 512)               0           ['bidirectional_32[0][1]',
                                                                     'bidirectional_32[0][2]']

 decoder_gru (GRU)            [(None, None, 512),  873984      ['decoder_data[0][0]',
                              (None, 512)]                        'concatenate_9[0][0]']

 decoder_softmax (Dense)      (None, None, 55)          28215       ['decoder_gru[0][0]']

==================================================================================================
```

FIGURE 5.12: Model Summary

figure 5.13 below shows the preferred model, BiGRU architecture diagram dawn from the implementation in keras. It consists of two input layers for encoder and decoder (encoder_data, decoder_data), three hidden layers (Bidirectional(GRU)) with 256 number of neurons and one output layer (decoder_softmax).

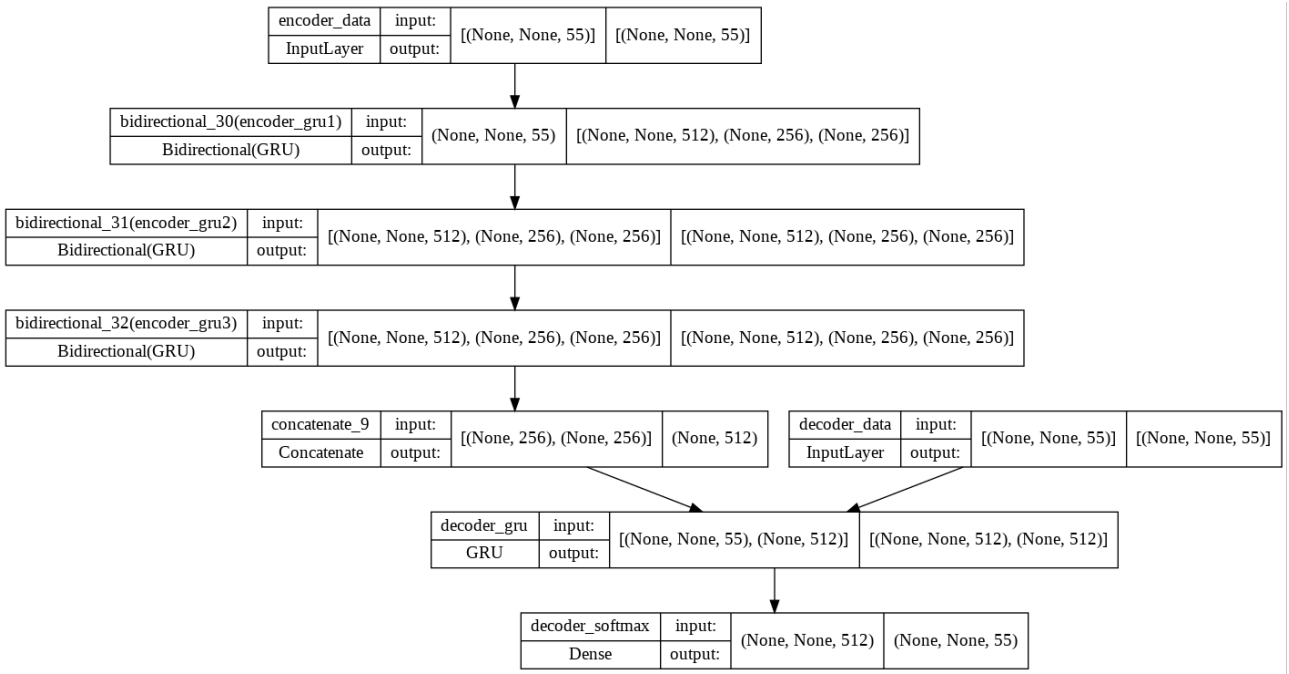


FIGURE 5.13: Model Diagram

5.8 Prototype

We implemented a graphical user interface for the final model using TKinter python GUI implementation library. Figure 5.8 below is a sample screen shoot from the interface. As we we can see from the text entry field, we entered a sample test sentence (*"**Sireessituu sirna barreeffama qubtee afaan oromoo.**"*). As it is seen from the sentence the model detected three words as a misspelled words(red underlined) which one of them is wrongly detected (the word "*oromoo"* was correctly spelled but the model detect it as incorrect). On the auto-correct field the model automatically corrected them. The model detect and correct if not correct the word when we press a space bar after typing a word on the text editor field and

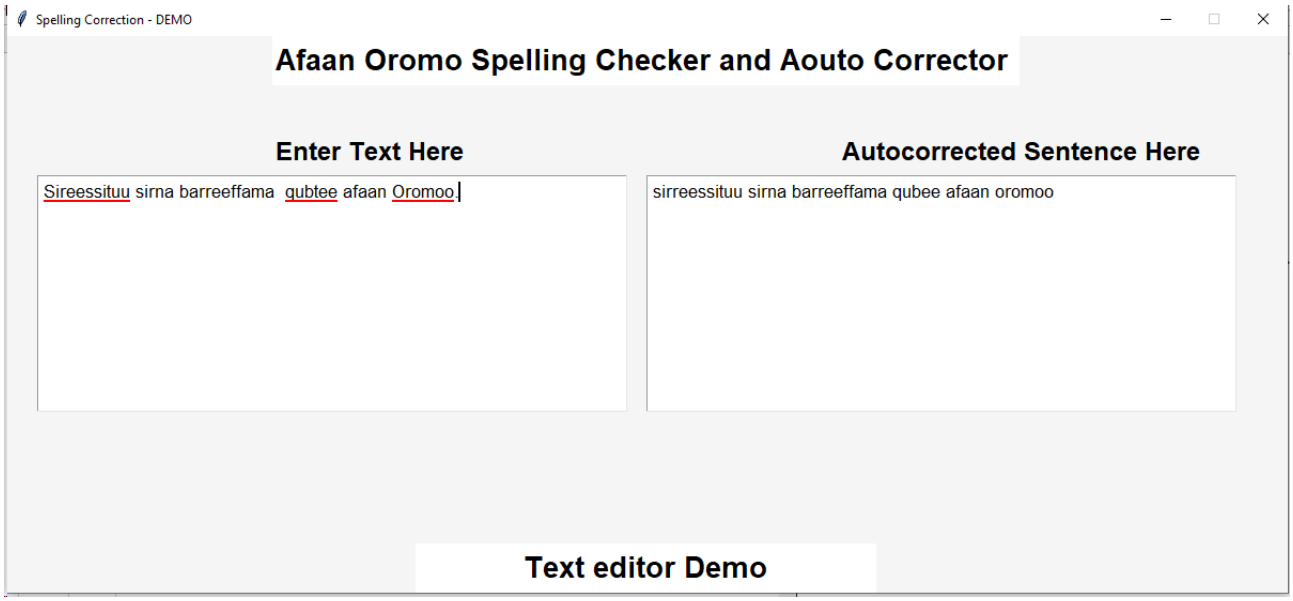


FIGURE 5.14: Prototype sample

Chapter 6

Conclusion and Future Work

6.1 Introduction

In this chapter, we will discuss the overall summary of the thesis work and future recommendation regarding our work on designing and implementing spelling checker and corrector for Afaan Oromo using deep learning. All what we tried to cover, our thesis outcome, some challenges we faced and what we recommend for the future extension of the work to the interested researcher in the same area, specifically for Afaan Oromo spelling checker and corrector are discussed in this chapter.

Throughout the whole of our work in this paper, we are tried all our bests starting from data gathering needed for the proposed system to the final result. It is actually obvious that in deep learning model development, the most expensive and crucial thing is preparing dataset needed for the desired problem to solve. Especially, this task is more tedious for those with low resourced languages like almost most all languages in our country, Ethiopia. As commonly known, Afaan Oromo is the first largely spoken in Ethiopia and even it is the 4th ranked having large number of speakers in Africa. However, it lacks many digital usability for several tasks like machine translation, speech recognition, grammar checker and corrector, spelling checker and corrector and many others. The leading problem for all this lack is as a result of dataset shortage to work with. In our thesis context, we have tried to the worst trial to get more data as much. We have collected 37311 Afaan Oromo sentences consisting 609311 words from several sources including news data from different broadcasting corporations in Afaan Oromo like FBC, OBN, OMN, EBC Afaan Oromo, BBC Afaan Oromo. we also used several Afaan Oromo books (educational, fiction, general books and others) and holy bible in Afaan Oromo. Passing through preprocessing as needed to our task, these data are used in our model development.

For the implementation of the model, we used different environments, computational resources, programming language, type of model architecture development and model evaluation metrics. As a development environment we used a google collaboratory with its GPU accelerator. Python 3.8 with all its required libraries are used as an implementation programming language. We tried to develop Afaan Oromo word spelling error checker and corrector model using four RNN architecture namely, Gated Recurrent Unit (GRU), Bidirectional Gated Recurrent Unit (BiGRU), Long Short Term Memory (LSTM) and Bidirectional Long Short Term Memory (BiLSTM) approaches using sequence to sequence approach. We tuned and assigned best performing hyper-parameters for all of these models. finally, we compared their accuracy performance using Precision, Recall and F1 score evaluation metrics which is one of scikit learn python library, and preferred the outstanding model among them. With the test data, which is 20% of our dataset(121863 words), GRU performed ***0.95624, 0.82350*** and ***0.90320*** of *precision, recall* and *F1 score* respectively, BiGRU performed ***0.92325, 0.83500 and 0.91008*** for *precision, recall and F1 score* respectively, LSTM performed ***0.84215, 0.82420*** and ***0.90363*** *precision, recall and F1 score* and BiLSTM performed ***0.92054, 0.82420*** and ***0.90363*** *precision, recall and F1 score*.

Accuracy of all candidates are nearly equal. In all cases the precision accuracy is higher than recall and F1 score. this indicates the model evaluated on test data predicted very less False Positive (FP). BiGRU performed well compared to the others, so that we preferred it for the last model for the Afaan Oromo spelling checker and corrector with ***0.92325, 0.83500 and 0.91008*** for *precision, recall and F1 score* respectively.

6.2 Future Work and Recommendation

To this specific area, spelling checker and corrector we would like to recommend the following points for future extension.

- We only used GRU and LSTM with their bidirectionalities but it is better if an attention is added for better accuracy improvement.
- We developed a text editor field for the demonstration but it is worthy if it is used as a plugin in professional word processing tool like MS office.

- An android app makes this tool more advantageous if it developed with more accuracy.

Appendix A

Last 10 epoches training summary for Bi-GRU model

Main Epoch 91/100

Shuffling data.

488/488 [==============================] - 120s 246ms/step - loss: 0.0335 - accuracy: 0.9906 - truncated_acc: 0.9830 - truncated_loss: 0.0606 - val_loss: 0.0294 - val_accuracy: 0.9916 - val_truncated_acc: 0.9848 - val_truncated_loss: 0.0532

-

Input tokens: ['jechoota', 'hiZ', 'qamama', 'keenyaraa', 'baka']

Decoded tokens: ['jechoota', 'hin', 'qaamaa', 'keenyarraa', 'bakka']

Target tokens: ['jechoota', 'hin', 'qaama', 'keenyarraa', 'bakka']

-

Main Epoch 92/100

Shuffling data.

488/488 [==============================] - 118s 241ms/step - loss: 0.0331 - accuracy: 0.9906 - truncated_acc: 0.9831 - truncated_loss: 0.0600 - val_loss: 0.0297 - val_accuracy: 0.9918 - val_truncated_acc: 0.9851 - val_truncated_loss: 0.0538

-

Input tokens: ['afuucfuuf', 'maaln', 'jedhee', 'Maalna', 'lamaan']

Decoded tokens: ['afuufuuf', 'maalan', 'jedhee', 'Maalan', 'lamaan']

Target tokens: ['afuufuuf', 'malan', 'jedhee', 'Maalan', 'lamaan']

-

Main Epoch 93/100

Shuffling data.

488/488 [==============================] - 119s 244ms/step - loss: 0.0328 - accuracy: 0.9906 - truncated_acc: 0.9831 - truncated_loss: 0.0595 - val_loss: 0.0294 - val_accuracy: 0.9917 - val_truncated_acc: 0.9850 - val_truncated_loss: 0.0532

- Input tokens: ['fi', 'ilaalcha', "ba'e", 'aqbuuf', 'miraneesse']

Decoded tokens: ['fi', 'ilaalcha', "ba'e", 'qabuuf', 'mirkaneesse']

Target tokens: ['fi', 'ilaalcha', "ba'e", 'qabuuf', 'mirkaneesse']

-

Main Epoch 94/100

Shuffling data.

488/488 [==============================] - 119s 244ms/step - loss: 0.0331 - accuracy: 0.9906 - truncated_acc: 0.9830 - truncated_loss: 0.0600 - val_loss: 0.0293 - val_accuracy: 0.9918 - val_truncated_acc: 0.9851 - val_truncated_loss: 0.0531

-

Input tokens: ['oon', 'daafga', 'yoo', 'yoo', 'AcOhiin']

Decoded tokens: ['oon', 'daagaa', 'yoo', 'yoo', 'Achiin']

Target tokens: ['yoon', 'daanga', 'yoo', 'yoo', 'Achiin']

-

Main Epoch 95/100

Shuffling data.

488/488 [==============================] - 119s 244ms/step - loss: 0.0329 - accuracy: 0.9908 - truncated_acc: 0.9833 - truncated_loss: 0.0596 - val_loss: 0.0290 - val_accuracy: 0.9918 - val_truncated_acc: 0.9852 - val_truncated_loss: 0.0525

-

Input tokens: ['Sirni', 'garu ', 'keeIii', 'irGaa', 'aha']

Decoded tokens: ['Sirni', 'garuu', 'keetii', 'irmaa', 'haa']

Target tokens: ['Sirni', 'garuu', 'keetii', 'irraa', 'haa']

-

Main Epoch 96/100

Shuffling data.

488/488 [==============================] - 121s 247ms/step - loss: 0.0329 - accuracy: 0.9907 - truncated_acc: 0.9831 - truncated_loss: 0.0596 - val_loss: 0.0295 - val_accuracy: 0.9918 - val_truncated_acc: 0.9852 - val_truncated_loss: 0.0535

-

Input tokens: ['nammoonni', 'waliin', 'dubbatZe', 'fuan', 'dhukkubaa']

Decoded tokens: ['nammoonni', 'waliin', 'dubbate', 'fufan', 'dhukkubaa']

Target tokens: ['namoonni', 'waliin', 'dubbatee', 'fufan', 'dhukkubaa']

-

Main Epoch 97/100

Shuffling data.

488/488 [==============================] - 121s 248ms/step - loss: 0.0327 - accuracy: 0.9907 - truncated_acc: 0.9833 - truncated_loss: 0.0592 - val_loss: 0.0301 - val_accuracy: 0.9917 - val_truncated_acc: 0.9850 - val_truncated_loss: 0.0545

-

Input tokens: ['dhirsa', 'fi', 'kaasaniiru', 'saan', 'ar emiis']

Decoded tokens: ['dhirsa', 'fi', 'kaasaniiru', 'saan', 'arcemiis']

Target tokens: ['dhirsa', 'fi', 'kaasaniiru', 'isaan', 'arxemiis']

-

Main Epoch 98/100

Shuffling data.

488/488 [==============================] - 119s 245ms/step - loss: 0.0332 - accuracy: 0.9906 - truncated_acc: 0.9830 - truncated_loss: 0.0601 - val_loss: 0.0293 - val_accuracy: 0.9919 - val_truncated_acc: 0.9853 - val_truncated_loss: 0.0531

-

Input tokens: ['aGka', 'BiliseeS', 'fsaa', 'gaarOii', "saa'oyiin"]

Decoded tokens: ['akka', 'Biliseen', 'isaa', 'gaarii', "saa'ooiin"]

Target tokens: ['akka', 'Biliseen', 'isaa', 'gaarii', "saa'oliin"]

-

Main Epoch 99/100

Shuffling data.

488/488 [==============================] - 121s 248ms/step - loss: 0.0330 - accuracy: 0.9907 - truncated_acc: 0.9831 - truncated_loss: 0.0598 - val_loss: 0.0293 - val_accuracy: 0.9917 - val_truncated_acc: 0.9850 - val_truncated_loss: 0.0531

-

Input tokens: ['diqisiifannaadhaan', 'irEa', 'fayyadamoo', 'ofiLf', 'wana']

Decoded tokens: ['dinqisiifannaadhaan', 'iraa', 'fayyadamoo', 'ofiif', 'wanna']

Target tokens: ['dinqisiifannaadhaan', 'irra', 'fayyadamoo', 'ofiif', 'waan']

-

Main Epoch 100/100

Shuffling data.

488/488 [==============================] - 122s 250ms/step - loss: 0.0322 - accuracy: 0.9909 - truncated_acc: 0.9835 - truncated_loss: 0.0583 - val_loss: 0.0288 - val_accuracy: 0.9919 - val_truncated_acc: 0.9853 - val_truncated_loss: 0.0522

-

Input tokens: ['bshannanaaf', 'of', 'isaani', 'lja', 'idraa']

Decoded tokens: ['bishannanaaf', 'of', 'isaani', 'laa', 'diraa']

Target tokens: ['bashannanaaf', 'of', 'isaanii', 'ija', 'irraa']

-

Saving full model to

/content/drive/MyDrive/AO_Spelling_Checker/LSTM/LSTM_Models/seq2seq_T1_100.h5

References

[1] Teferi Degeneh Bijiga. "The Development of Oromo Writing System". PhD thesis. University of Kent, 2015.

[2] Said Samatar. "THE JOURNAL OF OROMO STUDIES". In: ().

[3] Fikadu Belda and Fanose Mengistu. "Orthographic Interference Errors from Afan Oromo to English: The Case of Selected Schools of East and West Hararge Zones, Grade Nine Students". In: *East African Journal of Social Sciences and Humanities* 6.1 (2021), pp. 59–76.

[4] Gurjit Kaur, Kamaldeep Kaur, and Parminder Singh. "Spell Checker for Punjabi Language Using Deep Neural Network". In: *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE. 2019, pp. 147–151.

[5] Barbara Kitchenham. "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26.

[6] Pravallika Etoori, Manoj Chinnakotla, and Radhika Mamidi. "Automatic spelling correction for resource-scarce languages using deep learning". In: *Proceedings of ACL 2018, Student Research Workshop*. 2018, pp. 146–152.

[7] Kathleen M Chen et al. "Selene: a PyTorch-based deep learning library for sequence data". In: *Nature methods* 16.4 (2019), pp. 315–318.

[8] Johan Gudmundsson and Francis Menkes. *Swedish Natural Language Processing with Long Short-term Memory Neural Networks: A Machine Learning-powered Grammar and Spell-checker for the Swedish Language*. 2018.

[9] Wubetu Barud Demilie. "Dictionary Based Spelling Corrector System: The Case of Six Ethiopian Languages". In: *International Journal of Future Generation Communication and Networking* 13.4s (2020), pp. 1759–1776.

[10] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. "A review on the long short-term memory model". In: *Artificial Intelligence Review* 53.8 (2020), pp. 5929–5955.

[11] Chanjun Park, Chanhee Lee, and Heuiseok Lim. "Comparison of the Evaluation Metrics for Neural Grammatical Error Correction With Overcorrection". In: *IEEE Access* PP (May 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2998149.

[12] Menno Zaanen and Gerhard Van Huyssteen. "Various Uses of a Spelling Checker Project: Practical Experiences, Teaching, and Learning". In: *Southern African Linguistics and Applied Language Studies* 21 (Nov. 2009). DOI: 10.2989/16073610309486352.

[13] Gerhard B Van Huyssteen, E Roald Eiselen, and Martin J Puttkammer. "Re-evaluating evaluation metrics for spelling checker evaluations". In: *Proceedings of First Workshop on International Proofing Tools and Language Technologies*. 2004, pp. 91–99.

[14] Hsuan Lorraine Liang et al. "Spell checkers and correctors: A unified treatment". PhD thesis. University of Pretoria, 2009.

[15] Shashank Singh and Shailendra Singh. "Systematic review of spell-checkers for highly inflectional languages". In: *Artificial Intelligence Review* 53.6 (2020), pp. 4051–4092.

[16] Karen Kukich. "Techniques for automatically correcting words in text". In: *Acm Computing Surveys (CSUR)* 24.4 (1992), pp. 377–439.

[17] Gaddisa Olani Ganfure and Dida Midekso. "Design And Implementation Of Morphology Based Spell Checker". In: *vol* 3 (2014), pp. 118–125.

[18] Fred J Damerau. "A technique for computer detection and correction of spelling errors". In: *Communications of the ACM* 7.3 (1964), pp. 171–176.

[19] Robert A Wagner. "Order-n correction for regular languages". In: *Communications of the ACM* 17.5 (1974), pp. 265–268.

[20] Vladimir I Levenshtein et al. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics doklady*. Vol. 10. 8. Soviet Union. 1966, pp. 707–710.

[21] Thomas H Cormen et al. "Introduction to algorithms second edition". In: *The Knuth-Morris-Pratt Algorithm* (2001).

[22] Richard Bellman. "Dynamic programming treatment of the travelling salesman problem". In: *Journal of the ACM (JACM)* 9.1 (1962), pp. 61–63.

[23] Ramon D Faulk. "An inductive approach to language translation". In: *Communications of the ACM* 7.11 (1964), pp. 647–653.

[24] Victoria J Hodge and Jim Austin. "A comparison of standard spell checking algorithms and a novel binary neural approach". In: *IEEE transactions on knowledge and data engineering* 15.5 (2003), pp. 1073–1081.

[25] Abebe Abeshu. "Analysis of Rule Based Approach for Afan Oromo Automatic Morphological Synthesizer". In: *Science, Technology and Arts Research Journal* 2.4 (2013), pp. 94–97.

[26] Asanilta Fahda and Ayu Purwarianti. "A statistical and rule-based spelling and grammar checker for Indonesian text". In: *2017 International Conference on Data and Software Engineering (ICoDSE)*. IEEE. 2017, pp. 1–6.

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] Sepp Hochreiter et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[29] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[30] Shaona Ghosh and Per Ola Kristensson. "Neural networks for text correction and completion in keyboard decoding". In: *arXiv preprint arXiv:1709.06429* (2017).

[31] Shashank Singh and Shailendra Singh. "HINDIA: a deep-learning-based model for spell-checking of Hindi language". In: *Neural Computing and Applications* 33.8 (2021), pp. 3825–3840.

[32] Damar Zaky and Ade Romadhony. "An LSTM-based spell checker for indonesian text". In: *2019 International Conference of Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*. IEEE. 2019, pp. 1–6.

[33] S Sooraj et al. "Deep learning based spell checker for Malayalam language". In: *Journal of Intelligent & Fuzzy Systems* 34.3 (2018), pp. 1427–1434.

[34] Zijia Han et al. "Chinese spelling check based on sequence labeling". In: *2019 International Conference on Asian Language Processing (IALP)*. IEEE. 2019, pp. 373–378.

[35] A Cumhur KINACI. "Spelling Correction Using Recurrent Neural Networks and Character Level N-gram". In: *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. IEEE. 2018, pp. 1–4.

[36] Jon Degenhardt et al. "ECOM'19: The SIGIR 2019 Workshop on eCommerce". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2019, pp. 1421–1422.

[37] Licia Sbattella and Roberto Tedesco. "How to simplify human-machine interaction: A text complexity calculator and a smart spelling corrector". In: *Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good*. 2018, pp. 304–305.

[38] Jason PC Chiu and Eric Nichols. "Named entity recognition with bidirectional LSTM-CNNs". In: *Transactions of the association for computational linguistics* 4 (2016), pp. 357–370.

[39] Sina Ahmadi. "Attention-based encoder-decoder networks for spelling and grammatical error correction". In: *arXiv preprint arXiv:1810.00660* (2018).

[40] MELAKU TILAHUN. "AUTOMATIC SPELLING CHECKER FOR AMHARIC LANGUAGE". PhD thesis. 2020.

[41] Andargachew Mekonnen Gezmu, Andreas Nürnberger, and Binyam Ephrem Seyoum. "Portable spelling corrector for a less-resourced language: Amharic". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[42] BERIHUN HADIS. "SPELL CHECKER FOR TIGRIGNA LANGUAGE USING RULE BASED MORPHOLOGICAL ANALYZER AND UNSUPERVISED APPROACH". PhD thesis. 2020.

[43] Lul Farah Abdullahi. "Spell Checker for Somali Language Using Knuth-Morris-Pratt String Matching Algorithm". In: *Recent Trends in Data Science and Soft Computing: Proceedings of the 3rd International Conference of Reliable Information and Communication Technology (IRICT 2018)*. Vol. 843. Springer. 2018, p. 249.

[44] Workineh Tesema and Duresa Tamirat. "Investigating Afan Oromo Language Structure and Developing Effective File Editing Tool as Plug-in into Ms Word to Support Text Entry and Input Methods". In: *American Journal of Computer Science and Engineering Survey* (2017).

[45] Yehuwalashet Bekele Tesema. "Hybrid Word Sense Disambiguation Approach for Afaan Oromo Words: published Master's Thesis". In: *Department of Computer Science, Addis Ababa University, Addis Ababa, Ethiopia* (2016).

[46] Tilahun Gamta. "Qube Afaan Oromo: Reasons for Choosing the Latin script for Developing an Oromo Alphabet". In: *Journal of Oromo Studies* 1.1 (1993).

[47] R David Zorc. "SUGGESTIONS AND OVERALL PRINCIPLES FOR THE DEVELOPMENT OF AN OROMO DICTIONARY". In: ().

[48] Wakweya Olani. *Inflectional morphology in Oromo*. 2017.

[49] Debela Tesfaye. "Designing a Stemmer for Afaan Oromo Text: A Hybrid Approach". PhD thesis. Addis Ababa University, 2010.

[50] MultiMedia LLC. *NLP: Word Embedding Techniques for Text Analysis*. 2020. URL: https://medium.com/sfu-cspmp/nlp-word-embedding-techniques-for-text-analysis-ec4e91bb886f (visited on 06/04/2022).

[51] Allen Chieng Hoon Choong and Nung Kion Lee. "Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method". In: *2017 International Conference on Computer and Drone Applications (IConDA)*. 2017, pp. 60–65. DOI: 10.1109/ICONDA.2017.8270400.

[52] Charles R Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020), pp. 357–362.

[53] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.