



**Setswana Grammar Checker for Declarative Sentences using LSTM-Recurrent
Neural Networks**

by

Kesegofetse Amon

Reg. No: 16100671

BSc (Hons) (Computer Systems Engineering) (University of Sunderland)

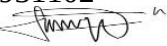
Department of Computer Science and Information Systems

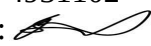
Faculty of Sciences

Botswana International University of Science and Technology

kesegofetse.amon@studentmail.biust.ac.bw, (+267) 76060345

A Dissertation Submitted to the College of Sciences in Partial Fulfilment of the
Requirements for the Award of the Degree of Master of Science in Computer
Science of BIUST

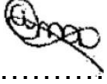
Supervisor: Keletso Letsholo,
Ph.D
Department of Computer Science
and Information Systems
BIUST
letsholok@biust.ac.bw
Tel: +267 4931102
Signature: 
Date: 11/11/21.....

Co-Supervisor: Dimane
Mpoeleng, Ph.D
Department of Computer Science
and Information Systems
BIUST
mpoelengd@biust.ac.bw
Tel: +267 4931102
Signature: 
Date: 11/11/21.....

November, 2021

Declaration and Copyright

I Kesegofetse Amon, declare that this dissertation is my own original work and it has not been presented and will not be presented to any other university for similar degree award.

Signature  11/11/21.....

This dissertation is copyright material protected under the Berne Convention, the copyright Act of 1999 and other international and national enactments, in that behalf, on intellectual property. It must not be reproduced by any means, in full or in part, except for short extracts in fair dealing; for researcher private study, critical scholarly review or discourse with an acknowledgement, without the written permission of the office of the Provost, on behalf of both the author and BIUST.

Certification

The undersigned certifies that he has read and hereby recommends for acceptance by the College of Sciences titled: Setswana Grammar Checker for Declarative Sentences using LSTM Recurrent Neural Networks, in fulfilment of the requirements for the degree of Masters of Science (Computer Science) of the BIUST.



Dr K. Letsholo

(Supervisor)

Date: ...11/11/21



Dr D. Mpoeleng

(Co-Supervisor)

Date: ...11/11/21...

Table of Contents

Declaration and Copyright	i
Certification.....	ii
Table of Contents	iii
Abstract	vii
LIST OF ABBREVIATIONS	viii
Chapter 1 - Introduction.....	9
1.1 Introduction	9
1.2 Problem Statement	11
1.3 General Objective	11
1.4 Specific Objectives.....	12
1.5 Research Questions.....	12
1.6 Contributions.....	12
1.7 Justification of the Research Study	13
1.8 Dissertation Structure.....	14
Chapter 2 - Literature Review	15
2.1 Introduction	15
2.2 Motivation of Study.....	15
2.3 Literature Review	15
2.3.1 Syntax-based Approach.....	16
2.3.1.1 Syntax-based Latvian Grammar Checker.....	16
2.3.1.2 Syntax-based Grammar Checker for English	17
2.3.2 Rule-based Approach	18
2.3.2.1 Rule-based Grammar Checker for Afan Oromo	18
2.3.2.2 Swedish rule-based grammar checker; Grammatifix	19
2.3.3 Statistical-based Approach	19

2.3.3.1	N-based statistical grammar checker for Bangla and English.....	19
2.3.3.2	RNN-Based English Grammar Checker	20
2.3.3.3	BI-LSTM Neural Network Chinese Grammar Checker	21
2.3.4	Hybrid Grammar Checkers	23
2.3.4.1	Swedish grammar checker-Granska.....	23
2.3.4.2	CoGrOO: A Brazilian-Portuguese Grammar Checker.....	23
2.3.5	Grammar Checking Approach by each Technique	24
2.3.5.1	Syntax-based Approach	24
2.3.5.2	Rule-based Approach.....	26
2.3.5.3	Statistical-based Approach.....	27
2.3.6	Comparison of the Grammar Checkers	30
2.3.6.1	Training and Accuracy	30
2.3.6.2	Scalability	30
2.3.6.3	Learning Ability	31
2.3.6.4	Implementation and Application of Grammar Checkers.....	31
2.3.6.5	Performance Metrics used in the Evaluation of Grammar Checkers	32
2.4	Chapter Summary	35
Chapter 3 - Methodology and the Proposed Setswana Grammar Checker.....		36
3.1	Introduction	36
3.2	Setswana Linguistic Resources	36
3.2.1	Setswana Language and Grammar	36
3.2.2	Setswana Grammar Detection and Correction Approaches	38
3.2.3	Setswana Corpus and its Annotation	39
3.3	Experimental setup and Data History.....	40
3.3.1	Data Collection.....	40
3.3.2	Data pre-processing and Exploratory Analysis	40
3.4	LSTM-RNN Model and its Performance Metrics	41

3.4.1	LSTM-RNN Mathematical Theory and its Architecture	41
3.4.2	LSTM-RNN Implementation Algorithms and Procedure.....	44
3.4.3	Systems Configuration and Hyper Parameters Tuning	45
3.4.3.1	Integrated Development Environment Setup	45
3.4.3.2	Data Loading	46
3.4.3.3	Data Processing	46
3.4.3.4	LSTM-RNN Model	46
3.4.4	Employed Performance Metrics	48
3.4.5	Systems Integration	49
3.4.5.1	Systems API	49
3.4.5.2	Systems Implantation and its Deployment.....	49
Chapter 4 –Results and Analysis		51
4.1	Introduction	51
4.2	Model Hyper Parameters	51
4.2.1	Results of the Model (LSTM-RNN) without Tuning Hyper Parameters	51
4.2.2	Results of the Model (LSTM-RNN) with Tuned Hyper Parameters	52
4.3	Results Obtained from Performance Metrics.....	52
4.3.1	Binary cross entropy loss function	52
4.3.2	Classification accuracy	53
4.4	Results with Existing Models.....	53
4.5	Chapter Summary	54
Chapter 5 - Conclusions and Recommendations		55
5.1	Introduction	55
5.3	Summary of the Study.....	56
5.4	Recommendation for Future Work	57
5.5	Challenges	58
5.6	Contributions to the Body of Knowledge	58

References.....	59
Appendix 1: The Setswana Grammar Checker	66

Abstract

This research is aimed at developing a Setswana grammar checker for Setswana declarative sentences using Long Short-Term Memory Recurrent neural networks (LSTM-RNNs). The research was motivated by the fact that Setswana is recognized as one of the under-resourced languages in the world and the language lacks Natural language processing (NLP) tools such as grammar checkers; this delays the language's technological progress. A Setswana grammar checker is a pre-requisite to the development of other Human Language (HTL) applications such as machine translators and parsers that are necessary for the language to exist in the web, hence contributing to the language's technological progress or improvement. Various techniques have been implemented to develop grammar checkers for different languages. These techniques include the rule-based approach, but the downfall associated with this technique is that it is language-specific and many rules have to be developed to satisfy all the grammatical rules available in that specific language; this may be tedious and time-consuming. Another technique is the syntax-based approach, and the disadvantage associated with this approach is that it depends on the availability of a language parser. This research implements the statistical-based approach to grammar checking. The grammar checker in this research is developed using Long Short-Term Memory Recurrent neural networks (LSTM-RNNs). The advantage of this technique lies in the fact that it enables the development of language-independent grammar checkers and the developer does not need to have deep knowledge of the underlying grammar of the language they are working with. The Setswana grammar checker was developed by the use of 1700 Setswana sentences; 750 incorrect sentences and 750 correct sentences. The training module had a Validation accuracy of 0.95, a Validation loss of 0.05, and a Training loss of 0.1. The testing module had a testing accuracy of 0.96. and a testing loss of 0.06. Results of this study indicate that Long Short-Term Memory Recurrent neural networks (LSTM-RNNs) can extract the pattern or word order followed by Setswana sentences and use this information to determine the grammatical correctness of Setswana text as compared to the rule and syntax-based grammar checking techniques.

LIST OF ABBREVIATIONS

NLP- Natural Language Processing WWW- World Wide Web

HLT- Human Language Technologies RNN- Recurrent Neural Networks

LSTM-RNN- Long Short Term Memory Recurrent Neural Networks

POS- Part of Speech

CFGs - Context Free Grammars

CYK - Cocke-Younger-Kasami algorithm BCELoss- Binary Cross Entropy Loss Function

Chapter 1 - Introduction

1.1 Introduction

Jones et al [1] stated that the development of Human Language Technologies (HTLs) over the last 40 years has indeed revolutionized communication between computers and humans. HLTs do not only enable human-computer interaction, they also permit humans to use computers in a more natural way and in their mother language. Deksné et al [2] mentioned that current research revolves around the development of HTL applications such as spelling and grammar checkers, machine language translators, speech recognition, text summarization and information extraction software. The authors further stressed that grammar checkers have gained popularity over the years and that grammar checkers help improve the accuracy of natural language understanding between computers and humans. Therefore, the findings above indicate the necessity of grammar checkers in resource-scarce languages such as Setswana. According to Alama et al [3], grammar checkers are usually integrated into common word processors such as Microsoft Office word processor and text editors, the authors further mention that stand-alone grammar checkers have also been developed and that some grammar checkers exist as plugins on common web browsers to verify written text for grammatical correctness, hence producing syntactic error-free textual content. Park et al [4] argues that amongst all these grammar checkers the Microsoft Office's word English grammar checker is the commonly used grammar checker. This grammar checker has been used as a plugin to Microsoft Office's word processor and Email Outlook. Manchanda et al [5] stated that the three most common techniques that have been implemented in the development of grammar checkers include the rule-based approach, syntax-based approach and statistical-based approach, however, the authors further stressed that hybrid approaches that make use of a combination of the aforementioned techniques also exist.

Currently no grammar checker for Setswana exists. Setswana is the National language of Botswana spoken by almost 78.2% of the people in Botswana, Pretorius [6]. Setswana is recognized as one of the under-resourced languages in the world; Jones et al [21] mentioned that under-resourced languages are languages that have a partial or no presence at all in the World Wide Web (WWW); they lack the resources necessary for the implementation of Human

Language Technologies (HTLs). The authors further mentioned that the internet has become an essential part of our day to day lives and that proofing tools such as grammar checkers are needed in order for languages to have an existence in the web. According to Deksne et al [2], grammar checkers help correct syntax errors in written text hence improving the accuracy of natural language understanding between computers and humans. Arrpe [7] stated that languages differ in morphological structure therefore grammar checkers developed for a specific language cannot be applied on a different language.

Manchanda et al [5] stated that various techniques have been adopted in the development of grammar checkers and these include; the rule-based approach, however, the downfall associated with this approach is the fact that rule-based models may require a lot of manual work to cover all types of grammar rules present in all the available sentence types and that the difficulty of language grammar grows exponentially as we try to resolve different kinds of errors. The authors further mentioned that another possible option in the development of grammar checkers is the syntax- based approach; however, the authors stressed that the disadvantage associated with this technique is that robust and vigorous parsers with satisfactory linguistic rules are not present for most resource-scarce languages like Setswana. Additionally, the authors mentioned that the statistical-based approach is another technique that can be used in the development of grammar checkers; the strength associated with this approach is that the developer does not necessarily have to have deep knowledge of language grammar.

Mesnil [8] stressed that recurrent neural networks as a statistical-based approach have been victoriously applied in the field of Natural language processing (NLP) because of their ability to capture word order information; they have been used in the development of grammar checkers. Mesnil [8] further mentioned that artificial neural networks have gained popularity because they are language-independent, meaning they can work with any language. Supporting the author above, Lawrence et al [9] stated that different techniques have been implemented to develop grammar checkers for different languages and each grammar checker has a unique success rate, on overall recurrent neural networks have shown remarkable performances. However, it is vital that robust grammar checkers with high classification accuracies are produced to ensure that quality texts are produced.

1.2 Problem Statement

Research has been done on the development of grammar checkers for various developed languages but currently, there no grammar checker exists for the Setswana language. The grammar checkers that currently exists for other languages cannot be used to check for grammatical correctness in Setswana because languages differ in morphological structure, therefore, grammar checkers developed for specific languages cannot be applied to a different language. Cole [10] states that similar to other Bantu languages, Tswana varies drastically in structure from European languages. Supporting the sentiments above Berg et al [11] states that Setswana is a semi-disjunctively written agglutinating language that is characterized by a complex morphology and that the language has a rich system of verbal inflections. Currently, English grammar checkers that adopt RNN architectures exist, these statistical grammar checkers make use of the Elman, Williams and Zipser (W&Z) recurrent network architectures. However, these grammar checkers suffer from the vanishing gradient problem meaning these grammar checkers cannot remember past data with longer sequences in the memory of the RNN. On the contrary, grammar checkers that implement the Long Short-Term Memory (LSTM) Recurrent Neural Networks architecture remember information for an extended period of time as opposed to the RNN architectures mentioned above. There currently exists a Chinese grammar checker that was developed using the Bi-LSTM recurrent neural network architecture and this grammar checker has demonstrated remarkable performances in capturing the word order in sentences with longer sequences. This research is aimed at exploring how LSTM-RNNs can be implemented in a different language, specifically Setswana. Another motivation for this research was to fill the technological void in the Setswana language's Natural language pipeline by developing a Setswana grammar checker that implements the LSTM-RNN architecture. Not having a Setswana grammar checker delays the language's technological advancement because a grammar checker is a pre-requisite to the development of other HTL applications such as parsers and machine translators that are necessary for the language to have existence in the web.

1.3 General Objective

To develop a statistical Setswana grammar checker using LSTM-RNNs that classifies Setswana Declarative sentences as grammatical or ungrammatical

1.4 Specific Objectives

- To identify and adopt the LSTM-RNN architecture that will be used to implement the LSTM-RNN based Setswana grammar checker
- To convert the words in the Setswana corpus into short numerical word vector representations via the LSTM-RNN word embedding model
- To train the LSTM-RNN model on grammar classification via the Adam Optimization Training (AOT) algorithm
- To evaluate the grammar checker on learnability, accuracy and memorability

1.5 Research Questions

- Which deep learning techniques have been implemented or adopted in the development of existing grammar checker for various languages? What methods do they use?
- What are the strengths and limitations of each of these deep learning techniques?
- How well do LSTM-RNNs perform when adopted to capture the pattern or word order followed by Setswana sentences and classify Setswana sentences as grammatical or ungrammatical?
- What is the performance change of LSTM-RNNs in grammar checking when applied to Setswana compared to other languages; how do they perform in terms of learnability, accuracy and memorability compared to other languages?

1.6 Contributions

The contribution to the body of knowledge of computer science from this research is the development of a Setswana grammar checker that adopts the LSTM-RNN architecture for grammar classification. Deksne et al [2] stated that grammar checkers help correct syntax errors in written text, hence improving the accuracy of natural language understanding between computers and humans. Alama et al [3] further stated that grammar checkers are typically integrated into common word processors and text editors. In addition, the authors mentioned that some grammar checkers exist as plugins on common web browsers to verify written text for grammatical correctness, hence producing syntactic error-free textual content.

1.7 Justification of the Research Study

According to Tesfaye [12] proofing tools such as grammar checkers play a vital role in document compilation. Grammar checkers help highlight syntax errors in text hence producing error-free content. Grammar checkers have been designed and developed for other languages but there exists no grammar checker for the Setswana language. Manchanda et al [5] state that statistical-based grammar checkers are easier to produce as compared to rule-based techniques mainly because;

- With the statistical-based approach, the grammar rules do not have to be manually constructed as opposed to the grammar rules in the rule-based approach.
- The grammar rules used in statistical-based grammar checkers are learnt by the machine from the annotated corpus; the corpus is tagged or labelled with Part of Speech (POS) tags.
- Furthermore, the authors state that the statistical-based approach has always proved to be most suitable especially for resource-scarce languages like Setswana that currently do not have any existence on the web.

Studies conducted by Mesnil [8] have also indicated that artificial neural networks work best for languages with a complex morphology in natural language processing tasks. On the other hand, De Mulder et al [13] state that LSTM-RNNs have shown remarkable performances over other RNN architectures specifically in the task of sequence modelling. This is because LSTM-RNNs do not suffer or experience the vanishing gradient problem. The authors further mention that LSTM- RNNs can capture word order information and retain in their memory for a longer time; they can handle arbitrarily long contexts. Jones et al [1] further state that the development of natural language processing tools such as morphological analysers, grammar and spelling checkers for less-resourced languages serve as a starting point for the development of a fully fleshed language parser. This study will be focusing on the development of a Setswana grammar checker using LSTM-RNNs. Grammar checking will be limited to

- Identifying grammatical errors in simple Setswana Declarative sentences; the grammar checker will be limited to classifying the given Setswana Declarative sentences as either grammatically correct or wrong
- Identifying only word ordering errors in Setswana sentences; sentences that do not conform to the word order or pattern followed by Setswana sentences will be considered incorrect

and those that conform to the word order followed by Setswana sentences will be considered correct.

1.8Dissertation Structure

The dissertation is organized into five chapters, i.e. the Introduction, Literature review, Research methodology, Evaluation and Conclusion & Recommendations

- Chapter 1 introduces the research. The chapter clearly specifies the motive behind conducting this study. This Chapter consists of the Introduction, Problem Statement, Research Objectives, Research Questions, Justification of the Study, Research Contribution and finally the structure of the dissertation.
- Chapter 2 lays down the literature review; this chapter explores existing literature in relation to the various techniques adopted in the development of grammar checkers. Conference papers and journals are critically revised and analysed. Each of the discovered grammar checking techniques was analysed based on the following factors; the grammar checking approach adopted by each technique, implementation, learnability, scalability, training, and accuracy.
- Chapter 3 specifies the research methodology adopted in this research. The Design Science framework was adopted when developing the LSTM-RNN based Setswana grammar checker. Using the above-mentioned framework as a guide, the following are discussed; Setswana linguistic resources, experimental setup and data history, LSTM-RNN model and its performance metrics and Systems Integration.
- Chapter 4 is the Results and Analysis Chapter. This chapter discusses the results of the LSTM-RNN model before and after turning hyper parameters together with the results obtained from performance metrics. The chapter goes on to evaluate the results of the grammar checker against existing grammar checking models.
- Chapter 5 is the conclusion of the study. The chapter indicates the conclusion of the study and a summary of how the defined objectives were attained or realized. Lastly, the chapter indicates future work research areas that can be conducted to advance this study.

Chapter 2 - Literature Review

2.1 Introduction

This chapter explores existing literature in relation to the development of grammar checkers. Various grammar checkers were explored and each grammar checker was evaluated based on the following; grammar checking technique adopted in the development of the grammar checker, its implementation, learnability, scalability, training, and accuracy. The chapter also explores the performance metrics used to evaluate NLP tools. Before discussing the factors above, the motivation for the study is given.

2.2 Motivation of Study

Botswana is a landlocked country in Southern Africa with an estimated population of 2,351,627. The National language spoken in Botswana is Setswana. 78.2% of the population speaks Setswana, Pretorius [6]. Setswana is used as a national language. It is also included as a subject offered in the school curriculum. Furthermore, the media relays information to the public in Setswana through sources like newspapers and magazines, Otlogetswe [14]. While writing text in Setswana, it is possible that people might make errors. Grammar checking by humans may prove to be inconvenient, especially when dealing with large documents, hence there is a need to automate the process of grammar checking, Manchanda et al [5]. A Setswana grammar checker would enable the automatic detection of errors in Setswana texts.

2.3 Literature Review

Manchanda et al [5] state that various techniques have been used for the development of grammar checkers. These techniques include the syntax-based technique, rule-based technique and statistical-based technique. Grammar checkers that have been developed for different languages based on the three techniques mentioned above are discussed.

2.3.1 Syntax-based Approach

2.3.1.1 Syntax-based Latvian Grammar Checker

Deksne et al [2] developed a syntax-based Latvian grammar checker. The problem statement that this paper is addressing is the deficiencies found in the first Latvian grammar checker. Latvian is a highly inflected language with a high morphological ambiguity. There are various long-distance agreements between words and phrases in a sentences for which there is a need for deep syntactic analysis of phrases and sentence to find potential errors. The grammar checker developed by Deksne et al [2] covers 200 language rules and its deficiencies include the grammar checker's inability to describe long distance errors and errors that describe complex syntactic structures. It only approved of pattern matching near words. In addition, the pattern matching algorithm employed by the grammar checker was slow and the addition of new grammar rules affected the speed of the grammar checker. This grammar checker is aimed at addressing the problems stated above. The implementation and development of this Latvian grammar checker is based on a parser that operates with two sets of rules i.e. rules that describe the Latvian grammar, e.g. rules concerned with correct syntactic structures (G rules) and rules that describe or focus on grammar errors (E rules). The grammar checker solely functions with two grammars. The first grammar encompasses 260 rules that describe improper capitalization patterns in expressions or phrases and style errors. The second grammar comprises a set of 477 rules that define syntactically correct grammar construction rules and 237 error rules. The errors are categorized within 21 categories based on the present error types. The parser used makes use of CFG grammar and it was developed based on the CYK algorithm; the CYK algorithm permits partial parsing if the sentence cannot be fully parsed. Firstly, the input text is split into sentences in the tokenizer module. Token types are then allocated to every token object i.e. abbreviation, word, punctuation, or numeric in each sentence. Simple formatting or composition errors are then identified using regular expressions in the error location module. Then the analyzer module incorporates morphological analysis to every token. The parser component then executes parsing based on a given rule set. Error trees from the parse result matrices are extracted by the parse walker component then suggestions for error fixing are generated. The results from the simple error locator and parse walker component are forwarded to the result preparation module which combines results and returns to a calling application. Two sets of corpora were used for performance testing; firstly, the development corpus was used for setting the rules that were to be followed during the error detection process and the test corpus

was used to measure the existing quality of the grammar checker. The test corpus contained 4814 sentences and the development corpus contained 9364 sentences. Recall results were only given for the development corpus because the test corpus had not been previously annotated or marked. The recall rate for the developed corpus was 0.290 and the precision rate was 0.833 while the precision rate of the test corpus was 0.710. The downfall associated with this grammar checker is that it does not list all the syntactic rules of the Latvian language therefore some grammatically correct sentences may be labelled as incorrect due to unsuccessful parsing of sentences; this is usually caused by the insufficient coverage of grammatical rules. Due to the ambiguities in natural languages, it is impossible to explicitly list all the syntax rules of a natural language.

2.3.1.2 Syntax-based Grammar Checker for English

Park et al [4] proposed a syntax-based English grammar checker, the grammar checker was specifically designed to detect grammatical errors commonly made by university students when writing English essays. The proposed grammar checker is a client – server system developed for real-time grammar checking. The system comprises a web-based user interface that can be distributed or spread over the internet or other networks. When the client enters text into the real-time sentence detector, the sentence is detected and passed to the stochastic POS tagger; the Stanford Maxent Part of speech tagger. The tagger annotates the input text with suitable tags and generates a stream of tokens for parsing. The parser then generates a parse tree if the parser successfully generates the parse tree then we assume that the input text entered is grammatically correct. On the contrary, if the parser fails to build the parse tree, then further processing is done to the incorrectly built parse tree, and an error report is produced and displayed to the user. A sentence is considered to be correct if the relationship or association found between tokens agree with the acceptable relationships governed by English grammar during syntactic parsing, here in this case Context Free Grammars (CFGs). However, Dekse et al [2] stated that the CFG formalism performs badly on languages that have rich morphological features or structure. The parsing algorithm used was the Cocke- Younger-Kasami (CYK) algorithm. The CYK algorithm parses Context Free Grammars in Chomsky Normal Form (CNF). Park et al [4] stressed that this

algorithm is considered amongst the highest performing algorithms based on worst-case running complexity. The CYK algorithm has a time complexity of $O(n^3)$ for clear grammars and the algorithm does not work with unclear grammars. The grammar checker was implemented to specifically address the grammatical mistakes found in a chosen set of 9 essays. The grammar checker detected the following types of errors; Wrong capitalization, missing fragments and wrong agreement and verb form. The grammar checker was tested on 8 new essays each containing an average of 11 sentences. Amongst them, all grammatical sentences (27 of them) were acknowledged as correct. Three ungrammatical sentences were incorrectly accepted as grammatical. The authors further emphasized that project on the development of this grammar checker is still ongoing and that much focus is on extending the coverage of the predicted mistakes and the performance of the parser.

2.3.2 Rule-based Approach

2.3.2.1 Rule-based Grammar Checker for Afan Oromo

Tesfaye [12] developed a rule-based language grammar checker for Afan Oromo; an agglutinative language that is spoken in Ethiopia. The problem addressed by this research is the non-existence of a grammar checker for Afan Oromo. The language has a unique grammatical structure, and the grammar checkers that are used for other languages cannot be applied or used in this language. The development of this grammar checker will also serve as a contribution to the language's NLP pipeline. The language lacks NLP advanced tools. To develop the Afan Oromo rule-based grammar checker, 123 error detection rules were devised so that the system could detect errors in input texts in the form of paragraphs. The rules were defined based on the grammatical relations between subject and adjective, subject and verb, main verb and subordinate verb in terms of tense, number and gender of the Afan Oromo language. This grammar checker provides correct sentence alternatives after detecting grammatical errors. To check the performance of the grammar checker, a student's graduation thesis of 2011 written in Afan Oromo was used and the grammar checker had a precision of 88.89%, and 80% recall rate. The development of language rules for this grammar checker was only limited to simple sentences but the grammar checker does not cover linguistic rules associated with complex sentences. The grammar checker could be improved by adding more rules; however, this may be tedious and time consuming because it needs a lot of work.

2.3.2.2 Swedish rule-based grammar checker; Grammatifix

Arppe [7] launched a Swedish rule-based grammar checker; Grammatifix, in 1999, the grammar checker was developed at Lingsoft. Arppe [7] developed Grammatifix to address the problems found in the Swedish version of the former Inso's International Proof Reader proofing tools software, developed in cooperation with IBM in the early 1990s. This grammar checker was implemented in the sense that it was only limited to checking noun phrase internal agreements, based on shallow parsing; Grammatifix was developed with the intent of developing a fully-fleshed grammar checker for the Swedish language. In addition to grammar checking, Grammatifix checks on three other kinds of phenomena; convention errors, stylistically marked words, and graphical writing. Constraint Grammar (CG) was used for grammar error detection, so this grammar checker used the CG rule formalism. CG is a framework for shallow syntactic analysis and part-of-speech disambiguation that was proposed by Karlsson (1990). The Swedish rule-based grammar checker covers 43 error type checks; 26 are syntactic in nature, 14 address number, date formatting conventions and punctuation, and 3 cover word specific non-standard stylistic usage. To test for precision, the test data was a 1,000,504-word extract from the Göteborgs-posten 1998, a Swedish newspaper. A precision rate of 70% was achieved. For the recall rate, 87,713-word extract from the same newspaper was acquired and a recall rate of 85% was achieved. The short-coming associated with this grammar checkers is that it cannot detect split compounds and also it cannot be used in real-time, in direct interaction with users mainly because of the underlying grammar checking technique (rule-based technique) adopted in the development of this grammar checker.

2.3.3 Statistical-based Approach

2.3.3.1 N-based statistical grammar checker for Bangla and English

Alam et al [3] developed an N-based statistical language grammar checker for Bangla and English. This paper is aimed at introducing a grammar checker that checks the syntax of both Bangla and English sentences; the N-based statistical grammar checker puts into consideration the n-gram based analysis of words and Part of speech tags to determine if the grammar of the sentence is correct or incorrect. The grammar model makes use of trigram analysis to determine the probabilities of the tag sequences occurring. A threshold of 0 was set and if the probability

is greater than 0 then the sentence is considered to be correct in terms of grammar. In addition, it is considered to be wrong when two or more successive tags cannot be fit together. The model was trained with 5000 words from the Prothom-Alo corpus. To check the performance of the grammar checker, manually tagged English sentences were used to test the grammar checker, and the grammar checker had a classification accuracy of 63%. When testing the grammar checker's performance on manually tagged Bangla sentences, the grammar checker had a classification accuracy of 53.7%. However, the grammar checker had a lower performance when tested on sentences that had been tagged by the automated Bangla POS tagger. The grammar checker had a 38% accuracy. The authors argue that the low performance was mainly a result of the training data used. The corpus that was adopted to train the model, and it did not have wide coverage of syntactic and grammatical rules. The corpus mainly consisted of large compound sentences in the English training set, and the tag set used did not have enough agreement features. The choice of a wrong language model was also one of the factors contributing to the low performance of the grammar checker. A trigram model was chosen, but experimental results indicated that an N-gram model that functions at word level could have worked better. Alam et al [3] further stated that the gram to use for a language is dependent on the average length of the sentences in the language.

2.3.3.2 RNN-Based English Grammar Checker

Lawrence et al [9] have also applied artificial neural networks as a statistical approach to grammar checking. The paper is aimed at training various recurrent neural network architectures to classify English sentences as grammatical or ungrammatical deprived of the classification into learned vs. innate components assumed by Chomsky in an attempt to produce the same judgments as a native speaker in terms of the grammatical correctness of sentences. The recurrent neural networks that were in consideration include the Zipser, Elman and Williams and the W&Z networks. According to Deksne et al [2] RNNs have a powerful computational ability and they also have the representational power found necessary for natural language structures. The findings showed that both Elman and Williams and Zipser (W&Z) recurrent neural networks were able to acquire or pick up the correct, or recommended grammar after executing techniques that refine the convergence of the gradient descent based back-propagation through-time training algorithm. The dataset used when developing this grammar checker consisted of 552 English positive, and 552 negative samples of sentences extracted from an introductory

GBLinguistics textbook by Lasnik and Uriagereka. The raw and untagged data, which include words from the textbook were first categorized. These words were categorized by the use of a readily available parser into the major syntactic classes assumed under GB-theory. Each English word was tagged with its rightful part- of-speech (POS) tag. In preparation for feeding the data to the neural network, the data was prearranged into a fixed length window comprising of categories containing eight separate inputs. These categories corresponded to the classifications of the identified Part of speech tags. Two outputs were defined in the neural networks. These outputs corresponded to grammatical and ungrammatical categorizations of input sentences. To feed the data into the recurrent neural networks, a window was passed over the sentence in chronological sequence from the start to the end of the sentence. The size of the window varied and it was dependent on the length of the sentence. However, much focus was on small input windows. This is mainly because the greater the input window, the greater the ability of a network to appropriately classify the training data without forming or developing a grammar. However, if the input window is relatively small, then the network must learn to store information. The aim of the research was limited to training a recurrent neural network using a small temporal input window. Five simulations were done for each RNN. Each simulation took roughly four hours on a Sun Sparc 10 server. High accuracy was achieved on training the Elman networks. The sequences of the last two words served as input to the network and classification accuracy of 99.6 percent that averaged over five trials on the training data was achieved. The test data resulted in an accuracy of 74.2 percent correct classification on average. W&Z recurrent networks followed with a 91.7% classification accuracy on training data, and 71.3% classification accuracy on test data. Research results indicated that the more powerful and robust Elman and W&Z networks demonstrated high performance amongst other RNNs shows that they were successful in finding the structure or sequence in the data hence learn the correct grammar for classifying between the grammatical or ungrammatical pairs used by GB-linguists. This research was limited to training a recurrent neural network using a small temporal input window and not a large one. These architectures would not yield the same results on longer sentences, hence the need to explore further how the architectures would perform on longer sentences.

2.3.3.3 BI-LSTM Neural Network Chinese Grammar Checker

Huang and Wang [15] developed a Chinese grammar checker; the grammar checker was developed based on the Bidirectional Long Short-Term Memory (LSTM) neural network

architecture. The authors stressed that Chinese grammatical error diagnosis has always been a complex problem because of the morphology of the Chinese language in attempt to solve this problem this paper introduces a BI-LSTM Neural Network Chinese grammar checker which resolves this problem by treating the task of grammatical error detection as a sequence labelling problem. The grammar checker was specifically developed to handle four specific errors; word ordering errors, word selection errors, redundant errors and missing word errors. The Bi-LSTM Chinese grammar checker consists of the following layers; the Embedding layer, Encoding layer and Decoding layer. On detecting grammatical errors in Chinese texts, the grammar checker firstly segments input Chinese sentences into separate words through the CKIP Chinese Segmentation System provided by Taiwan Academia. Each word is then assigned its POS tag, and these tags are assigned using the window tagging approach the tag of each word is mainly determined by its neighbouring words and the context of the word. These POS tagged words are then passed on to the embedding layer. The embedding layer then maps each of the given words into a dimensional embedding space via a lookup table, the result of this are vector representations of each word in the given input sentences. These vectors are then fed to the Encoding layer, which is the BI-LSTM layer; Huang and Wang [15] argue that this layer makes use of two LSTM layers mainly because a single LSTM layer could only make use of previous information while in some cases the grammatical error could be inferred in the words that follow. This problem is solved by introducing the Bi-LSTM layer which comes in the form of a Forward LSTM layer and a Backward LSTM layer. The output from the encoding layer is then fed to the decoding layer which makes use of the sigmoid function that helps to detect whether the input Chinese sentences are erroneous or not based on the types of errors defined above. The sigmoid function makes use of two possible outputs, which are 0 and 1. The number '0' represents incorrect, which simply means that the input sentence is erroneous while 1 represents correct, which implies that the sentence is correct. On evaluating the grammar checker, it had an accuracy of 0.5442 with a precision rate of 0.5701 and a recall rate of 0.5680. The grammar checker's training and testing data was obtained from the NLP-TEA dataset; a total of 30 359 Sentences were used. To improve the precision and recall rate of the model, feature selection should be performed on the input data (Chinese sentences) before feeding them to the embedding layer to further increase the model's performance and accuracy.

2.3.4 Hybrid Grammar Checkers

2.3.4.1 Swedish grammar checker-Granska

Another Swedish grammar checker, Granska was proposed by Domeij et al [19]. This grammar checker was designed to rectify grammatical errors that occur on a split compound and the treatment of agreement errors. There are the two types of errors that commonly occur in Swedish texts, hence addressing the problems found in Grammatifix, an earlier version of the Swedish grammar checker developed in 1999 by Arppe [7]. The system uses a hybrid of probabilistic and rule-based methods to detect errors and suggests corrections for several Swedish texts. The authors emphasized that the combination of these two methods is a requirement for a grammar checker suitable for use in real-time in direct interaction with users than systems that use rule-based parsing alone. The Swedish grammar checker (Granska) is composed of the following components; Tokenizer which is used to split sentences into words and special characters. Tagger is used to assign parts of speech tags to each word; The Granska system uses a hidden Markov model to tag and disambiguate all words in the input text; the input text is a corpus of 160 000 words. The next component is the Error rule component error rules are matched with the text to search for specified grammatical problems. The error rule component also generates error corrections and instructional information about detected problems that are presented to the user in a graphical interface. Furthermore, the system contains a spelling detection and correction module which can handle Swedish compounds. The whole system had about 20 rule types for 250 error rules and processes 2800 words per second. The preliminary tests conducted with the grammar checker indicated that the grammar checker could have a precision rate of above 90% and a recall rate exceeding 50%. The initial preliminary tests above indicate that the system needs to be further optimized to improve the current precision and recall rate.

2.3.4.2 CoGrOO: A Brazilian-Portuguese Grammar Checker

CoGrOO (Grammar Checker for OpenOffice) is a Brazilian-Portuguese grammar checker that was developed by Kinoshita, Salvador and Menezes [16]. The grammar checker handles the task of grammatical error detection by using a hybrid of rule-based and statistical-based techniques for identifying grammatical errors in Brazilian-Portuguese texts. CoGrOO was specifically developed to handle nominal and verbal agreement grammatical errors. The development or formation of the grammar checker was based on the CETENFOLHA

Portuguese-Brazilian corpus; training and testing data were obtained from the CETENFOLHA Portuguese-Brazilian corpus. The grammar checker consists of four modules which are: The Sentence boundary detector, Part of Speech (POS) tagger, and the Grammatical relation finder. Error detector module is embedded in each step of sentence analysis. The Sentence boundary detector is the first module and its function is to split the input text into sentences. After the input text has been split into sentences, the POS tagger assigns morphological POS tags to each word in the sentences; the most likely tag suitable for each word is determined by the context of the word in relation to other words in the defined sentences. The POS tagged sentences are then passed on to the Chunker which identifies and separately groups noun phrases and verbal phrases from the sentences. To achieve this, the Chunker uses patterns or structures extracted from the noun/verbal phrases in the CETENFOLHA corpus. The noun and verbal phrases from the chunker are then submitted or fed to the Grammatical Relation Finder. The Grammatical Relation finder finds grammatical relations between noun and verbal phrases in each of the input sentences e.g. subject to verb or verb to object. This is achieved statistically by automatically extracting patterns from the initially tagged input sentences in the corpus; a pattern that repeatedly occurs many times is considered valid. After identifying grammatical relations, the structural error checker module then checks errors involving these relations; any errors found are highlighted in the text. CoGroo had a precision rate of 60.03% and a recall rate of 14.41% when tested on the PROBI corpus. The grammar checker also had a precision rate of 44.12% and a recall rate of 28.30% when tested on the Metro corpus. Results indicate that the grammar checker has a lot of false positives this is due to the fact that error rules defined to identify grammatical errors in sentences are not complete, this indicates the need for improvement.

2.3.5 Grammar Checking Approach by each Technique

2.3.5.1 Syntax-based Approach

Manchanda et al [5] state that the development of a syntax-based parser is entirely dependent on the availability of a parser. Alam [3] states that with this approach full parsing is executed on a given text. The commonly used parsing algorithm is the Cocke-Younger-Kasami (CYK) algorithm. This algorithm passes Context Free Grammars in Chomsky Normal Form (CNF). The CYK algorithm starts by populating a triangular table with appropriate grammar entries and then works in a bottom up fashion to populate the table. The lowest row has strings on length 1, and

the next row has strings of length 2 and so on. If the uppermost row has a Start variable from the designed CFGs in it, then it shows that the specified sentence can be generated or created by the grammar. If the top row does not have the Start symbol, then it indicates that the String cannot be formed or generated by the given grammar. After parsing, each sentence is allocated a tree structure based on the grammatical structure of the underlying language. The text is considered incorrect if complete parsing is not achieved. Manchanda et al [5] stressed that in order to decrease the number of false alarms, the parser must be as complete as possible. The strength of this technique lies in the fact that errors in ungrammatical sentences can be simply recognized based on the constraints which are relaxed throughout the parsing of the sentence. However, the disadvantage of this technique is the fact that robust parsers with satisfactory linguistic rules are not present for most resource scarce languages. The following flow chart (Figure 2.1) shows the algorithm of the syntax-based approach in grammar checking:

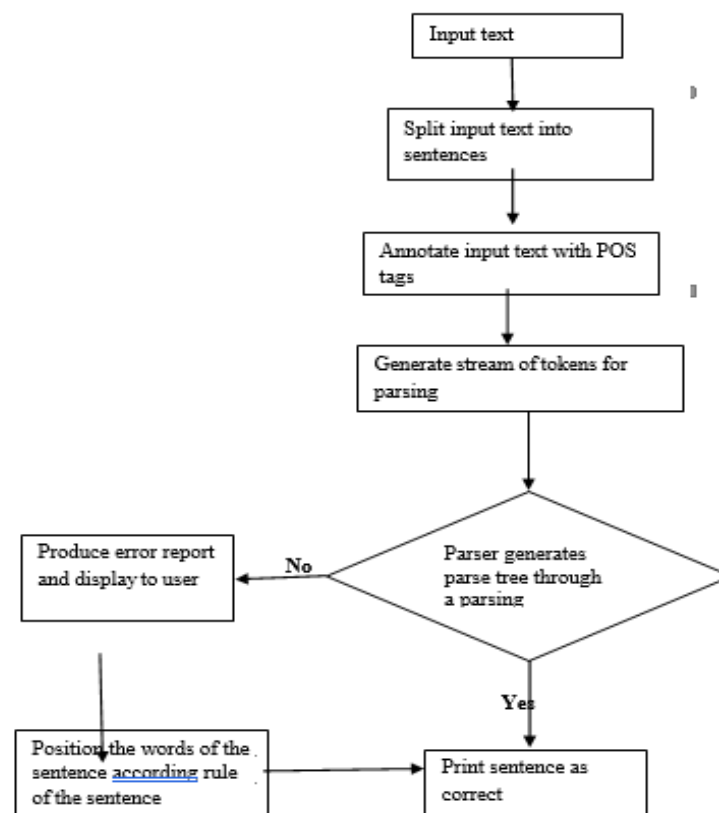


Figure 2.1.: Algorithm for syntax-based checking approach

2.3.5.2 Rule-based Approach

Manchanda et al [5] state that with the rule-based technique error detection rules to be used in grammar checking are manually constructed based on the morphology of the language. These rules are then used to detect grammatical errors in the text that has been POS tagged. The text is considered flawed if a match of any of the predefined rules is found. These rules often contain suggestions or recommendations on how the errors encountered can be rectified. Sharma [17] stressed that, unlike syntax-based grammar checkers, a rule-based model will always be incomplete. In addition, the author supported this by mentioning that even if numerous rules are defined, there will always be some errors that are hard to find, and it is not easy to foresee all the grammatical irregularities. However, the author further mentions that one of the drawbacks associated with a syntax-based model is the fact that an incomplete parser may raise annoying false alarms. Manchanda et al [5] further stressed that rule-based models may require a lot of manual work to cover all types of grammar rules present in all the available sentence types and that complexity of language grammar increases exponentially as we try to resolve various types of errors. Sharma [17] states that the rule-based approach can be advantageous over the corpus based approach when it comes to less-resourced languages for which large corpora is probably not present and for languages with a rich morphology, which even with the presence of corpora suffer from data scarceness or sparseness. Figure 2.1 below presents the algorithm for rule-based checking approach;

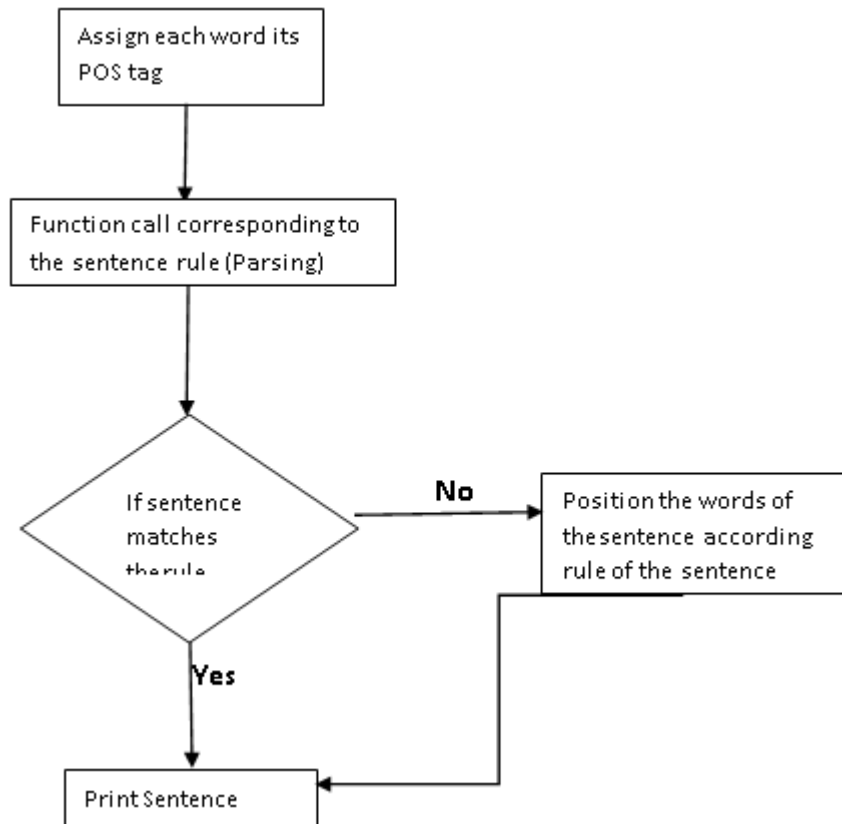


Figure 2.2: Algorithm for rule-based checking approach

2.3.5.3 Statistical-based Approach

Manchanda et al [5] state that in this approach, Part-Of-Speech tags are extracted from an annotated corpus. The machine then has to learn the pattern or sequence followed by these tags. The sequences that frequently occur are the ones that are considered correct whereas the sequences that do not occur at all are considered erroneous. Under this sub-heading we look at two main statistical-based grammar checking approaches, mainly the N-gram analysis and Neural Network approach to grammar checking.

- N-gram analysis Approach

Rahman et al [18] further mention that one of the commonly used statistical approaches to grammar checking is N-gram analysis, the N-gram analysis algorithm is applied on POS tagged words of a sentence. A specific language model has to be chosen based on the length of the sentences. The algorithm then uses the chosen language model to determine the likelihood of

the tag sequence in the sentence and finally decides on the grammatical correctness of the sentence based on the likelihood of the tag sequence. If the probability is above the set threshold, then the sentence is classified as grammatically correct. The trigram model looks at the two previous tags to decide on the likelihood of the tag sequence. Manchanda et al [5] mention that when using this approach, it is vital that the POS tags reflect the grammatical properties essential for testing agreement in the underlying language. Figure 2.3 below shows the algorithm for n-gram grammar checking approach;

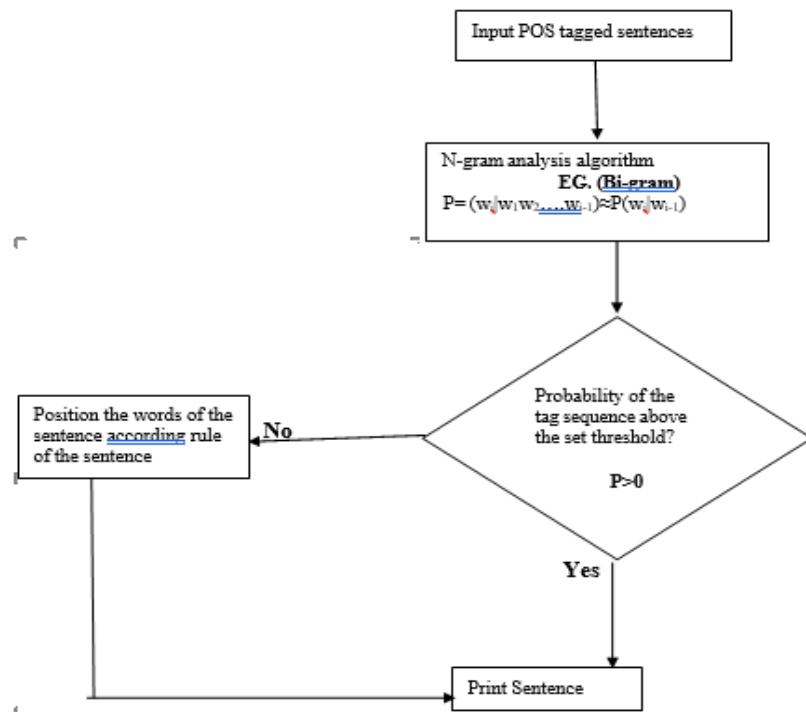


Figure 2.3 : Algorithm for n-gram grammar checking approach

- Recurrent Neural Networks Approach

According to Huang and Wang [15], with recurrent neural networks words from the sentence are fed into the network through the input layer. These words pass through the hidden layer for processing before they reach the output layer. The storage of the sequential order of words is achieved by ensuring that at each time step, a combination of the input data and the hidden layer is added to the previous time step recursively. The output from the network is then compared with the test data to get the error rate; this is achieved by the error function. After the

error rate is obtained, the back-propagation training algorithm is used to modify the initial weights of the network. Adjusting the weights is a technique that helps the neural network learn better. In other words, by adjusting the weights we are training the network on what it is expected to learn i.e. the correct sequence that the words should follow. According to Manchanda et al [5], one of the drawbacks associated with statistical based approaches is the fact that the results from their computations are always hard to interpret. There are no specific error messages given as a justification of why a false alarm may have been given for a specific input. Another drawback of this approach is that a threshold has to be set to separate the unusual but correct constructs from unusual and incorrect ones. However, Manchanda et al [5] also stressed that as opposed to the rule-based approach, the advantage of the statistical based approach lies in the fact that the system developer does not necessarily have to have deep knowledge of grammar. The authors further mention that statistical-based approaches are language independent, and as a result, language independent grammar checkers can be successfully developed. Figure 2.4 below shows the algorithm for the recurrent neural network grammar checking approach;

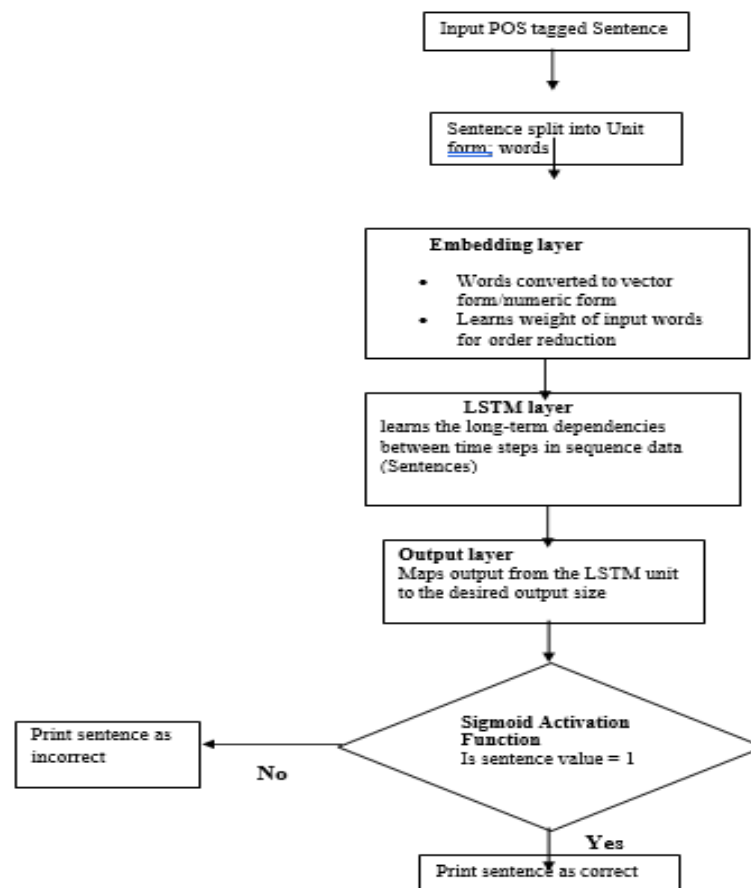


Figure 2.4 : Algorithm for the recurrent neural network grammar checking approach

2.3.6 Comparison of the Grammar Checkers

2.3.6.1 Training and Accuracy

According to Feng [19], statistical based systems have demonstrated more accuracy as compared to rule-based systems, especially when dealing with big data. The velocity, complexity and variety of data makes it nearly impossible for rule-based systems to give out accurate results as opposed to intelligent statistical-based systems. However, Manchanda et al [5] argued that rule-based approaches are ideal when manipulating a small amount of data as opposed to statistical approaches, the authors further mention that a higher accuracy can be achieved by providing clean training data and also changing or combining algorithms to specifically suit the task that one wants to achieve. On the other hand, Dharmik et al [20] mentioned that the accuracy of the Cocke- Younger- Kasami (CYK) algorithm in grammar checking is dependent on the set of grammar rules defined for the parser. The language defined has to be in its context-free grammar form. To achieve higher accuracy, the parser has to be fed with complete error-free grammar. In addition, an increase in the data size of the grammar allows better syntax analysis and improves the average accuracy of the parse tree. A parser with incomplete grammar can lead to false alarms.

2.3.6.2 Scalability

Feng [19] argued that rule-based systems are not scalable as opposed to statistical models. Scaling up the system can become troublesome, especially when one is dealing with big data that is dynamic. Updating the system may not only be time consuming but also complex because rules have to be explicitly defined to avoid false alarms. In contrast, statistical based systems are scalable by nature this is because they work well with large datasets. The process of scalability is simpler in these systems because they automatically learn from the data they are provided with. On the other hand, Dharmik et al [20] state that statistical-based parsers implementing the CYK algorithm have proved to be more scalable than parsers implementing the rule-based approach. This is mainly because the probabilistic rules used during grammar checking are automatically learnt by the machine from a text corpus.

2.3.6.3 Learning Ability

According to Kumar and Supriya [21], statistical-based systems have the capability of creating generalizations on the data through the power of machine learning during the training process and learn frontiers between decision boundaries. During decision making, machine learning algorithms can use the information they were trained with to effectively interpolate or infer new situations without being explicitly taught every possibility. Dharmik et al [20] mention that in contrast, every decision made in the rule-based approach is entirely dependent on the expert knowledge, or rules that were defined. What the Cocke-Younger-Kasami (CYK) algorithm learns is also limited to the context-free grammar rules constructed for the language.

2.3.6.4 Implementation and Application of Grammar Checkers

According to Arppe [7], the Swedish grammar checker has been launched in Microsoft Office Word 2000 and also as a stand-alone system called Grammatifix. The system was implemented in C++ under Unix and it has also been launched as a simple web interface for testing. Currently there is work in progress aimed at designing a graphical interface for PCs that will enable interactive use by users during writing. The PC system will be used as an investigation tool for learning usability characteristics with real users. The Short-coming associated with this grammar checker its inability to detect split compounds. In addition, it cannot be used in real-time for direct interaction with users mainly because of the underlying grammar checking technique (rule-based technique) adopted in the development of this grammar checker. Dekšne and Skadiņš [2] stated that the Latvian grammar checker is built into Microsoft Word and Open Office Writer text editors and it works as a background process. The downfall associated with this grammar checker is that it does not list all the syntactic rules of the Latvian language therefore some grammatically correct sentences may be labelled as incorrect due to unsuccessful parsing of sentences; this is caused by the insufficient coverage of grammatical rules. English grammar checkers such as Grammarly and Pro Writing Aid both operate as add-ins to Word applications and as standalone websites. Both grammar checkers fail to detect semantic errors, the grammar checkers scored an accuracy of 23% and 18% respectively when tested for their ability to identify semantic errors. Microsoft Office's word grammar checker has been used as a plugin to Microsoft Office's word processor and Email Outlook. Kinoshita, Salvador and Menezes [20] stated that the CoGrOO: Brazilian-Portuguese grammar checker was developed via the Perl programming language while the interface to OpenOffice was developed by the Java programming language and currently

exists as a standalone grammar checker. Research indicates that this grammar checker has a lot of false positives this is due to the fact that error rules defined to identify grammatical errors in sentences are not complete, this indicates the need for improvement. The advantage of the proposed grammar checker is that the grammar checker adopts the use of LSTM RNNs, hence the grammar checker will have high accuracy results in terms of grammar classification mainly because neural networks have a powerful computational ability, and they also have the representational power found necessary for natural language structures. LSTM RNNs are cable of handling long sequential data found in sentences they are also capable of learning long-term dependencies necessary for grammatical error classification. The grammar checking technique used in the grammar checker above is ideal for a larger coverage of language rules for the defined language mainly because the algorithm used learns the grammatical patterns in each sentence by itself via training; greater coverage will be achieved by training the algorithm with a large dataset hence improving classification accuracy.

2.3.6.5 Performance Metrics used in the Evaluation of Grammar Checkers

- Classification Accuracy

According to Novikova et al [22] classification accuracy is the easiest and most common performance metric applied to binary classification problems in NLP. The authors define classification accuracy as the rate of correctly classified samples. It works well only if there are an equal number of samples belonging to each of the defined classes. However, `argue that classification accuracy often gives users a false sense of achieving high accuracy. The authors further emphasized that the actual problems rise, when the cost of misclassification of the minor class samples are very high; classification accuracy can hide the detail or feature you need to analyze the performance of your model. Classification Accuracy has been used to evaluate the performance of popular grammar checking tools such as Grammarly, Ginger, Pro Writing Aid and Language Tool. Classification Accuracy is shown in equation (1) below;

$$\text{Classification accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \times 100$$

- Precision and Recall

Precision and Recall Sahu et al [23] stated the precision and recall is one of the common model evaluation metrics applied in machine learning. Precision in classification simply refers to the probability of all the samples that have been classified as correct actually being correct. Recall in classification refers to the probability that the classifier will be able to properly classify positive samples as certainly positive. The authors mentioned that this performance metric is also usually applied when dealing with binary classifications. There are two classes which are usually defined, and these are the positive and negative class. Additionally, Sahu et al [23] state that in the context of NLP specifically sentence classification, a sentence in the Positive class indicates that the sentence has been classified as grammatically correct whereas a sentence in the Negative class indicates that the sentence is not grammatically correct. However, Novikova et al [22] stated that it is not possible to maximize both metrics simultaneously, as one comes at the cost of another. The authors further emphasized that for problems where both precision and recall are important, one can select a model which maximizes both metrics. Precision and recall have been applied in the evaluation of popular grammar checking tools such as Grammarly, Ginger, Pro Writing Aid and Language Tool. The formulas for both Precision and recall are shown below as Equation (3) and Equation (4);

Formula for Precision

$$Precision = TP / (TP + FP) \quad (3)$$

Formula for Recall

$$Recall = TP / (TP + FN) \quad (4)$$

Key

TP = True Positives above refers to samples the classifier has correctly classified as positives
 TN = True Negatives above refers to samples the classifier has correctly classified as negatives
 FP = False Positives above refers to samples the classifier has incorrectly classified as positives
 FN = False Negatives above refers to samples the classifier has incorrectly classified as negatives

- F1 Score

Novikova et al [22] state that the F1 Score comprises the measurement of Precision and Recall scores which when combined form the F1 score. Therefore, the F1 score is the harmonic mean of precision and recall. The authors further mentioned that the F1 score should be used in cases where both Precision and Recall are important mainly because both metrics are maximized. F1 Score is shown below as equation (5):

Formula for F1 Score

$$Score = 2 / (1/Precision + 1/Recall) \quad (5)$$

- Binary Crossentropy Loss Function (BCELoss)

According to Wan et al [24], BCELoss is used in binary classifications tasks to compare each of the predicted probabilities to the actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. The smaller the loss, the better the model. The BCELoss Formula is represented below as equation (6);

~~BCELoss~~ BCELoss Formula

$$L = -\sum_{i=1}^n y_i \log(p_i) = -(y \log(p) + (1 - y) \log(1 - p)) \quad (6)$$

Table :Comparative Literature Review of Grammar Checking Techniques adopted in different grammar checkers

Model	Technique	Data	Accuracy
Latvian grammar checker	Syntax-based approach; CYK parsing algorithm	- test corpus (4814 sentences) - development corpus (9364 sentences)	- Recall rate 0.29 - Precision rate 0.83
Afan oromo grammar checker	Rule-based approach; 123 grammar rules	Student Graduation thesis text	- Recall rate 80% - Precision rate 88.89%
Swedish grammar checker (Grammatifix)	Rule-based approach; 43 error rules	1, 000, 54-word extract form the Goteborgs-posten 1998; Swedish newspaper	- Recall rate 85% - Precision rate 70%
Bangla and English grammar checker	Statistical-based approach; N-gram (trigram analysis)	5000 sentences from the Prothom-Alo corpus	Classification accuracy: 63% on English sentences and 53.7% on Bangla sentences
English grammar checker	Statistical-based approach; Elman RNNs	552 English sentences extracted from the GB Linguistic textbook by Lasnik and Uarigereka	Classification accuracy: 99.6% on training data and 74.2% on testing data
Chinese grammar checker	Statistical-based approach; Bi-LSTM-RNNs	30 359 sentences from the NLP-TEA dataset	Accuracy of 0.54 with a recall rate of 0.57 and a precision rate of 0.57
Brazilian-portuguese grammar checker (CoGroo)	Hybrid of rule and statistical-based approach techniques	800 sentences from the Metro corpus	- Recall rate 28.30% - Precision rate 44.12%
Setswana grammar checker	Statistical-based approach; LSTM-RNNs	850 positive and 850 negative Setswana sentences	Validation accuracy of 0.95 and Testing accuracy of 0.96

2.4Chapter Summary

From the findings above, it can be concluded that all techniques have their own benefits and shortfalls. Therefore, the choice of the technique to be adopted in the development of a grammar checker is dependent upon the language's morphology and grammatical features together with the existing NLP resources of the language.

Chapter 3 - Methodology and the Proposed Setswana Grammar Checker

3.1 Introduction

This chapter discusses the methodology that was adopted when administering this study and the proposed Setswana grammar checker. Before discussing the proposed statistical grammar checker, the Setswana linguistic resources are defined. The chapter goes on to explore the Experimental setup and data history, the proposed LSTM-RNN model and its performance metrics, and lastly, the System implementation and its deployment.

3.2 Setswana Linguistic Resources

3.2.1 Setswana Language and Grammar

Cole [10] states that, like other Bantu languages, Tswana varies drastically in structure from European languages. Supporting the sentiments above, Berg et al [11] states that Setswana is a semi-disjunctively written agglutinating language which is characterized by a complex morphology, and the language has a rich system of verbal inflections. The verbal prefixes are presented in a disjoint manner, while verbal suffixes are presented in a conjoined manner to the verbal root. The verbal prefixes and suffixes of Setswana are essential parts of the morphological structure of verbs and have morphological status in the c-structure of the Lexical Functional Grammar (LFG).

This dissertation is limited to the identification of grammatical errors in Setswana declarative sentences. Berg et al [11] state that simple Setswana declarative sentences also known as predicative sentences employ the SVO word order where 'S' symbolizes the subject, 'V' the verb, and 'O' the object. The authors go on to mention that in a simple transitive Setswana sentence, the subject appears before the verb then the object follows the verb. The authors define transitive sentences as sentences that are able to take a direct object. The examples in the sentences

below give possible word orders that can be followed by predicative sentences with a simple sentence structure. The sentence below is an example of predicative structure:

Example 1

English: Masego has bought a dress

Setswana: Masego o rekile mosese

Analysis

Subject: Masego Predicate: o rekile Object: mosese

Simple transitive Setswana sentence may also follow the SVOX word order, the subject appears before the verb then the object follows the verb followed by the adjunct.

Example 2

English: The students speak Setswana at school

Setswana: Baithuti ba bua Setswana ko sekolong

Analysis

Subject: Baithuti Verb: ba bua Object: Setswana Adjunct: ko sekolong

In the sentence above the subject 'Baithuti' goes before the verb 'ba bua' and the object 'Setswana' follows the verb. Finally, the adjunct 'ko sekolong' follows the object.

Simple declarative sentences in Setswana may take ditransitive, intransitive and transitive verbs and all these sentences can be altered with one or more than one adjunct. In a simple transitive Setswana sentence, the object in the sentence may be substituted by the object agreement morpheme which acts as an object marker.

Example 3

English: Students speak it at school Setswana: Baithuti ba se bua ko sekolong Analysis

Subject: Baithuti Verb: ba se bua Adjunct: ko sekolong

In the example above, the object 'Setswana' has been substituted by the object agreement morpheme 'se' which acts as the object marker. This modification then leads to a change in the

overall sentence structure. The sentence now consists of the subject 'Baithuti' followed by the verb 'ba se bua' which is then followed by the adjunct 'ko sekolong'.

Setswana sentences can also include verbs which can take two objects; the object instantly following the ditransitive verb is the indirect object, and it is followed by the direct object.

Example 4

English: The girl is buying a dress for the woman Setswana: Mosetsana o rekela mosadi mosese
Analysis

Subject: Mosetsana Verb: o rekela Indirect Object: mosadi Direct Object: mosese

In the example above, the subject 'Mosetsana' is followed by the ditransitive verb 'o rekela' which is then followed by the indirect object 'mosadi', the indirect object is then followed by the direct object 'mosese'.

Intransitive verbs do not take a direct object but they may possibly take adjuncts which would then follow the verb.

Example 5

English: The students speak it now

Setswana: Baithuti ba se bua jaanong

Analysis

Subject: Baithuti Verb: ba se bua Adjunct: jaanong

In the sentence above 'Baithuti' is the subject, 'ba se bua' is the intransitive verb and 'jaanong' is the adjunct.

3.2.2 Setswana Grammar Detection and Correction Approaches

According to Wagner et al [25], the most common grammatical errors that occur in sentences include Additional errors, Omission errors, Misordering errors and Misinformation errors.

Additional errors refer to errors that occur due to the presence of a word or any element which must not appear in a given context i.e. sentence in this case. Omission errors are caused by the absence of an element or item that must appear in a sentence e.g. omission of a certain word in a sentence. Misinformation errors refer to errors that are caused by the usage of an incorrect form of a morpheme in a given sentence. Lastly, Misordering errors refer to errors caused by the inappropriate placement or positioning of a morpheme or group of morphemes in a given utterance or sentence. This dissertation will be limited to identifying only Additional, omission, and misinformation errors that may possibly appear in Setswana Declarative sentences.

3.2.3 Setswana Corpus and its Annotation

A text corpus that contains Setswana sentences will be used as the dataset that will be used to train the LSTM model defined above. The corpus makes use of two text files; one text file contains a total of 1700 Setswana sentences; 850 correct sentences and 850 incorrect sentences. The second text file contains the labels, and these they contain only two values i.e. 850 negative and 850 positive labels. The negative label shows that a sentence is incorrect, and positive label shows that a sentence is correct. Each of the labels corresponds with the sentences given in the text file containing sentences. However, the sentences in the text files are not annotated, and the words in the sentences are not tagged with their part-of-speech tags. Setswana is an under-resourced language and has limited NLP tools, the language does not have a POS tagger. The focus of this dissertation is to develop a Setswana grammar checker only. The research makes use of LSTM-RNNs so the input words are in the form of word vectors hence the sentence composition is in the form of vectors. The vectors fed into the LSTM-RNN's embedding layer alone are enough to conduct all the processing that is needed to classify if a sentence is grammatically correct or not. The word vectors are formulated from the position of each word in relation to neighbouring words hence sequence modelling is achieved by extracting the word order or pattern followed by Setswana Sentences; emphasis is on the sequence of words (syntax) and how they are arranged to create well-formed sentences. The corpus is attached in Appendix A as figure 3.1

3.3 Experimental setup and Data History

3.3.1 Data Collection

The participants (Setswana Experts) of this study who were chosen to give the researcher information on the syntax of Setswana sentences were selected through the purposive sampling technique. The Setswana experts (teachers) were chosen based on their applicability to the research study, so the researcher had to limit the sample to the experts in the field of Setswana. Etikan, Musa and Alkassim [26] state that purposive sampling is a non-probabilistic sampling technique. The technique allows the researcher to make decisions on the information they need and then finds people that are knowledgeable and well-informed with the subject under study.

3.3.2 Data pre-processing and Exploratory Analysis

The data will be processed by removing all punctuation marks from sentences and removing new line characters in the given corpus. In addition, the Outliers will be removed from the data to ensure that the model does not get values that are too big or too small; hence, the resulting sentences are of the same length. The information in the text files will then be converted into data structures that can be consumed by LSTM models, hence all the words in the sentences in the Setswana text corpus are converted into their numeric (integer form). This is achieved via the use of the counter method. The counter method generates word indexes (in integer form) and encodes the sequence of each sentence in the corpus. The output from this generation are strings of integers representing the sequence of each sentence. The two labels will also be encoded and the positive labels are encoded into 1 and negative labels to 0. The data is then split into training, validation and testing datasets that will be used to achieve the task of Sentence classification via the LSTM model to classify Setswana sentences as grammatically correct or incorrect. This task of data splitting will be achieved through the `split_frac` method. The `split_frac` method is used to define how much of the data should be in the training, validation and test datasets. The `split_frac ()` will be set as follows; 80% of the data sample will be defined as the training data, and the outstanding 20% sample data will be split amongst the validation and test dataset, 50% validation and 50% test dataset.

3.4 LSTM-RNN Model and its Performance Metrics

3.4.1 LSTM-RNN Mathematical Theory and its Architecture

The artefact to be developed will make use of a Recurrent Neural network model, specifically the Long Short Term Memory (LSTM) recurrent neural network model because of its ability to learn and predict from sequential data. LSTMs have also proven to have the ability to handle sequential data better than recurrent neural networks; LSTMs do not suffer the vanishing gradient problem as compared to RNNs. The Keras and TensorFlow neural network libraries will be used when implementing this RNN LSTM model; these libraries enable the implementation of LSTMs via python. The design of the proposed LSTM model is constructed as follows; The LSTM has an Embedding layer, LSTM layer and Output layer. A detailed description of these modules is given below;

- Embedding Layer

Data in its integer form from the input layer is fed as input to the Embedding layer. The embedding layer is responsible for the conversion of textual data from its integer form into its short numerical vector representations; The embedding layer learns the weights of the input data in this case words (in their integer form) for order reduction. The embedding layer helps decrease the dimensionality of the word-space from the length of the corpus vocabulary to a much smaller length. The conversion details are captured in an embedding matrix. Every word gets replaced by an index which is used to look up the vector of each word in the embedding matrix. This is computational and it reduces the size of the dataset that is achieved via the use of the Word2Vec algorithm. This algorithm is used to produce word embeddings. Once the model is trained, the learned weight matrix $WV \times N$ can be easily extracted and used to extract the word vectors. Word2Vec is shown below as Equation (7);

Word2Vec Formula

$$\text{Word2Vec} = WV * N$$

In the formula above V represents the size of vocabulary and N represents the number of hidden units or length of word embeddings

- LSTM layer

The output from the embedding layer is fed as input to the LSTM layer. The LSTM layer learns the long-term dependencies between time steps in sequence data. In this case, the sequential data refers to the Setswana sentences provided as input to the LSTM layer in their vector

representations (reduced in size via the embedding layer). The function of the layer is to perform additive interactions, which can help increase gradient flow over longer structures during training. The LSTM layer is made up of a cell, input gate, output gate, and forget gate. The components that make up the LSTM layer are explained below;

I. Cell

The cell acts as the memory unit of the LSTM layer. The cell is accountable for monitoring and observing the dependencies between the components in the input sequence i.e. relationship of words in the input Setswana sentences.

II. Input Gate

The input gate controls the degree to which a new value flows into the cell. The formula utilized by the input gate during this process is shown below as equation (8);

$$\begin{aligned} i & \\ t &= \sigma (w_{i,[h_{t-1}, x]} + bi) \\ \dot{C} &= \tanh (w_{c,[h_{t-1}, x]} \\ &+ bc) \end{aligned} \quad (8)$$

III. Forget Gate

The forget gate is responsible for regulating the extent to which a value is stored or present in the cell. The formula utilized by the forget gate during this process is shown below as equation (9);

$$t = \sigma (w_{f,[h_{t-1}, x_t]} + bf) \quad (9)$$

IV. Output Gate

The output gate regulates the degree or level to which the value in the cell is used to calculate the output activation of the LSTM unit. The formula utilized by the output gate during this process is shown below as equation (10);

$$\begin{aligned} o & \\ t &= \sigma (w_{o,[h_{t-1}, x]} + \\ &bo) \end{aligned} \quad (10)$$

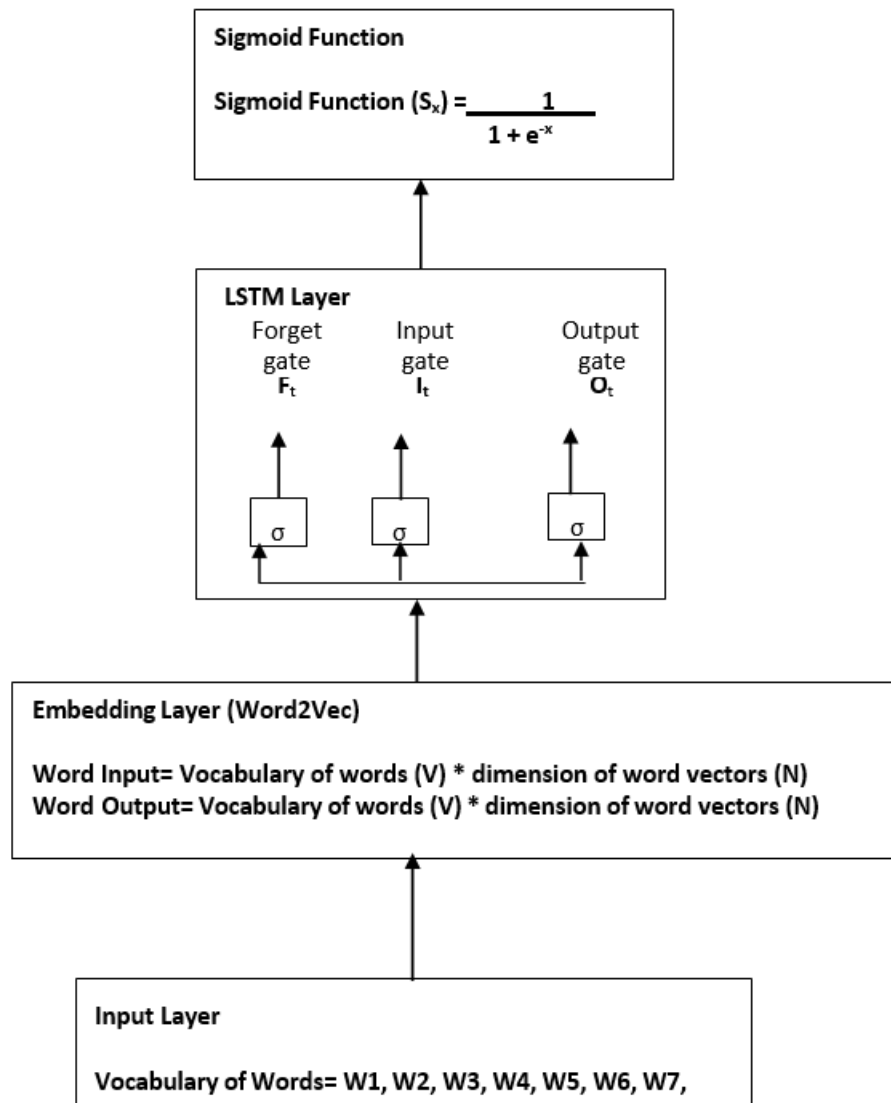
- Output layer

The output from the LSTM layer is fed as input to the output layer. This layer maps output from the LSTM unit to the desired output size. This layer also includes the sigmoid activation function which is used to classify sentences in the value range of 0-1. Sentences that are classified as 1 are considered grammatically correct, while those that are classified as 0 are considered being erroneous, hence grammatically wrong. The formula for the Sigmoid Activation Function is displayed below as equation (11);

$$(sx) = \frac{1}{1 + e^{-x}} \quad (11)$$

- LSTM-RNN Algorithm

Figure 3.2 below shows the architecture of the LSTM-RNN and its mathematical theory



3.4.2 LSTM-RNN Implementation Algorithms and Procedure

The proposed Setswana Grammar checker has three key modules, and these modules can be defined in an algorithmic form. The modules consist of the Embedding layer, LSTM layer, and the Sigmoid module. The grammar checker takes input in the form of Setswana sentences. Text pre-processing is then performed on the input text to remove new line characters, punctuation marks, and outliers. Sentences are then split into their unit form i.e. words. The words are then converted into their numeric form because LSTM models can only process numeric data. The converted sentences in their integer form are then passed on to the embedding layer where they are converted to their vector form, and then finally passed to the LSTM layer and eventually the

sigmoid layer for sentence classification. Figure 3.3 below is a flowchart that indicates the procedure of the LSTM- RNN implementation.

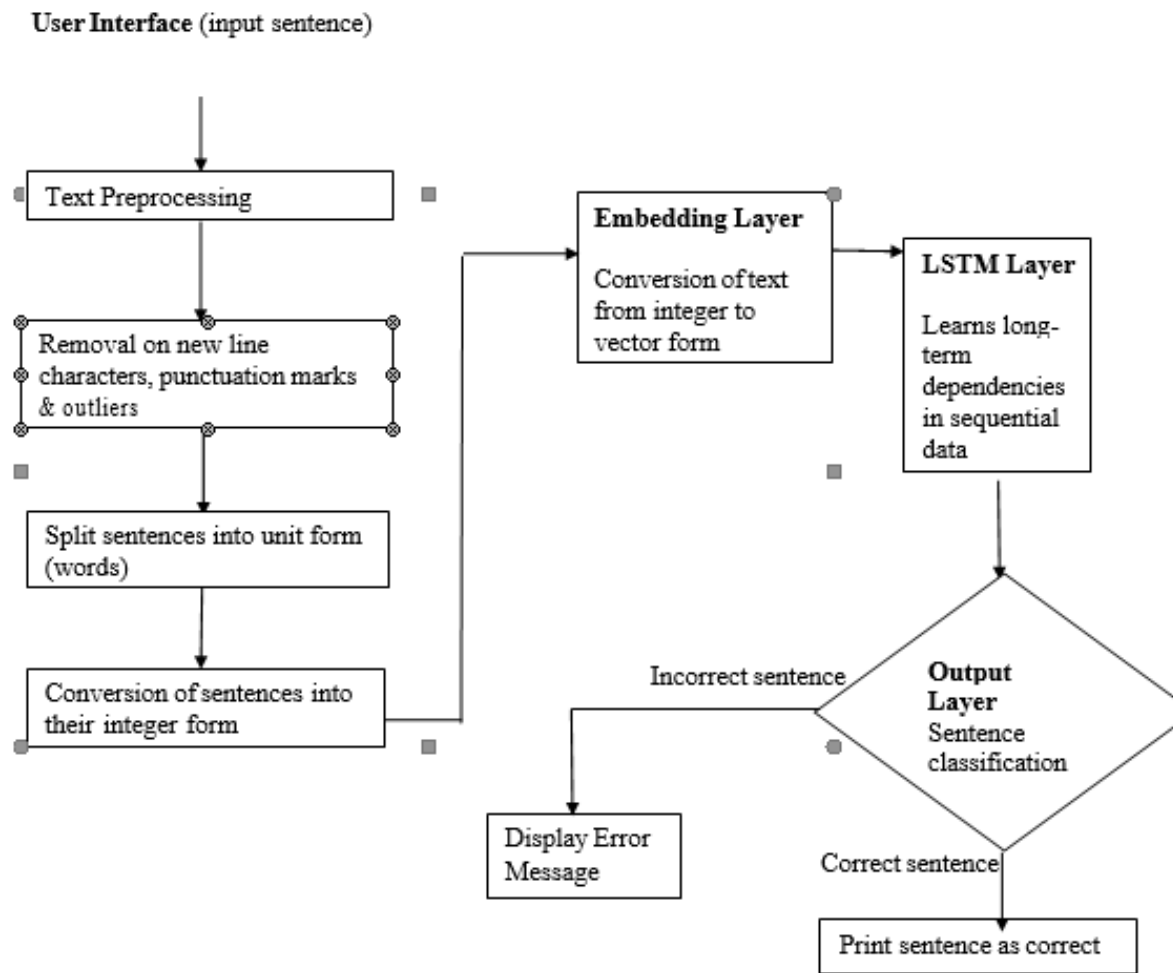


Figure 3.3: LSTM-RNN Implementation Algorithms and Procedure

3.4.3 Systems Configuration and Hyper Parameters Tuning

3.4.3.1 Integrated Development Environment Setup

The Integrated Development Environment (IDE) was set up before starting up the actual implementation implemented of the proposed Setswana grammar checker. Anaconda python distribution was downloaded and installed to have access to the Jupyter Notebook which was used for development. The justification for choosing the Jupyter notebook is its ability to integrate code and its output into a single that combines visualizations, mathematical equations, narrative

text, and other rich media. The Keras and TensorFlow neural network libraries were used to develop and implement the LSTM-RNN model. These libraries enable the implementation of LSTMs through python. These libraries are open source, hence all the necessary functions needed to develop the LSTM-RNN were imported from these libraries. The imported functions are displayed by the code snippet in figure 3.4 in appendix A.

3.4.3.2 Data Loading

Initially, the text files (Corpus) containing the Setswana sentences and their corresponding labels were stored in a file within the researcher's PC (these are found in appendix A Figure 3.1). The text files had to be loaded into the Jupyter Notebook IDE in order to enable their use within this platform. Figure 3.5 in Appendix A depicts the data loading process and figure 3.6 (Appendix A) gives a code snippet of the results gathered after printing six sentences. Code snippet serves as an example of how the data from the corpus looks like within the Jupyter notebook. The sentences are printed together with their corresponding labels.

3.4.3.3 Data Processing

After loading the data, the data was processed by removing all punctuation marks and removing new line characters from the Setswana sentences in the given corpus. Outliers were also removed from the data to ensure that the model does not get values that are too big or too small, and that the resulting sentences are of the same length. Figure 3.7 in Appendix A shows the code snippet that shows the process of the removal of all punctuation marks and removing new line characters from the Setswana sentences in the corpus. Figure 3.8 (Appendix A) shows the code snippet used for removing outliers from the data (Corpus).

3.4.3.4 LSTM-RNN Model

The proposed LSTM network model follows the following design: The LSTM has an Embedding layer, LSTM layer, Training module, Testing module and Output layer. The output of one layer serves as input to the next layer in the respective order defined above. The development of each of the aforementioned modules are defined below:

I. Embedding Layer

The embedding layer was developed to convert the word tokens (integers) to word vectors. The dimension of the embedding layer was defined, and in this case, it refers to the number of columns in the embedding lookup table i.e. size of our vectors. The dimension of the embedding

layer was defined as (73,100). Vocabulary size of length (vocab2int) +1 was defined. The vocabulary size in this case refers to the size of the size or range of values for our input i.e. Setswana word tokens. Figure 3.13 in Appendix A shows code snippets of the RNN-LSTM model (embedding layer) instantiated with the hyper parameters.

II. LSTM Layer

The output from the embedding layer is fed as input to the LSTM layer. The output from the embedding layer comes in the form of short numerical vector representations of words. The purpose of the LSTM layer is to learn the long-term dependencies between time steps in the sequential data. When developing the LSTM layer the number of hidden layer dimension in the LSTM cells was set to 128 and the number of LSTM layers was set to 1. Batch-first was set to true. This was done to ensure that the input and output tensors are provided in batches to the LSTM layer. The dropout in the LSTM layer was set to 0.3. It is necessary for hidden layers to dropout some of the units or nodes during training. This is necessary for regulating the deep neural network and avoidance of overfitting in the neural network. Dropout prevents overfitting due to a layer's "over-reliance" on a few of its inputs. Because these inputs are not always present during training i.e. they are dropped at random, the layer learns to use all of its inputs, improving generalization. Figure 3.13 in Appendix A shows a code snippet of the parameters of the LSTM layer defined above and the instantiation.

III. Training Module

The training module was developed with the following hyper parameters defined for training. The learning rate was set as 0.001, the number of epochs was set to 11 and the maximum value to clip was set to 5. The training algorithm that was used to train the LSTM-Recurrent Neural Network model on the task of grammar classification was the Adam Optimization Training algorithm. The model followed the supervised learning approach. Figure 3.14 in Appendix A shows the Training module together with its defined and instantiated parameters.

IV. Testing Module

In the testing module, the Binary Cross Entropy Loss function was used to optimize the performance of the model in terms of classification ability. The Binary Cross Entropy Loss function is suitable for binary classification problems such as the one in this study. Sentences that were labelled as 1 were classified as grammatically correct, while those that are labelled as

0 were considered to be erroneous, hence classified as grammatically wrong. In this setting, correct sentences belong to the positive class while incorrect sentences belong to the negative class. The loss function will return high values for poor predictions and small values for good predictions. Figure 3.16 in Appendix A shows the testing module, together with its parameters.

V. Output layer

The output layer was developed to map output from the LSTM unit to the desired output size. The output from the LSTM layer was fed as input to the output layer. This layer makes use of the sigmoid activation function which is used to classify sentences in the value range of 0-1. This function was imported from the PyTorch package. Sentences that were labelled as 1 were classified being grammatically correct, while those that are labelled as 0 were considered to be erroneous, hence they are classified as grammatically wrong. Figure 3.17 in Appendix A shows the Output layer and its defined parameters as stated above.

3.4.4 Employed Performance Metrics

The Classification Accuracy and Binary Cross Entropy loss function were used as the performance metrics to evaluate the performance of the Setswana grammar checker. The Binary Cross Entropy Loss Function was employed in both training and testing module to optimize the model hence increasing its classification ability. The loss function returns high values for poor predictions, and small values for good predictions. The implementation of the Binary Cross Entropy Loss function is shown in Figure 3.14, Appendix A. Classification Accuracy results for each input sentence are given. Several sentences were fed as input to the grammar checker, and they were either classified as grammatically correct or incorrect, and for each prediction a classification accuracy was given. These sentences and their results will be presented in the Chapter 4 (Results and Analysis).

3.4.5 Systems Integration

3.4.5.1 Systems API

The grammar checking system makes use of the Keras API. This API runs on top of the TensorFlow machine learning platform. TensorFlow packages puts together a collection of machine learning algorithms and deep learning models i.e. Neural networks and makes them useful by way of a common metaphor. TensorFlow uses Python to deliver a convenient front-end API for building applications with the framework while executing those applications in high-performance C++. Python just directs traffic between the pieces and provides high-level programming abstractions to hook them together. These APIs aided in the development of an LSTM-RNN grammar checker with high abstraction. The functions that were imported and installed to make use of the Keras API are shown in Figure 3.4, Appendix A

3.4.5.2 Systems Implantation and its Deployment

System implantation and deployment of the proposed grammar checking model followed the steps below;

I. Preparation

In the preparation stage, all python libraries, configuration files, Integrated Development Environment (IDE) and the code itself will be gathered and bundled into one software package. These include the Anaconda setup, Jupyter notebook setup, text files that make up the Setswana Corpus used in this grammar checker, the grammar checker in python code. Python libraries and APIs that need to be imported for the code to function without producing any errors will also be specified. These comprises of the Keras API, the Numpy and TensorFlow Python libraries.

II. Testing

To test the functionality of the grammar checker, the grammar checker will be deployed to a test server. The test environment is a server that runs a windows server 2012 operating system and 2 user computers both running windows 10 operating systems. A software package that comprises of the Anaconda setup, Jupyter notebook setup, text files that make up the Setswana Corpus, Python libraries and APIs will also be deployed into this server. A shared folder containing these packages will then be shared with a few users for testing so that they can install and make use of the grammar checker and the software package. This will help in checking the

functionality of the grammar checker in an environment different from the one it was developed on. In this stage, errors and bugs can be identified and fixed before the final deployment.

III. Deployment

In the Deployment stage, the grammar checker will be deployed for public use. The grammar checker will be deployed in a phased manner. This is mainly because currently we do not have an existing Setswana grammar checker so introducing a phased deployment process allows the users to embrace the transition; i.e. finally making use of a grammar checker in their Setswana texts and hence avoiding user resistance. In Phase 1, the grammar checker will be made available to individuals that have an interest in using the application. Only a sample of 20 people will be selected Phase 1. These individuals will be randomly selected. The researcher will make the public aware of the Setswana grammar checker via platforms such as social media and avail the application to the people are interested in the using the application. The grammar checker will be dispensed to the selected sample using different media such as hard drives and disks (CDs and DVDs). Phase 2; which is the last phase, involves rolling out the application to the public. The software will be packaged in CDs and distributed to the public based on the licensing agreements.

Chapter 4 –Results and Analysis

4.1 Introduction

This chapter is aimed at evaluating the performance of the proposed Setswana grammar checker. Firstly, the chapter presents the results of the LSTM-RNN model before and after the actual tuning of its Hyper Parameters. The chapter then goes on to present the results obtained from performance metrics. Lastly, the chapter presents an evaluation of the grammar checker against existing models.

4.2 Model Hyper Parameters

To optimize the model, the following hyper parameters were considered for tuning: the dimension of the embedding layer was defined, the number of LSTM layers, the number of hidden layer dimension in the LSTM cells, batch-first, dropout - applied to the layers (dropout), the learning rate of the optimizer, the number of epochs, and the maximum value to clip.

4.2.1 Results of the Model (LSTM-RNN) without Tuning Hyper Parameters

Table 4.1 below shows the default values of the individual hyper parameters before tuning the LSTM-RNN model.

Dimension of embedding layer	Number of LSTM layers	Hidden layer dimension	Batch_first	Dropout	Learning_rate	Number of epochs	Maximum valueclip
100	2	256	true	0.5	0.002	25	5

Figure 3.9 and Figure 3.10 in Appendix A shows a code snippet of the hyper parameters above and their instantiation; these are the hyper parameters of the model before tuning. Figure 3.11 (Appendix A) shows the results after executing the LSTM-RNN model (training module) before tuning hyper parameters. The results were obtained after training the LSTM-RNN model with 1700 Setswana sentences of which 850 were correct Setswana sentences and 850 were incorrect Setswana sentences. The training module had a Validation accuracy of 0.57, Validation loss of 0.25, and a Training loss of 0.35. Figure 3.12 in Appendix A shows the results of the testing module that produced a testing loss of 0.49, and a testing accuracy of 0.43. The results above indicate that hyper parameter tuning is necessary to ensure that the LSTM-RNN gives better results.

4.2.2 Results of the Model (LSTM-RNN) with Tuned Hyper Parameters

Table 4.2 below shows the values of the individual hyper parameters after tuning.

Dimension of embedding layer	Number of LSTM layers	Hidden layer dimension	Batch_first	Dropout	Learning_rate	Number of epochs	Maximum value_clip
100	1	128	true	0.3	0.001	11	5

Figure 3.13 and Figure 3.14 in Appendix A shows a code snippet of the hyper parameters above and their instantiation; these are the hyper parameters of the model after tuning. Figure 3.15 (Appendix A) shows the results after executing the LSTM-RNN model (training module) with tuned hyper parameters. The results were obtained after training the LSTM-RNN model with 1700 Setswana sentences of which 850 were correct Setswana sentences and 850 were incorrect Setswana sentences. The training module had a Validation accuracy of 0.95, Validation loss of 0.05 and a Training loss of 0.1. Figure 3.16 in Appendix A shows the results of the testing module. The testing module produced a testing loss of 0.06 and a testing accuracy of 0.96. The results above were obtained after tuning the hyper parameters of the LSTM-RNN model. An optimal combination of hyper parameters was achieved to give better results. After tuning, the LSTM- RNN model's validation accuracy improved by increasing from 0.57 to 0.95, and the testing accuracy increased from 0.43 to 0.96.

4.3 Results Obtained from Performance Metrics

The metrics used to evaluate the performance of the LSTM-RNN model include the Binary cross entropy loss function and Classification accuracy.

4.3.1 Binary cross entropy loss function

The Binary cross entropy loss function was applied to evaluate the performance of both the training and the testing module. The results indicate that the training module had a Validation accuracy of 0.95, Validation loss of 0.05 and a Training loss of 0.1. The testing module produced a testing loss of 0.06, and a testing accuracy of 0.96. The Binary cross entropy loss function returns high values for poor predictions and small values for good predictions; hence this LSTM-

RNN model had a low Validation loss of 0.05 and a test loss of 0.06. This indicates that the LSTM-RNN model has a good prediction.

4.3.2 Classification accuracy

Classification Accuracy results for each input sentence are given. Sentences were fed as input to the grammar checker. These sentences went through the classification process and were either classified as grammatically correct or incorrect, and for each prediction, a classification accuracy was given. Figure 3.18 in Appendix A shows an example of a sentence that has been classified as correct with a prediction accuracy of 0.948. Another example of a sentence that has been classified as incorrect with a prediction accuracy of 0.003 is also presented. These results depict that a higher classification accuracy is given provided the sentence conforms to the grammatical rules of the Setswana language. If the sentence does not match the grammatical rules of the Setswana language, then a lower classification accuracy is generated.

4.4 Results with Existing Models

The results of the proposed grammar checker are compared with those of existing models. The performance metrics reported by the Setswana grammar checker indicate that the grammar checker has a satisfactory accuracy just like other statistical grammar checkers. Table 4.3 below gives a comparison of the results of the grammar checker with existing models.

Model	Technique	Data	Accuracy
Latvian grammar checker	Syntax-based approach; CYK parsing algorithm	- test corpus (4814 sentences) - development corpus (9364 sentences)	- Recall rate 0.29 - Precision rate 0.83
Afan oromo grammar checker	Rule-based approach; 123 grammar rules	Student Graduation thesis text	- Recall rate 80% - Precision rate 88.89%
Swedish grammar checker (Grammatifix)	Rule-based approach; 43 error rules	1, 000, 54-word extract form the Goteborgs-posten 1998; Swedish newspaper	- Recall rate 85% - Precision rate 70%
Bangla and English grammar checker	Statistical-based approach; N-gram (trigram analysis)	5000 sentences from the Prothom-Alo corpus	Classification accuracy: 63% on English sentences and 53.7% on Bangla sentences
English grammar checker	Statistical-based approach; Elman RNNs	552 English sentences extracted from the GB Linguistic textbook by Lasnik and Uarigereka	Classification accuracy: 99.6% on training data and 74.2% on testing data
Chinese grammar checker	Statistical-based approach; Bi-LSTM-RNNs	30 359 sentences from the NLP-TEA dataset	Accuracy of 0.54 with a recall rate of 0.57 and a precision rate of 0.57
Brazilian-portuguese grammar checker (CoGroo)	Hybrid of rule and statistical-based approach techniques	800 sentences from the Metro corpus	- Recall rate 28.30% - Precision rate 44.12%
Setswana grammar checker	Statistical-based approach; LSTM-RNNs	850 positive and 850 negative Setswana sentences	Validation accuracy of 0.95 and Testing accuracy of 0.96

4.5 Chapter Summary

It is worth noting that the results from the models above were influenced by various factors such as the grammar checking technique implemented by the grammar checker, morphology of the language and the dataset, hence the large variance in results. Research has shown that results of the Setswana grammar checker can be improved by making use of a larger corpus because neural networks work best with large amounts of data. Improving the performance of the model by choosing the ideal number of hidden layers, the number of neurons and learning rate are also some of the factors that can be considered when improving the results of the model; parameters of the model have to be tuned to achieve optimal results from the given dataset. The proposed Setswana grammar checker makes use of a small dataset limited to simple declarative sentences because of the unavailability of Setswana NLP resources. More types of sentences and data can be added to make the model more robust.

Chapter 5 - Conclusions and Recommendations

5.1 Introduction

This chapter is aimed at discussing the overview, conclusions, limitations, and recommendations on the future works of this study. The challenges which were encountered throughout this study are also discussed.

5.2 Overview of the study

The aim of this study was to develop a Setswana grammar checker that classifies Setswana sentences as grammatically correct or incorrect. The grammar checker adopts the statistical approach to grammar checking and was developed using LSTM Recurrent Neural networks. The objectives achieved to fulfil the research objective of this study are discussed below;

- Objective 1

The first objective of this study was to identify and adopt the LSTM-RNN architecture that will be used to implement the LSTM-RNN based Setswana grammar checker. The reason for choosing the RNN architecture for implementation is their ability to process long sequential data. This is because the LSTM-RNN units include a memory cell that can keep information in memory for longer time periods.

- Objective 2

The second objective of this study was to convert the words in the Setswana corpus into short numerical word vector representations via the LSTM-RNN word embedding model. A Setswana corpus was created in the form of a text file containing declarative Setswana sentences that was used as the dataset in this study. The information in the text corpus was then converted into data structures that can be consumed by LSTM models, hence all the words in the sentences in the Setswana text corpus were converted into their numeric (integer) form. This transformation was done to transform data into a format that could be easily understood and processed by neural network i.e. numeric data. Data in its integer form was then fed as input into the embedding layer. The embedding layer served the purpose of converting one-hot-vectors into short numerical vector representations of words. This is computationally efficient for reducing the size

of the dataset hence enabling easy data processing. Output from the Embedding layer was then fed as input to the LSTM layer.

- **Objective 3**

The third objective was to train LSTM-RNN model on grammar classification via the Adam Optimization Training algorithm (AOT). This objective was achieved via the functionality of the LSTM and Output layer. The purpose of the LSTM layer is to perform additive interactions that can help increase gradient flow over long data sequences during training. The grammar checker achieved a validation accuracy of 0.94 during training, and 0.97 testing accuracy. The output from the LSTM layer was then fed as input to the output layer. The function of the output layer is to map output from the LSTM unit to the desired output size. This layer also comprises of the sigmoid activation function which was used to classify sentences in the value range of 0-1.

Sentences that are classified as 1 were considered grammatically correct, while those that were classified as 0 were considered being erroneous, hence grammatically wrong. Furthermore, a classification accuracy prediction percentage was given each time the grammatical class of a given sentence was predicted.

- **Objective 4**

The last objective of this study was to evaluate the grammar checker on learnability, accuracy, and memorability. To evaluate the LSTM-RNN model results of the LSTM-RNN model before and after turning, hyper parameters are discussed together with the results obtained from performance metrics. The results of the grammar checker are then evaluated against existing grammar checking models. The results from the evaluation proved that the Setswana grammar checker was able to achieve its goal of classifying Setswana sentences as grammatical or ungrammatical at a high successful detection rate.

5.3 Summary of the Study

Based on the study, the researcher found out that LSTM recurrent neural networks are language independent and that the grammatical pattern or word order followed by Setswana sentences was determined by the position of each word in relation to the words surrounding it. Furthermore, LSTM- RNNs work better with large amounts of data. The performance of the model improved

when the number of sentences in the corpus were increased. Furthermore, a larger number of hidden layers provide better performance. The number of neurons in the model also have an impact on the performance of the model, hence an adequate amount of neurons should be used to improve the performance of the neural network. If too many neurons are used, then the training time of the model may become too long or the model may over fit. There is no defined number of neurons or hidden layers that should be used in a neural network model. The choice on the number of neurons and hidden layers is experimental. The learning rate of the model also has an influence on the performance of the model. When the learning rate is small, then the neural network will need numerous epochs to converge, hence it can be trapped in local minima easily. When the learning rate is large, then gradient descent will overshoot the minima and diverge. Therefore, to develop a robust classification model, all of the attributes discussed above should be optimally chosen and clearly defined.

5.4 Recommendation for Future Work

Recommendations for future works on the Setswana grammar checker developed for classifying Setswana sentences as either grammatically correct or incorrect could be;

- Adding a module that highlights where the grammatical error is in the sentence, and adding a module that predicts and displays the correct grammatical structure when the sentence is incorrect.
- Introducing other types of sentences to enable the grammar checker to have a larger sentence coverage than the ones used in this model. This will help in increasing its robustness.
- The LSTM-RNN model adopted in the development of this Setswana grammar checker can be used to develop or improve existing Natural Language Processing (NLP) tools specifically Grammar checkers in any other resource-scarce languages. This is mainly because artificial neural networks can be applied in any other language irrespective of the languages' morphological structure or type because they are language-independent. Emphasis is on the sequence of words (syntax) and how they are arranged to create well- formed sentences and not their semantics or morphological structure. The hyper parameters defined in this model can be modified to fit the resources and needs of the language in consideration.

5.5 Challenges

Unlike other machine learning algorithms, artificial neural networks work in a “black box” nature. There is no step by step explanation of how the decision or outcome was reached on a defined problem. In contrast to neural networks, algorithms like decision trees are interpretable.

However, measures were taken to ensure that the results produced were accurate i.e. a sample of sentences from the test set with predefined classification labels were punched into the model and used to verify if they will produce the same results as defined in the label. Another challenge faced was the unavailability of Setswana experts. Some experts had busy schedules, hence they could not attend meetings to explain certain concepts.

5.6 Contributions to the Body of Knowledge

This research is the first ever work for developing a grammar checker for Setswana using Deep Learning i.e. LSTM-RNNs. A corpus was created using theoretical information from Setswana experts, and it was used to train the LSTM-RNN model on grammar classification. The Setswana grammar checker had a validation accuracy of 0.95, and a testing accuracy of 0.96. Grammar checkers are essential because they verify written text for grammatical correctness, hence producing syntactic error-free textual content. This is a major contribution for Setswana which is currently considered as one of the under resourced languages in terms of Natural language processing tools.

References

1. Jones, J., Bosch, S. E., Pretorius, L., & Prinsloo, D. (2005). Development of reusable resources for Human Language Technologies (HLT) applications: practice and experience. *SouthAfrican Journal of African Languages*, 25(2), 141-159.
2. Deksne, D., Skadiņa, I., & Skadiņš, R. (2014, April). Extended CFG formalism for grammar checker and parser development. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 237-249). Springer, Berlin, Heidelberg.
3. Alam, M., UzZaman, N., & Khan, M. (2007). N-gram based statistical grammar checker for Bangla and English.
4. Park, J. C., Palmer, M. S., & Washburn, C. (1997, March). An English Grammar Checker as a Writing Aid for Students of English as a Second Language. In *ANLP* (p. 1747 24).
5. Manchanda, B., Athavale, V. A., & kumar Sharma, S. (2016). Various Techniques Used For Grammar Checking. *International Journal of Computer Applications & Information Technology*, 9(1), 177.
6. Pretorius, R., Berg, A., Pretorius, L., & Viljoen, B. (2009, March). Setswana tokenisation and computational verb morphology: Facing the challenge of a disjunctive orthography. In *Proceedings of the First Workshop on Language Technologies for African Languages* (pp. 66- 73). Association for Computational Linguistics.
7. Arppe, A. (2000). Developing a grammar checker for Swedish. In *Proceedings of NODALIDA* (Vol. 99, pp. 13-27).
8. Mesnil, G., He, X., Deng, L., & Bengio, Y. (2013, August). Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech* (pp. 3771-3775).
9. Lawrence, S., Giles, C. L., & Fong, S. (2000). Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1), 126-140
10. Cole, D.T. (1952). *An Introduction to TSWANA Grammar*. Longman Penguin Southern Africa.
11. Berg, A., Pretorius, R. P. L., Butt, M., & King, T. H. (2013). The representation of Setswana double objects in LFG. In *Proceedings of the LFG13 conference* (pp. 111-1760 130).
12. Tesfaye, D. (2011). A rule-based Afan Oromo Grammar Checker. *IJACSA Editorial*.
13. De Mulder, W., Bethard, S., & Moens, M. F. (2015). A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech &*
14. Otlogetswe, T. (2001). The BNC design as a Model for a Setswana language corpus.extraction, 1.

15. Huang, S. and Wang, H., 2016, December. Bi-LSTM neural networks for Chinese grammatical error diagnosis. In Proceedings of the 3rd Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA2016) (pp. 1931 148-154).
15. Domeij, R., Knutsson, O., & Severinson Eklundh, K. (2002). Different ways of evaluating a Swedish grammar checker.
16. Kinoshita, J., do Nascimento Salvador, L. and de Menezes, C.E.D., 2006, May. CoGrOO: a Brazilian-Portuguese Grammar Checker based on the CETENFOLHA Corpus. In LREC (pp. 2190-2193).
17. Sharma, S. K. (2016). Rule Based Grammar Checking Systems (A Survey). International Journal of Computer Applications & Information Technology, 10(1), 217-220.
18. Rahman, M. R., Habib, M. T., Rahman, M. S., Islam, G. Z., & Khan, M. A. A. (2020). An exploratory research on grammar checking of Bangla sentences using statistical language models. International Journal of Electrical and Computer Engineering, 10(3), 32
19. Feng, F., Yang, Y., Cer, D., Arivazhagan, N., & Wang, W. (2020). Language-agnostic bert sentence embedding. arXiv preprint arXiv:2007.01852.
20. Dharmik, R. C., Bhanuse, R. S., & Gaikwad, A. D. (2016, February). Ambiguity of Context Free Grammar using the CYK algorithm. In 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave) (pp. 1-6). IEEE
21. Kumar, N. S., & Supriya, L. P. (2018). Survey on grammar checking and correction using deep learning for indian languages. International Research Journal of Engineering and Technology (IRJET), 5(11).
22. Novikova, J., Dušek, O., Curry, A. C., & Rieser, V. (2017). Why we need new evaluation metrics for NLG. arXiv preprint arXiv:1707.06875.
23. Sahu, S., Vishwakarma, Y. K., Kori, J., & Thakur, J. S. (2020). Evaluating performance of different grammar checking tools. International Journal, 9(2).
24. Wan, H., Gao, L., Su, M., Sun, Q., & Huang, L. (2021). Attention-Based Convolutional Neural Network for Pavement Crack Detection. Advances in Materials Science and Engineering, 2021.
25. Wagner, J., Foster, J., & van Genabith, J. (2007, June). A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL) (pp. 112-121
26. Etikan, I., Musa, S. A., & Alkassim, R. S. (2016). Comparison of convenience sampling and purposive sampling. American journal of theoretical and applied statistics, 5(1), 1-4.
27. O’Riain, S., Curry, E., & Buitelaar, P. (2012, December). Engaging Practitioners within Design Science Research: A Natural Language Processing Case Study. In European Design Science Symposium (pp. 155-169). Springer, Cham.
28. Liu, D., Li, Y., & Thomas, M. A. (2017, January). A roadmap for natural language processing research in information systems. In Proceedings of the 50th Hawaii

International Conference on System Sciences.

29. Indulska, M., & Recker, J. (2010). 13. Design science in IS research: a literature analysis Information systems foundations: The role of design science, 285-302.
30. Offermann, P., Levina, O., Schönherr, M., & Bub, U. (2009, May). Outline of a design science research process. In Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology (p. 7). ACM.
31. Pfeffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2006). The design science research process: A model for producing and presenting information systems research. In Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006), Claremont, CA, USA (pp. 83-106).
32. Bhirud, N. S., Bhavsar, R., & Pawar, B. (2017). Grammar checkers for natural languages: a review. International Journal on Natural Language Computing (IJNLC), 6(4), 51-62.
33. Naber, D. (2003). A rule-based style and grammar checker (pp. 5-7). GRIN Verlag.
34. Birn, J. (2000, December). Detecting grammar errors with Lingsoft's Swedish grammar checker. In Proceedings of the 12th Nordic Conference of Computational 1842 Linguistics (NODALIDA 1999) (pp. 28-40).
35. Tschichold, C., Bodmer, F., Cornu, E., Grosjean, F., Grosjean, L., Kubler, N., ... & Tschumi, C. (1997). Developing a new grammar checker for English as a second language. In From Research to Commercial Applications: Making NLP Work in Practice.
36. Carlberger, J., Domeij, R., Kann, V., & Knutsson, O. (2004). The development and performance of a grammar checker for Swedish: A language engineering perspective. Natural language engineering, 1(1).
37. Lawley, J. (2003). The developement of a grammar checker for Spanish secondary students of English as a foreing language. Revista española de lingüística aplicada, (16), 127-138.
38. Shaalan, K. F. (2005). Arabic GramCheck: A grammar checker for Arabic. Software: Practice and Experience, 35(7), 643-665.
39. Wagner, J., Foster, J., & van Genabith, J. (2007, June). A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-1867 CoNLL) (pp. 112-121).
40. Bal, B. K., Shrestha, P., Pustakalaya, M. P., & PatanDhoka, N. (2007). Architectural and system design of the Nepali grammar checker. PAN Localization Working Paper.
41. Ehsan, N., & Faili, H. (2011). Statistical Machine Translation as a Grammar Checker for Persian Language. In Proceedings of the Sixth International Multi-Conference on Computing in the Global Information Technology (pp. 20-26).

42. Kaur, J., Garg, K., & Tech, M. (2014). Hybrid approach for spell checker and grammar checker for Punjabi. vol, 4, 62-67.
43. Madi, N., & Al-Khalifa, H. S. (2018). A Proposed Arabic Grammatical Error Detection Tool Based on Deep Learning. *Procedia computer science*, 142, 352-355.
44. Kernick, P., & Powers, D. M. (1996). A Statistical Grammar Checker (Doctoral dissertation, Honours Thesis, Flinders University of South Australia).
45. Go, M. P., & Borra, A. (2016, October). Developing an unsupervised grammar checker for Filipino using hybrid N-grams as grammar rules. In *Proceedings of the 30th Pacific Asia Conference on Language, Information and Computation: Oral 1888 Papers* (pp. 105-113).
46. Huang, C. C., Chen, M. H., Huang, S. T., & Chang, J. S. (2011, June). Edlt: a broad-coverage grammar checker using pattern grammar. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations* (pp. 26-31). Association for Computational Linguistics.
47. Mozgovoy, M. (2011, September). Dependency-based rules for grammar checking with Language Tool. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)* (pp. 209-212). IEEE.
48. Gill, M. S., & Lehal, G. S. (2008, August). A grammar checking system for Punjabi. In *22nd International Conference on Computational Linguistics: Demonstration Papers* (pp. 149-152). Association for Computational Linguistics.
49. Schmidt-Wigger, A. (1998). Grammar and style checking for German. In *Proceedings of CLAW (Vol. 98, pp. 76-86)*.
50. Giri, K. K., & Tekchandani, R. G. (2017). Efficient Rule-Based Grammar Checker with Word Sequencing (Doctoral dissertation, Thapar University).
51. Oco, N., & Borra, A. (2011, November). A grammar checker for Tagalog using Language Tool. In *Proceedings of the 9th Workshop on Asian Language Resources* (pp. 2-9).
52. Prat, N., Comyn-Wattiau, I., & Akoka, J. (2014, June). Artifact Evaluation in Information Systems Design-Science Research-a Holistic View. In *PACIS* (p. 23).
53. Monge, R. A., Marco, F. S., & Cervigon, F. T. (2002). An assessment of the consistency for software measurement methods. *arXiv preprint cs/0204014*.
54. Silva, W.D.C.M. and Finger, M., 2013. Improving CoGrOO: the Brazilian Portuguese Grammar Checker. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*.
55. Huang, S. and Wang, H., 2016, December. Bi-LSTM neural networks for Chinese grammatical error diagnosis. In *Proceedings of the 3rd Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA2016)* (pp. 148-154).
56. Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication*

association.

57. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug), 2493-2537.
58. Sundermeyer, M., Ney, H., & Schlüter, R. (2015). From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3), 517-529.
59. Soutner, D., & Müller, L. (2013, September). Application of LSTM neural networks in language modelling. In *International Conference on Text, Speech and Dialogue* (pp. 105-112). Springer, Berlin, Heidelberg.
60. Tran, K., Bisazza, A., & Monz, C. (2016). Recurrent memory networks for language modeling. *arXiv preprint arXiv:1601.01272*.
61. De Mulder, W., Bethard, S., & Moens, M. F. (2015). A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1), 61-98.
62. Sundermeyer, M., Schlüter, R., & Ney, H. (2014). *rwthlm—the RWTH Aachen University neural network language modeling toolkit*. In *Fifteenth annual conference of the international speech communication association*.
63. Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.
64. Yao, K., Zweig, G., Hwang, M. Y., Shi, Y., & Yu, D. (2013, August). Recurrent neural networks for language understanding. In *Inter speech* (pp. 2524-2528).
65. Sundermeyer, M., Oparin, I., Gauvain, J. L., Freiberg, B., Schlüter, R., & Ney, H. (2013, May). Comparison of feedforward and recurrent neural network language models. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 8430-8434). IEEE.
66. Sutskever, I., Martens, J., & Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 1017-1024).
67. Inan, H., Khosravi, K., & Socher, R. (2016). Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*.
68. Birn, J. (2000, December). Detecting grammar errors with Lingsoft's Swedish grammar checker. In *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)* (pp. 28-40).
69. Holan, T., Kubon, V., & Plátek, M. (1997, March). A prototype of a grammar checker for Czech. In *Fifth Conference on Applied Natural Language Processing* (pp. 147-1987 154).
70. Holan, T., Kubon, V., & Plátek, M. (1997, March). A prototype of a grammar checker for Czech. In *Fifth Conference on Applied Natural Language Processing* (pp. 147-1991 154).

71. Fortmann, C., & Forst, M. (2004). An LFG grammar checker for CALL. In InSTIL/ICALL Symposium 2004.
72. Starlander, M., & Popescu-Belis, A. (2002, May). Corpus-based Evaluation of a French Spelling and Grammar Checker. In LREC.
73. Burston, J. (2013). Review of BonPatron: An online spelling, grammar, and expression checker. *Calico Journal*, 25(2), 337-347.
74. Carlberger, J., Domeij, R., Kann, V., & Knutsson, O. (2002). A Swedish grammar checker.
75. Kinoshita, J., Salvador, L. N., Menezes, C. E., & Silva, W. D. C. (2007, October). Cogroo-an open office grammar checker. In *Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)* (pp. 525-530). IEEE.
76. Oliva, K. (1997, March). Techniques for accelerating a grammar-checker. In *Fifth Conference on Applied Natural Language Processing* (pp. 155-158).
77. Kaur, J., & Garg, K. (2014). Hybrid approach for spell checker and grammar checker for Punjabi. *International Journal*, 4(6).
78. Lin, N. Y., Soe, K. M., & Thein, N. L. (2011, December). Developing a chunk-based grammar checker for translated English sentences. In *Proceedings of the 25th Pacific Asia Conference on Language, Information and Computation* (pp. 245-254).
79. Bal, B. K., Shrestha, P., Pustakalaya, M. P., & PatanDhoka, N. (2007). Architectural and system design of the Nepali grammar checker. *PAN Localization Working Paper*.
80. Temesgen, A. (2013). *Design and Development of Amharic Grammar Checker* (Doctoral dissertation, Addis Ababa University).
81. Go, M. P., & Borra, A. (2016, October). Developing an unsupervised grammar checker for Filipino using hybrid N-grams as grammar rules. In *Proceedings of the 30th Pacific Asia Conference on Language, Information and Computation: Oral Papers* (pp. 105-113).
82. Go, M. P., Nocon, N., & Borra, A. (2017, November). Gramatika: A grammar checker for the low-resourced Filipino language. In *TENCON 2017-2017 IEEE Region 10 Conference* (pp. 471-475). IEEE.
83. Lin, N. Y., Soe, K., & Thein, N. (2011). Chunk-based grammar checker for detection translated English sentences. *International Journal of Computer Applications*, 28(1), 7-12.
84. Jones, J., Bosch, S.E., Pretorius, L. and Prinsloo, D., 2005. Development of reusable resources for Human Language Technologies (HLT) applications: practice and experience. *South African Journal of African Languages*, 25(2), pp.141-159.
85. Salis, C. and Edwards, S., 2008. Comprehension of wh-questions and declarative sentences in agrammatic aphasia: The set partition hypothesis. *Journal of Neurolinguistics*, 21(5), pp.375- 399.
86. Yilmaz, K., 2013. Comparison of quantitative and qualitative research traditions: Epistemological, theoretical, and methodological differences. *European Journal of Education*, 48(2), pp.311-325.

87. R.Pretorius and L.Pretorius, "Setswana Tokenisation and Computational Verb Morphology : Facing the Challenge of a Disjunctive Orthography," Computational, no. March, pp. 66–73, 2009.
88. Dulay, H. B., Burt, M., & Krashen, S. S.(1982). Language two.
89. S. Rajasekar, P. Philominathan, and V. Chinnathambi, "Research Methodology," ISRC, no. 10298, pp. 1–22, 2006.
90. Nfila, B. I. (2006). Standard in Setswana in Botswana (Doctoral dissertation).
91. Keet, C. M. (2016). An assessment of orthographic similarity measures for several Africanlanguages. arXiv preprint arXiv:1608.03065.
92. Tesfaye, D. (2011). A rule-based Afan Oromo Grammar Checker. IJACSA Editorial.
93. Zachman, J. A. (1987). A framework for information systems architecture. IBM systemsjournal, 26(3), 276- 292.

Appendix 1: The Setswana Grammar Checker

Figure 3.1a: Corpus that contains correct and incorrect Setswana sentences

p tshwanetse go tlhaloganya dipuo tse pedi tse o ranolelang mo go tsone.
Go o tsone tlhaloganya dipuotshwanetse go tse pedi tse o ranolelang mo .
Fa o ranola o seka wa ranola lefoko ka lefoko, lebelela bokao jwa se go buiwang ka sone.
Bokao fa jwa o seka wa ranola lefoko, lebelela se go buiwang ranola o ka sone lefoko ka.
Fa go se na lefoko le le tsamaelanang le le o batlang go le ranola, itirele le la gago.
Fa la gago go se na lefoko le le le o le ranola, itirele le le tsamaelanang batlang go.
Maina a mafelo le a batho ga a ranole.
Maina a mafelo le a batho ga a ranole .
Ga gana thamolo e go tweng e phepa.
Ga e go phepa gana thamolo tweng e.
Mafisa ke dikgomo tse di neelwang mongwe yo o se nang natso go di tlhokomela, le go ithusa ka tsone.
Mafisa ithusa ka di tlhokomela, ke dikgomo yo o se nang natso go le go tsone di neelwang tse mongwe.
Leruo le le isiwang mafisa ke dikgomo.
Leruo dikgomo le mafisa ke le isiwang.

Figure 3.1b: negative and positive text labels each corresponding to the grammatical statues of sentencesgiven in the text corpus

positive
negative
positive
negative
positive
negative
positive
negative
positive
negative
positive
negative
positive
negative
positive
negative

Figure 3.4: Functions imported from the Keras and TensorFlow libraries

```
In [90]: import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset
from torch multiprocessing import Process
import torchvision.transforms as transforms
import torchvision.datasets as dsets
```

Figure 3.5: Data loading process (text files being loaded into the jupyter notebook IDE)



Figure 3.6: A code snippet of the results gathered after printing six sentences, the sentences are printed together with their corresponding labels

```
In [172]: print(set[:500])
          print()
          print(classification[:200])
```

O tshwanetse go tlhaloganya dipuo tse pedi tse o ranoletang mo go tsone.
Go o tsone tlhaloganya dipuotshwanetse go tse pedi tse o ranoletang mo .
Fa o ranola o seka wa ranola lefoko ka lefoko, lebelela bokao jwa se go buiwang ka sone.
Bokao fa jwa o seka wa ranola lefoko, lebelela se go buiwang ranola o ka sone lefoko ka.
Fa go se na lefoko le le tsamaelanang le le o batlang go le ranola, itirele le la gago.
Fa la gago go se na lefoko le le le o le ranola, itirele le le tsamaelanang batlang

positive
negative
positive
negative
positive
negative

Figure 3.7: A code snippet that shows the process of the removal of all punctuation marks and removing new line characters from the Setswana sentences in the corpus

```
In [165]: #removing punctuations
          from string import punctuation
          print('Punctuations: ', punctuation)

          set = set.lower()
          all_text = ''.join([ch for ch in set if ch not in punctuation])

          Punctuations: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Figure 3.8: Shows the code snippet used for removing outliers from the data (Corpus)

```
In [130]: print('Number of sentences before removing the outliers: ', len(set_ints))

#removing the sentences with zero length
non_zero_idx = [ii for ii, rev in enumerate(set_ints) if len(rev)!=0]

set_ints = [set_ints[ii] for ii in non_zero_idx]
classification_encoded = np.array([classification_encoded[ii] for ii in non_zero_idx])

print('Number of sentences after removing the outliers: ', len(set_ints))

Number of sentences before removing the outliers: 69
Number of sentences after removing the outliers: 66
```

Figure 3.9: A code snippet of the embedding layer of the RNN-LSTM instantiated with its hyperparameters before tuning

```
In [97]: #instantiate the model with hyper parameters
vocab_size = len(vocab2int)+1 #+1 for the padding 0
output_size = 1
embedding_dim = 100
hidden_dim = 256
n_layers = 2

model = GrammarRNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)
print(model)
```

Figure 3.10: The Training module together with its defined and instantiated parameters before tuning

```
In [98]: def train(model, trainloader, validloader, batch_size, clip=5, print_every=2, epochs=25, lr=0.002):

    if cuda_available:
        model = model.cuda()

    criterion = nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    counter = 0
    for e in range(epochs):
```

Figure 3.11: Results after the execution of the training module before tuning hyper parameters

```
In [49]: train(model, trainloader, validloader, batch_size)
Epoch: 22/25 Training Loss: 4.768396593135549e-06 Validation loss: 0.18107135593891144 Validation Acc: 0.571
Epoch: 22/25 Training Loss: 0.2240230143070221 Validation loss: 0.17261210083961487 Validation Acc: 0.571
Epoch: 22/25 Training Loss: 4.410759629536187e-06 Validation loss: 0.16287943720817566 Validation Acc: 0.571
Epoch: 22/25 Training Loss: 6.52079188512289e-06 Validation loss: 0.15941151976585388 Validation Acc: 0.571
Epoch: 22/25 Training Loss: 6.914210189279402e-06 Validation loss: 2.121927764164866e-06 Validation Acc: 0.714
Epoch: 23/25 Training Loss: 4.0054605960904155e-06 Validation loss: 2.0861648408754263e-06 Validation Acc: 0.714
Epoch: 23/25 Training Loss: 6.806913006585091e-06 Validation loss: 0.16616874933242798 Validation Acc: 0.571
Epoch: 23/25 Training Loss: 0.07702189683914185 Validation loss: 1.9907970454369206e-06 Validation Acc: 0.714
Epoch: 23/25 Training Loss: 2.3603511181136128e-06 Validation loss: 0.16954800486564636 Validation Acc: 0.571
Epoch: 23/25 Training Loss: 0.12705084681510925 Validation loss: 2.0265599687263602e-06 Validation Acc: 0.714
Epoch: 24/25 Training Loss: 2.5868453121802304e-06 Validation loss: 0.1808966100215912 Validation Acc: 0.571
Epoch: 24/25 Training Loss: 6.556542757607531e-06 Validation loss: 0.18577606976032257 Validation Acc: 0.571
Epoch: 24/25 Training Loss: 0.2431056797504425 Validation loss: 0.19117292761802673 Validation Acc: 0.571
Epoch: 24/25 Training Loss: 6.198922164912801e-06 Validation loss: 0.19850635528564453 Validation Acc: 0.571
Epoch: 24/25 Training Loss: 2.3961104034242453e-06 Validation loss: 0.20950749516487122 Validation Acc: 0.571
Epoch: 25/25 Training Loss: 3.743182332982542e-06 Validation loss: 0.21917064487934113 Validation Acc: 0.571
Epoch: 25/25 Training Loss: 0.03776707872748375 Validation loss: 0.23333041369915009 Validation Acc: 0.571
Epoch: 25/25 Training Loss: 4.112733222427778e-06 Validation loss: 0.24610674381256104 Validation Acc: 0.571
Epoch: 25/25 Training Loss: 0.15345944464206696 Validation loss: 0.2562762200832367 Validation Acc: 0.571
Epoch: 25/25 Training Loss: 0.35225623846054077 Validation loss: 0.2547113299369812 Validation Acc: 0.571
```

Figure 3.12: The testing module together with the results obtained after its execution before tuning hyper parameters

```
In [28]: test_losses = []
num_correct = 0
criterion = nn.BCELoss()

test_h = model.init_hidden(batch_size)
for x, y in testloader:
    if cuda_available:
        x, y = x.cuda(), y.cuda()

    test_h = tuple([each.data for each in test_h])

    output, test_h = model(x, test_h)
    test_loss = criterion(output.squeeze(), y.float())

    test_losses.append(test_loss.item())

    pred = torch.round(output)
    correct = pred.eq(y.float().view_as(pred))
    correct = np.squeeze(correct.cpu()).numpy() if cuda_available else correct.numpy()

    num_correct += np.sum(correct)

print('Test Loss: {}'.format(np.mean(test_losses)))
print('Test Acc: {:.3f}'.format(num_correct/len(testloader.dataset)))

Test Loss: 0.4865645468235016
Test Acc: 0.429
```

Figure 3.13: A code snippet of the LSTM layer after tuning hyper parameters

```
In [22]: #instantiate the model with hyper parameters
vocab_size = len(vocab2int)+1 #+1 for the padding 0
output_size = 1
embedding_dim = 100
hidden_dim = 128
n_layers = 1

model = GrammarRNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)
print(model)

C:\Users\moe\Anaconda3\lib\site-packages\torch\nn\modules\rnn.py:51: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.3 and num_layers=1
  "num_layers={}".format(dropout, num_layers))

GrammarRNN(
  (embedding): Embedding(73, 100)
  (lstm): LSTM(100, 128, batch_first=True, dropout=0.3)
  (fc): Linear(in_features=128, out_features=1, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
)
```

Figure 3.14: The Training module after tuning hyper parameters

```
def train(model, trainloader, validloader, batch_size, clip=5, print_every=2, epochs=11, lr=0.001):

    if cuda_available:
        model = model.cuda()

    criterion = nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    counter = 0
    for e in range(epochs):

        hidden = model.init_hidden(batch_size)
        for x, y in trainloader:
            model.train()
            counter += 1

            if cuda_available:
                x, y = x.cuda(), y.cuda()

            hidden = tuple([each.data for each in hidden])
            output, hidden = model(x, hidden)
            loss = criterion(output.squeeze(), y.float())
            optimizer.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), clip)
            optimizer.step()

        if counter % print_every == 0:

            val_h = model.init_hidden(batch_size)
            val_losses = []
            num_correct = 0
            model.eval()

            for x, y in validloader:

                val_h = tuple([each.data for each in val_h])

                if cuda_available:
                    x, y = x.cuda(), y.cuda()

                output, hidden = model(x, val_h)
                loss_valid = criterion(output.squeeze(), y.float())
                val_losses.append(loss_valid.item())

                #convert output to probabilities 0 or 1
                pred = torch.round(output.squeeze())
                correct = pred.eq(y.float().view_as(pred))
                correct = np.squeeze(correct.cpu().numpy()) if cuda_available else correct.numpy()
                num_correct += np.sum(correct)

            print('Epoch: {}/{}'.format(e+1, epochs),
                  'Training Loss: {}'.format(loss.item()),
                  'Validation loss: {}'.format(np.mean(val_losses)),
                  'Validation Acc: {:.3f}'.format(num_correct/len(validloader.dataset)))
```

Figure 3.15: Results obtained from executing the training module after tuning hyper parameters

```
In [126]: train(model, trainloader, validloader, batch_size)
Epoch: 11/11 Training Loss: 0.28596431010921997 Validation loss: 0.047831929135474426 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.11727525293827057 Validation loss: 0.047974792956237626 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.1683734953403473 Validation loss: 0.0480856527633745 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.13892164826393127 Validation loss: 0.04816826474699851 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.00013448116078507155 Validation loss: 0.048284323949166506 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 2.6727106160251424e-05 Validation loss: 0.048379441800320555 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 3.817168908426538e-05 Validation loss: 0.04849407353942618 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 5.320550917531364e-05 Validation loss: 0.048626590610878984 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.1467558741569519 Validation loss: 0.04871110577583108 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 2.244741153845098e-05 Validation loss: 0.04871076838903451 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 2.0575771486619487e-05 Validation loss: 0.04871046099769421 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.13814815878868103 Validation loss: 0.04877474687702274 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.16235078871250153 Validation loss: 0.04887952035668979 Validation Acc: 0.976
Epoch: 11/11 Training Loss: 0.15723223984241486 Validation loss: 0.0489699023833322 Validation Acc: 0.953
Epoch: 11/11 Training Loss: 0.32878485321998596 Validation loss: 0.04907671934442453 Validation Acc: 0.953
Epoch: 11/11 Training Loss: 0.13068287074565887 Validation loss: 0.04924267259700761 Validation Acc: 0.953
Epoch: 11/11 Training Loss: 3.322426346130669e-05 Validation loss: 0.04940800521955466 Validation Acc: 0.953
Epoch: 11/11 Training Loss: 0.15064077079296112 Validation loss: 0.049403365774143074 Validation Acc: 0.953
Epoch: 11/11 Training Loss: 2.9051781893940642e-05 Validation loss: 0.04938135749141699 Validation Acc: 0.953
Epoch: 11/11 Training Loss: 0.13955703377723694 Validation loss: 0.04933326316885916 Validation Acc: 0.953
```

Figure 3.16: The testing module together with the results obtained after its execution

```
In [127]: test_losses = []
num_correct = 0
criterion = nn.BCELoss()

test_h = model.init_hidden(batch_size)
for x, y in testloader:

    if cuda_available:
        x, y = x.cuda(), y.cuda()

    test_h = tuple([each.data for each in test_h])

    output, test_h = model(x, test_h)
    test_loss = criterion(output.squeeze(), y.float())

    test_losses.append(test_loss.item())

    pred = torch.round(output)
    correct = pred.eq(y.float().view_as(pred))
    correct = np.squeeze(correct.cpu().numpy()) if cuda_available else correct.numpy()

    num_correct += np.sum(correct)

print('Test Loss: {}'.format(np.mean(test_losses)))
print('Test Acc: {:.3f}'.format(num_correct/len(testloader.dataset)))

Test Loss: 0.06110359087773866
Test Acc: 0.959
```

Figure 3.17: The Output layer and its defined parameters

```
In [160]: def predict(model, set, seq_length):

    rev = pre_process(set)
    rev = torch.from_numpy(rev)
    if cuda_available:
        model = model.cuda()
        rev = rev.cuda()

    p_h = model.init_hidden(1)
    output, hidden = model(rev, p_h)

    pred = torch.round(output.squeeze())

    if pred.item() == 1:
        print('correct')
    else:
        print('Wrong.')

    print('The sentence was predicted with an accuracy of {}'.format(output.item()))
```

Figure 3.18: Correct and Incorrect sentences fed as input to the grammar checker for classification

i). Example of a Correct Setswana sentence

The following correct Setswana sentence was fed as input to the grammar checker;

```
In [158]: set = 'o tshwanetse go tlhaloganya dipuo tse pedi tse o ranolelang mo go tsone.'
```

Results

The sentence above was classified as “correct” because it conforms to the word order followed by Setswanasentences.

```
In [161]: predict(model, set, 200)

correct
The sentence was predicted with an accuracy of 0.9485407471656799
```

ii). Example of an incorrect Setswana sentence

The following incorrect Setswana sentence was fed as input to the grammar checker;

```
In [178]:  
set = 'Go o tsone tlhaloganya dipuotshwanetse go tse pedi tse o ranoletang mo.'
```

Results

The sentence above was classified as “Wrong” because it does not conform to the word order followed by Setswana sentences.

```
In [181]: predict(model, set, 200)  
  
Wrong.  
The sentence was predicted with an accuracy of 0.0033421460539102554
```