JIMMA UNIVERSITY

JIMMA INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATICS

AFAAN OROMOO DEPENDENCY PARSER USING RECURRENT NEURAL NETWORKS

By

Mendasa Tesfa Hirkisa

A Thesis Submitted to School of Graduate Studies in Partial Fulfillment for the Degree of Master of Science in Information Technology

Jimma, Ethiopia

June 2022

Jimma University

Jimma Institute of Technology

Faculty of Computing and Informatics

AFAAN OROMOO DEPENDENCY PARSER USING RECURRENT NEURAL NETWORKS

By

Mendasa Tesfa Hirkisa

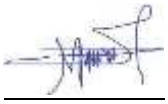
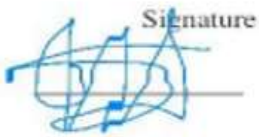
ADVISOR: Dr. Getachew Mamo (Ph.D.)

CO-ADVISOR: Mr. Mizanu Zelalem (MSc)

APPROVAL FORM

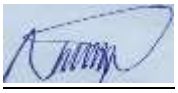
This is to confirm that the thesis prepared by **Mendasa Tesfa**, entitled **Afaan Oromoo Dependency Parser Using RNN,** submitted in partial fulfillment of the requirement for the Master's Degree of Science in *Information Technology* compiles with university regulation and meets the acceptance necessities with worthy to originality and quality.

This work was approved by the university advisors and board examiners.

Advisor: Getachew Mamo (Ph.D.)		11/11/2022
Name	Signature	Date
Co-Advisor: Mizanu Zelalem		11/11/2022
Name	Signature	Date
Chairperson: Muluken Yohannis	_____	11/11/2022
Name	Signature	Date
Internal examiner: Mamo Fideno	_____	11/11/2022
Name	Signature	Signature
External examiner: Dr. Teklu Urgessa		11/11/2014 E.C
Name	Signature	Date

DECLARATION

I state that the work which is being presented in this research entitled **Afaan Oromoo Dependency Parser Using RNN**, provided for evaluation and this thesis is my own, original work at school of computing, *Information Technology* department, and in no university or institute has been awarded for degree and that it has not been previously evaluated, and all the resource materials used for this thesis had been accordingly acknowledged.

MENDASA TESFA  _____

Signature Date

DEDICATION

This work is dedicated to my loving parents, particularly my father **Tesfa Hirkisa**, mother **Boggale Terefa**, and grandfather **Tefera Hika**, for their constant love, support, and words of encouragement.

ACKNOWLEDGMENT

In life, three sects of persons are most important, first the almighty God, the second parent, and the third friend. God almighty for the wisdom he bestowed upon me, the strength, peace of my mind, infinite grace, unfailing love, unwavering faith, good health, and for his shower of blessing throughout my thesis work to complete successfully. So, I thank my, GOD, forever.

I would like to convey my heartfelt gratitude to my research advisor, Dr. Getachew Mamo (Ph.D.), for inspiring me with his vision, sincerity, energy, and enthusiasm. He also educated me on how to conduct research and present my thesis work most straightforwardly and concisely possible. I also greatly full thank my co-Advisor **Mr. Mizanu Zelalem(**MSc**)**, for his limitless effort in showing directions through my work and for providing me with useful materials. I'd like to express my appreciation to all members of the school of computing, especially **Mr. Hambisa Mitiku (**Ass. professor), chair of the Information Technology department.

Dear all, I am very grateful to you who have shared your insights on the AO language. Particularly, I appreciate **Mr.Dereje Regassa**(*DEd candidate*), for your explanations of the AO morpheme and its structure.

The greatest blessing, I enjoyed in all these years is the selfless, unwavering, and unconditional love, strength, and support given to me by my parents especially my mother **Bogale Terefa**, my father **Tesfa Hirkisa,** and my relatives to complete this work. My thanks also to my caring siblings whose advice worked for this thesis paper.

At the last, but not least, I am thankful to everyone who assisted me directly or indirectly to complete this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AO: Afaan Oromoo

AODP: Afaan Oromoo dependency parser

AOTB: Afaan Oromoo treebank

POS: Part of speech

AOPOS: Afaan Oromoo part of speech

SOP: Statement of the problem

SOV: subject-object-verb

NLP: Natural language process

CONLL: Conference on computational natural language learning

NLU: Natural language understanding

NLTK: Natural Language Tool-Kit

CP: Constituent parsing

DP: Dependency parsing

XDP: Extendable dependency

DG: Dependency grammar

RA: Right arc

LA: Left arc

σ: Stack

β: Buffer

S: Shift

Ci: initial configurations

Cm: intermediate configurations

Cf: Final configurations

Ti: Initial transition

Tm: Intermediate transitions

Tf: Final transitions

H/h: head

D/d: dependent

R: Relations

IT: Information Technology

JU: Jimma University

RNN: Recurrent neural network

LSTM: long short-term memory

BI-LSTM: Bi-directional long short term memory

$\mathbf{i}_{t=}$ Input gate

$\mathbf{f}_{t=}$ Forget gate

$\mathbf{o}_{t=}$ Output gate

GD: Gradient descent

VGD: Vanishing gradient decent

LAS: Labeled attachment score

UAS: Unlabeled attachment score

DEd: Doctor of education

DEFINITIONS

The first model (model one (1)): Transition predictor model

The second model (model two (2)): Relation predictor model

Stream POS: Particular part of speech represented as one

Specific POS: Represents each particular part of speech

Stream Relations: Particular relations types are represented as one

Specific Relations: Represents each relations types

Sub-model: either transition predictor or relation predictor model

M1: model one

M2: model two

ABSTRACT

Dependency parsing is an act of extracting the relations among the words or morphemes using dependency type to resolve ambiguities among the head and its modifiers. Humans use facial expressions, tone of speech, body language, and others to make a natural language more clear and understandable. Unlike humans, machines need well-formed and studied language structures for both natural language understanding and generations. This was achieved through developing natural language processing applications. Hence Afaan Oromoo dependency parser was developed to resolve and clarify misunderstandings among Afaan Oromoo morphemes. Even though, constituent parsers and universal dependency parsers exist they are not effective to handle morpheme information. Among dependency parser approaches, data driven approach was selected in Afaan Oromoo dependency parser to obtain morphemes and word order features in Afaan Oromoo. From a data-driven approach transition system was selected for its simplicity and fast performance than graph-based dependency parsers. Particularly arc-standard is used to generate an unlabeled dependency graph. Afaan Oromoo dependency parser was developed from two sub-models that work self-reliantly. The first one is used to predict the transition and then generates an unlabeled dependency graph (tree). The second one is used to predict the relation types and generate a labeled dependency graph. RNN algorithm was selected to handle sequences of Afaan Oromoo morphemes and extract the language patterns. The treebank was constructed from **500** sentences and in the first model **3480** and **1740** instances of configurations were used for training and test data. In the second model **1000** and **415** (head-dependents) were used for training and test purposes. Consequently, LSTM and BILSTM had experimented and the BILSTM has shown better accuracy for classifications of both transitions and relations. The first model performs an accuracy of **90**% using BILSTM and **89**% using LSTM. Next, the second model scored **71**% for BILSTM and **69**% for LSTM. Additionally, using BILSTM the model scores **60**% for **UAS** and **40**% for **LAS**. To sum up, the performance of the deep learning models is directly proportional to corpus size. And also increasing dependency labels enhances clarifications between the morphemes.

Keywords: *Transition predictor**,** Relation predictor, Root*

CHAPTER ONE

1. INTRODUCTION

1.1 Background

Language is a means of communication mainly used for transmitting ideas, persuading, opinions expression, giving orders, asking for information, and also it serves as a symbol of national identity [1]. Languages are broadly categorized as artificial languages and natural languages. Artificial languages are created for a specific purpose while natural language is any ordinary language used by humans to coordinate with each other in their day-to-day daily activities [2]. Communication is carried out in the form of spoken which is the primary medium for human beings and written mainly used to pass the knowledge from one generation to the next [2]. The level of development of the language matters for effective use of the language. In today's world, not only humans but machines can understand and generates natural languages if well studied. Linguists are responsible to study and state the implicit and explicit structures of the language. Natural language processing is a branch of computer science and artificial intelligence that studies how computers and humans interact in natural language [3].

NLP's ultimate goal is to enable computers to understand language as humans do. Today's natural language processing applications are more powerful using deep learning algorithms and possible to handle the language patterns and solve the misunderstanding or presence of two or more possible meanings of the relations between the AO morphemes [4]**.** Virtual assistants, speech recognition, sentiment analysis, automatic text summarization, and machine translation are all excellent NLP applications that are employed in developed languages. So, lower-level NLP applications are utilized to feed higher-level NLP applications. Due to AO's limited computing resources, additional study is needed at phases of NLP such as lexical, syntactic, semantic, discourse, and pragmatic analysis [5]. Consequently, resolving ambiguity at the syntactic level is critical for AO. Parsing is a method of analyzing the sentence structure, content, and meaning for a better understanding of the meaning, structure, content, and syntactical or semantic relationship of constituents within a sentence [5]. It is possible to see the parsing concept in two major classes syntactic and semantic parsing. Particularly, syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it while semantic analysis refers to meaning representations assigned to sentences solely based on knowledge gained from the lexicon and grammar. There are two common types of syntactic

parsing: dependency and constituency parsing [6]. A constituency parser breaks a sentence into sub-phases based on phrase structure grammar while a dependency parser links words in a sentence and assigns labels to these relations based on dependency grammar [6]. So, the focus of this research is on dependency parsing for Afaan Oromoo sentences.

Dependency parsing characterizes the head-dependent relationship between words in a sentence classified by functional categories or dependency relation type [7]. This enhances semantic information and is better for ambiguity resolution because of the labeled relation among the words. AO is a morphologically rich language [8]. Then, it is not simple to understand the relations between AO morphemes. For instance, in "Caalaa n sang oota bit e." the issue of understanding each morpheme feature is not easy. But the relation types explicitly clarify them as Caalaa n: **NCM** (noun class marker), sang oota: **NUM** (number), bit e: **TNS** (tense) [8]. Additionally, words may represent different meanings in a given sentence based on their arrangement. This implies the prevalence of misinterpretations of the word within a sentence. Here, 'Caalaa' is in **subject** place, 'sang' is in **object** place and 'bit' is the **Root** word of the sentence [9] [8].

The dependency parsing follows data-driven or grammar-based approaches [7]. The grammar-based approach requires more knowledge of rules about the grammar of the language. In a data-driven approach, the language patterns are obtained from the data set of the language. In this approach, two methods are used to develop the dependency parser. Those are transition-based methods and graph-based methods. The transition method is fast, simple, has less storage, and works for projective languages [10]. While graph-based is more accurate than transition systems [7]. In the case of the transition-based methods, transition techniques such as arc standard, arc eager, arc hybrid, and arc swifts are seen. This arc standard uses three transition actions and is faster and more effective than others [10]. A recurrent neural network that is state of the art for relation and transition classifier is used. It learns from and performs on AO sequential data. Additionally, among the RNN algorithms, the one which improves accuracy is experimented with and identified for Afaan Oromoo dependency parser.

As a whole, it is fast as parsing is straightforwardly done on words without considering the phrase structure of the sentences, grammar rules of words, or sentence formation [11]. It has great advantages in other NLP application developments.

1.2 Motivation

Afaan Oromoo has a higher number of speakers [12]. It is also a regional working language. Additionally, it is going to be the second working language in the federal working language. So many peoples are also eager to learn the language. The relationships between words, and among components of the words, have to be clearly stated for simplicity in learning. Also, it is a good start to make or develop into computational language, hence upper levels used this parser which states the relations between the words and within the word. Hence, developing applications that clarify misunderstanding is very important. Furthermore, Google company has recently recognized the importance of this language, and Google Translator has been developed just for it. This is a decent start, but it isn't quite enough for the development of the language. More effective NLP applications will be necessary to advance the language [3]. As a result, the researcher motivated and conducted his study with syntactic analysis using AODP.

1.3 Statement of the problem

Afaan Oromoo is a popular language that is spoken by over 50 million people [12]. Currently, it is the official language of the Oromia regional state. Both governments and societies are attempting to develop the language to be added to the country's official language. The attempt is to make both conventional and computer language. Even though Afaan Oromoo has a high number of speakers, it is not matured enough and requires further study [12]. Many are enthusiastic to learn the language but it is difficult to understand the details of the morphemes information in the language. Knowing the patterns or cores of morpheme features is key to mastering the language. Although constituent parsers were developed earlier, no one has tried the dependency parser for Afaan Oromo previously. So the relations between the words and within the words (at morpheme level) were unable to be clear for the language users. Hence AODP was developed to fill the gaps in constituent parsers, and universal dependency parsers by handling AO morphemes features. Next, in comparison with other parsers, compatible, effective, and fast transition system (arc-standard transitions technique) was selected. Additionally, in terms of the methodology using BILSTM for AODP makes it special in handling the detailed relations in sequences of morphemes in the past and future [13]. The details are discussed as follows by comparing with the previous works.

In syntactic analysis, there were earlier constituent parsers. It's slower because it's based on phrase structure principles to analyze sentence structure [14]. Its relation isn't properly represented by relation types, therefore it's difficult to provide the precise meaning between words [15]. The relations between the morphemes and words are not stated using this parsing technique but as phrase structure the way it works to form a sentence.

Language independent (universal dependency parsers) exist but are ineffective for morphologically complex languages such as AO [16]. It was developed for English languages and shows the ineffective result for such languages [17]. For morphologically rich languages, adapting corresponding morphemes POS and the corresponding relationship is critical. If so possible to handle the relationship between the morphemes.

Some morpheme-based parsers [18] were ineffective for local languages as it is not effective to handle the details of language-dependent morpheme features. As a result, is not possible to handle all the relations among the morphemes of the languages using other language-oriented relations types. A limited number of the relationship between the morphemes restrict the prediction of relations effectively for morphologically rich languages [15].

Some parsers [19] used rule-based traditional techniques in parsing, it is unable to extract the language patterns and exhaustive to have the whole knowledge of the language to derive rules. Some parsers used machine learning algorithms that are poor in feature extractions and less language pattern recognition. The data-driven graph-based is used but not fast as a transition based [20]. Here arc-standard transition technique is more effective from AO SOV word arrangements, as the core word or root is found at the end of the sentence.

Eventually, dependency parsing acts as a prerequisite for NLP applications such as semantic role labeling, relation extraction, machine translation, and language users can use it starting from individuals to the whole society, etc. For instance, the grammar checker used it as input and enhances its effectiveness both in social media and in organizations. [7]. As it has clear which element is the head and which element is the dependent one explicitly through labels. More semantic information.

1.3.1 Research questions

1. To what extent arc-standard transition system handles AO morphemes?
2. Why BILSTM is more effective than LSTM in AODP?

1.4 Objectives of the study

1.4.1 General objective

The general objective of the study is to develop Afaan Oromoo dependency parser using RNN

1.4.2 Specific objectives

To achieve the general objective stated above the following specific tasks are required.

- ✓ To review literature
- ✓ To develop the dependency Treebank.
- ✓ To develop a model for Afaan Oromoo dependency parser.
- ✓ To train and test the model.
- ✓ To evaluate the result of model performances

1.5 Methodologies

1.5.1 Literature review

Reviewing papers enables the researcher to understand the level of knowledge in the area of the study and identify the gap that is not covered by previous works. Journal articles, books, conference papers, and websites were kinds of literature reviewed.

1.5.2 Data collection and preparation

The data source was identified and collected to construct Afaan Oromoo treebank. So The data was collected from various domains such as newspapers (FBC, VOA, OBN), books, websites, magazines, etc [21]. Then, Afaan Oromoo words and sentences were collected from Afaan Oromoo books such as sanyii, caasluuga, furtuu, and catrina [8] [9]. The books also served us to get a better understanding of Afaan Oromoo language structure as well as to derive sentences for our corpus. The data was prepared and tagged based on linguists' advice [22] [23] [2] [24]. First AO treebank was developed, then from the treebank, both configurations and head-dependent corpus were constructed. It was classified into training and test data. Then, it was used in the transition predictor and relation predictor sequentially.

1.5.3 Developmental tools

The model development tools include both hardware and software tools. Anaconda platform with Python 3.7 programming language was used [25]. Deep learning framework, TensorFlow is used at the back end and the Keras library is used at the front end [26]. Tensorflow stores tensor data and have the flexible numerical computation core and flow of the operation which can be useful in our case [26]. Keras library is very useful to work with RNN and convolutional neural networks [25] [27]. It provides a clean and easy way to create deep learning models based on Tensor flow. Additionally, we used python libraries such as Numpy, Matplotlib, etc in the study. Ms word, notepad++, and Jupiter notebooks were editors used in this study. Edraw max to design the model. Computer hardware CPU with Windows 10 OS was used.

1.5.4 Experimentation and testing

The models were constructed using prepared training and test data sets for both models. The features were extracted and prepared for the models. RNN algorithms were used and we experimented with LSTM and BILSTM to classify the transition type in model one and classify the relation type in the relation predictor model [4].

1.5.5 Evaluation

Accuracy is used to measure the effectiveness of transition prediction and relation prediction. The evaluation metrics such as Un labeled attachment score and labeled attachment score were used to evaluate the results head, dependent from the first model, and head-dependents, the label from the second model [7].

1.6 Scope and Limitation

The scope of the research includes dependency parsing for Afaan Oromoo projective morphemes types. It is good at parsing inflectional and derivational morphemes. Not includes non-projective and transitivity property of AO adjectives. This parser is not very effective for special case morphemes. The model is unable to parse hyphened words and punctuation marks.

1.7 Application of results

The result of the developed model helps or acts as a preprocessor for higher-level Afaan Oromoo NLP applications. Among them, semantic role labeling, identifies the argument, especially, in the very first stage the pruning stage [3]. Next, it also helps in question answering how a question is phrased and how sentences in documents are structured and potentially provides important clues for the matching of the question and answer candidates in the sentences. It also plays a great role in Grammar checking as it identifies whether a given sentence will be parsed or not according to the grammar of the language. The messages between the morphemes are seen, if correct accept, if not possible to understand from explicit relations and have to be corrected accordingly. So, if parsed it is grammatically correct for that language. Else, it has some errors [15].

1.8 Research Organization

The thesis is organized into seven chapters comprising an Introduction, Literature review, related works, an overview of Afaan Oromoo dependency parser design, Experimentation & Evaluation, and Conclusion & Recommendations. The first chapter gives a general introduction to the study. The second chapter presents a review made on different works(literature) regarding dependency parser (gives elaborations, knowledge on study area) together with its approaches and different machine learning techniques. The third chapter discusses the related works, the family of the study stated here to gain gab and fill it. The fourth chapter discusses the overview of Afaan Oromo's background knowledge and details of the language structures. All about techniques of word-formation are mainly seen. The fifth chapter discusses AODP design. It presents the algorithms and techniques used for parser development. Chapter six tells about the experimentation and discusses the results. Chapter seventh presents the conclusion and the recommendations as well as some directions for future works.

CHAPTER TWO

2. LITERATURE REVIEW

In this chapter, literature reviews elaborate on basic concepts in dependency parsing. Then it enables the reader to understand the knowledge in dependency parsing. Later it guides to select the articles and find out the research gap in the related work.

2.1 Morphology

Morphology is the study of the way words are built up from smaller meaning-bearing units, and morphemes [35]. Words are produced by bringing together base forms and affixes (stems and affixes). Affixes are also classified according to the result they produce. Derivational affixes produce new words that may change the part of speech. They are usually also distinct semantically from the original word [3]. Inflectional affixes produce new surface forms of a given word. They don't change its part of speech; they annotate the word with additional syntactic information [2]. Languages generally vary in the degree to which they use word order and morphological markers to highlight syntactic relationships.

2.2 Dependency Parsing

Dependency Parsing is the process to analyze the grammatical structure in a sentence and find out related words as well as the type of relationship between them [7] [3]. The speakers of the language have rules for calling morphemes with their patterns and the order of the words in their minds. This implies that there is grammar in the speaker's mind traditionally, Linguistics has stated these rules earlier as the relationship between the words has been stated based on some conditions or behaviors stated as characters of the head, dependents. hence the traditional grammarians started to state forms for syntactic representation as paniaias [3] [15].

Unlike traditional grammar, today some languages are computational languages and machines extract the relationships accordingly. Consequently, our final goal is also to enable machines to generate the dependency tree or graph for a given sentence. Dependency structure is used to show the head and modifiers for a given sentence [6]. If it was labeled it tells more about the relation type among the head and modifiers.

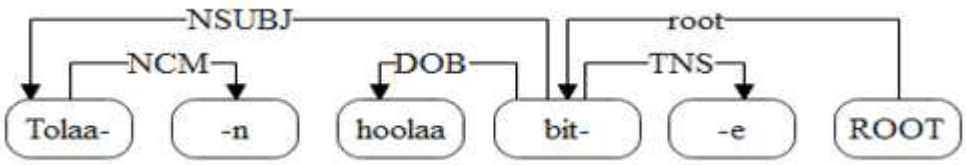


Figure 2-1: Sample dependency structure

In the above figure, the arrows represent the dependency between two words in which the word at the arrowhead points to dependent, and the word at the end of the arrow is head. The root word can act as the head of multiple words in a sentence but is not a child of any other word. You can see above that the word 'bit' has multiple outgoing arrows but none incoming. Therefore, it is the root word. 'Tolaa' head for modifier '**n'**, '**e**' is a modifier for root 'bit' [36] [8] [2]. Constraints must be fulfilled by Dependency structures including Connectedness, single-headiness**,** rooted, acyclicity, and projectivity behaviors of the words [15].

A dependency relation is a binary relation where a word, the dependent, depends on another word, the head [37]. These dependency relations can be further divided into various types of dependency, thus used to label the dependencies. Every word must have exactly one head and can have zero or more dependents. To satisfy this requirement, an artificial word called root is always added to each sentence which then acts as the head to the real head.

In dependency parsing, the tags represent the relation type between two words in a sentence. Among word-formation morphemes, compounding, and word order not only rules but explicit relation types are used. The number of tags is language-dependent. As of now, there are 37 universal dependency relations used in Universal Dependency, but languages can develop their tags in addition to universal tags based on the language structures [17]. In the above figure, the dependency relation is shown using the labeled graph. The relation type between '*bit'* and *'Tolaa'* is nominal subject **('*NSUBJ'),*** '*bit'* and *'Hoolaa'* is the direct object **('*DOB')***, '*bit'* and '*e'* is tense **('*TNS')** [2]. As a whole, the dependents modify the headwords through the labels and the root word from a given sentence is 'bit'. Binary, directed or antisymmetric, and anti-reflexive anti-transitive are conditions that must be fulfilled by the relation types [15].

2.2.1 Approaches of dependency parsing

The approaches used in dependency parsing are roughly classified into grammar-based and data-driven approaches [38]. The grammar-based is the traditional one. But today as deep learning techniques using data-driven is preferable. The details will be as follows.

2.2.1.1 Grammar-based versus data-driven

In contrast to the data-driven methods, grammar-based methods rely on explicitly defined formal grammar [11]. It is exhaustive and requires more knowledge about the language grammar to generate head, dependent relations among a given sentence [7].

2.2.1.2 Data-driven approach

The language structure and patterns will be obtained from the data. Dependency parser follows either transition-based method or graph-based method with such working techniques [10]. Due to this data-driven approach transition-based method performs well for projective languages and the graph-based method works for non-projective ones [39]. The sequences of configurations and transitions were used in the transition-based systems. Contrary, in the graph-based method, the global information or the weight of the edge or arc is in consideration and the optimal weight is selected [15]. An arc from a head to a dependent is said to be projective if there is non-cross a path from the head to every word that lies between the head and dependent in the sentence [15]. While projective is crossing arc pairs in a sentence. Mainly seen in long sentences in which modifiers further head rather than nearby words in sequential order of modifying [38].

2.2.1.2.1 Transition systems

A transition system is an abstract machine, consisting of a set of configurations (or states) and transitions between configurations [40]. In such a manner the dependency graph is constructed starting from the initial configuration to the final configuration by applying the sequences of transition actions. The four frequently raised transition techniques used are arc standard, arc eager, arc swift, and arc hybrid. They perform in different ways and their selection depends on the structures of the languages [10].

Arc standard transition system is a simple, effective, and fast transition technique when the operation is carried out within the stack only [10]. It uses 3 transition types shift, left arc, and right arc [10].

The arc eager uses 4 transition types particularly shift, right arc, left arc, and reduce [40]. The operation is carried out between the top of the stack and the buffer. Shift operation from the word on top of the buffer to the top of the stack when the word has no relationship. The left arc removes the word on top of the stack which is dependent on the headword. The right arc shifts the word on top of the buffer to the top of the stack if there is a relation between the top of the stack word and the top of the buffer. Reduce operation helps to remove already predicted relation from the stack and is used to arrive at the root word [10].

2.3 Classifiers for transition types and relation types

The researcher focused on a neural network (RNN) that handles the natural language patterns unlike the traditional techniques (rule-based and other machine learning techniques).

2.3.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a mathematical model that tries to simulate the structure and functionalities of natural neural networks [27]. At the entrance of an artificial neuron, the inputs are weighted. The next section of the artificial neuron is a sum function that sums all weighted inputs and biases. At the end of the artificial neuron, the sum of previously weighted inputs and bias is passed through an activation function and output is generated [41].

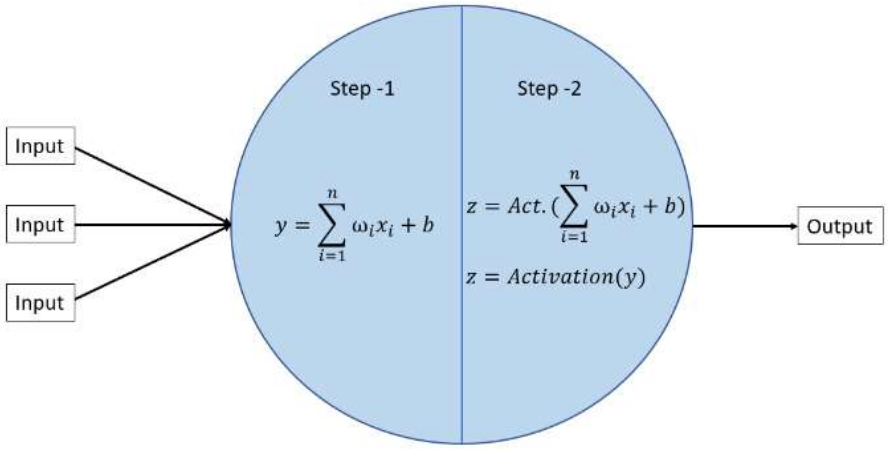


Figure 2-2: Architecture of ANN (Perceptron)

Deep learning algorithms are based on the concept of artificial neural networks. When many hidden layers of the neural networks are used its performance also increases [26] [32]. The deep learning approach is preferable as it can learn patterns from the data without explicit feature extractions like other machine learning algorithms [42]. A deep neural network provides state-of-the-art accuracy in many tasks. Deep Learning works in two phases forward propagation to generate the output and backward propagation improving the model by updating

the weight by applying a mathematical method known as derivative techniques. The general deep learning models follow the following pipelines [26].

Figure 2-3: How deep learning models work

At first, understand the problem you are going to solve using a deep learning approach. Next, identify sources and develop a dataset. Select the best performing deep learning algorithm for your data. Then, develop and train the deep learning model. Finally, test it and use it if it does well.

2.3.1.1 *Recurrent neural networks*

Traditional neural networks such as multilayer neural networks have a major limitation in considering sequential data [13]. There are dependencies among the words in a sentence. But multilayer neural network accepts them as independent of each other. To solve this constraint recurrent neural networks (RNN) are proposed. Recurrent neural networks (RNNs) have been widely used for processing sequential data [4]. Recurrent neural networks are multi-layered neural networks that can store information in context nodes, allowing them to learn data sequences and output a number or another sequence [41] [13]. Recurrent neural networks have a memory about what has been calculated so far and use it on current output computation. RNN works more based on nearby information while the initially inserted information will not mostly be considered. This indicates that RNNs suffer from long-term dependency problems [41].

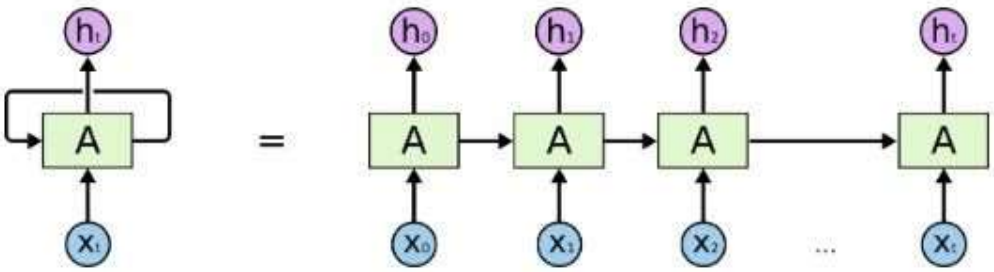


Figure 2-4: An unrolled recurrent neural network.

2.3.1.1.1 LSTM

But there are also cases where we need more context information. Initially, LSTMs handle the problems of long-term dependency, vanishing gradient, and exploding gradient in recurrent neural networks [43]. The heart of a network is its cell state which provides a bit of memory to the LSTM so it can remember the past. The LSTM can remove or add information to the cell state, which is prudently regulated by configurations called gates [29]. They are composed of an appoint-wise multiplication operation and a layer of sigmoid neural nets. Three of these gates are present in LSTMs to monitor and adjust the cell state, and the sigmoid layer produces numbers between 0 and 1 that indicate how much of each component should be permitted to move [43]. LSTM's first step is to choose which information from the cell state to reject. The forget gate layer, a sigmoid layer, decides on this activity. A number between 0 and 1 is output for each number in the cell state **C_t-1** after checking at h_t-1 and x_t. 1 characterizes 'completely preserve this' while a 0 represents 'completely get clear of this '. In the forget gate, a sigmoid function takes in x_t and h_t−1 to decide how much information should be kept or dropped.

$$f_t = \boldsymbol{\sigma}(wf([ht-1,xt]+bf)) \quad \mathbf{2}$$

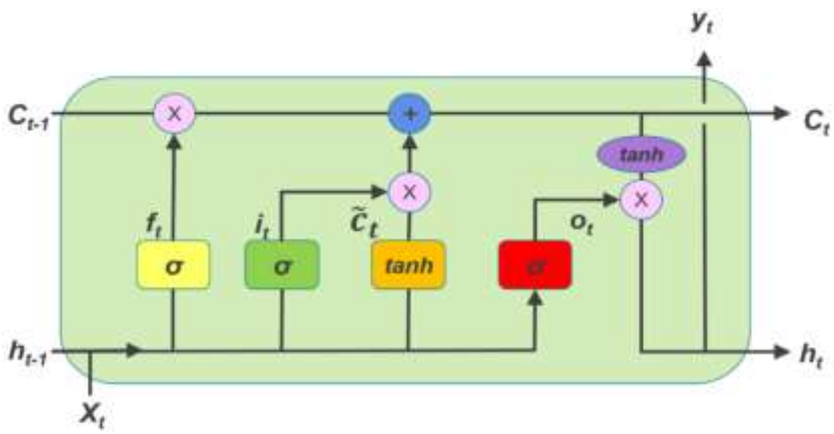


Figure 2-5: A block of LSTM at any timestamp {t}

Among the types of cells forget gate cell determines to what extent to forget the previous data. The input gate determines the extent of information being written onto the internal cell State The next step is to decide what new information we're going to store in the cell state [13].

$$i_t = \sigma(w_i([h_{t-1}, x_t] + b_i)) \quad 1$$

Finally, the output gate determines what output (next hidden state) to generate from the current internal cell state. Here are equations of gates that describe how it takes input and gives the final output [13] [29].

$$o_{t} = \sigma(w_o([h_{t-1}, x_t] + b_o) \quad \ldots 3$$

2.3.1.1.2 BILSTM

Bidirectional LSTM is based on the idea that the output at the time may not only depend on the preceding elements in the sequence, but also on future elements. It [29] resolves the problem by having two different LSTM(BILSTM). From the two, the first LSTM is fed with the input sequence while the second LSTM is fed with the input sequence in reverse. Next, the hidden state is composed of both the forward and backward states. Each state representation consists of the token information along with sentential context from both directions which has exposed improved results than LSTM [13].

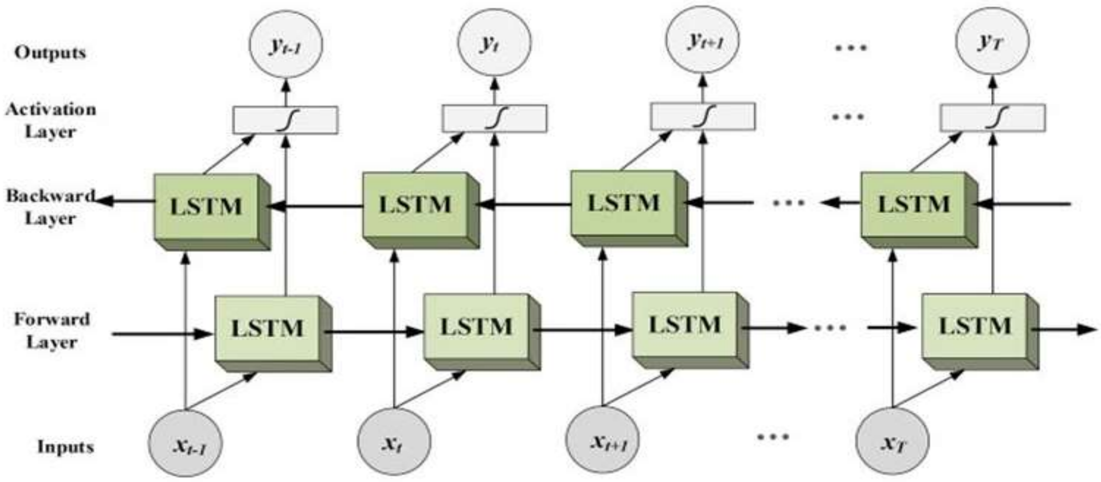


Figure 2-6: Forward and backward propagations in BILSTM

2.4 RNN model development phases

This gives the general pipelines to develop the RNN model in a simple way using the Keras framework [4]. The network architectures have to be clearly stated and have to be configured. So as indicated in deep learning model development the first step is to identify the data, based on the data going to be trained, the deep learning algorithm was selected which can handle the problem at hand [13]. Then the network or algorithm will be configured. Next, the designed or configured network will be trained with the data and tested for performance. In addition to this hyper-parameter are used and tuned in the training phase of model development. The models accept different input features and follow Keras functional API model development techniques.

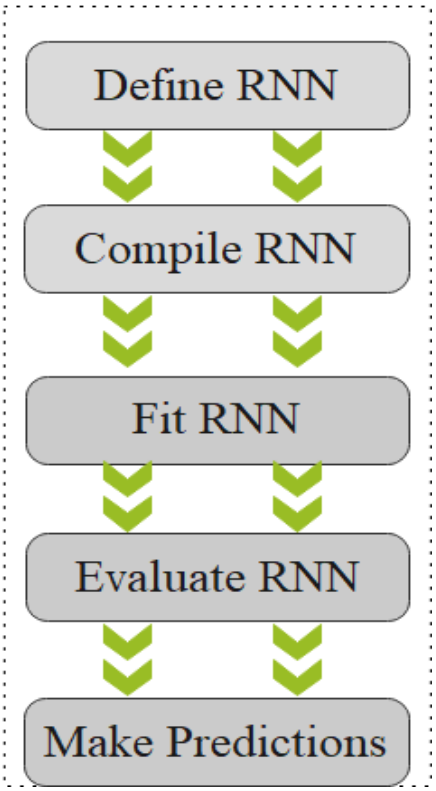


Figure 2-7: Steps in RNN model development using Keras

Based on this flow RNN model was developed. Defining the model is the initial step. Input and output layers will be defined. The first layer becomes an input for the second one and arrives at the output layer in the functional model [26] [25]. The next step is compiling the model which refers to transforming the layers into a matrix to be executed by CPU or GPU. Next, fitting the model is adapting the weight on a training dataset. Finally, evaluate the performance of our model on a separate dataset that is unseen. Once satisfied with the performance of our model then use it for the prediction of new data [25].

CHAPTER THREE

3. RELATED WORKS

Earlier, several studies have been carried out for both foreign language and local language dependency parsers. It is possible to see the others work under universal dependency parsers, morpheme-based dependency parsers, and transition-based dependency parsers using deep learning as follows.

3.1 Universal dependency parsers

Stanford dependency parser [17] was implemented using a graph-based method which is not as fast as the transitions system. It used LSTM for word embedding techniques and BILSTM, baffiane classifier is used for relation predictions. It also used a single model, unlike others that use two models in transition systems. Phase structure to dependency structure conversion takes a long time and makes it not fast [44]. But it acts as a baseline in dependency parsers for others. Those parsers can't support morpheme information for morphologically rich languages. So, scores less performance for such languages.

MST parser is also another graph-based parser. Those parsers can't support morpheme information for morphologically rich languages [39]. It scores less performance for such languages. It is implemented using complex machine learning techniques which affect its speed.

In malt parser a manual feature selection mechanism and adjusted by the user. It is machine learning, not deep learning, and can't support morphemes [16]. But, it can solve issues of projective and non-projectivity as a strong side.

It(ensemble) [45] combines the parsing algorithms such as nivres arc eager, Nivre's arc-standard, and Covington's non-projective model parses from left or right and vice versa. Seven different models are used and each parser runs in its thread. It performs voting mechanisms after parsing for the result. Then well-formed dependency tree is returned. This makes different it different from malt parser. This indicates that it can't support morphemes.

It(Clear Parser) [40] is more efficient and accurate than the malt parser because Clear Parser differentiates between projective and non-projective structures and can avoid unnecessary

searches. Gold-standard annotation and automatically parsed trees increase the corpus. Hence differ from malt parser.

3.2 Morpheme-based dependency parsers

Hebrew dependency parser is constructed from both transitions-based and graph-based parsers [18]. They used POS features and morphological information but doesn't show significant improvement. Morpheme information has to be more and more for the sake of increasing its performance.

It(Turkish) [19]is Rule-based dependency parsers are very restricted and can't be applied to other languages. But more performs if the rules are settled for the language accordingly. This technique is not state of the art in terms of methodology hence deep learning can handle the learn the features by itself. Setting the rules for the languages is also exhaustive.

The (Amharic) [20] investigated by using the arc eager transition system and LSTM network. The study introduced another technique of building a labeled dependency structure by using a separate network model to predict dependency relations. Evaluation of the parser model results in 91.54 and 81.4 unlabeled and labeled attachment scores respectively on the Amharic dependency tree bank.

Gasser [46] tried to develop the first dependency parser which is rule-based for the Amharic language for the first language. but it was not implemented. The work introduced the grammar of Amharic with the extensible dependency grammar framework of Debusmann.

3.3 Deep learning and Transition-Based Dependency Parsers

The researcher [47]introduced bidirectional transition-based dependency parsing. As a result, Transition-based dependency parsing is a fast and effective approach for dependency parsing. Bidirectional transition-based parsing learns a left-to-right parser and a right-to-left parser separately. To parse a sentence, they perform joint decoding with the two parsers. Finally, Experimental results show that this method leads to competitive parsing accuracy and this method based on dynamic oracle consistently achieves the best performance.

Two models were used in this parser [48]. It is a transition-based parser using LSTM. and it achieves competitive results with minimal features. But, the issue of considering future sequences is not considered which initiates others to try BILSTM [48].

A Simple LSTM model for Transition-based Dependency Parsing was implemented using LSTM and achieved a 93.06% unlabeled and 91.01% labeled attachment [49]. Additionally, LSTMs were also replaced with GRUs and Elman units in the model to explore the effectiveness methods [49].

Graph-based dependency parsing with a graph neural network was a powerful dependency tree node representation that captures high-order information concisely and efficiently [50]. The researcher has used use graph neural networks (GNNs) to learn the representations and discuss several new configurations of GNN's updating and aggregation functions. Then, the experiments on PTB show that our parser achieves the best UAS and LAS on PTB (96.0%, 94.3%) among systems without using any external resources [50].

3.4 Summary

From the wholes, the earlier parsers don't fulfill Afaan Oromoo dependency parser considered factors such as morpheme information, Afaan Oromoo word order, fast and accurate transition system particularly arc standard technique and better state of the art classifier BILSTM. So Afaan Oromoo dependency parser was developed to fill the gap seen in such points of view. The number of relation types that helps to tag the relations also increased in our study.

Table 3-1 : Summary of related works

No	Articles	Category	Approach	Method	Performance
2	Stanford dependency parser	Universal	DL	Graph-based	81.30%, unlabelled 76.30% labelled
2	Malt-parser	Universal	ML	Transition based	80–90%
3	A Dependency Grammar for Amharic	-	Rule-based	-	Not implemented
4	Transition-based dependency parser for Amharic using DL	Morpheme	DL	Transition system	91.54 and 81.4 ULAS,LAS
5	A Simple LSTM model for Transition-based Dependency Parsing	Transition	DL	Transition system	93.06% , 91.01% ULAS,LAS
6	Hebrew	Both Transitions-based and graph-based	-	Transitions-based and graph-based	Doesn’t show significant improvement
7	Graph-based dependency	-	DL	Graph	96.0%, 94.3% UAS and LAS

CHAPTER FOUR

4. AFAAN OROMOO OVERVIEWS

Afaan Oromoo is a member of the Cushitic branch of the Afro-Asiatic language family ranking the third most widely spoken language in Africa, next to Hausa and Arabic [36] [2]. It was mainly spoken in parts of Ethiopia and neighboring countries. It is also the regional working language of Oromia as well as the educational language for elementary schools. Additionally, it was given as a field of study in the universities. Currently, it is pending to be added to the national working language next to Amharic [51]. Despite Afaan Oromoo having a long history and well-developed oral tradition, it remained an unwritten language for a long period [9]. But, today, Afaan Oromoo is a written language, public media, social issues, religion, political affairs, technology, and a working language, and then it is further under study to make a computational language. For instance,Afaan Oromoo Google Translate is applied. Afaan Oromoo uses Latin letters (qubee) for writing purposes [24]. In addition to 26 English alphabets, Qubee uses a combination of characters (Qubee dachaa), which is pronounced as a single character with the tongue curled back slightly. Those Afaan Oromoo Qubee dachaa are ‘ch’, ‘dh’, ‘ny’, ‘ph’, ‘sh’, and ‘ts’ [24]. Like English, Qubee uses consonants and vowels (a, e, i, o, and u) [24]. next, Afaan Oromoo words are formed using those letters based on the language details.

4.1 Afaan Oromoo Word formation

Morphology and compounding are the two main ways Afaan Oromoo words are formed. Derivational morpheme changes the word class and enhances the vocabulary of the languages [36] [23]. While inflectional morphemes add features to the word class but don’t change it.

4.1.1 Afaan Oromoo morphology

Afaan Oromoo is morphologically-rich as it has a high morpheme-per-word ratio. This implies the number of both derivational and inflectional morphemes added to the base morpheme or lexicon is very high and needs more linguists to study [8]. The corpus is developed based on this ground then machines enable to train and understand the language. For more understanding, the following AO morpheme types are very important. The two main types of AO morphemes [8].

4.1.1.1 Free morphemes (independent morphemes)

Those are the types of morphemes that can't be derivate or inflected but are used to Keep language structures. An independent morpheme is composed of one morpheme hence it can be subdivided [22].

Table 4-1: Lists of some AO independent morphemes

NO	Morphemes	NO	Morphemes
1	Copula	10	Adverbs
2	Cardinal numbers	11	Comparative
3	Indefinite quantifiers	12	Superlative
4	Pronouns	13	Adjectives
5	Pre-positions	14	Indefinite pronoun
6	Postpositions	15	Independent parapositions
7	Jussive	16	Pre-positions
8	Conjunctions	17	Negation
9	Independent conjunctions	18	Cardinal numbers

4.1.1.2 Dependent morphemes

Those are the types of morphemes that can be subdivided into stems and affixes. The affixes will be either derivational or inflectional morphemes [8].

4.1.1.2.1 Derivational Affixes in Afaan Oromoo

Afaan Oromoo word classes mainly derived are nouns, verbs, and some adjectives. It is helpful to increase the vocabulary of the language [23]. The process of word-formation from nouns, verbs, and adjectives is termed nominalization, verbalization, and adjectivization in order. Different root words and affixations result in various types of nouns, verbs, and adjectives [8].

4.1.1.2.1.1 Nominalization

Nominalization is the process of forming nominals from different word categories. In Afaan Oromoo, there is a large stock of nominal derived from adjectival, verbal, and nominal bases [23]. The noun root or stem will be from a noun, verb, or adjective hence called a stream root, and the suffix will be stream suffixes. The relation generated between the root word and suffix will be described as a stream. As the types of nouns differ, the noun formation suffixes also differ. For instance, an Abstract noun has abstract forming suffixes. **nort** represents all types

of words formed from words with a noun, verbal and adjective parts of speech and serve root words in noun formation and **nofsu** are suffixes used on those root words, and the details are described in the following table [36].

Table 4-2: Sample-derived nominals

No	Types of noun	Root word	Nofsu	Examples
1.	Abstract noun	Adjectives	-ummaa -uma -ennaa	Gaar- ummaa: Gaarummaa Gamn- uma: Gamnuma Add-eenna : Addeenna
2.	Process or action noun	Verbal	-icha -aatii -sa	Fiigicha: Fiigicha Dhug-aatii: Dhugaatii Qot-iisa : Qotiisa
3.	Result nominals	Verbal	-sa -aatii -aa	Abaar-sa: Abaarsa Dhug-aatii: Dhugaatii Kenn-aa: Kennaa

Here, representing nominalization using the following word formation rules.

Figure 4-1: Nominalization of word formation rules

4.1.1.2.1.2 Verbalization

Verbalization is a process of forming verbs using various Afaan Oromoo word categories. Verbs are mainly derived from nouns, verbs, and adjectives [23]. Then it is possible to get various types of verbs which include causatives, states, reflexives, passives, etc.

Table 4-3: Sample-derived verbs

No	Types of verb	Root word	Nofsu	Examples
1.	Auto benefective	Verbal	-at-	Bit-ate : Bitate
2.	Causative	Verbal Nominal Adjectival	-siis- -is- -s	Mur-siis : Mursiis Raff-is : Raffis Malaa-s : Malaas

Here, representing Verbalization using the following word formation rules

Figure 4-2: Verbalization word formation rules

4.1.1.2.1.3 Adjectivization

Adjectivization is a process of forming adjectives from different word categories. In Afaan Oromoo, Adjectives will be derived from adjectival, verbal, and nominal bases[23]. The problem is how to determine the category of the roots or base to which suffixes are attached.

Table 4-4: Sample-derived adjectives

No	Types of adjectives	Root word	Nofsu	Examples
1.	Nominal adjective	Noun	-essa- -ttii	Soor-essa : Sooressa Soor-ettii : Soorettii
2.	Verbal adjectives	Verbal	-at-	Diim-at-aa : Diimataa

Here, representing Adjectivization using the following word formation rules

Figure 4-3: Adjectivization word formation rules

4.1.1.2.2 Inflectional affix, number, gender, and tense are seen

4.1.1.2.2.1 Inflectional affixes in Afaan Oromoo

Those are morphemes that add features or information to the root words but can't change the word category [2]. The features will give the needed messages based on the root they attached. For example, the tense can support tense, number and gender features, and the like.

Table 4-5: Sample feature information in AO

No	Root word	Suffixes	Features	Examples
1.	noun	-oota, -lee, -ota, -wwan -n	Number Case	Hool-ota: Hoolota Tolaa-n : Tolaan
2.	verb	-an -t- -e	Number Gender Tense	Deem-an: Deeman Bit-t-e: Bitte

Figure 4-4: AO features representations

4.1.1.2.3 Afaan Oromoo Special morphemes

It was categorized as it is not familiar in the above cases but used in this study. Afaan Oromoo free morphemes are inclusive under this category [22]. Independent morphemes and dependents other than derivational and inflectional are seen here. For instance, particles, comparatives, and focus markers are the few ones [22].

4.1.2 Compounding

Compounding is the second way of Afaan Oromoo word formation as combining various Afaan Oromoo word classes [23]. The following table shows sample Afaan Oromoo compound word formations.

Table 4-6: Sample AO compound words

No	Compound words	First word	Next word	Examples
1.	Noun noun	Noun	Noun	Abbaa buddeenaa
2.	Noun adjective	Noun	Adjective	Sanbata guddaa
3.	Verb noun	Verbal	Noun	Qotee bulaa

Figure 4-5: AO compound word formation rules

4.2 AO Treebank

AO treebank was constructed from those details of morphemes, POS, relations, features, etc. Developing Afaan Oromoo treebank requires more study, time, and budget.

4.2.1 Afaan Oromoo POS

It is mandatory to tag each morpheme with the intended part of the speech. Then later it is used as one feature in the development of AO dependency parser. It is possible to develop a more specific POS which represents each morpheme or generalize related morphemes with one representative POS as stream POS. The following specific POS tags are derived from each first word while the stream AOPOS generalizes the related morphemes into a single POS.

Table 4-7: Specific and Stream AOPOS tag

No	Specific AOPOS	Abbreviations	Stream AOPOS
1.	Abstract noun	abno	nort
2.	Process or action noun	acno	
3.	Result noun	reno	
4.	Gerundive noun	geno	
5.	Manner noun	mano	
6.	Instrumental noun	inno	
7.	Agentive noun	agno	
8.	Auto benefective verb	auv	vert
9.	Causative verb	cav	
10.	Stative verb	stv	
11.	Passive verb	pav	
12.	Noun noun	nncon	cono
13.	Noun adjective	nacon	
14.	Adposition noun	ancon	
15.	Verb noun	vncon	
16.	Noun noun	Nncoa	coad
17.	Noun adjective	nacoa	
18.	Noun numeral	nucoa	

4.2.2 Afaan Oromoo Relations

Generated based on the above morphemes POS and word orders. It will be either stream relations or specific ones. Here are some of AO relations in the following tables.

Table 4-8: Afaan Oromoo sample relation types

No	Relations	Abbreviations
1.	Personal pronoun	PEPN
2.	Interrogative pronoun	WHPR
3.	Pronoun	PRON
4.	Negative	NEG
5.	Focus class marker	FCM
6.	Person	PERS
7.	Adjective complement	ADCOM
8.	Adposition	ADP
9.	Adverb	ADV
10.	Order	ORDER
11.	Demonstrative pronoun	DEPR
12.	Auxiliary	AUX
13.	Cardinal number	CANO
14.	Case	CASE
15.	Coordinating conjunction	CCONJ
16.	Conjunct	CNJT
17.	Compound noun	CNON
18.	Compound adposition	COAP
19.	Copula	COPULA
20.	Definite article	DART
21.	Direct object	DOB
22.	Double genitive	DOGE
23.	Gender	GEND
24.	Interjection	INJ
25.	Instrument	INST
26.	Indirect object	IOBJ

4.3 Sentence structure

Afaan Oromoo sentence structures follow subject-object-verb (**SOV**) word arrangements [2]. In this case, the subject comes first, the object second, and the third verb [22]. Here it is possible to predict the relations ship based on the word sequences. For instance,' **Margaan re'ee qale**.', '**Margaa'** is a subject place, '**re'ee'** is the object place and the '**qale**' verb which carries the messages of the sentence comes at the final place [8]. This implies the Afaan Oromoo **Root** word comes at the end of the sentence

CHAPTER FIVE

5. AODP MODEL DESIGN

This chapter describes the architecture of the overall AODP system. The parser was developed from the two sub-models transition predictor model and the relation predictor model. Next, each model was developed from both Afaan Oromoo dataset and RNN models configured. The dataset is both word and POS parts and it was represented in vectors. Then, the first model was trained for transition prediction while the second was trained for relation prediction. In the first model, it was trained using an arc-standard transition system. Finally, both models were tested to check their performance. Generally, the models, input features, output features, RNN algorithm, and the flows of activities were shown as follows.

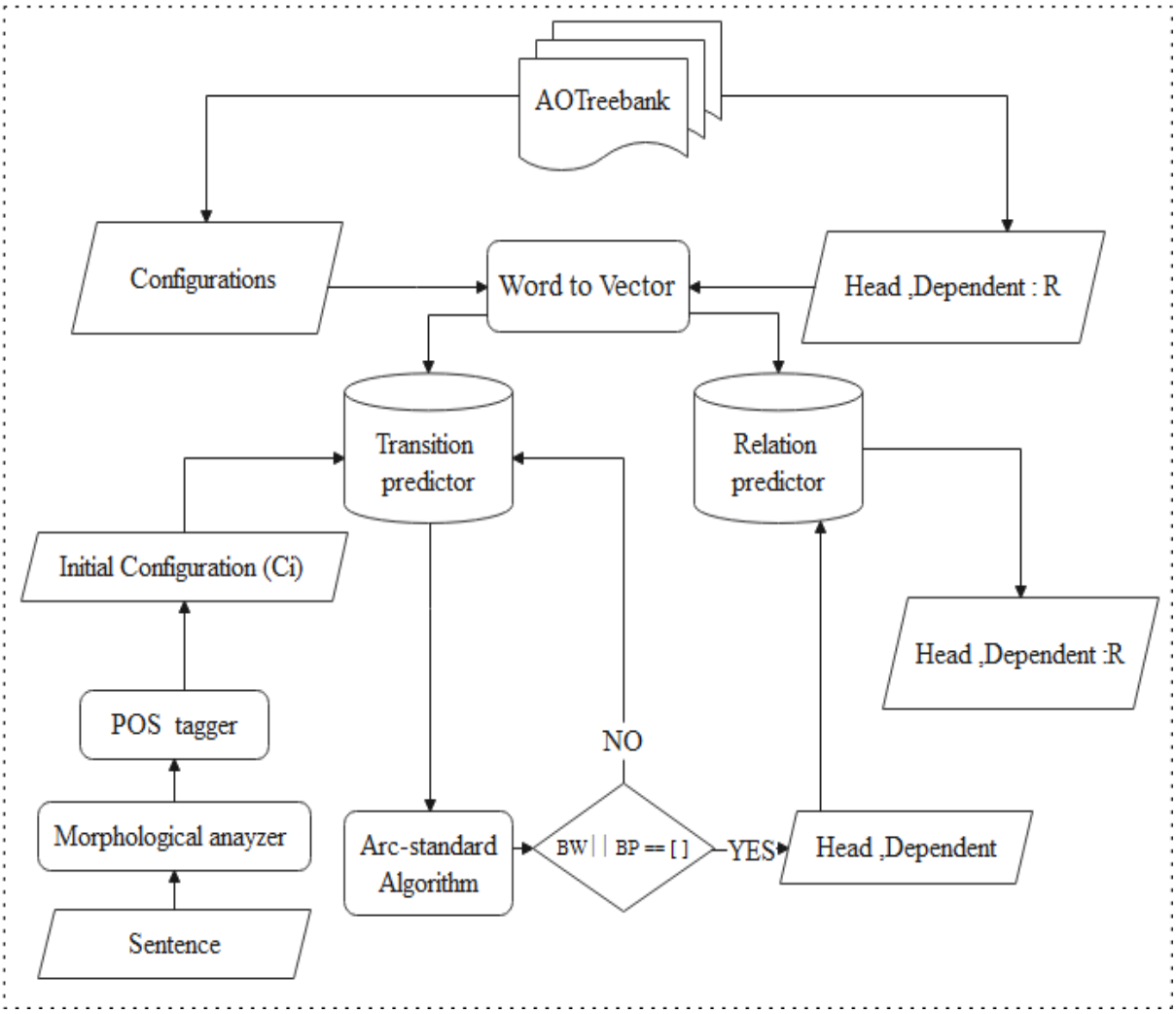


Figure 5-1: General architecture for the AODP system

5.1 AO Treebank

AO treebank was constructed from the details of AO morphemes types and features, POS, Relations, etc. [8]. The data was collected, preprocessed, and converted into vectors input features for the two models as configurations and head-dependents. It was more understandable when it was represented using the **CONLL** format [7]. Unlike universal parsers, the researcher used a few attributes of CONLL as NO, WORD, POS, HEAD, and RELATIONS. So, '**No'** represents lists of index numbers for words or morphemes. '**Word'** shows the words or morphemes used. Part of the speech of each word was stated under the '**POS**' column. The numbers under the '**HEAD'** columns indicate the index number of a word it modifies, Finally, the relation between the word and its head was stated under the '**RELATION'** field [7]. Both models' feature was extracted from this AOTB and described in the following manner.

Table 5-1: Sample AO Treebank

#Caaltuun dheengadda huccuu haphii bitte.

NO	WORD	POS	HEAD	RELATION
1	Caaltuu	prno	6	NSUBJ
2	n	noca	1	NCM
3	dheengadda	adv	6	ADV
4	huccuu	noun	6	DOB
5	haphii	adjb	4	ADJ
6	bit	vert	-	-
7	t	gnd	6	GEND
8	e	tens	6	TNS
9	.	punc	6	PNC

5.1.1 Configurations

Configurations are constructed from stack, buffer, and transitions [49]. For the sake of training the first model, the configurations are derived from the general AOTB and input features are stated using stack, buffer, and its corresponding transitions. The set of configurations (**Ci** to

Cf) and the set of corresponding transitions (**Ti** to **Tf**) were used for model development. But, for the test model, the configurations were used without having transitions.

5.1.2 Head-dependents dataset

Here the labeled head-dependents are extracted from the AOTB for training the second model. Head is the governing or the basic word while dependents are the modifiers of the headwords within a given sentence. In the test phase the head, dependent features without relation type are used. The number of relation types varies in different dependency parsers. So, the relation predictor model predicts the relations for a new sentence from the given alternatives of relation types.

5.1.3 Data Collections

A balanced corpus is very important to handle the details of morphemes. So, the data was collected from diversified sources such as newspapers (FBC, VOA, OBN), books, websites, magazines, etc. [21] [22]. For instance, FBC (fana broad casting corporate) includes sport, health, business, technology domains, etc. Additionally, exemplary sentences are obtained from Afaan Oromoo books such as Furtuu, Semmoo, Catrina, and Sanyii which represent morpheme types [8] [9]. In summary, AOTB was developed from 500 sentences. From the treebank around 5220 configurations and 1415 head–dependents were generated.

5.1.4 Data Preprocessing

Both features for the two sub-models have to be represented accordingly in both training and test phases. In the training phase, the features were derived from the AOTB. But for parsing a new sentence, the sentences have to be morphologically analyzed, POS tagged and represented into vectors before being fed to the models. In this study, sentences were manually tokenized into their morphemes and tagged. But, the existing AO morphological analyzers and POS taggers have been seen and referred as a preliminary point. But, it is not fully comprehensive to handle the issues in this study [52] [53].

5.1.4.1 Word to vectors

The main goal of word-to-vector representation is to increase network capacity for learning from textual data [4]. The two major types of word representation means are prediction-based approaches that assign probabilities to measure the degree of relatedness and frequency or

count-based approaches that use co-occurrence of words [13] [4]. Particularly one-hot encoding and word embedding (Keras embedding layer) was applied for this study as follows.

5.1.4.1.1 One-hot encoding

The words are encoded into a one-hot vector using this encoding method [54]. The one-hot vector, which is a series of one and zero, indicates words that are the same size as the vocabulary size of the input data. If the word is available, one is allotted; otherwise, zero is given to all sequences. These are sparse, high-dimensional vectors [4].

5.1.4.1.2 Word embedding

Word embedding is also used to convert words into their corresponding vectors [4]. These methods are employed to produce low-dimensional models. Low-dimensional vectors are those in which the vocabulary size, as opposed to the dimensionality size, can be specified. In contrast to one-hot vectors, dense vectors have no zeros in any of their components (features).

5.1.4.1.2.1 Keras embedding layer

Words were transformed into their appropriate vector representations using the word embedding layer of the Keras Python package [39]. The word embedding layer was created in Keras as a component of deep learning. With each iteration, the representation of all words is gradually learned by assigning a random value to the first layer. This layer is used to create dense representations and low-dimensional models [4].

5.2 Transition predictor model development

This was the first sub-model from the general AODP. The model was developed from both configurations and designed RNN algorithm. The transition predictor model is not only configured RNN but when designed RNN was trained with the dataset then it makes the transition predictor model. For this model development, Afaan Oromoo treebank was developed. From this general treebank, specific input features for this model were derived. This feature is configurations. The configurations used for the model training include both word and part of speech part of the morphemes. The words are stored in elements of configurations (stack, buffer) based on the types of transitions applied to each configuration. The training and test data were prepared from such Configurations. Hence, it was split into training and test data. So input features for the model developments are specifically, stack word, stack POS, buffer word, buffer POS with the types of transitions. Then, the features were represented into vectors as the deep learning model understands the numeric representations (vector) form of words. As a result, it enables the model to extract the features from the language patterns. So, the model trains from the given data set, using parameterized RNN algorithm, and in such a way transition predictor model was developed. On another hand, the trained model was tested for unlabeled configurations and the model performance was seen. The model was tuned with hyper parameters and saved with effective performance. Then evaluate the model performance using accuracy to measure the effectiveness of transitions predictions and Unlabeled attachment score (UAS) to measure the number of correctly attached head-dependents for a given sentence [7]. Finally, the model was seen as effective in transition predictions for new input sentences. The first model's overall output is used as input for the second one to gain the needed relation type between the words.

Table 5-2: Sample input features for the model

Stack	Buffer	Transitions
[]	[Caaltuu n dheengadda huccuu haphii bit t e .]	S
[Caaltuu]	[n dheengadda huccuu haphii bit t e .]	S
[Caaltuu n]	[dheengadda huccuu haphii bit t e .]	RA
[Caaltuu]	[dheengadda huccuu haphii bit t e .]	S
[Caaltuu dheengadda]	[huccuu haphii bit t e .]	S
[Caaltuu dheengadda huccuu]	[haphii bit t e .]	S
[Caaltuu dheengadda huccuu haphii]	[bit t e .]	RA
	[bit t e .]	S
[Caaltuu dheengadda huccuu]	[t e .]	LA
[Caaltuu dheengadda huccuu bit]	[t e .]	LA
	[t e .]	LA
[Caaltuu dheengadda bit]	[t e .]	S
[Caaltuu bit]	[e .]	RA
[bit]	[e .]	S
[bit t]	[.]	RA
[bit]	[.]	S
[bit e]	[]	RA
[bit]	[]	RA
[bit .]		
[bit]		
[]		

5.3 Relation predictor model development

This was the second sub-model in AODP which was designed to predict the relationship between the morphemes and tag them with relation type. The Relation predictor model is not only configured RNN but when configured RNN was trained with (head, dependent: label) which generates a Relation predictor model. This model development activity follows similar pipelines to the first model but input features used in this model are lists of the labeled heads and dependent words with their POS tag. From this general Afaan Oromoo treebank, specific input features (head, dependent: label) were derived. This feature contains lists of both words and parts of speech from the data. The training and test data were prepared from such a dataset. Hence, it was split into training (head, dependent: label) and test (head, dependent). Then, the features were represented into vectors using a one-hot encoding and embedding layer before being fed to the RNN algorithms for training and testing. Next, the model trains from the given data set, using parameterized RNN algorithm, and in such a way Relation predictor model was developed. On another hand, the trained model was tested for unlabeled heads-dependents and the model performance was seen. Then evaluate the model performance using accuracy to measure the effectiveness of relation predictions and labeled attachment score(**LAS**) to measure the number of correctly attached head-dependents: relations types for a given sentence [15] [38]. Finally, the model was seen as effective in relation prediction of new input sentences accordingly.

Table 5-3: Sample input features for the mode2

List of head-dependent	POS of head-dependents	Relation type
[caaltuu n]	[prno noca]	NCM
[huccuu haphii]	[noun adjb]	ADJ
[bit huccuu]	[vert noun]	DOB
[bit dheengadda]	[vert adv]	ADV
[bit caaltuu]	[vert prno]	NSUBJ
[bit t]	[vert gnd]	GNDR
[bit e]	[vert tens]	TNS
[bit .]	[vert punc]	PNC

5.4 Parsing phase System

This phrase describes the actual process to extract the relationships types among the words for a given new sentence. To realize this phase, a given sentence has to be morphologically analyzed. Each detail of the morpheme information within the words has to be handled. Next, it was very important to assign part of the speech tag for those morphemes. Currently, the researcher used manual techniques for both morphological analysis and tagging due to the lack of an Afaan Oromo natural language processor in the case. The configurations include features such as stack, buffer, and the relation types used. So the first configurations are obtained from input sentences. Then, parsing starts from initial configurations.

Deep learning algorithms only deal with vector representations of words [4]. Hence, those features (first model and second model) were represented in the numeric form using the one-hot encoding technique [4]. Each Vocabulary in the features was represented in vector form. Later it was embedded using embedding layers also [4]. After formatting the features for the model accessible, the first configurations **Ci** (test feature) is sent for the first model. Then, the model predicts the transition types going to be applied to the configuration. Next, using the arc standard transition system the next configuration (**Cm**) was generated. Then, the processes follow similar conditions until to get the final configurations (**Cf)**. Eventually, from the first model full configurations (**Ci** to **Cf**) and transitions (**Ti** to **Tf**) were obtained. In the end, an unlabeled tree (dependency graph) was generated from the first model. Then the task of relation types predictions for head-dependents generated by the first model was performed by the second sub-model. Finally labeled head, dependent with relation was generated as the final output for AODP as follows.

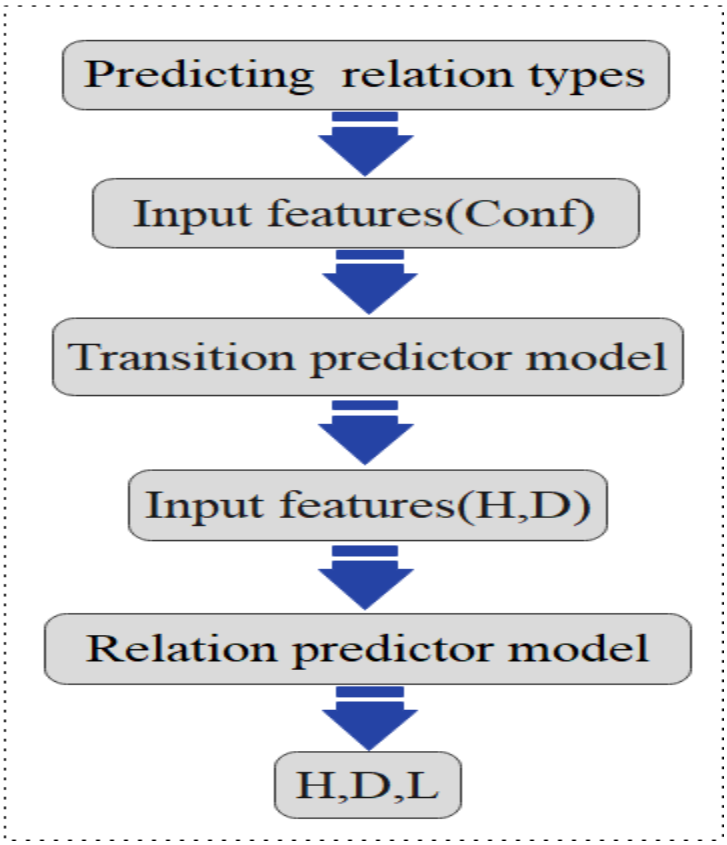


Figure 5-2: Generalized parsing steps(phases)

5.4.1 How does the arc-standard transition system work?

Arc standard transitions system works on sets of configurations and transitions [10]. It applies the transitions to arrive from **Ci** to **Cf**. From this to gain the next configuration or intermediate configurations we use the selected transitions. Among them we may use three types of transitions do so. The initial configurations have an empty stack, but an artificial root (**ROOT**) is added to it. The buffer has a list of words (w_1, w_2, w_3…w_n). It has no arc or transitions(**Ti**) assigned at this level. From this **Ci** feature, the transition is predicted and the next configurations (**Cm**) were constructed. Finally, the last configurations were generated as it has artificial **Root** only in its stack, empty buffer **([]),** and no transitions also [10]. Generally, the three transition types used in arc-standard works are as follows. Assuming **i**, and **j** are the two nodes on the top of the stack and the types of arcs **l** (transitions) are stored to A (arc).

A. **LEFT-ARC** (**LA**) adds a dependency arc (**j, l, i**) to A, where i is the node on top of the stack (**σ**) and j is the second node on top of the stack. So it pops the second node **j** from the stack.

B. **RIGHT-ARC** (**RA**) adds a dependency arc (i, l, j) to **A**, where i is the second node on top of the stack(σ) and j is the first node on top of the stack. So it pops the first node **j** from the stack.

C. The transition **SHIFT** removes the first node i in the buffer β and pushes it on top of the stack (**σ)**.

In such a way, for each **Ci** to **Cf**, the corresponding **Ti** to **Tf** was predicted. Based on this the arc standard algorithm generates an unlabeled tree with a head-dependents.

Representations of transitions in the following table (0: SHIFT, 1:LA and 2 : RA).

Table 5-4: Arc-standard transition system on configurations

Stack	**Buffer**	**Transitions**
[ROOT]	[margaa n foon dheedhii hin nyaat u .]	0
[ROOT margaa]	[n foon dheedhii hin nyaat u .]	0
[ROOT margaa n]	[foon dheedhii hin nyaat u .]	2
[ROOT margaa]	[foon dheedhii hin nyaat u .]	0
[ROOT margaa foon]	[dheedhii hin nyaat u .]	0
[ROOT margaa foon dheedhii]	[hin nyaat u .]	2
[ROOT margaa foon]	[hin nyaat u .]	0
[ROOT margaa foon hin]	[nyaat u .]	0
[ROOT margaa foon hin nyaat]	[u .]	1
[ROOT margaa foon nyaat]	[u .]	1
[ROOT margaa nyaat]	[u .]	1
[ROOT nyaat]	[u .]	0
[ROOT nyaat u]	[.]	2
[ROOT nyaat]	[.]	0

CHAPTER SIX

6. IMPLEMENTATION AND EVALUATION

The designed Afaan Oromoo dependency parser was implemented and evaluated in this chapter. This phase describes ways to arrive at the result. Moreover, implementation setups used in this study are dataset(features), word to vector representations algorithms, hyper-parameters, RNN algorithms, etc. Initially, from AO treebank features were derived and represented into features for the two models. Next, the features were represented into vectors and then fed to the models. RNN algorithms (LSTM and BILSTM) had been experimented and the best performing one was selected for the two sub-models. In the end, the result was discussed in comparison with other parsers using evaluation metrics.

6.1 AO Treebank

From the Afaan Oromoo developed treebank input features for the two models were extracted and used. The first model uses a set of configurations while the second one uses a labeled head, dependent. The data was split into the training and test data for the two sub-models.

6.1.1 Word to vector representations of the data

One hot encoder and word embedding specifically the Keras embedding layer was used in this study for the two sub-models of AODP.

6.1.1.1 One hot-encoding

All vocabularies found in the features are represented into vectors using Keras one-hot encoding library. This results in sparse dimension representations of the vocabularies [43] [4].

6.1.1.2 The embedding layer

This embedding layer takes one-hot encoding features and minimizes the sparse dimensions of vector representations into dense ones. So this layer is used to extract the semantic relationships among the vocabularies(words) [4] [13].

6.2 Hyper-parameters used in the two sub-models

Hyper-parameters are variables that verify the structure of the neural network models and determine their performance [48] [10]. The hyper-parameter used for the two sub-models of AODP is summarized in the following table.

Table 6-1: Hyper-Parameter used

Hyper-parameter	**value**
LSTM units	256
BLSTM units	256
Epoch	100
Dropout rate	0.5
Word embedding dimensions	100
Loss functions	Categorical cross-entropy
Optimization algorithm	Adam
Activation function out-put layer	softmax
Activation function hidden layer	Relu
Batch size	32

6.3 Transition predictor model development

This sub-model was the result of both dataset and deep learning algorithms configured(RNN). To gain this model first, the training data sets were identified and represented. Next, the RNN model was defined and trained with the prepared data.

6.3.1 Data set (features) used

Neural network or deep learning algorithms works with numeric values only. There were 3840 instances configurations for training data and 1170 for evaluation purposes. Hence, the features were represented with numbers using various techniques. For instance, one hot encoder is used in this study. Configurations were converted to features such as stack word, stack POS, buffer word, and buffer POS, with corresponding transitions converted to numeric values using a one-hot encoder [4]. Next, the sequences were padded and represented with a Numpy array.

6.3.2 Developing the RNN model

Next to data preparations the network model was developed through the following steps [13].

6.3.2.1 Define the model

Defining the model means, selecting, arranging, parameterizing, and tuning the neural network algorithm as its optimal for problem-solving [4]. So, here were the details for AODP using Keras functional API. The input layer was defined and the dimensions of input features with their sequence length are used as a parameter. Next to embedding layer accepts input features from input layers. Then, Keras embedding layers were also used to represent the words into vectors. The dot product of word and POS parts of each feature (stack, buffer) was obtained. Then the stack and buffer of those features are fed to independent RNN algorithms (LSTM or BILSTM). Then, they concatenated and generates another hidden layer output. Finally, this was fed to the output layer TDD for transition predictions. Through this, important parameters for the RNN algorithms were used. The number of nodes (256), Relu-activation functions for hidden layers, and softmax activation for the output layer was used.

6.3.2.2 Compile the model

In this phase, the Adam optimizer was used for weight updating, and the categorical cross-entropy loss function to calculate the difference between target output and model output [54].

```
model.compile(optimizer="Adam",loss="categorical_crossentropy" ,metrics=["accuracy"])
```

Figure 6-1: Compile transition prediction model

6.3.2.3 Fit the model

Fit the model with four configuration features and its label transitions, 100 epochs used, validation data (unlabeled configurations with transitions were used), etc. Hence with such tuning parameter of the model then saved it with its better performance [54].

```
history=model.fit([SWPA,SPPA,BWPA,BPPA],YT,epochs=100,validation_data=([SWPE,SPPE,BWPE,BPPE], YYT))
```

Figure 6-2: Fit transition prediction model

6.4 Relation predictor model development

This model development follows transition predictor model development pipelines except for the types of input features and parameters used in RNN development [4]. Under this model development, input features were a set of head-dependents labeled with relations for training.

6.4.1 Data set (features) used

There were 1000 instances labeled (head, dependents) for training and 415 for evaluation purposes. Features were converted to lists of the word, POS of the (head, dependents) features with corresponding relations. Then it was converted to numeric values using a one-hot encoder and represented with a Numpy array [13].

6.4.2 Developing the RNN model

This follows the same steps as done in the transition predictor model.

6.4.2.1 Define the model

The model was defined using Keras functional API [13]. The word, POS features of the head, POS was fed to the input layer. The input layer was defined and the dimensions of input features (1000 for training and 415 for the test were used) with their sequence length (20) are used as a parameter. Next to embedding layer accepts it. The dot product of the word and POS parts of each feature was obtained. Then, the dot product of word and POS was obtained and fed to

RNN algorithms (LSTM or BILSTM). Next, it generates another hidden layer output. Finally, this was fed to the output layer TDD for relation predictions.

6.4.2.2 Compile the model

In this case, Adam optimizer for weight updating, and the categorical cross-entropy loss function were used to calculate the difference between target output and model output [25].

```
model.compile(optimizer="Adam",loss="categorical_crossentropy" ,metrics=["accuracy"])
```

Figure 6-3: Compile relation prediction model

6.4.2.3 Fit the model

Fit the model with head, dependents feature, and its relation, epochs 100, and validation data (unlabeled head, dependents used) [25].

```
model.fit([RWTA,RPTA],YRW,epochs=100,validation_data=([RWEA,RPEA], YRWE))
```

Figure 6-4: Fit relation prediction model

6.5 Parsing phase

The sequence of input words has to be synthesized into morphemes and tagged accordingly. The first configuration (**Ci**) is generated initially. Then, the transition predictor model uses these features and predicts the next transition. Furthermore, using the arc standard transition system, the next configuration is constructed. Again the configuration is sent for the transition predictor model. The process continues in such a manner until final configurations are gained.

```
['[]', '[margaa n foon dheedhii hin nyaat u .]', '[]', '[prno noca noun adjb ne
g vert tens punc]', 'SHIFT']

['[margaa]', '[n foon dheedhii hin nyaat u .]', '[prno]', '[noca noun adjb neg
vert tens punc]', 'SHIFT']

['[margaa n]', '[foon dheedhii hin nyaat u .]', '[prno noca]', '[noun adjb neg
vert tens punc]', 'RIGHT_ARC']

['[margaa]', '[foon dheedhii hin nyaat u .]', '[noca]', '[noun adjb neg vert te
ns punc]', 'SHIFT']
```

Figure 6-5: Sample configuration generated **Ci** to **Cf** with its transitions

Based on the above-generated configurations the following un-labeled trees were generated. It is used as input for the second model for predicting its relations. Next, **head** and **dependents** are obtained as output from this model and used by the relation predictor model as input. Finally, it was labeled and (**Head**, **dependent**, **relation**) was obtained.

```
[['margaa', 'n'], ['foon', 'dheedhii'], ['hin', 'foon'], ['nyaat', 'hin'], ['ny
aat', 'u'], ['margaa', 'nyaat'], ['margaa', '.']]
[['prno', 'noca'], ['noun', 'adjb'], ['neg', 'adjb'], ['vert', 'neg'], ['vert',
'tens'], ['noca', 'tens'], ['tens', 'punc']]
```

Figure 6-6: Sample **head**, **dependent** generate

6.6 Experimentations and Models Evaluation

Here experimentations were carried out among RNN algorithm LSTM and BILSTM using the same parameters and input features for the two models. Hence in the two models, the accuracy of predicting the transition type and relation type increased using BILSTM. For transition prediction, the experiment was carried out using 3840 instances of training data and 1170 evaluation data. For relations predictions, 1000 labeled (**head**, **dependents**) for training and 415 (**head**, **dependents**) were used for evaluation. Even though RNN algorithms was effective in transition prediction models the experimental result shows the following differences in using LSTM and BILSTM.

```
Epoch 99/100
109/109 [==============================] - 2s 18ms/step - loss: 0.2709 - accuracy: 0.8974 - val_loss: 1.7130 - val_accuracy:
0.7034
Epoch 100/100
109/109 [==============================] - 2s 18ms/step - loss: 0.2707 - accuracy: 0.8991 - val_loss: 1.8357 - val_accuracy:
0.7051
```

Figure 6-7: Show result of using LSTM for transition predictions

The first model or transition predictor shows more effective results using BILSTM than using LSTM as indicated below.

```
Epoch 99/100
109/109 [==============================] - 5s 43ms/step - loss: 0.2863 - accuracy: 0.8914 - val_loss: 1.6962 - val_accuracy:
0.6795
Epoch 100/100
109/109 [==============================] - 5s 42ms/step - loss: 0.2767 - accuracy: 0.9017 - val_loss: 1.9008 - val_accuracy:
0.6855
```

Figure 6-8: Show result of using BI LSTM for transition predictions

On the other hand, the effectiveness of relation prediction in the second model was shown using accuracy as evaluation metrics as follows.

```
Epoch 99/100
32/32 [==============================] - 0s 6ms/step - loss: 0.9248 - accuracy: 0.6850 - val_loss: 2.1741 - val_accuracy: 0.3
855
Epoch 100/100
32/32 [==============================] - 0s 5ms/step - loss: 0.9180 - accuracy: 0.6900 - val_loss: 2.1703 - val_accuracy: 0.4
072
```

Figure 6-9: Show the result of using LSTM for relation predictions.

On another hand, like the transition predictor model, using BILSTM is more effective in relation predictions also.

```
Epoch 99/100
32/32 [==============================] - 0s 9ms/step - loss: 0.8568 - accuracy: 0.7150 - val_loss: 2.9576 - val_accuracy: 0.2
867
Epoch 100/100
32/32 [==============================] - 0s 9ms/step - loss: 0.8519 - accuracy: 0.7160 - val_loss: 2.9656 - val_accuracy: 0.2
843
```

Figure 6-10: Show result of using BILSTM for relation predictions

On another hand, evaluation metrics such as unlabeled attachment score (**UAS**) and labeled attachment scores (**LAS**) were used to measure the output of both the first and second models sequentially. From the transition predictor model, it was evaluated how many heads are attached to their dependents correctly using the unlabeled attachment score. Then, the relation predictor was evaluated for the number of correctly attached heads, and dependents with their label.

$$\text{LAS} = \frac{\text{The number of correctly predicted heads,dependents with their dependency relations}}{\text{Total heads,dependents,and dependency relations.}} * 100 \qquad [7]$$

$$\text{UAS} = \frac{\text{The number of correctly predicted heads,dependents}}{\text{Total heads,dependents}} * 100 \qquad [7]$$

On the whole, the following result using BILSTM.

```
Unlabeled attachment score : 60.0%
Labeled attachment score : 40.0%
```

Figure 6-11: shows the results of labeled and unlabeled attachment score

6.6.1 *Sample output for input sentences*

Here is a sample output for sentences the first six were correctly parsed and assigned relationships but the last two were incorrect. The first four relations correctly parsed represent adjectives (ADJ), and the next two relations represent the tense (TNS). The last two were incorrectly parsed as noun class modifiers but the first one actual relation type represents tense and the last one represents gender.

```
CORRECTLY LABELLED RELATIONS
------------------------------
ati goota => ADJ
nuti goota => ADJ
isin goota => ADJ
inni goota => ADJ
deem e => TNS
xuux e => TNS

INCORRECTLY LABELLED RELATIONS
--------------------------------
deem t => NCM
fiig d => NCM
```

Table 6-2: Sample output for input sentences

6.7 Result and Discussion

6.7.1 *Results*

Generally, from the experimentation, we have got the following results. The result has summarized in the following table for the two models and values obtained by applying the RNN algorithm to them. On the whole, BILSTM is seen as it performs better and the score for the **head-dependent** attachment without relation is **60% UAS,** and for the correctly labeled **head-dependents** score is **40% LAS.**

Table 6-3: Accuracy for the two models using LSTM and BILSTM

No	RNN Algorithms	Transition predictor model (first model) accuracy	Relation predictor model (second model) accuracy
1	LSTM algorithm	89%	69%
2	BILSTM algorithm	90%	71%

6.7.2 Discussions

In this study, the Afaan Oromoo dependency parser was implemented using RNN. The model was created to solve the ambiguity of the morphemes at the syntax level. So, two sub-models were implemented separately for the success of the parser. Afaan Oromoo treebank was developed from scratch, and input features configurations and labeled (head-dependent) were derived from the treebank for the sub-models development. In this study, the number of Afaan Oromoo POS tags and relation types used was increased. Although the model performance decreased with increasing the number of relation types, the parser used **61** relation types. Universal dependency parsers and others are not effective for Afaan Oromoo. They used fewer relation types than the Afaan Oromoo dependency parser. It also works at a word level, this decreased the level of morpheme clarity or ambiguity resolutions.

Amharic [20] transition-based dependency parser used *LSTM* algorithm, *arc-eager* transition system. But Afaan Oromoo word order is more related to *arc-standard* algorithm. Both Afaan Oromoo word orders and most Afaan Oromoo morphemes types did not show transitivity properties [8] [2]. Transitivity properties are seen if the head of the morpheme has another head [15]. For instance, this behavior was seen in some Afaan Oromoo morphemes especially in derivational Afaan Oromoo adjectives [8]. For such seldom cases, the *arc-standard* transitions system is not effective. Moreover, using such a transition system enhances its importance in the study for its effectiveness and simplicity [10].

In this sub models , RNN algorithm were used and overcomes the limitations of multilayer neural networks by capturing the information in the sequences of words [4] [13]. Hence, the RNN algorithm which is state of the art to learn sequential data was implemented [4]. The experimental result shows using BILSTM for the Afaan Oromoo dependency parser is more effective than using LSTM. LSTM has memory or cells which are called gates to manipulate the input and output information [55]. This enables it to manage long-term dependency more than RNN. Next, using BILSTM enhances LSTM performance by considering the past and future information of the morphemes [29]. The effectiveness of the model increases with the amount of Afaan Oromoo treebank. Additionally, increasing the number of relations in this study enables the model to predict the relations among the morphemes specifically more [8]. The effectiveness of the transition predictor model affects the results of relation predictions. From model one, if the trees are generated correctly, it enhances the quality of the next model for relation predictions. At the end correctly attached head-dependents generated from model

one leads model two to predict correct relations. The developed model performs **60% UAS** from model one and **40% LAS** from the second model. Generally, Afaan Oromoo dependency parser is a competitive parser as it is applied on AO morphologically rich language by experimenting with RNN algorithm and identified that BILSTM is best performing algorithm. It scores **90%** accuracy in first model and **71%** accuracy in the second model using BILSTM.

CHAPTER SEVEN

7. CONCLUSION AND RECOMMENDATION

7.1 Conclusions

Afaan Oromoo dependency parser was developed using RNN. The parser was developed via a data-driven approach, particularly by implementing an arc-standard transition system. It is a fast, effective for projective part of the language. As a result, it is suitable to generate a dependency graph among the morphemes. The characters of Afaan Oromoo morphemes and their word order allow the arc-standard transitions system to perform more for Afaan Oromoo dependency parser than the others.

Two sub-models were used in AODP. The first one is the transition predictor model while the second is a relation predictor. The two models were trained independently on different features. The first model was trained on a set of configurations and used to create a dependency graph. While the second model was trained on the head-dependent labeled with its relations, and used to predict relations for the unlabeled dependency graphs. Next, an experiment was conducted on the two RNN algorithms (LSTM and BILSTM) to identify the better-performing algorithm for AODP. As a result, BILSTM shows better performance for the two sub-models. Specifically, the first model has scored **89%** accuracy for LSTM and **90%** for BILSTM. Then, the second model scored **69%** accuracy for LSTM and **71%** for BILSTM. Generally, AODP performs **40%** for labeled attachments score and **60%** for unlabeled attachment scores using BILSTM. In general, RNN captures the patterns from the morphemes without explicit manual feature extraction or setting rules for the morphemes. Furthermore, it was seen that deep learning approaches work more for natural languages. Particularly, in this study, RNN works more than multilayer neural networks and BILSTM is the best algorithm in comparison to simple RNN and LSTM for AODP.

Eventually, the result of AODP will also increase with increasing the corpus size. Additionally, the details of morphemes with their specific POS and corresponding specific relations enhances clarity although it is challenging to include all detail of morpheme in AOTB.

7.2 Recommendations

As a recommendation, a clear Afaan Oromoo POS tagger and morphology analyzer are very important as low-level NLP applications for Afaan Oromoo dependency parsers. Because, Afaan Oromoo morphology analyzer is very useful to partition words into their meaningful constituents. Based on this, effective POS tagger is used to assign word class for the morpheme parts. This makes AODP robust to parse the details in AO morphemes. There is directly proportional relation among Afaan Oromoo morphology analyzer, POS tagger, Transition predictor model, and Relation predictor model in terms of performance. As a result, the more the details of morphemes identified, POS tag assigned, transitions predicted indicates the more effective relation types predicted for a given sentence. Although deep learning algorithms are the state of the arts in NLP, it requires huge data in model development. But, Afaan Oromoo treebank was developed from scratch in this study. So the researcher strongly recommends developing a large and standardized tree bank for Afaan Oromoo is crucial. In general, using effective NLP applications for preprocessing and standardized AO tree bank makes the AODP powerful.

7.2.1 Contributions of the study

- ✓ The best performing RNN algorithm for AODP was identified.
- ✓ Afaan Oromoo morphemes which are handled with an arc-standard transition system have been identified.
- ✓ Afaan Oromo treebank was developed from the scratch

7.2.2 Future works

- ✓ Non-projective part of Afaan Oromoo language will be implemented using graph methods.
- ✓ Afaan Oromoo treebank is developed by us from scratch but it is not enough and has to be developed.
- ✓ The basic issue is considering other special morphemes other than inflections and derivations and representing relation types for each of them.
- ✓ Trying the parser in other transition methods.

8. REFERENCES

[1] T. G. Ayana, "Afaan Oromo Parser using Hybrid Approach," JU, Jimma, 2017.

[2] SEENSA AFAANIIFI XIINQOOQAA, FINFINNEE, ITOOPHIYAA, 2008.

[3] S. B... H. M. Daniel Jurafsky, Speech and Language Processing: An Introduction to, Prentice-Hall, 2006.

[4] J. Brownlee, Deep Learning for Natural Language Processing, 2017.

[5] B. Plank, "Domain adaptation for parsing," University of Groningen.

[6] L. Kohorst, "Constituency vs. Dependency Parsing," 12 december 2019. [Online]. Available: https://medium.com/@lucaskohorst/constituency-vs-dependency-parsing8601986e5a52. [Accessed 28 1 2020].

[7] R. M. a. J. N. Sandra Kübler, Dependency Parsing, Graeme Hirst, University of Toronto, 2009.

[8] A. Barkeessaa, Sanyii, Finfinnee, 2013.

[9] A. Barkeessaa, Semmoo, Finfinnee, 2016.

[10] R. M. E. P. a. J. M. Bernd Bohnet, "Generalized Transition-based Dependency Parsing".

[11] G. T. R. M. Alymzhan Toleu, "Comparison of Various Approaches," OPCS, Almaty, Kazakhstan, 2019.

[12] B. Erena, "Afaan Oromoo," [Online]. Available: https://scholar.harvard.edu/erena/oromo-language-afaan-oromoo. [Accessed 23 4 2022].

[13] J. Brownlee, Long Short-Term Memory Networks With Python, 2017.

[14] G. H. DABALO, "AUTOMATIC SYNTACTIC PARSER FOR AFAAN OROMO COMPLEX," AAU, Addis Abeba, 207.

[15] A. Volokh, "Performance-Oriented," vol. 37.

[16] J. N. J. H. J. Nilsson, "MaltParser: A Data-Driven Parser-Generator for Dependency Parsing".

[17] T. Dozat, "Stanford's Graph-based Neural Dependency Parser," 2017.

[18] Y. G. a. M. Elhadad, "Hebrew Dependency Parsing: I," 2014.

[19] J. Nivre∗∗, "Dependency Parsing of Turkish".

[20] M. Zelalem, "TRANSITION BASED DEPENDENCY PARSER FOR AMHARIC LANGUAGE USING DEEP LEARNING," Bahirdar University, Bahirdar, 2019.

[21] "FBC," [Online]. Available: https://www.fanabc.com/afaanoromoo/. [Accessed 29 4 2021].

[22] Catrina, Afaan Oromoo language structure, Addis Abeba.

[23] T. Negassa, "word formation in Oromo".

[24] G. Rabbirraa, Furtuu.

[25] " keras tutorial point," 2020. [Online]. Available: https://www.tutorialspoint.com/keras/index.htm. [Accessed 20 may 2020].

[26] "tensorflow tutorails point," 2020. [Online]. Available: https://www.tutorialspoint.com/tensorflow/index.htm. [Accessed 21 may 2020].

[27] Oinkina, "Understanding LSTM Networks," colah's blog, 2015. [Online]. Available: file:///C:/Users/user/Downloads/Understanding%20LSTM%20Networks%20--%20colah's%20blog.htmldd.html. [Accessed 7 5 2020].

[28] J. L. ¨. 1 and 2. a. R. Collobert∗, "Deep Neural Networks for Syntactic Parsing of Morphologically Rich," Idiap Research Institute, Martigny, Switzerland.

[29] K. M. Kurniawan, "Exploring Recurrent Neural Network," University of Edinburgh, 2017.

[30] K. a. Klein, "Constituency Parsing with a Self-Attentive Encoder," University of California, Berkeley, 2018.

[31] L. a. Zhang, "In-Order Transition-based Constituent Parsing," *Transactions of the Association for Computational Linguistics*, vol. Volume 5, no. 10.1162/tacl_a_00070, p. 413–424, 2017.

[32] K. Rajasekaran, "Deep Learning techniques," Koblenz Landau, 2020.

[33] M. e. al, "Rethinking Self-Attention: An Interpretable Self-Attentive Encoder-Decoder Parser," University of California, 22019.

[34] Z. a. Zhao, "Head-Driven Phrase Structure Grammar Parsing on Penn Treebank," Shanghai Jiao Tong University, Shanghai, China, 2019.

[35] "Natural Language Processing Tutorial," 2020. [Online]. Available: https://www.tutorialspoint.com/natural_language_processing/index.htm. [Accessed 25 1 2020].

[36] "Afaan Oromo," wikibook, 7 june 2018. [Online]. Available: https://en.wikibooks.org/wiki/Afaan_Oromo. [Accessed 25 11 2019].

[37] C. G.-R. Daniel Fernández-González, " Left-to-Right Dependency Parsing with Pointer Networks," *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, p. 710–716, June 2019.

[38] S. K¨ubler, "Why is German Dependency Parsing More Reliable," Indiana University.

[39] R. M. F. Pereira, "Non-projective Dependency Parsing using Spanning Tree Algorithms".

[40] M. P. Jinho D. Choi, "Getting the Most out of Transition-based Dependency Parsing," 2011.

[41] Oinkina, "Understanding LSTM Networks," 27 August 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 15 may 2020].

[42] E. M. Ponti, "MACHINE LEARNING TECHNIQUES," University of Pavia.

[43] " Language Understanding with Recurrent Networks," Sphinx, 2017. [Online]. Available: https://www.cntk.ai/pythondocs/CNTK_202_Language_Understanding.html. [Accessed 10 May 2020].

[44] †. B. M. Marie-Catherine de Marneffe, "Generating Typed Dependency Parses from Phrase Structure Parses".

[45] P. D. D. J. N. Prof. Dr. Jonas Kuhn, "Ensemble Dependency Parsing," 28. September 2020.

[46] M. Gasser, "A Dependency Grammar for Amharic".

[47] Y. J. K. T. Yunzhe Yuan, "Bidirectional Transition-Based Dependency Parsing".

[48] M. ELKAREF, "DEEP LEARNING APPLICATIONS FOR," January 2018.

[49] B. B. E. B. B. Mohab Elkaref, "A Simple LSTM model for Transition-based Dependency Parsing," vol. v1, 2017.

[50] G.-b. D. P. w. G. N. Networks, "Tao Ji, Yuanbin Wu, Man Lan," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 2475–2485, July 2019.

[51] G. Melbaa, "Afaan oromoo language," Gadaa.com, [Online]. Available: http://gadaa.com/language.html. [Accessed 11 12 2019].

[52] G. Mamo, "Parts of Speech Tagging for Afaan Oromo," *IJACSA.*

[53] R. Regasa, "MORPHOLOGICAL SEGMENTATION USING NEURAL," 2022.

[54] T. Point, Keras tutorials point, Pvt. Ltd., 2019.

[55] D. Thakur, "LSTM and its equations," 6 July 2018. [Online]. Available: https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af. [Accessed 2021 9 10].

[56] 2. Choe and Charniak, "Parsing as Language Modeling," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing: Association for Computational Linguistics*, p. 2331–2336, November 2016.

[57] D. e. al, "Recurrent Neural Network Grammars," *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 199–209, June 2016.

[58] E. C. a. M. J. David McClosky, "Effective Self-Training for Parsing," Brown University, 2006.

[59] C. D. M. A. Y. N. Richard Socher, "Learning Continuous Phrase Representations and," Stanford University.

[60] J. A. a. D. Klein, "How much do word embeddings encode about syntax," University of California, Berkeley.

[61] B. Megyesi, "data-driven syntactic analysis methods and application for Swedish," Stockholm, Sweden.

[62] "What Is the Importance of Computer Technology in Everyday Life?," Techwalla, 19 September 2018. [Online]. Available: https://www.techwalla.com/articles/what-is-the-importance-of-computer-technology-in-everyday-life. [Accessed 8 12 2019].

[63] B. E. Seyoum1, "Universal Dependencies for Amharic," Addis Ababa University1,3, National Institute of Informatics.

[64] J. B. a. P. Liang, "Imitation Learning of Agenda-based Semantic Parsers," *Transactions of the Association for Computational Linguistics,* vol. 3, p. 545–558, 2015.

[65] J. K. a. T. Mitchell, "Weakly supervised training of semantic parsers," *EMNLP/CoNLL),* p. 754–765., 2012.

[66] "HornMorpho: a system for morphological processing of Amharic, Oromo, and Tigrinya".

[67] A. Sharma, "Analytics adhya : How Part-of-Speech Tag, Dependency and Constituency Parsing Aid In Understanding Text Data?," 29 July 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/07/part-of-speechpos-tagging-dependency-parsing-and-constituency-parsing-in-nlp/. [Accessed 9 4 2021].

[68] A. Ö. Saziye Betül Özate¸, "A Hybrid Approach to Dependency Parsing," 2020.

[69] P. Q. C. D. Manning, "Arc-swift: A Novel Transition System for Dependency Parsing," 2017.

[70] Y. W.. a. M. L. Tao Ji, "Graph-based Dependency Parsing with Graph Neural Networks," 2017.

[71] "Hyperparameter Tuning for Machine Learning Explained," 6 Aug 2020. [Online]. Available: https://www.mygreatlearning.com/blog/hyperparameter-tuning-explained/. [Accessed 4 October 2020].

[72] "bb afaanoromoo," [Online]. Available: https://www.bbc.com/afaanoromoo. [Accessed 3 6 2021].

[73] "NLP," NLP, 5 May 2021. [Online]. Available: https://www.datarobot.com/blog/what-is-natural-language-processing-introduction-to-nlp/. [Accessed 3 2 2022].

9. APPENDIX

9.1 Arc standard algorithm

Based on transition predicted we can gain head and dependents from model1 and use as input for model2.

```
conf_at_each_step.append([list_to_flat(test_word_stack),
                          list_to_flat(test_word_buffer),
                          list_to_flat(test_pos_stack),
                          list_to_flat(test_pos_buffer),label_of_transition])
    if predicted_transition == 0:
        if len(test_word_buffer)==0:
            print('SHIFT cannot be applied for empty word buffer!')
            break;
        poped_word=test_word_buffer.pop(0)
        poped_pos=test_pos_buffer.pop(0)
        test_word_stack.append(poped_word)
        test_pos_stack.append(poped_pos)
    if predicted_transition == 1:
        if len(test_word_stack)<2:
            is_completed=False
            print ('LEFT_ARC cannot be applied for word stack with size  less than two!')
            break;
   head_dependent_list_of_word.append([test_word_stack[len(test_word_stack)-
1],test_word_stack[len(test_word_stack)-2]]);
        head_dependent_list_of_pos.append([test_pos_stack[len(test_pos_stack)-
1],test_pos_stack[len(test_pos_stack)-2]]);
        poped_word=test_word_stack.pop(len(test_word_stack)-2)
        poped_pos=test_pos_stack.pop(len(test_pos_stack)-2)
    if predicted_transition == 2:
        if len(test_word_stack)<2:
            is_completed=False
            print ('RIGHT_ARC cannot be applied for empty word stack!')
```

```
            break;
   head_dependent_list_of_word.append([test_word_stack[len(test_word_stack)-
2],test_word_stack[len(test_word_stack)-1]]);
        head_dependent_list_of_pos.append([test_pos_stack[len(test_pos_stack)-
2],test_pos_stack[len(test_pos_stack)-1]]);
        poped_word=test_word_stack.pop(len(test_word_stack)-1)
        poped_pos=test_pos_stack.pop(len(test_word_stack)-1)
for x in conf_at_each_step:
    print(x)
    print('')
print(head_dependent_list_of_word)
print(head_dependent_list_of_pos)
```

9.2 Sample Afaan Oromoo treebank

#Boontuun dheeneadda xalayaa katabde. Boontuu wrote a letter (the day) before

NO	WORD	POS	HEAD	RELATION
1	boontuu	noun	5	SUBJ
2	n	noca	1	NCM
3	dheengadda	adv	5	ADV
4	xalayaa	noun	5	DOB
5	katab	vert	-	-
6	d	gnd	5	GEND
7	e	tens	5	TNS

#Hundeen saffisaan mana haxaawe. Hundee cleaned the house quickly.

NO	WORD	POS	HEAD	RELATION
1	hundee	noun	5	SUBJ
2	n	noca	1	NCM
3	saffisaan	adv	5	ADV
4	mana	noun	5	DOB
5	haxaaw	vert	-	-

9.3 AOPOS tags

No	Specific AOPOS	Abbreviations	Stream AOPOS
19.	Abstract noun	abno	Nort
20.	Process or action noun	acno	
21.	Result noun	reno	
22.	Gerundive noun	geno	
23.	Manner noun	mano	
24.	Instrumental noun	inno	
25.	Agentive noun	agno	
26.	Auto benefective verb	auv	Vert
27.	Causative verb	cav	
28.	Stative verb	stv	
29.	Passive verb	pav	
30.	Noun noun	nncon	Cono
31.	Noun adjective	nacon	
32.	Adposition noun	ancon	
33.	Verb noun	vncon	
34.	Noun noun	Nncoa	Coad
35.	Noun adjective	nacoa	
36.	Noun numeral	nucoa	

37.	Adjective noun	ancoa	
38.	Positive copula	poco	Copula
39.	Negative copula	neco	
40.	Neutral sex noun	neno	Gnno
41.	Masculine noun	mano	
42.	feminine noun	feno	
43.	Natural female gender	nafe	
44.	Feminine verb	feve	Gnve
45.	Masculine verb	masve	
46.	Neutral sex verb	neve	
47.	Masculine adjectives	maad	Gnadj
48.	Feminine adjectives	fead	
49.	Neutral sex adjective	nead	
50.	Plural noun	plno	PL
51.	Plural adjectives	plad	
52.	Plural verb	plve	
53.	Cardinal numbers	Canu	NUM
54.	Ordinal numbers	Ornu	
55.	Indefinite pronouns	Inpr	
56.	Indefinite pronouns persons	inpr(quan)	
57.	Post position	popo	ADP

58.	Prepositions	prpo	
59.	Para positions	papo	
60.	Coordinating conjunctions	cconj	CONJ
61.	Subordinating conjunctions	sconj	

9.4 Afaan Oromoo Relations

No	Relations	Abbrevatiosn
27.	Noun complement	NCOM
28.	Personal pronoun	PEPN
29.	Interrogative pronoun	WHPR
30.	Pronoun	PRON
31.	Negative	NEG
32.	Focus class marker	FCM
33.	Person	PERS
34.	Punctuation	PNC
35.	Adjective complement	ADCOM
36.	Adposition	ADP
37.	Adverb	ADV
38.	Order	ORDER
39.	Demonstrative pronoun	DEPR
40.	Auxiliary	AUX
41.	Cardinal number	CANO
42.	Case	CASE
43.	Coordinating conjunction	CCONJ
44.	Conjunct	CNJT
45.	Compound noun	CNON
46.	Compound adposition	COAP

47.	Comparative case suffix	COCS
48.	Comparative degree	CODR
49.	Copula	COPULA
50.	Definite article	DART
51.	Direct object	DOB
52.	Double genitive	DOGE
53.	Gender	GEND
54.	Interjection	INJ
55.	Indefinite pronoun	INPR
56.	Instrument	INST
57.	Indirect object	IOBJ
58.	Jussive	JUSS
59.	Main verb	MVER
60.	Number	NUM
61.	Noun class marker for the pronoun	NCMP
62.	Nominalization	NOMZ
63.	Noun subject	NSUBJ
64.	Ordinal subject number	ORNO
65.	Part of the word	PART
66.	Passive	PASS
67.	Pronoun subject	PSUBJ
68.	Positive degree	PODG
69.	Positive	POS
70.	Quantifier	INQU
71.	Questions	QUES
72.	Subordinate conjunction	SCONJ
73.	Superlative adjective	SUAD
74.	Subject	SUBJ
75.	Superlative quantifier	SUQU
76.	Tense	TNS
77.	Verb formation	VEF

9.5 Configurations for first model POS part

```
#kuullaniin arjaa dha.

[ ]                         [ prno noca adjb poco punc ]          0
[ prno ]                    [ noca adjb poco punc ]         0
[ prno noca ]               [ adjb poco punc ]          2
[ prno ]                    [ adjb poco punc ]          0
[ prno adjb ]               [ poco punc ]           2
[ prno ]                    [ poco punc ]           0
[ prno poco ]               [ punc ]            1
[ poco ]                    [ punc ]            0
[ poco punc ]               [ ]             2
[ poco ]                    [ ]             2


#Arrabsoon gadheedha.

[ ]                         [ noun noca adjb poco punc ]          0
[ noun ]                    [ noca adjb poco punc ]         0
[ noun noca ]               [ adjb poco punc ]          2
[ noun ]                    [ adjb poco punc ]          0
[ noun adjb ]               [ poco punc ]           2
[ noun ]                    [ poco punc ]           0
[ noun poco ]               [ punc ]            1
[ poco ]                    [ punc ]            0
[ poco punc ]               [ ]             2
[ poco ]                    [ ]             2
```

9.6 Head dependent for second model POS part

```
[prno noca]   1
[noun adjb]   2
[vert neg]   3
[vert noun]   4
[vert prno]   5
[vert tens]   6
[vert punc]   7
[prno noca]   1
[noun adjb]   2
[vert noun]   4
[vert adv]   8
[vert prno]   5
[vert gnd]   11
[vert tens]   6
[vert punc]   7
[prno noca]   1
[noun adjb]   2
[poco noun]   4
[poco prno]   5
[poco punc]   7
[noun noca]   1
[noun adjb]   2
[vert neg]   3
[vert noun]   10
[vert gnd]   11
[vert tens]   6
[vert punc]   7
[prno noca]   1
[prno adjb]   2
[vert prno]   5
```

9.7 Algorithm to convert treebank data to initial configurations

```
with open ('data/tabletree.txt') as f:
    lines=f.readlines()
new_sentence=True
words_pos=[]
words=[]
pos=[]
for line in lines:
```

```
    line=line.strip()
    if line=='':
        new_sentence=True
        if len(words) >0:
            words_pos.append([words,pos])
        words=[]
        pos=[]
        continue;
    if  line[0] == '-':
        continue;
    if  line[0] == '#':
        continue;
    if  line[0] == 'N' and line[1] == 'O':
        continue;
    splitted=line.split('|')
    words.append((splitted[1]).strip())
    pos.append((splitted[2]).strip())
if len(words) >0:
    words_pos.append([words,pos])
data=''
maxlen=0;
for x in words_pos:
    x[0]=flat_word=list_to_flat(x[0])
    x[1]=flat_word=list_to_flat(x[1])
    if len(x[0]) > maxlen:
        maxlen=len(x[0])
for x in words_pos:
    data=data+x[0]+(' '*(max len-len(x[0])))+x[1]+'\n\n'
fo=open('data/reversed_data.txt',"w")
fo.write(data)
fo.close(
```