ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL AND COMPUTATIONAL SCIENCES

SCHOOL OF INFORMATION SCIENCE

Morphological Segmentation for Amharic Verb Class Using Recurrent Neural Network (RNN)

A Thesis Submitted to the School of Information Science of Addis Ababa University in Partial Fulfillment of the Requirement for the Degree of Master of Science In Information Science and Systems (Language Technology)

MSc Thesis

By

Wondimagegnhue Tsegaye Tufa

Advisor:

Wondwossen Mulugeta Gewe(PhD)

September, 2019

Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL AND COMPUTATIONAL SCIENCES

SCHOOL OF INFORMATION SCIENCE

Morphological Segmentation for Amharic Verb Class Using Recurrent Neural Network (RNN)

A Thesis Submitted to the School of Information Science of Addis Ababa University in Partial Fulfillment of the Requirement for the Degree of Master of Science In Information Science and Systems (Language Technology)

MSc Thesis

By

Wondimagegnhue Tsegaye Tufa

Advisor:

Wondwossen Mulugeta Gewe(PhD)

September, 2019

Addis Ababa, Ethiopia

Addis Ababa University

Collage of Natural and Computational Science

School of Information Science

Morphological Segmentation for Amharic Verb Class: Using Recurrent Neural Network (RNN)

MSc Thesis By: **Wondimagegnhue Tsegaye Tufa**

Name and signature of Members of the Examining Board

Wondwossen Mulugeta(PhD)		Oct 01/2019
Advisor	Signature	Date
Million Meshesha (PhD)		Oct 01/2019
Examiner	Signature	Date
Melkamu Beyene (PhD)		Oct 01/2019

Declaration

I declare that this research is my original work and has not been presented for a degree in any university, and that all sources of material used for the research have been properly acknowledged.

Declared by:

Name: Wondimagegnhue Tsegaye Tufa

Signature: _____ 30/2019

This research has been submitted for Examination with my approval as university advisor.

Name: Wondwossen Mulugeta (PhD), Advisor

Signature: _____

Date: Sept 30/2019

Addis Ababa, Ethiopia

September, 2019

Acknowledgment

Firstly, I would like to express my gratitude to Addis Ababa University for hosting one of the most existing master program and my university, Bahir Dar University, for fully sponsoring my study and make my learning experience more exiting. Secondly I would like to thank my advisor for showing me the problem direction and gave me unreserved guidance, supervision and constructive suggestion during my research work. I would also like to think all of my teachers and classmates for helping me explore the unknown. Finally, I would like to thank God for giving me the chance to start and finish my thesis.

Contents

List of Table

List of figure

Acronyms

ANNs Artificial Neural Networks

CWS Chinese Word Segmentation

FNN Feedforward Neural Networks

FST Finite State Transducers

GRNN Gated Recursive Neural Network

GRU Gated Recurrent Unit

HMM Hidden Markov model

ILP Inductive Logic Programming

IPA International Phonetic Association

LSTM Long Short-Term Memory

MBL Memory Based Learning

MDL Minimum Description Length

MLP Multilayer Perceptron

MRL Morphologically Rich Languages

NLM Neural Language Models

NLP Natural Language Processing

OOV Out of Vocabulary

PGM Probabilistic Generating Model

POS Part of Speech

RNN Recurrent Neural Network

seq2seq sequence to sequence

SERA System for Ethiopian. Representation in ASCII

SRNN Simple-RNN

SVM Support Vector Machine

TLM "Two Level" Morphology

Abstract

Due to the dependency of higher-level NLP task on morphological analysis, lack of an appropriate tool for morphological analysis is a major bottleneck for research work conducted on high level NLP application such as machine translation, speech processing, text summarization and many more. Currently most research works done on morphological analysis for morphologically rich languages (MRL) like Amharic are based on techniques that require high supervisions or rely on rule-based techniques that require detailed enumeration of the rule of the language to be crafted manually. Both of these techniques require a high-quality data in terms of capturing the rules that exist in the language and it also require a significant quantity of training data for better generalization. Both of these requirements are challenging to overcome due to the fact that significant number of MRL like Amharic are under-resourced. Lack of training data in quality and quantity is a major obstacle for research work in low resourced and morphological rich languages. The low resource state and morphological complexity of the language demand techniques that can provide better learning with relatively small number of example and be able to capture the complexity of language. In this paper, we propose RNN based sequence-to-sequence model that provides an encouraging performance in learning complex segmentation with small number of example and with no linguistic annotation, using the state-of-the-art encoder-decoder architecture. We have approached the problem of morphological segmentation as transformation task by considering the surface word as an input and the segmentation as a transformation process to produce a list of segmented morpheme. We prepared a training data by selecting different class of verbs. We have explored different encoder unit in terms directionality, window size, encoder type and size, and different data representation paradigm. The experiment showed that our model can learn a complex segmentation with no linguistic annotation and with limited number of examples. The model showed 74.2% accuracy on segmentation and 98.7% on morpheme boundary accuracy. We have shown that it is possible to learn morphological segmentation without relying on linguistic annotation. These contribute towards a general solution that can work on language other than Amharic. Our work can be extended to include other common POS classes such as nouns and adjective. Extending the work to include analysis would make the work more relevant for higher level NLP applications such as machine translation, speech recognition and spell checker.

Key words: Recurrent Neural Network, Amharic Morphology, Encoder-Decoder Architecture.

Abstract

Due to the dependency of higher-level NLP task on morphological analysis, lack of an appropriate tool for morphological analysis is a major bottleneck for research work conducted on high level NLP application such as machine translation, speech processing, text summarization and many more. Currently most research works done on morphological analysis for morphologically rich languages (MRL) like Amharic are based on techniques that require high supervisions or rely on rule-based techniques that require detailed enumeration of the rule of the language to be crafted manually. Both of these techniques require a high-quality data in terms of capturing the rules that exist in the language and it also require a significant quantity of training data for better generalization. Both of these requirements are challenging to overcome due to the fact that significant number of MRL like Amharic are under-resourced. Lack of training data in quality and quantity is a major obstacle for research work in low resourced and morphological rich languages. The low resource state and morphological complexity of the language demand techniques that can provide better learning with relatively small number of example and be able to capture the complexity of language. In this paper, we propose RNN based sequence-to-sequence model that provides an encouraging performance in learning complex segmentation with small number of example and with no linguistic annotation, using the state-of-the-art encoder-decoder architecture. We have approached the problem of morphological segmentation as transformation task by considering the surface word as an input and the segmentation as a transformation process to produce a list of segmented morpheme. We prepared a training data by selecting different class of verbs. We have explored different encoder unit in terms directionality, window size, encoder type and size, and different data representation paradigm. The experiment showed that our model can learn a complex segmentation with no linguistic annotation and with limited number of examples. The model showed 74.2% accuracy on segmentation and 98.7% on morpheme boundary accuracy. We have shown that it is possible to learn morphological segmentation without relying on linguistic annotation. These contribute towards a general solution that can work on language other than Amharic. Our work can be extended to include other common POS classes such as nouns and adjective. Extending the work to include analysis would make the work more relevant for higher level NLP applications such as machine translation, speech recognition and spell checker.

Key words: Recurrent Neural Network, Amharic Morphology, Encoder-Decoder Architecture.

CHAPTER ONE

INTRODUCTION

1.1. Background

In natural language processing, we often require a smaller unit of a word-form for most of higher level task such as machine translation, information retrieval, spell-checker, speech recognition and text summarization [1][2][3]. Splitting a word to produce meaningful sub word, which contributes to the meaning of the word, is one of the important task for language processing. We do this with a process called Morphological Segmentation. Morphological Segmentation deals with reducing a surface form of a word into a sub component where this subcomponent has a contribution to the meaning of the whole word. Identifying this small unit is the primary task of a morphological processing and analysis.

Morphological segmentation is a fundamental task in a number of natural language processing. In statistical machine translation, languages that exhibit a higher word inflection impose a challenge due to the many form of a word can take which create a data sparsity problem [4]. This problem can be addressed by applying a morphological analyzer to reduce the word form which could give clue on the matching structure in the target language. In automatic speech recognition, Out Of Vocabuiary (OOV) words, which is the result of data sparsity, is reported to have a negative effect in the recognition performance [5]. Using morpheme as a basic unit of processing has showed a significant performance improvement in this regard. In spell-checker systems, the task of detecting a spelling error and suggesting a corrected word form could be assisted with a morphological analysis module. This is especially feasible for morphologically rich languages which are characterized by the complexity and higher degree of word production feature.

There are two major approach in computational morphology: Rule based and Corpus based [1]. A rule based approach requires a detailed linguistic knowledge and a representation of this knowledge in a computational unit with predefined algorithmic approach. A common computational paradigm for rule based approach is use of finite state model [3]. In such technique, the knowledge is used to dictate the formation of a word using states and transitions. This approach is reported to be an efficient way of implementation in computational morphology [3]. The major

downside of this approach is constructing a transducer require multiple rule that should be embedded in each unit of the transducer [1]. This in turn requires detailed rules of the language to be enumerated and hand-coded. Modifying and debugging FST models is also a challenging task.

Machine learning approach, on the other hand, learns the rules from a dataset with an example (supervised learning) or from a raw data (unsupervised learning) [1]. Unsupervised learning is learning from pattern in a dataset while supervised learning requires an annotated example to learn.

The pioneer work in unsupervised learning of morphology was proposed by Goldsmith [6] which used minimum description length (MDL) to learn the underlying morph segmentation. This approach however assumes concatenative characteristics of a language so it would only be applicable for concatenative languages [6]. Some of commonly explored supervised learning techniques include Inductive Logic Programming (ILP) and Memory Based Learning (MBL). Inductive Logic Programming (ILP) on the other hand learns from example and background knowledge in a relational representation [2]. The approach has been used for Amharic Verb Affix segmentation [2]. The work reveals how an incremental learning technique could be applied to learn affix segmentation using simple and complex examples with a multiple stage of learning. Memory Based Learning (MBL) uses a stored example in a memory to classify new data instance by comparing the similarity between the feature of an existing example data and the new data instance [1].

In recent years, several studies have pointed out the advantage and possibility of being able to learn linguistic structure from raw input features without any additional feature engineering. Wang et al. [7] explored Window LSTM Neural Networks for learning morphological boundary directly from raw input words and are subsequently able to predict morphological boundaries on Hebrew and Arabic languages. The approach rely on Long Short Term Memory (LSTM) units with windows of characters to capture contextual information. Reddy et al [8] explored deep sequence to sequence (seq2seq) model that takes raw input string as the input and predicts the segmented string for Sanskrit language. Reddy et al [8] rely on encoder-decoder framework for word segmentation problem, and propose a deep seq2seq model using LSTMs for the prediction task.

1.2. Statement of the Problem

The major task of a morphological analysis involves segmentation of a word into their constituent morpheme [9]. For morphologically rich languages like Amharic, analysis at morphology level is a significant first step for the high level natural language processing task[1][2][3].

As a family of Semitic languages, Amharic is characterized by complex and productive morphology [2]. This complexity is contributed to three factors: lexical content, templating and affixation. In addition to lexical information, the morphemes in Amharic verb convey different information which includes subject and object person, number, and gender; tense, aspect, and mood; various derivational categories; relativization; and a range of prepositions and conjunctions. Stem formation is a combination of root, vowel and template merger which result in a non-concatenative morphology.

This complexity imposes a challenge for almost all of NLP tasks that depends on dictionary and corpus (machine learning approaches). Due to high inflexion at the word level, using a surface form of a word in application such as speech recognition application would probably result in unrecognizable word because the chance to include all possible form of a word in pronunciation dictionary is impractical, memory intensive and expensive. This process is also significant for other higher level NLP tasks that depend on corpus. By avoiding the need to store many possible form of a word and relying on morphological analysis model, we can reduce data sparsity problem which is common to most NLP application.

Despite the relevance of such segmentation model, Morphological analysis of Amharic is not a trivial task and requires techniques that can successfully capture morphological complexities of the language. Until recently, different approach has been explored for Amharic. The two major approaches that have been explored are rule based and machine learning. A rule based approach requires a detailed linguistic knowledge and a representation of this knowledge in a computational unit. In such technique, the knowledge is used to dictate the formation of a word using states and transitions. This approach is an efficient way of implementation in computational morphology [9]. The major downside of this approach is the time and expertise it requires in crafting and encoding linguistics rule. Scaling up a rule based approach on the same language and to other similar languages is also hard due to the hand crafting of rules required to construct the FST.

Machine learning approach, on the other hand, learns from a dataset with an example: supervised learning or from a raw data: unsupervised learning. Two major supervised techniques have been explored for Amharic: Inductive Logic Programming (ILP) and Memory Based Learning (MBL). Inductive Logic Programming (ILP) has been used for Amharic Verb Affix segmentation [2]. The work explored how an incremental learning technique could be applied to learn affix segmentation using simple and complex examples with a multiple stage of learning. Memory Based Learning (MBL) uses a stored example in a memory to classify new data instance by comparing the similarity between the feature of an existing example data and the new data instance [1].

Despite these attempts, there still exist a gap in exploring approaches that matches the current state of the language. As MRL and under resourced at the same time, the major gap that exist in languages like Amharic is an insight, in terms of what techniques are more suitable for learning segmentation in a low resource setting. In this sense, low resource setting implies both the quantities: minimizing the number of examples required for training and the quality: minimizing the linguistic knowledge required for supervision. The low resource state of the language demand technique that can learn morphological segmentation with a small number of examples and minimum linguistic annotation.

In a recent development, several studies [10][11][12] have pointed out the advantage and possibility of being able to learn linguistic structure from raw data without any additional feature engineering [7]. In this study our aim is to design a morphological segmentation model without relying on linguistics feature of Amharic verb.

1.3. Research Questions

In this work, we explored encoder decoder architecture with the following research questions.

- ✓ What Parameters is optimal for learning morphological segmentation in terms of the training data size, evaluation and model implementation?

1.4. Research Objective

1.4.1. General Objective

The general objective of this research work is design RNN architecture that can be applied for learning Amharic morphology segmentation under a low resource setting.

1.4.2. Specific Objectives

To achieve the general objective, the following specific objective is conducted.

- ✓ To design an experiment with different type of features.
- ✓ To design an experiment with varying training data.
- ✓ To develop machine learning models using the selected architectures and parameters.
- ✓ To evaluate the performance of model based on a selected evaluation technique.

1.5. Scope and limitation of the study

This research work is limited to the following activities:

- ✓ We only used Amharic verb and other part of speech class such as nouns and adjectives are not included.
- ✓ Our approach depends on availability of surface word and segmentation pair which implies that we are limited on a supervised approach.
- ✓ We only focused on segmentation of surface word into valid morphemes and no further analysis information such as grammatical information is included.
- ✓ We also relied on small set of training data randomly selected from a word list organized by Biniam[1].
- ✓ We also focus on three consonant verbs for our training due to lack of a two consonant verbs in our source corpus.

1.6. Significance of the study

The significance of our work includes:

- ✓ We have showed the possibility of learning morphological segmentation from a raw text without relying on a linguistics annotation.
- ✓ We have showed the effect of size of training data on the task of morphological segmentation
- ✓ We have showed the effect of different encoder unit of the task of morphological segmentation

[1] [1] http://www.cs.ru.nl/~biniam/geez/crawl.php

- ✓ We have showed the effect of different encoder unit of the task of morphological segmentation.

1.7. Methodology

1.7.1. Research Design

Our objective in this research work requires building artifacts, algorithms and models, to produce an improved learning model. For such kind of research work, experimental research is a more suitable research method. Design science allows us to follow a framework in building and fine-tuning artifacts tailored to the objective of our research work.

1.7.2. Data Preparation

For supervised learning techniques, the primary data required are words in both unsegmented and segmented form. The segmented data is used as an example for training different learning models. We have prepared two data sets. The first data set has 780 words prepared using the word ሰብር/seber/break/, አለ/ale, ገደለ/gedel/kill, which are selected as a representative example: exhibit high production and are class of bi-radical and tri-radical. The second data set contain 3,000 words which were randomly selected from word list crawled from[2]. To prepare a segmented data, we have used HornMorpho-2.5, a rule based Amharic morphological analyzer that include a segmentation module. We have further processed the output to suit our representation. We use segmentation done by HornMorpho as a training example for our case. For training purpose, we used google's generic sequence to sequence framework, an open source neural network library written in Python. For other programming task we used python 2.7.

1.7.3. Implementation tools

In this research work, we have explored RNN architecture with Long Short Term Memory (LSTM) and gated recurrent unit (GRU) cells. A recurrent neural network (RNN) is a family of artificial neural network where connection between hidden units of a network is directed cycles. This network configuration of the network provides a capability to process complex sequencing relationship by using connections as internal memory. This makes it suitable for sequence to sequence problems such as language. Based on these architecture, we have explored variety pf

[2] http://www.cs.ru.nl/~biniam/geez/crawl.php

architecture based on directionality (unidirectional or bidirectional), depth (multi-layer); and type (Simple RNN, a Long Short-term Memory (LSTM), or a gated recurrent unit (GRU). We have explored variable window size (number of characters feed into encoder layer) by taking window size as a feature.

Python 2.7

As a programming environment we have used python 2.7. We have selected python due to its strong back from a neural network research community and availability of deep learning libraries and resources.

HORNMORPHO 2.5

It is a Python program that analyzes Amharic words into their constituent morphemes and generates words, given a root or stem and a representation of the word's grammatical structure. It is free software. We have used Amharic segmentation module from this program for preparing training and testing data. The program accept Amharic word in Amharic Fidel format (e.g ይፈልጋሉ) and output lists of morphemes with their grammatical information (e.g y(sb=3sm|3p)-{flg+1e2_3 (imprf)-al_u(aux,sb=3p)). As described data preparation section, we used a python program to further process this output to extract the morphemes.

GloVe(Global Vectors for Word Representation)

GloVe is an unsupervised learning algorithm for obtaining vector representations for characters or words. Training is performed on aggregated global word-word or character-character co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. We have used glove to build a character based embedding. We have selected glove over word-embedding's due to an availability of more parameter (e.g: window-size) to build a character embedding's.

Matplotlib

Matplotlib is a Python 2D visualization library which generated figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts. We have used this library along with python to visually observe learning performance change (Increase or decrease over time) during model training and to spot over fitting if occurred.

Numpy

NumPy is a Python library for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. We have used numpy along with tensor flow.

Google Seq2Seq

For building our segmentation model we used Google Seq2Seq, a general-purpose encoder-decoder framework based on Tensorflow that can be used for Machine Translation, Text Summarization and Conversational Modeling.

Tensor flow

TensorFlow is an open source library for high performance numerical computation. It support computations both on CPU'S and GPU's. It is used for machine learning applications such as neural networks.

Tensor-board

Tensor-board is a suite of visualization tools called. We used TensorBoard to visualize our TensorFlow graph, plot quantitative metrics about the execution of our graph, and show additional data like images that pass through it.

1.7.4. Evaluation Method

In evaluating our model, we have considered morphological segmentation as a two-step process: discovering the existence of a morpheme and generating a morpheme that matches the reference morpheme. We have evaluated the first step through evaluating Morpheme-existence, by counting and comparing the morpheme boundary, between the model output and the reference. On the second evaluation, Accuracy, we have compared the exact morpheme match between the model output and the gold standard reference.

1.7.5. Organization of the research

This research document is organized in seven chapters. In chapter one we present background of the research, statement of the problem, research question, research objective, significance of the study and research methodology. In chapter two we present a literature review on computational morphology, Amharic morphology, approaches to morphological segmentation and related

research work conducted on morphology. In chapter we present about the design of our model which include tools used, training data selection and preprocessing and model architecture. In chapter four, we present our experimental design, description of these experiments and the result for each of this experiment. In chapter five, we present summary and conclusion of our work. In chapter six, we included all of the reference we used and in the last chapter contain the appendix.

CHAPTER TWO

LITERATURE REVIEW

2.1. Overview

In natural language processing (NLP), we often require a small unit of a word for high level applications such as machine translation, information retrieval, spell-checker, and speech recognition and text summarization. The process of extracting this small unit is often a crucial task in language processing. We do this with a process called Morphological segmentation. Morphological segmentation is part of Computational Morphology that deals with splitting a word into a sub component where this subcomponent has a linguistic role to the meaning of the surface word. Computational morphology is a discipline under NLP that deals with understanding written and spoken language for automatic computational processing of language [13].The primary task of a computational morphology is to extract linguistic information from an input string for the higher level NLP task.

There are two paradigms under machine learning approach, supervised learning and unsupervised learning. In supervised learning, we use annotated data as way to supervise the learning process. In the case of unsupervised learning, we use raw data for training task. Unsupervised learning approach is appealing due to the fact that it can be applied to any language with sufficiently large dataset in electronic form [14]. Supervised learning techniques differ in the extent of supervision used in training phase. This variation dictate the annotation level required in the training data. There are different algorithm under supervised learning family such us supervised learning paradigm include inductive logic programming (ILP), support vector machine (SVM), hidden Markov model (HMM) and memory-based learning (MBL) [1]. Inductive Logic Programming (ILP) in generals learns from example and background knowledge in a relational representation. Memory Based Learning (MBL) uses a stored example in a memory to classify new data instance by comparing the similarity between feature of an existing example data and the news data instance.

As a family of Semitic languages, Amharic is characterized by complex and productive morphology [2]. The richness in morphology is contributed to the templating nature of the language and affixation process. Words in Amharic convey information such as subject and object

person, number, and gender; tense, aspect, and mood; various derivational categories; relativization; and a range of prepositions and conjunctions. The formation of word in Amharic follows templating. The stem is formed with root + vowels + template (e.g: sbr + ee + CVCVC, which results in the stem seber /broke/) [2].

2.2. Concept and Terminology

For a better understanding of the difficult concept used in this study, we have provided definition of commonly used terms as follows.

Word: It is a sequence of characters used in a language which represent a meaningful concept. In Amharic words are usually separated from each other using a white space or a known delimiter [2].

Stem: It is a part of a word form without prefixes and suffixes. In Amharic it could have internal structure which may affect the grammatical features of the word. The internal structure is left intact within the stem [2].

Root: In Semitic languages like Amharic, a Root is a sequence of consonants extracted from the Stem where all the vowels used to form the stem are removed [2]. The Root conveys the basic semantic meaning of the word.

Morphemes: Morphology is the study of the way words are built up from smaller meaning-bearing units [2]. This meaning bearing smaller unit is referred as morphemes. For example, the word cat consists of a single morpheme (the morpheme cat) while the word cats consist of two morphemes (cat and s). There are four types of morphemes [13]:

- ✓ Free morphemes: provide meaning by itself, e.g.: town and dog, can appear with other lexemes (as in town hall or dog house) or they can stand alone.
- ✓ Bound morphemes: provide meaning only when combined with other morphemes e.g "un-"appear only together with other morphemes to form a lexeme. Bound morphemes are mostly prefixes and suffixes.
- ✓ Derivational morphemes can be added to a word to derive another word. The addition of "-ness" to "happy" for example, give "happiness".

- ✓ Inflectional morphemes modify a words tense, number, aspect, and so on, without deriving a new word or a word in a new grammatical category (as in the "dog" morpheme if written with the plural marker morpheme "-s" becomes "dogs").

Morpheme boundary: The boundary represents the location of a character in a segmented word that would result in a valid morpheme [13].

Morphotactic: Morphotactic is the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. The order in which morphemes follow each other is strictly governed by a set of rules called morphotactic. Natural languages use different means to form a word: compounding, affixation, reduplication, conversion and stem alternation. In compounding, we combine two or more lexemes into one [13]. The morphemes of a word can only occur in a certain allowed order. The morphemes can be divided into a number of classes and the morpheme sequences are normally defined in terms of the sequence of classes.

Affixes: Affixes are class of morpheme that usually contribute additional meaning by combining with the main morpheme [13]. Affixes are classified based on the location of the form they are attached to. Prefixes are attached before a main morpheme and suffixes after the main form.

Types of Morphology: There are three broad classes of word formation from morpheme: Inflection, Derivation and Compounding.

- ✓ Inflection: is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem. It usually fills syntactic function and is productive. The meaning of the resulting word is easily predictable. Inflectional morphemes modify a word's tense, number, aspect, and the like.
- ✓ Derivation: is the combination of a stem with a grammatical morpheme which results in a word of a different class. It is hard to predict the meaning of the resulting new word. In case of derivation, the part of speech (POS) of the newly derived word may change.
- ✓ Compounding: is combining of two or more base units to form a new word. Compounds are formed by combining uninflected noun forms with semantic content with either different inflected verbal forms with no semantic content. An important concept in compounding is the notion of head. A compound noun is divided into head and modifier or modifiers. For instance, the compound noun watchtower in which watch and tower can be represented as a head and modifier.

2.3. Morphological Segmentation

Morphological process is set of task performed on sequence of characters with the goal of producing description of the surface form of a word or to generate the surface form from a given description [13]. Morphological Segmentation deals with splitting a word into valid components with the assumption that the sub-components play a role to the meaning of the whole word [2]. For most languages, the process involves identifying the prefix and suffix morphemes of a word. This process leaves two categories of morphemes, the affix and the stem. A deep level of segmentation includes further segmentation of the affix-sequences into smaller units and further analysis of the stem to separate the vocals from the root in template based morphology. In morphology, it is believed that the whole is made of the pieces and correct identification of the pieces is vital in building the whole or building another object by various combinations of these pieces. In segmentation, when words have non-concatenative and fusional behavior like in Amharic Language, the task would be challenging. The challenge is on identifying the boundary between adjacent morphemes and identifying orthographic changes. For example the Amharic word [t-gedy-alex] (ትገድያለሽ) from [gdel]ገድል where ል[l] is changed to ይ[Y].In such cases, it is common to see the morphemes change their surface form as a result of contact with other pieces. Morphophonemic alterations are typical examples of such changes. But it has to be noted that segmentation of words into smaller valid pieces help show associations between grammatical features and the segmented morphemes.

2.4. Amharic Morphology

2.4.1. Overview

Amharic is family of Semitic language which characterized by morphologically richness. It is the second most widely spoken Semitic language in the world [13]. It is also the official working language of the Federal Democratic Republic of Ethiopia. This language has been in use nationwide for many years where various printed and electronic documents have been produced. In addition, Amharic is also the working language of several of the regional states within the federal system of Ethiopia. Amharic is written using the Ethiopic script called Fidel, which is originated from Ge'ez abugida (alphabet, letters, or characters). The Ethiopic script is claimed to be developed from the Sabaean/Minean script [13]. Ethiopic has 34 basic characters/letters. Each of these letters can be modified by six vowels to create six syllabic symbols making a total of

seven letters for each entry. The letter of the 6th order is taken as the basic where all the rest are derivations. There are also other unique letters that are extended derivation of the basic character that are used to represent unique sounds. From the 34 characters 28 have such extended derivations. There are also few basic letters that have other extended derivations which are not exhibited by most of the letters. In a regular use most of these derivations are rare. The Ethiopic script is transliterated or Romanized using a conversion standard. There is no single standard for transliterating the Ge'ez script into the Latin alphabet. A relatively complete and documented transliteration standard for Amharic scripts are the System for Ethiopian. Representation in ASCII (SERA), the International Phonetic Association (IPA) conversions. Many researchers working on Ethiopic script however tends to use their own conversion standard or modified version of SERA for their convenience [13].

2.4.2. Amharic Writing System

Amharic is written using the Ethiopic script called Fidel, which is originated from Ge'ez abugida (alphabet, letters, or characters). Ethiopic contain 34 basic characters/letters. Each of these letters can be changed by one of the six vowels to create six syllabic symbols making a total of seven letters for each entry. The letter of 6th order is taken as the core and the rest are derivations. Amharic also have other distinct letters that are extended derivation of the core character and these are used to represent unique sounds.

Among the 34 characters 28 of them have such extended derivations. There are also few basic letters that have other extended derivations which are not exhibited by most of the letters. In a regular use most of these derivations are rare. The Ethiopic script is converted or romanized using a conversion standard. There is no single agreed upon standard for transliterating the Ge'ez script into the Latin alphabet. A relatively complete and documented transliteration standard for Amharic scripts are the System for Ethiopian. Representation in ASCII (SERA), the International Phonetic Association (IPA) conversions [13].

2.4.3. Amharic Part of Speech (POS)

The different parts of speech, formation and interrelationships which constitute the morphology of Amharic words have been studied in detail [13]. There are eight parts of speech in Amharic [13]. This include: Noun, Pronoun, Verb, Adverb, Adjective, Preposition, Conjugation and Interjection. The classification is based on the position of these words in a sentences and the type of morphemes

in their formation. Analysis of the Amharic word structure shows that verbs are most complex among the other part of speech category [13].

2.4.4. Amharic Nouns

Amharic nouns are categorized as simplex and derived. The latter are derived from verb roots, adjectives or other nouns. Affixation is the major process when deriving them from adjectives and other nouns. Nouns can be inflected for Number, Gender, Case and Definiteness. Most of plural nouns can be created by adding a plural marker affix (–oc or –woc) their distribution is determined phonologically) to the singular form. Some nouns from Geez language do not have such plural suffixes. Often, another operation in addition to plural marker affixation occurs. There are two genders in Amharic, masculine and feminine. For things that are not male or female, the gender female tends to be used when the entity is small or adorable; the gender male is used otherwise. Definiteness markers are suffixes that vary depending on the gender of the noun (–u or –wa for feminine and –u or –w for masculine) [15].

2.4.5. Amharic Verb

As a family of Semitic languages, verbs in Amharic have the most complex morphology [2]. In addition to the lexical information which carries the main meaning of the word, the morphemes in an Amharic verb provide different grammatical functions. The morphemes attached to the stem may refer subject and object person, negation, number, and gender; tense, aspect, and mood; and many more grammatical properties. They also have different derivational categories such as passive, causative, and reciprocal and polarity. In addition to, Amharic verb stems consist of a root +vowels + template. [13]. The main meaning of the word resides on the root which is a sequence of consonants/radicals. These consonants are merged with the vowel sequence in a predefined pattern (template) to construct the stem. The nature of the combining the three elements of the stem and its semantic is presented in Table 1

Aspect	Root	Vowel	Template	Stem	Example	Gloss
Perfective	gdl	e,e	CVCVC	gedel(ገደለ)	gedel-e	he kill
Imperfective	gdl	e	CVCC	gedl(ግደል)	y-gedl-al	he will kill
Jussive	gdl	e	CCVC	gdel(ግድል)	y-gdel	Let him kill
Gerundive	gdl	e	CVCC	gedl(ግደል)	gedl-o	He, having killed

Table 1 Template structure of Amharic verb

This non-concatenative (discontinuity) process makes Amharic verb morphology more complex than languages. In addition, affixes in Amharic also have a role. Affixes could prefixed, suffixed, and in-fixed and circumfixed as part of the stem. The morphological richness of the language can be extended by the sequence of morphemes that could be attached to the stem [9].

2.4.6. Verb Compositions

A verb, as a word, is a meaningful string which might be constructed from one or more smaller components. We refer these smaller parts as components of a word. A constituent is a part that acts as a single unit in a structure of a word. Each of this parts contribute in modifving the semantics or the function of the word. Apart from this role, these constituents could be related with each other to provide additional grammatical information. These relationships could take many forms changing the final word created in the process. Some components could act independently while some others might require coexistence with other components. Some components, when they come together, might also result in an orthographic change in either of the components which are interacting. For Amharic verbs, the components could be affixed morphemes, Stems, roots, vowels, and the stem template of the intercalation. In the following sections we explain these components [9].

2.4.7. Affix

An affix is a type of morpheme attached to a stem of a word to form a new word. They are classified as derivational and inflectional. Derivational affixes changes part of speech of a word while inflectional affixes change grammatical feature within a similar POS. Affixation is a way of creating different words by adding the morphemes at the beginning, the middle or the end. Prefix

and suffixes are common type of affixation. For example, by combining the English word like with a morpheme ly, we can create a new word likely. Morphologically rich languages also had additional kind of affixes, circumfix and infixes. Such classification of morphemes takes the position of the added morphemes relative to the stem.

2.4.8. Prefix

Prefixes are class of morphemes appearing before the stem to form new words or to show grammatical features. For example, in English word unlike, we have a prefix un. Prefix could be classified based on their similarity, position and impact on the grammatical category or features of the word. For Amharic there are six prefix slots with a number of morphemes going into each. The slots are cluster of three Prepositional/Conjugational morphemes, Relative, Polarity/Negation, and Subject agreement affixes. Just like suffixes, morphemes belonging to the same slot could represent the same grammatical feature under different categories [13].

2.4.9. Suffix

Suffixes are morphemes attached to a stem (positioned after) to form another word or express some grammatical function. For example, in English word likely, we have a suffix ly. These affixes come after the stem and are classified based on their similarity and effect on the grammatical functions of the stem. For Amharic there are six suffix slots with a number of suffix morphemes belonging to each slot.

2.4.10. Circumfix

Circumfixes are different from the other class of affix. They may be classified as prefix or suffix. Their co-occurring nature with each other in either side of the stem requires a different classification to emphasize their inter dependency. "One example affix of such type is the polarity morpheme al-m. The morpheme mis a polarity marker of a negative main clause. If the verb is not the main clause then the suffix mis not shown to mark the negative polarity. But, the suffix -m-, if it is marked, always occurs with the negative marker prefix-al-in perfective verbs as in al-te-seber-ku-m (I am not broken)" [13]. In this example, the verb is passive and if it is not in the main clause, the pre-fixal can be used alone to form verbs like al-te-seber-ku(I have not been broken)[9][13].

2.4.11. Infix

Infixes are not common in most languages and are properties of few languages. They resides inside of the stem of a word. Most Semitic language has a common infix for verbs. The stem-internal structure of Amharic verbs that requires the vowel sequences to take some of the positions within the root can also be considered as a form of in-fixation. Here, the vowels are inserted inside the consonantal roots in one or more positions. [13][9].

2.5. Stem and Template

In Semitic languages like Amharic roots, vowel sequence and template are crucial component of a verb. The stem in such languages is a composition of the consonantal root and the vowels that form the specific templates. For example, the word seber can be decomposed as: sbr+ee+CVCVC using the consonantal ROOT + Vowel + TEMPLATE structure. The vowels in different positions are used to describe the root-template structure of a stem in Semitic languages. These are referred as transfix. In the above example, the vowel -e- has been inserted twice in the root sbr creating the stem seber. These are called stem-internal structure of Amharic verbs [13]. The stem-internal structure has a significant role. It consists the root, which has the main meaning of the word. In addition, it contains the template which dictates the verb type and partly its grammatical feature. It also entails the number and positions of the vowels inserted in the root. For example, the pattern sbr + e + CCVC is the form for the jussive. But the verb would be imperfective if the template is CVCC[13].

2.6. Approaches to Morphological Analysis

Morphological Analysis has two major approach: Rule based and Machine learning approach [1]. This section describes these paradigms.

2.6.1. Rule based Approach

The rule-based approach follows the explicit theory of morphology formulated by an expert. This approach enables sophisticated linguistic theories such as generative phonology into computational morphology processes. Because of the solid knowledge from this linguistic theory, systems developed using rule-based approaches are often efficient and produce better quality outputs [16]. There are different rule-based methods used to develop morphological analyzer for different languages and some of these are, paradigm based and finite state automata. In paradigm

based method for a particular language, each word category like nouns, verbs, adjectives, adverbs and postpositions is classified into certain types of paradigms. Within computational morphology, a very significant advance came with the demonstration that phonological rules could be implemented as finite state transducers (FSTs) and that the rule ordering could be dispensed with using FSTs that relate the surface and lexical levels directly, so called "two level" morphology (TLM) to lexical output) to one that performs generation [13].

2.6.2. Finite State Technology

Finite-state technology (FST) denotes the use of finite-state devices, such as automata and transducers, in natural language processing. Since the early works which demonstrated the applicability of this technology to linguistic representation. FST is considered adequate for describing the phonological and morphological processes of the world's languages [3].

2.6.3. Finite State Machines

A finite-state machine (FSM) is an abstract machine that implements a regular language. Regular languages can be described formally in a concise notation, through regular expressions. A finite-state machine is a network consisting of states indicating one start state and one or more final states. Transitions between states are possible only if the required input is recognized. A path is a sequence of transition over arcs to a particular state. In computational morphology, a path is a set of alphabets equivalent to a word in natural language. So, it can be said that the technology that utilizes the finite-state network in the processing of creating an application is said to be a finite state technology. But, the finite state automata only accept word and checks if the word is valid words that are found in the language. It does not give or produces an output or generate.

In Finite-state transducers, the analysis of words in a network has simply yielded one of two responses, either accepts, indicating that the word is in the language of the network, or a reject, indicating that the word is not in the language. While this can be valuable, as for instance in spell-checking, finite-state networks are capable of storing and returning much more interesting information. Within computational morphology, a very significant advance came with the demonstration that phonological rules could be implemented as finite state transducers and that the rule ordering could be dispensed with using FSTs that relate the surface and lexical levels directly. Finite-state techniques are probably the most common approach employed by automatic morphology systems, as their simplicity and outstanding efficiency are better. FSAs can be used

to recognize particular patterns, but don't, by themselves, allow for any analysis of word forms. Hence for morphology, we use finite state transducers (FSTs) which allow the surface structure to be mapped into the list of morphemes. FSTs are useful for both analysis and generation, since the mapping is bidirectional.

2.6.4. Two Level of Morphological Analyzer

The two-level morphology approach to morphological analysis is a language independent general formalism for analysis and generation of word-forms [17]. This approach has only two levels, the lexical level and the surface level, hence the name Two-Level Morphology. This model has an added advantage of being bi-directional, both analysis and generation could be done using the same system with. These were not possible with the earlier approaches (uni-directional). Two-level morphology depends heavily on finite state methods, which are well known.

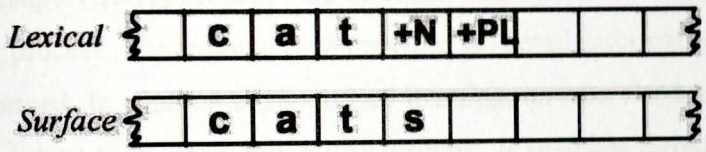


Figure 1 Example illustration of two level morphology

The two level approaches have already successfully been used to develop a comprehensive morphological analyzer for Amharic, Afan Oromo and Tigrign languages [9].

2.6.5. Machine Learning Approach

Machine learning is the use of example data or past experience to program computers for optimizing a specific performance criterion [18]. Machine learning is a suitable approach in cases where we cannot directly write a computer program to solve a specific problem, but need example data or experience. A classic example where learning is necessary is in case where human expertise is hard to find, or when humans are unable to explain their expertise [18]. For some tasks that algorithm might not exist —for example, to identify weather an email is a spam or not. In such case we know the input: an email document that in the simplest case is a sequence of characters. We alsc know the output should be: a yes/no output indicating whether the message is spam or not. We do not know how to transform the input to the output. What can be considered spam changes in time and from individual to individual. What we lack in knowledge, we make up for in data. We can easily compile thousands of example messages some of which we know to be spam

and what we want is to "learn" what constitutes spam from them. In other words, we would like the computer (machine) to extract automatically the algorithm for this task. There are many applications for which we do not have an algorithm but do have example data [18]. Machine learning uses the theory of statistics in develop mathematical Models, because the core task is making generalization from a sample. Machine learning approaches have two major paradigms: Unsupervised and Supervised. While the supervised approach learns from examples, supervised by the given target output, unsupervised approaches use raw data to identify pattern without explicit constraints on how to identify features.

2.6.6. Unsupervised morphology learning

In unsupervised learning, we use raw data for training task. Unsupervised learning approach is idea choice due to the fact that it can be applied to any language with sufficiently large dataset in electronic form. Unsupervised approaches are characterized by the form of the example they take during the learning process. The training data provided no detail information about the output or the pattern to be learned. In contrast to supervised learning, unsupervised machine learning does not demand target output to be provided during training. It searches through the data to extract pattern from the raw text. Unsupervised approaches are characterized by their demand of massive corpora to be able to generalize. Few occurrences of a specific pattern may not be sufficient for generalization. The huge demand of raw data comes from this basic principle [13]. Unsupervised learning had been applied for morphology learning on different languages some of which include:

Linguistica

It is a statistical tool developed based on linguistics to explore unsupervised learning of natural language, with major focus on morphology (word-structure). It explores morpheme-combinations words, without relying on the internal knowledge of the language. Morpheme segmentation is the initial task of this process. The program relies on statistics to locate the morpheme boundaries in the surface words. With a similar approach it identifies the stem and the suffix as well. This initial step of uses morpheme boundaries detection algorithm developed by [19]. The main strength of Linguistica is in identification of morpheme boundaries.

[16] conducted an experiment on Amharic morphology, the result have showed that Linguistica relies on a huge collection of training words with overlapping pattern for a better result. This contradicts the facts that Amharic is under resourced and characterized by word sparsity. A

performance improvement was reported for the same language by [16] based on some enhancement done on Linguistica algorithm. Due to the fact that the initial design for Linguistica was based on English language, the algorithm gives less weight to word candidate at the prefix side of a word. This assumption fails for languages characterized by a prefix morpheme, like Amharic. The other limitation of Linguistica is it cannot handle orthographic changes and stem-internal analysis which is the fundamental feature of Semitic morphology.

Morfessor

The primary goal in the development of Morfessor is to develop a language-independent unsupervised learning algorithms that can discover morpheme units from raw text material [9]. The first attempt focus on the task of segmentation of words in unsupervised manner where the segmented morpheme are taken to act as the label of the morpheme.

The second phase of the task in 2010 challenge with a similar theme of the first one, tries to provide a possibility for semi-supervised learning using a gold standard morpheme. The general principle behind the Morfessor is to discover as compact a representation of the data as possible. Sub-strings occurring frequently enough in several different word forms are proposed as morphs and the words are then represented as a combination as these morphemes. For example, from a corpus of English words, Morfessor builds a list of the form [hand, hand+s,left+hand+ed, hand+ful]. A balance is created between compactness of the morpheme lexicon versus the compactness of the representation of the corpus. The lexicon is a composed of all distinct morphemes (for our previous example, the lexicon includes: [hand, s, left, ed, ful] together with some stored properties of these morphemes. The representation of the corpus can be seen as a sequence of pointers to entries in the morpheme lexicon; e.g. the word 'left-handed' is represented as three pointers to morphemes in the lexicon [9]. Some of the research work conducted for Amharic and other languages based on unsupervised approach is discussed as follows.

Goldsmith [6] Explore unsupervised learning of morphology based on minimum description length (MDL) to learn the underlying morph segmentation. He develops heuristics to construct a probabilistic morphological grammar that best describe analysis prepared by a linguist. MDL is used to select the heuristics.

Spiegler et al [20] Explore an unsupervised learning based on probabilistic generating model. The approach works with an assumption that the segmentation boundary as a hidden state. And it tries

to discover this hidden state through the character transition probability at the segmentation position. The model was tested on English, Turkish, Finish and Arabic.

Bayu [16] Explore unsupervised learning technique to develop a morphological analyzer for Amharic based on Linguistica2001 (the work of [11]). This approach requires a large collection of words that have similarity in patterns. Amharic is morphological complex which makes the formation of words scarcer.

2.6.7. Supervised Morphology Learning

Supervised machine learning relies on different form of examples for the learning process. The training data should include information about what output is expected. Supervised learning in general involve an input X, an output, Y, and the task is to learn the mapping from the input to the output. The approach in machine learning is that we assume a model defined up to a set of parameters:

$$y=g(x/(\theta))$$

Where $g(\bullet)$ is the model and θ are its parameters. Y is a number in regression and is a class code in the case of classification problem. $g(\bullet)$ is the regression function or in classification, it is the discriminant function separating the instances of different classes. The goal of the machine learning program is to optimizes the parameters, such that the approximation error is minimized, that is, our estimates are as close as possible to the correct values given in the training set.[18]

Some literature shows that a closely related class of supervision called partially supervised exists. Such class of supervised learning is different from an ordinary supervised learning due to the fact that the information provided for training is not in a similar form as the final output expected from the final model [13].

Supervised machine learning approach have been applied for morphology learning, a recent work on Amharic has been conducted based on incremental learning approach using inductive logic programming (ILP)[2] and memory-based learning [1]. Inductive Logic Programming (ILP) learns from example and background knowledge in a relational representation. The work on Amharic affix segmentation reveals how an incremental learning technique could be applied to learn detailed segmentation using simple and complex examples with a multiple stage of learning. Memory Based Learning (MBL) uses a stored example in a memory to classify new data instance

by comparing the similarity between feature of an existing example data and the news data instance. They Explored Memory Based Learning for Amharic morphological analysis on nouns and verbs. Abate et al [3] explore Memory Based Learning for Amharic morphological analysis on Amharic nouns and verbs. The approach uses a stored example in a memory to classify new data instance by comparing the similarity between feature of an existing example and news data instance. Since we used deep neural network based architecture, on the next section we discussed in detail supervised machine learning approaches based on artificial neural network architecture.

2.6.8. Artificial Neural Networks (ANNs)

ANNs are family of supervised learning techniques originally developed as mathematical models of biological brain [18]. ANN is composed of small processing unit linked to each other with weighted connection. The network is activated by providing an input to some or all of the nodes, and this activation then spreads throughout the network along the weighted connections. The electrical activity of biological neurons typically follows a series of sharp 'spikes', and the activation of an ANN node was originally intended to model the average firing rate of these spikes [21].

While a typical machine learning approach can be characterized as learning to make predictions based on observations, a multi layered neural network (deep learning) approaches work by learning to not only predict but also to correctly represent the data, such that it is suitable for prediction.

With a large set of input and output mapping, deep learning approaches work by feeding the data into a network that produces successive transformations of the input data until a final transformation predicts the output. The transformations process produced by the network are learned from the given input-output mappings, such that each transformation makes it easier to relate the data to the desired label[21].While the human designer is in charge of designing the network architecture and training regime, providing the network with a proper set of input-output examples, and encoding the input data in a suitable way, a lot of the effort of learning the correct representation is performed automatically by the network, supported by the network's architecture [21].

There are different variations of ANN with different properties [21]. One fundamental classification is based on how nodes are connected to each other: cyclic and acyclic. In the case of cyclic ANN, also known as recurrent neural networks, nodes form a cycle with nodes in one layer

connected to nodes in previous and next layer. ANNs without cycles are referred to as feedforward neural networks (FNN). The most widely used form of FNN is the multilayer perceptron (MLP). Since the output of an MLP depends only on the current input, and not on any past or future inputs, MLPs are more suitable for pattern classification.

2.6.9. Multilayer Perceptron (MLP)

Feed-forward networks, in particular multi-layer perceptron's (MLPs), allow to work with fixed sized inputs, or with variable length inputs in which we can disregard the order of the elements. When feeding the network with a set of input components, it learns to combine them in a meaningful way. MLPs can be used whenever a linear model was previously used [21]. The units in a multilayer perceptron are arranged in layers, with connections feeding forward from one layer to the next. Input patterns are presented to the input layer, then propagated through the hidden layers to the output layer. This process is known as the forward pass of the network. An MLP with a particular set of weight values defines a function from input to output vectors. By altering the weights, a single MLP is capable of instantiating many different functions. Indeed, it has been discussed by [22] that an MLP with a single hidden layer containing a sufficient number of nonlinear units can approximate any continuous function on a compact input domain to arbitrary precision. For this reason, MLPs are said to be universal function approximates.

2.6.10. Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are class of artificial neural network architecture inspired by the cyclical connectivity of neurons in the brain| uses iterative function loops to store information [21]. RNN have different properties that make them suitable for language processing problem such as speech and handwriting recognition and part-of-speech tagging. Recurrent neural networks (RNNs) are specialized models for data which are sequential. The network components take an input a sequence of characters, and produce a fixed size vector that summarizes that sequence. The summarizing a sequence implies different things for different tasks [21].

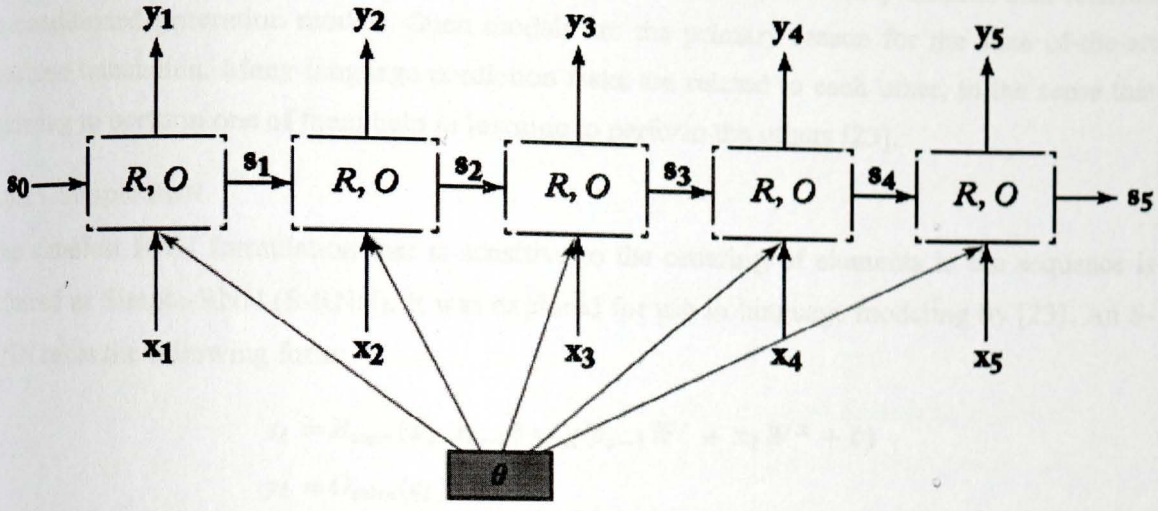


Figure 2 Graphical representation of an RNN

Recurrent networks are rarely used as standalone component, and their power is in being trainable components that can be fed into other network components, and trained to work in tandem with them. For example, the output of a recurrent network can be fed into a feed-forward network that try to predict some value. Mainly the recurrent network is used as an input-transformer that is trained to produce informative representations for the feed-forward network that operate on top of it. [21]

Recurrent networks are very impressive models for sequences, and are the most exciting offer of neural networks for language processing. They allow to bypass the markov assumption that was dominant in NLP for decades, and designing models that can contextualize on entire sentences, while taking word order into account when it is needed, and not suffering much from statistical estimation problems stemming from data sparsity. This is capability leads to impressive gains in language-modeling, the task of predicting the probability of the next word in a sequence (or, equivalently, the probability of a sequence), which is a cornerstone of many NLP applications. [21].

On a different account, Recursive networks extend recurrent networks from sequences to trees. Many of the problems in natural language are structured, requiring the production of complex output structures such as sequences or trees, and neural network models can accommodate that need as well, either by adapting known structured-prediction algorithms for linear models, or by

using novel architectures such as sequence-to-sequence (encoder-decoder) models, also referred as conditioned-generation models. Such models are the primary reason for the state of-the-art machine translation. Many language prediction tasks are related to each other, in the sense that knowing to perform one of them help in learning to perform the others [23].

2.6.11. Simple RNN

The simplest RNN formulation that is sensitive to the ordering of elements in the sequence is referred as Simple-RNN (S-RNN). It was explored for use in language modeling by [23]. An S-RNN takes the following form:

$$s_i = R_{\text{SRNN}}(x_i, s_{i-1}) = g(s_{i-1}W^s + x_i W^x + b)$$
$$y_i = O_{\text{SRNN}}(s_i) = s_i$$
$$s_i, y_i \in \mathbb{R}^{d_s},\ x_i \in \mathbb{R}^{d_x},\ W^x \in \mathbb{R}^{d_x \times d_s},\ W^s \in \mathbb{R}^{d_s \times d_s},\ b \in \mathbb{R}^{d_s}.$$

That is, the states are S_{i-1} and the input x_i are each linearly transformed, the results are added (With a bias term) and then passed through a nonlinear activation function (commonly tanh). The output at position i is the same as the hidden state in that position.

The S-RNN is only slightly more complex than the CBOW (bug of words), with the major difference being the nonlinear activation function. However, this difference is a central feature, as adding the linear transformation followed by the nonlinearity makes the network sensitive to the order of the inputs. A S-RNN has been reported to show a strong result for sequence tagging as well as language modeling [24].

2.6.12. Gated Architectures

The S-RNN is difficult to train more effectively because of the vanishing gradients problem [23]. The error signals (gradients) in consecutive steps in the sequence diminish quickly in the back-propagation process, and do not reach earlier input signals, making it hard for the S-RNN to capture long-range dependencies. Gating-based architectures, such as the LSTM [25] and the GRU [26] are designed to overcome this deficiency [21].

Consider the RNN as a general purpose computing device, with the states Si representing a memory. Each application of the function reads in an input xi+1, reads in the current memory. i, operates on them in some way, and writes the result into memory, resulting in a new memory states

S_i+1. In such a way, a typical problem with the S-RNN architecture is that the memory access is not controlled. At each step of the computation, the entire memory state is read, and the entire memory state is written. A controllable gating mechanism is the basis of the LSTM and the GRU architectures, to be defined next: at each time step, differentiable gating mechanisms decide which parts of the inputs is written to memory, and which parts of memory is overwritten. [21]

2.6.13. GRU

The LSTM architecture is very effective in overcoming the vanishing gradient problem, but it is quite complicated. The complexity of the architecture makes it hard to analyze, and also computationally expensive to work with. The gated recurrent unit (GRU) was recently introduced by [27] as an alternative to the LSTM. It was subsequently shown to perform comparably to the LSTM on several (non-textual) datasets. Like the LSTM, the GRU is based on a gating mechanism, but with fewer gates and no separate memory component is used.

$$
\begin{aligned}
s_j = R_{\mathrm{GRU}}(s_{j-1}, x_j) &= (1-z) \odot s_{j-1} + z \odot \tilde{s_j} \\
z &= \sigma(x_j W^{xz} + s_{j-1} W^{sz}) \\
r &= \sigma(x_j W^{xr} + s_{j-1} W^{sr}) \\
\tilde{s_j} &= \tanh(x_j W^{xs} + (r \odot s_{j-1}) W^{sg}) \\
y_j = O_{\mathrm{GRU}}(s_j) &= s_j \\
s_j, \tilde{s_j} \in \mathbb{R}^{d_s},\ x_i \in \mathbb{R}^{d_x},\ z, r \in \mathbb{R}^{d_s},\ W^{xo} &\in \mathbb{R}^{d_x \times d_s},\ W^{so} \in \mathbb{R}^{d_s \times d_s}.
\end{aligned}
$$

One gate (r) is used to control access to the previous state s_{j-1} and compute a proposed update sj. The updated state s_j (which also serves as the output y_j) is then determined based on an interpolation of the previous states s_{j-1} and the proposal s_j, where the proportions of the interpolation are controlled using the gate. The GRU was shown to be effective in language modeling and machine translation.

2.6.14. LSTM

The Long Short-Term Memory (LSTM) architecture [25] was designed to solve the vanishing gradients problem, and is the first architecture to incorporate a gating mechanism. The LSTM architecture explicitly splits the state vector Si into two halves, where one half is treated as

"memory cells" and the other is working memory. The memory cells are designed to preserve the memory, and also the error gradients, across time, and are controlled through differentiable gating components — smooth mathematical functions that simulate logical gates. At each input state, a gate is used to decide how much of the new input should be written to the memory cell, and how much of the current content of the memory cell should be forgotten mathematically, the LSTM architecture is defined as [21]:

$$
\begin{aligned}
s_j = R_{\text{LSTM}}(s_{j-1}, x_j) &= [c_j; h_j] \\
c_j &= f \odot c_{j-1} + i \odot z \\
h_j &= o \odot \tanh(c_j) \\
i &= \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
f &= \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
o &= \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \\
z &= \tanh(x_j W^{xz} + h_{j-1} W^{hz})
\end{aligned}
$$

$$
y_j = O_{\text{LSTM}}(s_j) = h_j
$$

$$
s_j \in \mathbb{R}^{2 \cdot d_h},\ x_i \in \mathbb{R}^{d_x},\ c_j, h_j, i, f, o, z \in \mathbb{R}^{d_h},\ W^{xo} \in \mathbb{R}^{d_x \times d_h},\ W^{ho} \in \mathbb{R}^{d_h \times d_h}.
$$

The state at time j is composed of two vectors, c_j and h_j, where c_j is the memory component and h_j is the hidden state component. There are three gates, i, f, and o, controlling for input, forget, and output. The gate values are computed based on linear combinations of the current input xj and the previous state hj, passed through a sigmoid activation function. An update candidate z is computed as a linear combination of x_j and h_j, passed through a tanh activation function. The memory cj is then updated: the forget gate controls how much of the previous memory to keep (f, and the input gate controls how much of the proposed update to keep. Finally, the value of h_j (which is also the output y_j) is determined based on the content of the memory cj, passed through a tanh nonlinearity and controlled by the output gate. The gating mechanisms allow for gradients related to the memory part cj to stay high across very long time ranges. LSTMs are currently the most successful type of RNN architecture, and they are responsible for many state-of-the-art sequence modeling results. The alternative architecture for LSTM-RNN is the GRU.

2.6.15. Conditioned Generation (Encoder-Decoder)

The power of the RNN is clear when moving to a conditioned generation framework. The generation framework generates the next token tj+1 based on the previously generated tokens $t_{1:j}$

$$\hat{t}_{j+1} \sim p(t_{j+1} = k \mid \hat{t}_{1:j}).$$

This is modeled in the RNN framework as

$$p(t_{j+1} = k \mid \hat{t}_{1:j}) = f(\text{RNN}(\hat{t}_{1:j}))$$
$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}),$$

Where f is a parameterized function that maps the RNN state to a distribution over words, for example, f (x) = softmax(xW+b). In the conditioned generation (encoder-decoder) framework, the next token is generated based on the previously generated tokens, and an additional conditioning context c.

At each stage of the generation process the context vector c is concatenated to the input t_j, and the concatenation is fed into the RNN, resulting in the next prediction. The following diagram illustrates the architecture.

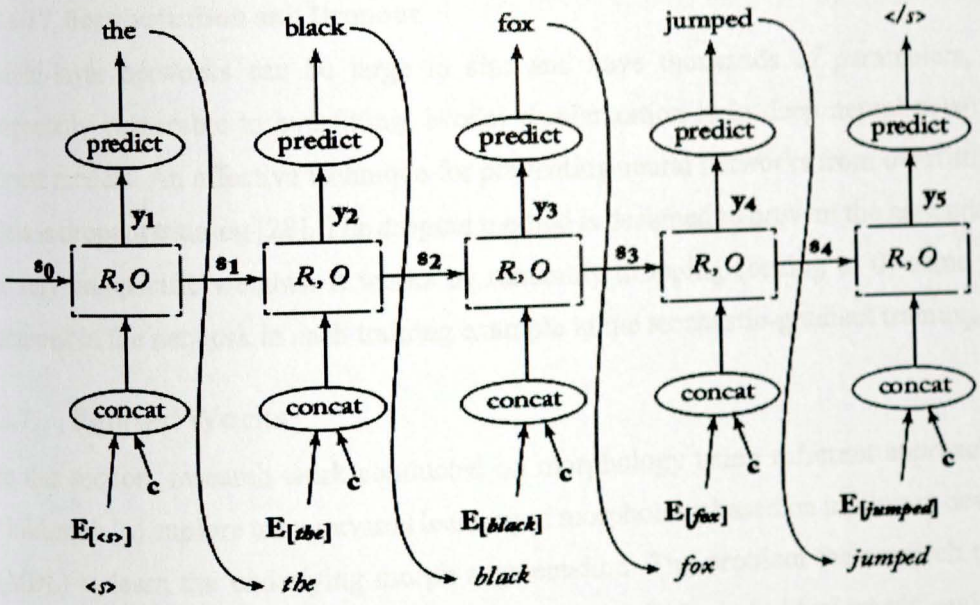


Figure 3 Conditioned RNN generator

We can put any kind of information in place of the context c. For example, if we have a large corpus of news items categorized into different topics, we can treat the topic as a conditioning context. Our language model is able to generate texts conditioned on the topic. If we are interested

in movie reviews, we can condition the generation on the genre of the movie, the rating of the review, and perhaps the geographic region of the author.

2.6.16. Sequence to Sequence model

The context c can have different forms. In the previous section, we described some fixed-length. Examples of conditioning contexts. Another popular approach takes c to be itself a sequence, most commonly a piece of text. This gives rise to the sequence to sequence conditioned generation framework, also known as the encoder-decoder framework [26].

In sequence to sequence conditioned generation, we have a source sequence $x_{1:n}$ (for example a sentence in French) and we are interested in generating a target output sequence t1:m (for example the translation of the sentence into English). This works by encoding the source sentence $x_{1:n}$ into a vector using an encoder function c= ENC(.x1:n), commonly an RNN: c=RNN enc.(x1:n). A conditioned generator RNN (decoder) is then used to generate the desired output $t_{1:m}$.

2.6.17. Regularization and Dropout

Multi-layer networks can be large in size and have thousands of parameters, making them especially vulnerable to overfitting. Model regularization is in deep neural networks as it is in linear models. An effective technique for preventing neural networks from overfitting the training data is dropout training [28]. The dropout method is designed to prevent the network from learning to rely on specific weights. It works by randomly dropping (setting to 0) some percent of the neurons in the network in each training example in the stochastic-gradient training.

2.7. Related Works

In this section, research work conducted on morphology using different approach is discussed. Goldsmith [6] explore unsupervised learning of morphology based on minimum description length (MDL) to learn the underlying morph segmentation. The problem the research tries to address involves the discovery of the correct morphological split for individual words, and the building of accurate categories of stems based different suffixes. The primary goal of this work is to produce analysis of word as closely related with what a human morphologist produces without relying on any kind of linguistic rule. The researcher uses minimum description length (MDL) analysis to model the unsupervised morphological segmentation. MDL relies on the analysis of a training data

that is optimal to provide a compact representation of the data and a compact means of extracting that compression from the training data. It requires a quantitative account whose parameters aligns the original corpus. The MDL is primarily used to select the heuristics. The main contribution of this work is the concept of signatures, which aid both in quantifying the MDL account and in incrementally building a satisfactory morphological grammar.

Spiegler et al. [20] explore an unsupervised learning based on probabilistic generating model(PGM). The researcher tries to address unsupervised morpheme analysis for words contained in a training data using a generic algorithm without any annotations. Probabilistic generative model (PGM) is used to explain the process of data generation based on an observation of variables X and hidden variables Y with the objective of forming a conditional probability distribution P (Y|X). In morphological segmentation, we attempt to split words into morphemes. Thus, the observed variables correspond to the surface words and the hidden variables to their segmentation. Knowing the parameters of the model we can find the best segmentation of a given word. The approach works with an assumption that the segmentation boundary as a hidden state. And it tries to discover this hidden states through the character transition probability at the segmentation position. The result showed that the model focuses on decomposing surface words and it predicts well the number of morpheme that exist between two words. The other strength is the algorithm works whether a language uses only suffixes or also prefixes. In addition, the algorithm can be used to different level of supervision. In a (semi-) supervised or unsupervised it can be utilized to average over a limited segmented set by learning its parameters from it. The other advantage is that instead of relying on a morpheme dictionary and morphosyntactic rules which are likely to be incomplete, it applies the statistics of a small training set to a larger test data set. The model was tested on Arabic (vowelized and non-vowelized), English, Finnish, German and Turkish. It achieved competitive results in the Morpho Challenge 2009.

Amsalu et al [15] explore finite state transducer for Amharic verbs, nouns and adjectives. The researcher tries to address the problem of Amharic morphological segmentation using Xerox Finite State Tools. Morphological segmentation using finite state transducers (FSTs) bases on the assumption that the mapping of surface words to their segmentation has a regular relation. This means the underlying surface forms include a regular set, the surface forms constitute a regular set, and there is a many–to–many regular relations between these sets. The main challenge here is applying FST to non–concatenative languages like Amharic due to the nonlinear word formation

with intercalation of consonantal roots with vocalic patterns. The work was reported to show an accuracy of 94% (verbs), 85 %(nouns) and 88 %(adjectives).

Gasser [3] explore a finite state model for three Ethiopian languages, Amharic, Tigrigna and Oromigna. The researcher attempt to address the lack of morphological analyzer for the three languages. The approach used relies on Amtrup [30] work. In Amtrup's approach, each of the path in a transducer may have weight of some sort with a feature structure. The feature structure are lists of grammatical pairs that contains feature and value. As the path in an FST are traversed, the feature structure is accumulated by unifying the current set with the value that appears on the path along the arc through the transducer. The feature-value pairs in these FST represent a memory for the path that has been traversed. A path whose structure fails to unify with the current set of feature-value pairs cannot be traversed. The basic idea behind Amtrup's approach is that the transitions in a traducers are weighted with grammatical feature pair. For evaluation words were selected randomly from different word list crawled from websites. A total of 200 Tigrinya verbs, 200 Amharic verbs, and 200 Amharic nouns and adjectives had been chosen. The output of the system was evaluated by a human reader familiar with the languages to be evaluated. An output is considered correct only if it matches all legal combinations of roots and grammatical structure for a given surface word form and included no incorrect roots or structures. The program scores 96% accuracy on Tigrinya verbs, 99% accuracy on Amharic verbs, and 9 errors on the 95.5% accuracy on Amharic nouns and adjectives.

Wondwossen et al, [2] explore Inductive Logic Programming (ILP) for Incremental Learning of Affix Segmentation for Amharic verb morphology. The researchers try to address the challenge of a detailed segmentation affix into smallest morpheme units. Incremental Learning of Affix Segmentation in generals learns from example and background knowledge in a relational representation. Incremental learning rely on the use of simple structures for learning at early stages and continues to a more complex structures using information of previous structures as a an input for the next operation. This learning process can be implemented using Inductive Logic Programming (ILP) which is a machine learning techniques that learns rules from positive and negative examples. The researchers used 140 training examples containing words, the stem and with morphological features, the system is able to learn and extract 6 prefixes and 25 suffixes. The work on Amharic affix segmentation reveals how an incremental learning technique could be applied to learn detailed segmentation using simple and complex examples with a multiple stage

of learning. The work is reported to show a segmentation accuracy with 0.94 Precision and 0.97 Recall. One critical drawback of approaches based in incremental learning for morphology is the need for an intelligent teacher to curate a complete positive and negative example.

Mesfin et al [1] explore Memory Based Learning for Amharic morphological analysis on Amharic nouns and verbs. Memory Based Learning (MBL) uses a stored example in a memory to classify new data instance by comparing the similarity between feature of an existing example data and the news data instance. Memory-based approaches have the advantages of probabilistic approach and knowledge based approaches. It is a classification algorithm and uses analogy to do the classifications. The researchers framed segmentation problem as a classification problem which retrieves the grammatical functions and properties of morphologically inflected words on nouns and verbs. The approach uses a stored example in a memory to classify new data instance by comparing the similarity between feature of an existing example and new data instance. The researchers used a corpus that contains 1022 words (841 verbs and 181 nouns). 1356 and 6719 instances are extracted from nouns and adjectives respectively. A total of 26 different class labels occur within these instances. The work is evaluated using 10-fold cross-validation with IB1 and IGtree algorithms with an accuracy of 93.6% and 82.3% respectively.

Wang et al [7] explored Window LSTM Neural Networks for learning morphological boundary directly from raw input words and are subsequently able to predict morphological boundaries on Hebrew and Arabic language. The approach relies on Long Short Term Memory (LSTM) units with windows of characters to capture contextual information. The researchers used 6,192-word example for training for each of the languages. The researcher has applied a standard stochastic gradient descent with mini batches of training instances, and utilize dropout regularization. The work also presents a comparison of three main architectures: regular LSTM, Window-LSTM, multi-window LSTM and bidirectional multi window LSTM.A comparison of the performance of these architecture with other approaches that rely on human-designed features, CRF, Poon and semi supervised Morfessor. The Bidirectional Multi-Window LSTM model outperforms the CRF method with relative improvements of 5.86%, 0.11%, and 2.87% for precision, recall, and F1, respectively. The work achieves good results on a number of languages without relying on linguistic knowledge.

Reddy et al [8] explored deep sequence to sequence (seq2seq) model that takes raw input string as the input and predicts the segmented string for Sanskrit language. The researchers rely on encoder-decoder framework for segmentation problem, and propose a deep seq2seq model using LSTMs for the prediction task. The approach follows the architecture from [31], which was originally applied for neural machine translation. The researchers use the pair of sandhied and unsandhied sentences as input (source) and output (target) sentences, respectively. The model uses a LSTM architecture with 3 layers each at the encoder and decoder with and without attention. The training phase used 4,200 sentences pairs with a batch size of 128 and learning rate was set at 0. 001.For optimization Adam is used and dropout is utilized for regularization. The result shows seq2seq with attention outperforms the current state of the art with a percent increase of 16.29 % in F-Score.

Chen [32] explores deep learning techniques for Chinese word segmentation (CWS) and POS tagging by deep learning. The study takes advantage of large raw data to improve internal representation of Chinese characters, and use these representations to enhance supervised word segmentation. The approach use a type of neural network architecture introduced by [10] for different NLP problems. The network accepts sentence and extracts multiple levels of feature from the raw inputs, with more abstract representation. The experiments were conducted on the Chinese Treebank data sets from Bakeoff-3(45,135 words), which contains a training and a test corpus for supervised word segmentation. The performance was evaluated with the closed test) and are comparable with other models in the literature.

Chen et al [24] explores a gated recursive neural network (GRNN) with a reset and update gates to incorporate the complicated combinations of the context characters for Chinese word segmentation. The researchers use three datasets, PKU, MSRA and CTB6, to evaluate the model with 90% sentences of the training data as training set and the rest 10% sentences as development set for PKU and MSRA datasets. The performance of segmentation significantly increasing by the use of gated recursive architecture, which shows that a better model of context characters has been learned.

Luong et al [33] explores recursive neural networks (RNNs) for building representations for morphologically complex words from their morphemes. This work utilizes recursive neural networks (RNNs) and neural language models (NLMs) in combination to learn better word

representations. The work address the problem common to word based vector representation (word embedding's) by relying on morphemes as a basic unit instead of a word. This has an advantage especially for words that exhibit morphological complexity. By treating each morpheme as a basic unit in the RNNs and the researchers were able to construct a representation for morphologically complex words from their morphemes. By training a neural language model (NLM) and applying the RNN structures for complex words, the work utilizes contextual information to learn morphemic semantics and their compositional properties. In this way the model has the capability of building representations for any new word composed of known morphemes. The model was evaluated using the word similarity task and outperforms its corresponding baseline significantly over the rare word dataset.

In our work we framed morphological segmentation problem as a sequence to sequence problem. By considering the surface word as an input and the morphemes as output, we can consider these task as input transformation problem. Such approach allowed to build upon on a more recent works that achieved an encouraging result on a similar task such as machine translation. The success of RNN based approaches for highly complex sequence-to-sequence problems such as machine translation is a good example. We choose this direction due to the fact that in neural network based approaches we can design our learning algorithm without relying on heavy feature engineering [10]. In our case we trained our model to learn a segmentation from a raw text example without any linguistic annotation. These is a significant advantage for under resourced languages like Amharic. Based on these thesis, we explored the possibility of learning a segmentation model from a raw text, how the size of the training data affects the performance of our model and the architectural parameter that would be effective to learn from a small training data.

CHAPTER THREE

DESIGN OF MODEL

3.1. Overview

Computationally, morphological segmentation can be framed as a sequence-to-sequence problem. The primary goal in such approach is to map a surface form of a word to a set of valid morphemes. The recent success of neural encoder-decoder architecture for highly complex sequence-to-sequence problems such as machine translation is the main inspiration for this work. While the success of the sequence-to-sequence approach in translation task is astonishing, it is important to note that such approach relies on a high resource configuration. For example, the pioneer work based in encoder-decoder architecture [29] requires eight layers of high-dimensional LSTMs, is computationally very expensive, and is hard to train effectively [21]. The literature also showed that these works are strongly concentrated on high resourced and morphological non-complex languages such as English and French with a large amounts of training data and computational resources [26] [30] [29]. Despite the success of these techniques for resource favored languages, there effectiveness on under resourced languages (smaller number of training examples and limited computational settings) and morphological rich language is not explored. Our aim in this research is to narrow these gap, by exploring RNN based architecture to show case the effectiveness of such architecture on a low resource setting. Our model is based on a generic sequence-to-sequence framework proposed by [29]. In this section we summarized the architecture we use for our model build.

3.2. Model Architecture

For our experiment, we rely on recurrent neural networks (RNNs). Recurrent neural networks seemed well suited for the task of morphological segmentation. RNNs are well-known to work well for learning problems where the input data is sequential. Morphological segmentation is class of such problems where both the input and the output are sequential. Sharing weights between hidden units across each time step, RNN architecture is a natural way to model time series data, where each time step of the input depends on those in the past(context). In addition to this, RNNs have the capability to handle variable-length inputs, eliminating the need for padding inputs. In contrast,

Multilayer perceptron's or convolutional neural networks are restricted to fixed-size inputs, fixed-size outputs, and a fixed-number of layers and computational steps. These would make RNN an ideal choice for morphological segmentation problem which are variable in input size both at the input and the output layer.

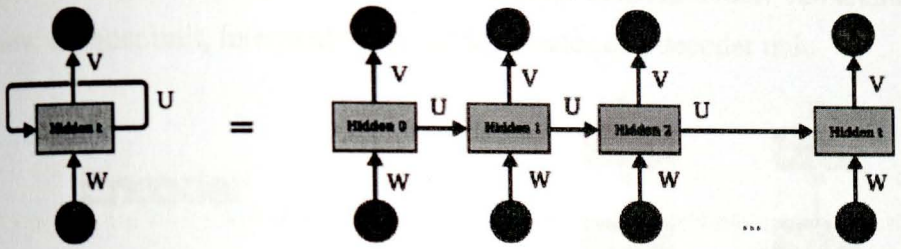


Figure 4 Example of Unrolling a RNN over the input time steps [21]

RNNs consist of an input layer, an output layer, and a recurrent layer, as shown in figure 4. They are comprised of a series of weight matrices and activation functions. The input data represented as X0, X1 etc. are the characters in the surface word and the output are the character in the segmentation. We usually use a vector representation of these characters.

Explicitly, the set of equations that maps a set of inputs, x to predicted outputs $y\hat{}$ is [21]:

$$h1 = (W\,x + bh) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

$$hi = (W\,x + Uhi{-}1 + bh) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$y\hat{}i = \text{softmax}(V\ hi) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Where σ is the logistic function, b are bias vectors, and W, U, V are weight matrices shared across time steps. Over the course of training, the model learns which setting of weight matrices W, U, minimizes an overall loss function.

3.3. Model Design

In the encoder-decoder architecture, an encoder reads a variable length input sequence which is a sequence of vectors ($x=x_1, x_2, \ldots, x_i, \ldots..x_m$) corresponding to a sequence of input symbols $x=x1$, $x_2, \ldots, x_i, \ldots..x_n$) and generates a fixed-dimensional vector representation of the sequence. The key advantage of the approach is the ability to train a single end-to-end model directly on source and target text and the ability to handle variable length input and output sequences of text. The decoder is trained to predict the next output y_t given the encoded input vector e and all the previously

predicted outputs y_1, y_{t-1}. In such architectures, the generation unit generates the next token t_{j+1} based on the previously generated tokens. Given a discrete alphabet Σ (e.g., lists of characters in a surface word), our objective is to map a surface word $W \in \Sigma*$ (e.g., w= unintentional), to a segmentation list $M \in \Omega*$ (e.g., c= un, intent, ion, al). We write the segmented form as $C=M_1 M_2 \ldots M_n$, where each segment $Mi \in \Sigma*$ and n is the number of morphemes. The architecture consists of three parts: encoder unit, intermediate (encoder) vector and decoder unit.

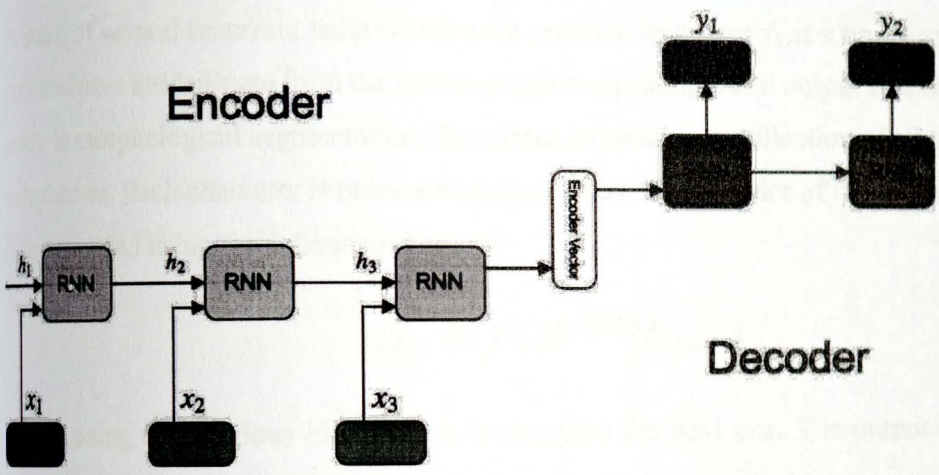


Figure 5 architecture of encoder-decoder

3.3.1. Encoder

A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward. In morphological segmentation, the input sequence is a collection of all characters from the surface word. Each character is represented as x_i where i is the order of that word. The hidden states h_i are computed using the following formula:

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

This formula represents the result of a recurrent neural network. As seen in the formula, we just apply the appropriate weights to the previous hidden state $h_{(t-1)}$ and the input vector x_t.

3.3.2. Encoder Vector

This is the final hidden state produced from the encoder part of the model. It is computed using the formula above. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions. It acts as the initial hidden state of the decoder part of the model.

3.3.3. Decoder

A stack of several recurrent units where each predicts an output y_t at a time step t. Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state. In morphological segmentation, the output sequence is a collection of all characters from the morphemes. Each character is represented as y_i where i is the order of that word. The hidden state h_i is computed using the following formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

We are using the previous hidden state to compute the next one. The output y_t at time step t is computed using the formula:

$$y_t = softmax(W^S h_t)$$

We compute the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which help us determine the final output.

3.4. Data preparation

For this work, we require lists of Amharic surface word along with their corresponding morphemes. We have prepared two data sets. The first data set has 780 words prepared using the verb ሰብር/seber/break/, አለ/ale, ገደለ/gedel/kill, which are selected as a representative examples: exhibit high production and are class of bi-radical and tri-radical. This help us in preparing a representative data set that can capture different inflation at morphology. Based on these, we have prepared 780 by applying grammatical information. To prepare distinct sets of inflected word form, we have used grammatical information based on person (first, second, third), Gender

(feminine, neuter, masculine), number (singular, plural), case (subjective/ Nominative), definiteness (definite, Indefinite), degree (positive, comparative, superlative) and tense (past, present, future). Sample training example are included in Appendix 7.1.

The second word list is composed of words randomly selected from word-list organized by Biniam[3] . We have included the algorithm we used to randomly select the word list on appendix 7.7. The source contain different words crawled from different sources such as news websites, blogs and forums. The second data set contain 3000 words. As presented in appendix 7.1, we have organize the word list in terms of input (surface word) and output (segmented word).In total we have prepared 3780 example words. Before proceeding to the training, we have randomly splitted the data-sets into three training data set data-set-I (520 words), data-set-II (1050 words) and data-set-III (1600 words). The algorithm for splitting the data set is presented in Appendix 7.2.

We have used these sub-data set to learn the effect of size of the training data on the overall performance of the model. For the rest of the experiments we have used data-set-I. As part of pre-processing, we have applied the following tasks on the word list.

Corpus Cleaning: In this process, we have removed a non-letter characters (characters outside of Amharic letters) and symbols that are attached to any of the word.

Normalizing characters: Some words contain characters with the same function. For example, the character 'U' (ha) and 'ሐ' (ha) have similar function in terms of meaning. So to prevent word that have same meaning to repeated in the training set, we have replaced such characters with their equivalent characters.

Conversion to SERA standard (Romanization): We have used SERA system for character mapping to convert words written in Amharic letter (Fidel) to Latin form. This is required due to the fact that the program we used for example preparation, HORNMORPHO, uses SERA standard for processing and for the final output. The algorithm for this task is included in the appendix 7.5.

[3] http://www.cs.ru.nl/~biniam/geez/crawl.php

3.5. Data Preparation

For preparing the training and testing data, we have used HORNMORPHO 2.5. The program has segmentation module which provide a segmentation of the surface word with the function named seg under a module named l3.

In the segmentation output, the stem is surrounded by braces, and prefixes and suffixes are separated by hyphens. The morphemes are described in terms of the grammatical features they represent; these descriptions appear in parentheses, with alternative interpretations for a morpheme separated by the symbol '|'.

Example: l3.seg('am', 'ይፈልጋሉ') display segmentation of the word 'ይፈልጋሉ' as y(sb=3sm|3p)-{flg+1e2_3}(imprf)-al_u(aux,sb=3p) and from this output we extract the morphemes as { flg+1e2_3 ,y, al_u}. We wrote a python program to extract the morpheme from the output. The final example look contain the Romanized form of the word (e.g ይፈልጋሉ – yIfelgalu) and the morpheme list. [Input: yIfelgalu output: y + fel_g + al_u]. The algorithm for this task is presented in the appendix 7.3.

Input	Output	Training Pair
ይፈልጋሉ	y(sb=3sm|3p)-{flg+1e2_3}(imprf)-al_u(aux,sb=3p)	YIfelgalu → y + fel_g + al_u
ብትሰብሪለትም	b(cnj1)-t(sb=2|3sf)-{sbr+1e23}(imprf)-i(sb=2sf)-l_(prep2)-et(ob=3sm)-m(cnj2)	bIn_IsebrIl_et → b 'n sebr l_ et
ቢሰብሩለት	b(cnj1)-y(sb=3sm|3p)-{sbr+1e23}(imprf)-u(sb=2p|3p)-l_(prep2)-et(ob=3sm)	bisebrul_et → b y sebr u l_ et

Table 2 Sample segmentation example

Since our objective of training is to minimize the feature engineering involved, we only rely on morpheme list and no grammatical information or other form of feature is included in the training data. In the training data, we have two pairs of input and output with the surface word (input) and morpheme list (output). The following table shows the input/output pair for the training data.

	Input(Surface-word)	Output(Segmentation)
ይፈልጋሉ	YIfelgalu	y fel_g al_u
ብትሰብሪለትም	bIn_IsebrIl_et	b 'n sebr l_ et
ቢሰብሩለት	bisebrul	b y sebr u l_ et

Table 3 Sample training example

3.6. Tools used

In this section we describe tools we used from data processing to model evaluation.

Python 2.7

As a programming environment we have used python 2.7. We have selected python due to its strong back from a neural network research community and availability of deep learning libraries and resources.

HORNMORPHO 2.5

It is a Python program that analyzes Amharic words into their constituent morphemes and generates words, given a root or stem and a representation of the word's grammatical structure. It is free software. We have used Amharic segmentation module from this program for preparing training and testing data. The program accept Amharic word in Amharic Fidel format (e.g ይፈልጋሉ) and output lists of morphemes with their grammatical information (e.g y(sb=3sm|3p)-{flg+1e2_3 (imprf)-al_u(aux,sb=3p)). As described data preparation section, we used a python program to further process this output to extract the morphemes.

GloVe(Global Vectors for Word Representation)

GloVe is an unsupervised learning algorithm for obtaining vector representations for characters or words. Training is performed on aggregated global word-word or character-character co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. We have used glove to build a character based embedding. We have selected glove over word-embedding's due to an availability of more parameter (e.g: window-size) to build a character embedding's.

Matplotlib

Matplotlib is a Python 2D visualization library which generated figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts. We have used this library along with python to visually observe learning performance change (Increase or decrease over time) during model training and to spot over fitting if occurred.

Numpy

NumPy is a Python library for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. We have used numpy along with tensor flow.

Tensor flow

TensorFlow is an open source library for high performance numerical computation. It support computations both on CPU'S and GPU's. It is used for machine learning applications such as neural networks.

Tensor-board

Tensor-board is a suite of visualization tools called. We used TensorBoard to visualize our TensorFlow graph, plot quantitative metrics about the execution of our graph, and show additional data like images that pass through it.

3.7. Feature Selection

Feature Selection is one of the fundamental concepts in machine learning that impacts the performance of our model. The feature selection decision affect what information we would be relying on for the training purpose. Since out research goal it to minimize feature engineering, we only on the raw text to decide on the features. We have used one major features for all of our experiment: characters in the surface word. These approach keep our solution generic to a different kind of morphological segmentation task irrespective of the language.

Surface Word	Example Features	Example Parameter
ብትሰብሪለትም	ብ,ት,ሰ,ብ,ሪ	Window size 1
ብትሰብሪለትም	ብት,ሰብ,ሪለ,ትም	Window size 2
ብትሰብሪለትም	ብትሰ,ብሪለ	Window size 3

Table 4 Sample feature description

3.8. Representation

Representations are critical to modeling using deep learning architecture. One-hot vectors are high-dimensional and sparse while character embedding's are low-dimensional and dense. When using one-hot vectors as a feature in a classifier, the feature vector grows with the vocabulary size (in our case the number of characters). character embedding's are more computationally efficient in this aspect. The alternative approach to using sparse representation like one-hot representation is to use a denser vector. Which is, each core feature (characters) is embedded into a d dimensional space, and represented as a vector in that space. The dimension d is usually much smaller than the number of features, i.e., each character in a vocabulary of 100 characters (encoded as 100-dimensional one-hot vectors) can be represented as 10 dimensional vector. The embedding's (the vector representation of each core feature) are treated as parameters of the network, and are trained like the other parameters.

The major change in the input when using deeper representation is, then, the move from sparse representations in which each feature is its own dimension, to a dense representation in which each feature is mapped to a vector. For our work, we have trained an embedding and used the output to convert the input in our training data. One advantage of using dense and low-dimensional vectors is computational efficiency: the majority of neural network libraries are slow with high-dimensional vectors. However, the main advantage of the dense representations is in generalization power. In our work, we assumed that embedding based representation might help the overall learning phase by capturing some relationships between the characters. So in our experiment, we have used a representation based on a character embedding's.

3.9. Evaluation Metrics

In evaluating our model, we have considered morphological segmentation as a two-step process, discovering the existence of a morpheme and generating a morpheme that matches the reference morpheme. We have evaluated the first step through evaluating Morpheme-existence, by counting and comparing the morpheme boundary, between the model output and the reference. On the second evaluation, Accuracy, we have compared the exact morpheme match between the model output and the reference.

3.9.1. Morpheme Boundary

In this measurement we have evaluated the model based on number of morphemes (both known and unknown). This is done by comparing the number of morpheme discovered by the model in a surface word with the number of morpheme in the reference. This evaluation help us in analyzing the model performance in terms of morph.

3.9.2. Morpheme Match

We evaluated the accuracy using exact matches on the morpheme level. An output counts as correct if and only if the output of the model exactly matches the reference morphemes, i.e., if all output characters are predicted correctly.

3.10. Learning Algorithm/Training

Encoder-decoder models vary in terms of their exact architectures. A common choice for sequential data is the recurrent neural network (RNN), used by most NLP tasks. RNN is used for both the encoder and decoder side. The RNN models itself differ in terms of: directionality – (unidirectional or bidirectional), depth (single- or multi-layer); and type (Simple RNN, a Long Short-term Memory (LSTM), or a gated recurrent unit (GRU). In our case, we have explored these architectures to find the generalization capability with limited number of examples.

We set the experimental parameters including the neuron type, encoder type (uni-directional and bidirectional) network depth and size, embedding's, and dropout rate for the models using the training data. For the first experiment we used the data sets I, II and III discussed on training data sections with data divided into (80% training, 10% development and 10% testing).

For the rest of the experiment we have used dataset I. Our initial setting includes: unidirectional LSTM unit with two layer and 30 network size, 0.2 dropout, softmax activation and greedy

decoding for the inference. On top of these setting, we have conducted four experiment: to learn effect training data size with three data sets, sub word size (number of character feed into the encoder layer), the effect of the different RNN model and encoder type.

3.11. Training Detail

For training our model we tried to minimize the computational requirement of the architecture. We used RNN with 4 layers, with 2 layer at the encoder layer and 2 layer at the decoder layer. We used a softmax activation at each output. In summary we have used the following setting for to perform the training:

- ✓ We have experimented with different number of hidden units starting from 256 (as suggested in the work of [33]) and with 128, 64 and 32. We settled with 32 because the training time is smaller without a loss in accuracy.
- ✓ We have experimented with different number hidden units starting from 1000 units on each layer as suggested in [33]. We significantly reduce the number of hidden units after observing a long training time. We settle with 32 hidden units which took our training under an hour to train.
- ✓ At the encoder side we have used two type of encoder types, unidirectional and bidirectional. We did this to understand the effect of directionality on the learning performance.
- ✓ We initialized all of our parameters with fixed seeds too ensure consistent model output.
- ✓ For optimization we have used two type of optimizer, Adam and SGD, stochastic gradient descent. For Adam, we have used a default of 0.01 for learning rate.
- ✓ After running a few training for 10000 steps we set the training 2000 steps. We have not observed any increase in the accuracy after these step.
- ✓ We have initialized the weight uniformly with a value of a default value of 0.1 and for regularization we used dropout with a value of 0.2.
- ✓ Fcr training the embedding's we have used glove by varying the context window size

3.12. Training procedure

(1) Randomly select N

(2) Read setting file

(3) Split data into training, development and testing.

(4) Train a character embedding using the embedding settings.

(5) Build a vocabulary for the input and the output characters

(6) Transform the source (input) and output using the trained embedding's.

(7) Fit the source/target pair to learn a model

(8) Save model parameter as a check-point

(9) Plot the learning stage as epoch(X) to accuracy(Y) graph, epoch(X) to cost(Y).

(10) Use the saved model and apply it to generate the

(1) The source file contains surface word to segmentation pairs.

(2) Select 720 pairs of examples, algorithm presented in App 7.7.

(3) Setting file include split ratio and embedding settings.

(4) We used 80:10:10 for training, validation and testing. Algorithm presented in App 7.2.

(5) The embedding setting contain the window size to use for a specific experiment. Embedding algorithm presented in App 7.8.

(6) The vocabulary is built to map both the input and the output to a vector value from the embedding's. The algorithm presented in App 7.9.

(7) Both the input and output should be transformed to a vector using the embedding's and the vocabulary mappings before.

(8) Initialize the parameters and pass the transformed source/target pairs to the encoder and decoder function.

(9) Use the parameter with the highest validation accuracy as a checkpoint to a model file.

(10) Use the graph to fine tune the parameter and to check whether overfitting occurred or not.

(11)The output morpheme lists will be used for external evaluation.

(12)Using the evaluation algorithm on App 7.6 for evaluation. This will be considered as a final accuracy measure for the model accuracy.

CHAPTER FOUR

EXPERIMENTAL AND RESULT

In this chapter we describe the experiments we conducted, the objective of each experiment and the result and discussion on each of the experiment. In total, we have conducted four experiments, we try to answer we have designed each of the experiment to address each of the research question. Experiment I address the first research question and experiment II, III and IV address the second research question.

In the first experiment we have explored how the sizes of the training data affect the overall performance of the model. The main object of this experiment is to learn how our architecture learns across different training data.

In the second experiment we have explored the effect of the context window by conducting experiment with different value of the context window. Our objective in this experiment is to find the optimal context window size by conducting experiment with varying context size.

On the third experiment we have explored different RNN units with the objective of understanding which units provide an optimal learning. In this experiment we have explored SRNN, LSTM and GRU units.

On the last experiment we have explored the effect of directionality of the encoder unit in the general learning performance. We have included a graphical illustration of the learning process. Each of the illustration shows the training accuracy (y-axis) vs training step (x-axis). The table shows the summary of the accuracy result. In the morpheme match accuracy, we have evaluated our model by comparing the output of our models' morpheme with gold standard. The evaluation is performed on a new test set.

4.1 Experimental Parameters

We have used the experimental parameter described on [29] as a baseline parameter. However to show the learning capability of the architecture we have significantly reduced the baseline parameter as shown in the table 4.1 below. The objective of reducing the baseline is to be able to train our model on a CPU computer with an acceptable training time.

Parameter	Parameter - Value (Baseline)	Parameter - Value (New - Parameter)
Number of layer	8	4
Network size	400	32
Number of encoder layer	2	2
Number of decoder layer	2	2
Attention – Mechanism	Yes	No
Batch size	128	32
Number of Iteration	12000	2000

Table 5Experimental parameters

Model build tool

For building our segmentation model we used Google Seq2Seq, a general-purpose encoder-decoder framework based on Tensorflow. The tool has the following sets of parameter to fine tune a model during training.

Parameter name	Description	Possible value
num_units	Network size	32,64,128
num_layers	Network depth	2,4,8
num_encoder_layers	Encoder depth	2,4,8
encoder_type	Encoder unit to be used	uni, bi, gnmt

Table 5 Selected Experimental parameters

Experiment I: Effect of training size

The purpose of this experiment is to understand how the training data composition affects the learning performance. These training sets have different character, word and morpheme composition. We have included a sample graphical illustration for data set I. Figure 6 shows the training accuracy (y-axis) vs training step (x-axis). The graph illustrates the increase in the accuracy from epoch 1k to 1.5k and saturates afterwards. The accuracy change is also symmetrical with the cost function graph which shows a proportional decrease. Figure 7 shows the cost (y-axis) with respect to the training epoch (x-axis).

Table 6 shows the summary of the accuracy result. In the morpheme match accuracy, we have evaluated our model by comparing the output of our models' morpheme with gold standard. The evaluation is performed on a new test set. The result shows the sensitivity of our learning algorithm for variability in the training data. The performance drop for morpheme match correlated to addition of vocabulary (new characters) as we increase the training data, from data set I → II → III. These results verify what is known in the literature as rare word problem (rare character vocabulary) in sequence to sequence learning [31].

Data-Set	Accuracy (Morpheme-Match)	Accuracy (Morpheme-Boundary)
Data-Set I	73.8%	98.62%
Data-Set II	71.3%	99.26%
Data-Set III	71.1%	99.73%

Table 6 Effect of training data size

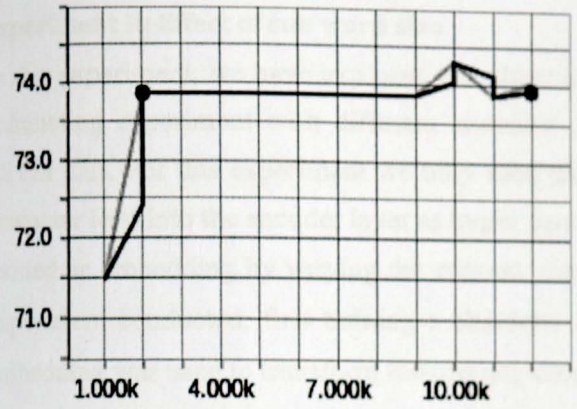


Figure 6 Graphical illustration of (Data-Set I) training (accuracy vs steps)

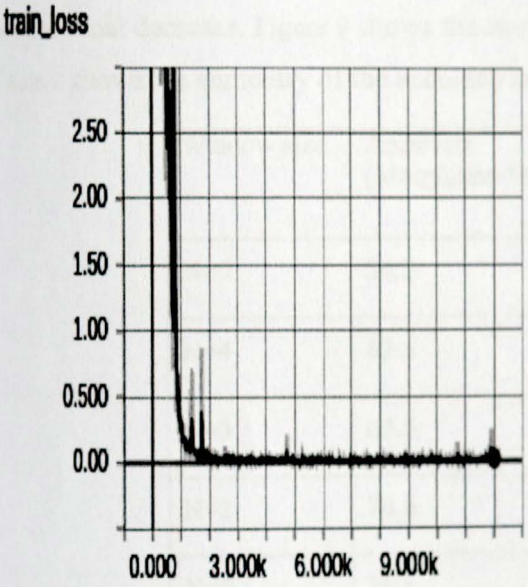


Figure 7 loss (loss vs steps)

In this first experiment showed the model performance with different training data size. We used a window size of 1, a unidirectional encoder and LSTM unit. The result showed a 2.5% decrease when we go from dataset I to dataset II but slight increase is observed in the morpheme boundary detection. Since morpheme boundary shows how many morphemes are available in a surface word it entails that the model has learned the structure of the word. This implies that our model has learned the word structure better as we increase the training words but suffers in finding the exact morpheme. Finding the exact morpheme is directly related to the vocabulary and with increased example data the probability of getting unknown vocabulary gets higher.

Experiment II: Effect of sub word size

In this experiment, we have explored the effect of the size of character feed into the encoder by conducting experiment with different character length. We used a unidirectional encoder and LSTM unit. For this experiment we only used data set I and we have considered the number of character feed into the encoder layer as hyper parameter. For each of the experiment, we have first trained an embedding by varying the context window. For instance, the value N =2 represent an experiment conducted, first training a character embedding by setting the window size 2. This embedding was used to transform the training data before feeding into the encoder layer. We have included a sample graphical illustration. Figure 8 shows the training accuracy (y-axis) vs training step (x-axis). The accuracy change is symmetrical with the cost function graph which shows a proportional decrease. Figure 9 shows the cost (y-axis) with respect to the training epoch (x-axis). Table 7 shows the summary of the accuracy result across varying window size.

Window size	Accuracy (Morpheme-Match)	Accuracy (Morpheme-Boundary)
N=5	24.3	94.75
N=4	62.5	97.19
N=3	63.5	98.63
N=2	70.6	99.16
N=1	**71.3**	**99.16**

Table 7 Effect of context window

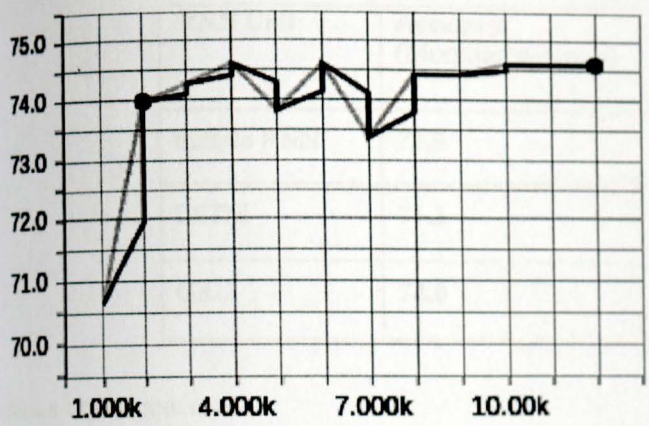


Figure 8 Graphical illustration of w-1) training (accuracy vs steps)

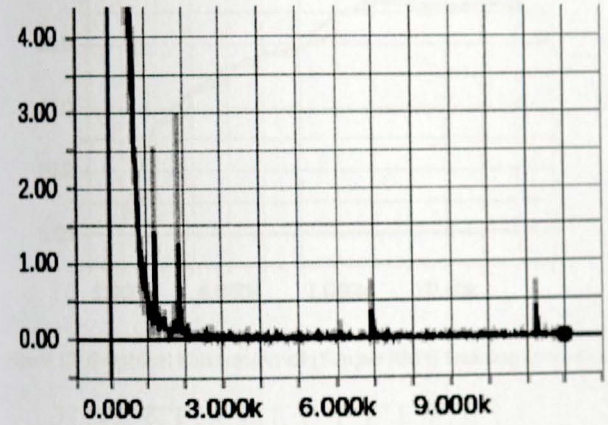


Figure 9 loss (loss vs steps)

In this experiment, we have explored the effect of different window-size. As suggested in a probabilistic sequence models used by [10], this approach shows to be effective to model dependencies in label sequences. Our result showed that using lower number of window size (one and two) have a clear advantage over a longer character size. A smaller character size increases the generalization capability of our network by learning a representation at a character level.

Experiment III: RNN units

In this third experiment, we have explored different types of RNN unit: Simple RNN, a Long Short-term Memory (LSTM), and a gated recurrent unit (GRU). We used a window size of 1 and a unidirectional encoder. Our objective in this experiment is to explore the learning capability of different RNN units. We have included a sample graphical illustration. Figure 10 shows the training accuracy (y-axis) vs training step (x-axis). The accuracy change is symmetrical with the cost function graph which shows a proportional decrease.

Table 8 shows the summary of the accuracy between the different RNN units.

RNN Unit	Accuracy (Morpheme-Match)	Accuracy (Morpheme-Boundary)
Simple RNN	73.8	98.62
LSTM	71.3	98.16
GRU	**74.0**	**98.67**

Table 8 Effect of RNN units

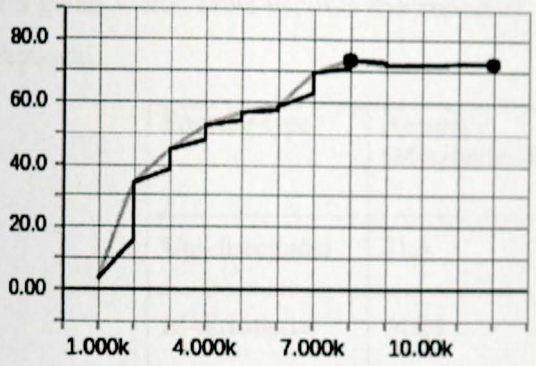


Figure 10 Graphical illustration of (Simple RNN) training (accuracy vs steps)

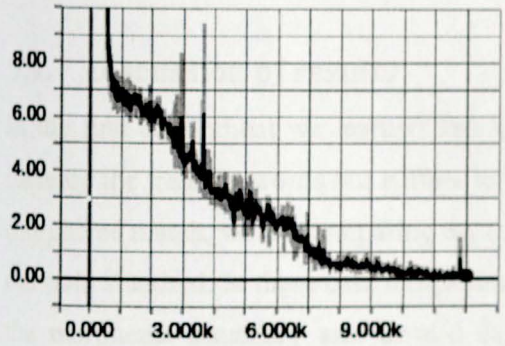


Figure 11 Graphical illustration of (Simple RNN) loss (loss vs steps)

In this experiment demonstrate the effect of different RNN units on the segmentation and morpheme boundary accuracy. GRU based RNN showed higher accuracy both in the segmentation output and almost on all of the morpheme boundary test data set. These results confirms the

conclusion of [32], which showed the performance advantage of using GRU with large dataset. Our result showed these to be an advantage as well for a small training data.

Experiment IV: Effect of encoder type

In this experiment, we have explored the effect of directionality of the RNN model. We used a window size of 1 and LSTM unit. The choice of directionality depends on the nature of sequence to sequence problem. Sequence that have a strong context relationship have more advantage from applying directionality in training. When accepting input character sequences, bidirectional model has a good capability to capture more information by making both a forward and a backward pass for a given word. Table 9shows the summary of the accuracy between Uni-directional and Bi-directional.

Encoder-type	Accuracy (Morpheme-Match)	Accuracy (Morpheme-Boundary)
Uni-directional	71.3	98.16
Bi-directional	74.20	99.86

Table 9 Effect of encoder type

3.1. Discussion of results

In the first experiment we learned that our model has learned the word structure better as we increase the training words but suffers in finding the exact morpheme. In the evaluating of exact morpheme match, we are comparing the morpheme generated by the model with the morpheme of our gold standard. In these case morpheme which is tagged as 'unk' would be counted as valid in the morpheme boundary and invalid in the morpheme match evaluation. Finding the exact morpheme is directly related to the available vocabulary.

The second experiment showed that using lower number of window size less than three have a clear advantage over a longer character size. A smaller character size increases the generalization capability of our network by learning a representation at a character level. This is also a good indicator that window size is a relevant feature choice to model both morphological boundary discovery and morphological segmentation.

The third confirms the conclusion of [32], which showed the advantage of using GRU with large a small training data as the case for large training data.

In the Fourth experiment we learned that modeling morphological segmentation would have advantage if we use bi-directional unit over uni-directional unit. The implication is that sequences of character in morphological have strong sequence dependency. When accepting input character sequences, bidirectional model capture more information by making both a forward and a backward pass for a given word. Analysis of segmentation error is presented in table 9.

No	Undiscovered Morphemes	Count
1	b_	62
2	e	55
3	u	51
4	h	46
5	b	34

Table 10 list of undiscovered morphemes sorted by their frequency

In table 10 we have presented a sorted list of morphemes that are missed (undiscovered) by our segmentation model. The sorting is based on the frequency of occurrence of the morphemes appearing in the total undiscovered morphemes list. As seen in the table, the morphemes that is not discovered by the model have some common characteristics. The first characteristics of these morphemes occur in relatively limited number in the training example. This have a negative effect in the decoding process since the network did not see these example in higher frequency, it would be excluded from the top candidate list during decoding, so it labels them as unknown. The second characteristics is the overlap of these morphemes with character of the surface word. All of these morphemes are single character morphemes and this has a negative impact in the trained network due to the fact that a character occurs in the surface word and in the multi character morphemes as well.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1. Conclusion

In this research we have explored the possibility building a morphological segmentation model without relying of a grammatical information. Higher-level NLP task such as machine translation, speech processing, text summarization and many more rely on morphological analysis. Currently most research works done on morphological are based on techniques that require high linguistic annotation or rely on rule-based techniques that require detailed enumeration of the rule of the language to be crafted manually. These techniques require a high-quality data in terms of capturing the rules that exist in the language and it also require a significant quantity of training data for better generalization. The low resource state and morphological complexity of the language demand techniques that can provide better learning with relatively small number of example and be able to capture the complexity of language. In this paper, we explored RNN based sequence-to-sequence model that provides an encouraging performance in learning complex segmentation with small number of example and with no linguistic annotation. We have framed the problem of morphological segmentation as transformation task by considering the surface word as an input and the segmentation as a transformation process to produce a list of segmented morpheme. Our experiment showed that our model can learn a complex segmentation with no linguistic annotation and with limited number of examples.

The main contribution of our work is as follows: Our model uses no linguistic annotation and learns morpheme structures from surface word and segmentation pairs. In comparison to rule based approach, this is a significant advantage to reduce the annotation effort required. Scaling up out approach is also cheaper than rule based approaches since in our case we only required surface word and morpheme pair. We have explored variable training data, different neuron units, decoders and representation approaches to learn and provide an insight on the best possible experimental setting for learning morphological segmentation in resource constrained setting. And, in all of the experiment we conducted we have significantly reduced the heavy parameters including number of layers, hidden units, attention and number of training steps) of the parameter suggested by the original work. These have enabled us to do all of the experiment on a CPU under

a 30 minutes of training time. In this work we have showed that encoder-decoder architecture is a promising learning algorithm to build morpheme segmentation and boundary detection model with little supervision and smaller training data requirement. In terms of the limitation, we have observed a high dependency on morpheme vocabulary. The accuracy of the model in finding a valid morpheme is closely tied to the size of the character vocabulary. Further exploration is required to understand how the character vocabulary. In summary, our research work has been limited to the following activities.

- We only explored Amharic verb POS with three consonants as these are the most common form of verbs in Amharic verb.
- We approached the segmentation problem only, which is one component of morphological analysis. Our model provides lists of morpheme of a surface word and does not process a grammatical information.
- Building our model also requires examples of pairs of surface word and a respective morpheme.

5.2. Recommendation

In this research work we have explored how recurrent neural network can be used to learn the task of Amharic morphological segmentation. To do that we relied on recurrent neural network based architecture.

- In our work we only focused on Amharic verb types with three consonants, other verb types such as two consonant and four consonant verb types can be handled as a post processing task or can be include in the training data set to build an extended segmentation model.
- Our work also requires a supervision, an example of surface word and example segmentation, to train a segmentation model. Future work should explore learning of morphological segmentation using unsupervised approach.
- In our work, we only focused on morphological segmentation of Amharic verb. Future works should include other Amharic part of speech class such as noun and adjective.
- Morphological segmentation is the one component of morphological analysis task. Future work should explore the possibility of extending these morphological segmentation task to do morphological analysis.

- Last but not least, future work should explore the possibility of extending our work to other Semitic languages to further investigate how language independent feature would be captured in models.

References

[1] M. Abate and Y. Assabie, "Development of Amharic Morphological Analyzer Using Memory-Based Learning," *Int. Conf. Nat. Lang. Process.*, pp. 1–13, 2014.

[2] W. Mulugeta, M. Gasser, and B. Yimam, "Incremental Learning of Affix Segmentation," *Coling-2012*, vol. 1, no. December 2012, pp. 1901–1914, 2016.

[3] M. Gasser, "HornMorpho : a system for morphological processing of Amharic, Oromo, and Tigrinya," no. May, pp. 2–5, 2011.

[4] M. Y. Tachbelie, S. T. Abate, and W. Menzel, "Morpheme-based and factored language modeling for amharic speech recognition," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6562 LNAI, pp. 82–93, 2011.

[5] Y. Lee, "Morphological Analysis for Statistical Machine Translation," *IBM T. J. Watson Res. Cent.*

[6] J. Goldsmith, "Unsupervised Learning of the Morphology of a Natural Language" *Comput. Linguist.*, vol. 27, no. 2, pp. 153–198, 2001.

[7] L. Wang, Z. Cao, Y. Xia, and G. de Melo, "Morphological Segmentation with Window LSTM Neural Networks," *Proc. 30th {AAAI} Conf. Artif. Intell. (AAAI 2016)*, pp. 2842–2848, 2016.

[8] V. Reddy, A. Krishna, V. D. Sharma, P. Gupta, M. R. Vineeth, and P. Goyal, "Building a Word Segmenter for Sanskrit Overnight," 2017.

[9] M. Gasser, "Semitic Morphological Analysis and Generation Using Finite State Transducers with Feature Structures," *Proc. 12th Conf. Eur. Chapter ACL (EACL 2009)*, no. April, pp. 309–317, 2009.

[10] J. Weston and M. Karlen, "Natural Language Processing (Almost) from Scratch," vol. 12, pp. 2493–2537, 2011.

[11] L. Wang, Z. Cao, Y. Xia, and G. De Melo, "Morphological Segmentation with Window LSTM Neural Networks," pp. 2842–2848.

[12] O. Hellwig, "Improving the Morphological Analysis of Classical Sanskrit," no. 2, pp. 142–151, 2016.

[13] W. M. Gewe, "Machine Learning of Complex Morphology : the Case of Amharic Verbs," no. July, 2015.

[14] C. Stab and I. Gurevych, "A Comparative Study of Minimally Supervised Morphological Segmentation," *Assoc. Comput. Linguist.*, vol. 42, no. October 2015, 2016.

[15] S. Amsalu and D. Gibbon, "Finite state morphology of {A}mharic," *5th Recent Adv. Nat. Lang. Process.*, no. May, pp. 47–51, 2005.

[16] T. BAYU, "Automatic Morphological Analyzer for Amharic an Experiment Employing Unsupervised Learning and Autosegmental Analysis Approaches," 2002.

[17] S. Wintner, "Natural Language Processing of Semitic Languages," pp. 43–67, 2014.

[18] E. Alpaydin, *Introduction to Machine Learning*. 2016.

[19] Z. S. Harris, "From Morpheme to Utterance Author (s): Zellig S . Harris Published by : Linguistic Society of America Stable URL : http://www.jstor.org/stable/410205," *Linguist. Soc. Am.*, vol. 22, no. 3, pp. 161–183, 1946.

[20] S. Spiegler, B. Golénia, and P. Flach, "Promodes: A probabilistic generative model for word decomposition," *CEUR Workshop Proc.*, vol. 1175, no. May 2014, 2009.

[21] S. Cohen, *Neural Network Methods for Natural Language Processing Synthesis Lectures on Human Language Technologies Bayesian Analysis in Natural Language Processing*.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[23] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," URL: arXiv:1211.5063v2 vol. 2, 2014.

[24] X. Chen, X. Qiu, C. Zhu, and X. Huang, "Gated Recursive Neural Network for Chinese Word Segmentation," pp. 1744–1753, 2015.

[25] S. Hochreiter and J. Urgen Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," 2014.

[27] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," pp. 1–9, 2014.

[28] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors arXiv : 1207 . 0580v1 [cs . NE] 3 Jul 2012," pp. 1–18.

[29] I. Sutskever, O. Vinyals, and Q. V Le, "Sequence to sequence learning with neural networks," *Adv. Neural Inf. Process. Syst.*, pp. 3104–3112, 2014.

[30] D. Britz, A. Goldie, M. Luong, and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," *Google Brain*, vol. 2, 2017.

[31] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, "Addressing the Rare Word Problem in Neural Machine Translation URL:http://arxiv.org/abs/1410.8206v4 ".

[32] X. Chen, X. Qiu, C. Zhu, P. Liu, and X. Huang, "Long Short-Term Memory Neural Networks for Chinese Word Segmentation," no. September, pp. 1197–1206, 2015.

Appendix

7.1. Training Data (Sample)

slleseb_erul_etlm	sle seb_er u l_ et m
keseb_erac_lhub_et	ke seb_er ac_hu b_ et
slleseb_erec_lm	sle seb_er ec_ m
ly_aseb_erul_etlm	'y_e as seb_er u l_ et m
ly_aseb_eru	'y_e as seb_er u
teseb_ernlm	te seb_er n m
lndlsebir	lnd ' sebr
bit_lsebrul_et	b t sebr u l_ et
leseb_erub_etlm	le seb_er u b_ et m
iy_eseb_erxlm	'y_e seb_er x m
slleseb_erkum	sle seb_er hu m
leseb_erkul_etlm	le seb_er hu l_ et m
slleseb_erklb_etlm	sle seb_er h b_ et m
slleseb_erx	sle seb_er x
bisebru	b y sebr u
slleseb_erul_etlm	sle seb_er u l_ et m

7.2. Data splitter

```
import codecs
import random
'''
A programm to split data into training(80%) dev(10%) and test(10%)/ Python 2.7 / Wende.T ''
src = 'gedele_full'
prefix = '4'

train_src_file = 'train' + prefix + '.src'
train_trg_file = 'train' + prefix + '.trg'
dev_src_file = 'dev' + prefix + '.src'
dev_trg_file = 'dev' + prefix + '.trg'
test_src_file = 'test' + prefix + '.src'
test_trg_file = 'test'  + prefix + '.trg'

def get_list(src_data):...

def write_to_file(file_,cont):...

def split(src_list,size):...

def toktrgize_to_char(word):...

def separate_data(cont_list):
    src='';trg='';repeate_check =[];multi_form =0
    for line in cont_list:
        src_word = line.split()[0]
        if src_word not in repeate_check:
            repeate_check.apptrgd(src_word)
            src += toktrgize_to_char(src_word) + '\n'
            trg += ' '.join(line.split()[1:]) + '\n'
        else:
            multi_form +=1
```

7.3. HornMorpho based segmentation

```
import codecs
from io import StringIO
import sys
import l3
'''
    HORNMORPH 2.5  based Amharic Word Segmentor / Python 3.5 / Wende.T
'''
old_stdout = sys.stdout
result = StringIO()

source_file = 'word_list_gedel'
out_file = source_file + '_seg'
#####################################################################
def read_data():...
def write_data(data,i):...
def segment(word_list):
    size =len(word_list)
    i=1006;tracker=i;k=100 #Iteration parameters
    while(tracker>1):
        sys.stdout = result #Store the output to a variable
        for word in word_list[i-k:i]:
            tracker =tracker-1
            try:
                l3.seg('am',word)
            except IndexError:
                pass
        sys.stdout = old_stdout
        result_string1 = result.getvalue()
        result_string =result_string1.split('\n')
        write_data(result_string,i)
        i= i-k
    ######################
```

7.4. HornMorpho pre-processor

```
'''
A programm to process output of hornomorpho  / Python 2.7 / Wende.T '''
import codecs
from io import StringIO
import sys
import string
import re
import l3
############################################################
# mapfile2 = codecs.open(r'syll', 'r', 'utf-8')
src_file = 'word_list_gedel_seg'
out_file = src_file + '_latin'
old_stdout = sys.stdout
result = StringIO()
############################################################
def read_data():
def write_data(cont):
def decode( maps ):
def mapper(mapstripped):
def stem(cont):
def to_horn(cont):
def word_to_horn(surf_word):
    sys.stdout = result #Store the output to a variable
    l3.phon('am', surf_word)
    sys.stdout = old_stdout
    result_string1 = result.getvalue()
    result_string =result_string1.split('\n')
    ind = len(result_string) -2
    return result_string[ind]
def extract(dic):
if __name__ == '__main__':
```

7.5. SERA standard convertor

```
  1  '''
  2  A programm to process output of hornomorpho  / Python 2.7 / Wende.T '''
  3  import codecs
  4  from io import StringIO
  5  import sys
  6  import string
  7  import re
  8  import l3
  9  ###########################################################
 10  # mapfile2 = codecs.open(r'syll', 'r', 'utf-8')
 11  src_file = 'word_list_gedel_seg'
 12  out_file = src_file + '_latin'
 13  old_stdout = sys.stdout
 14  result = StringIO()
 15  ###########################################################
 16  def read_data():
 25  def write_data(cont):
 31  def decode( maps ):
 34  def mapper(mapstripped):
 41  def stem(cont):
 81  def to_horn(cont):
 98  def word_to_horn(surf_word):
 99      sys.stdout = result #Store the output to a variable
100      l3.phon('am', surf_word)
101      sys.stdout = old_stdout
102      result_string1 = result.getvalue()
103      result_string =result_string1.split('\n')
104      ind = len(result_string) -2
105      return result_string[ind]
106  def extract(dic):
139  if __name__ == '__main__':
```

7.6. Model Evaluation

```
import codecs
''' A programm to evaluate model performance/ Python 2.7 / Wende.T '''
input_file = '../infer_inp2.vi'
model_output_file = '../nmt_model1_bi/output_infer2'
ref_file = 'train1.src'
error_log_file = 'error_log2'
def compute_score(data_gold,model_output):
    score = {}
    input_words = (codecs.open(input_file,'r','utf-8').read();.splitlines()
    error_log = ''
    for i in range(len(data_gold.splitlines())):
        model_seg = model_output.splitlines()[i]
        ref_seg = data_gold.splitlines()[i]
        print 'Computing accuracy @ line %d ' % i
        print ' Reference segmentation %s ' % ref_seg
        print ' Model segmentation %s ' % model_seg
        print ' Reference morph count %d ' % len(ref_seg.split())
        print ' Model morph count %d ' % len(model_seg.split(;)
        correct_morph = 0.0
        morph_length = len(ref_seg.split())
        eval_label = []
        undiscoverd_morphemes = ''
        for morph in ref_seg.split():
            if morph in model_seg.split():
                correct_morph += 1
                eval_label.append(True)
            else:
                undiscoverd_morphemes += morph + ' '
                eval_label.append(False)
        acc = float(correct_morph/morph_length)*100
        if acc < 100:
        print ' Overlap %s ' % eval_label
```

7.7. Random Data Selector

```
1  import codecs
2  from random import randint
3  '''
4  A programm to randomly select training data / Python 2.7 / Wende.T'''
5  data = codecs.open('words_720K_morph_p1_251K','r','utf-8').read()
6  sample_size = 3000
7
8  def write_data(file_name,cont):...
11
12 def generate_list(lines_list):
13     cont = ''
14     counter = 0
15     while counter < sample_size:
16         rand_index = randint(0,len(lines_list)-1)
17         cont +=  lines_list[rand_index] + '\n'
18         counter += 1
19     write_data('rand_example',cont)
20
21 def to_word(lines_list):...
25
26 def to_lines():...
30
31
32 if __name__ == '__main__':
33     lines_list = to_lines()
34     word_list = to_word(lines_list)
35     print len(lines_list)
36     print len(word_list)
37     generate_list(lines_list)
38
```

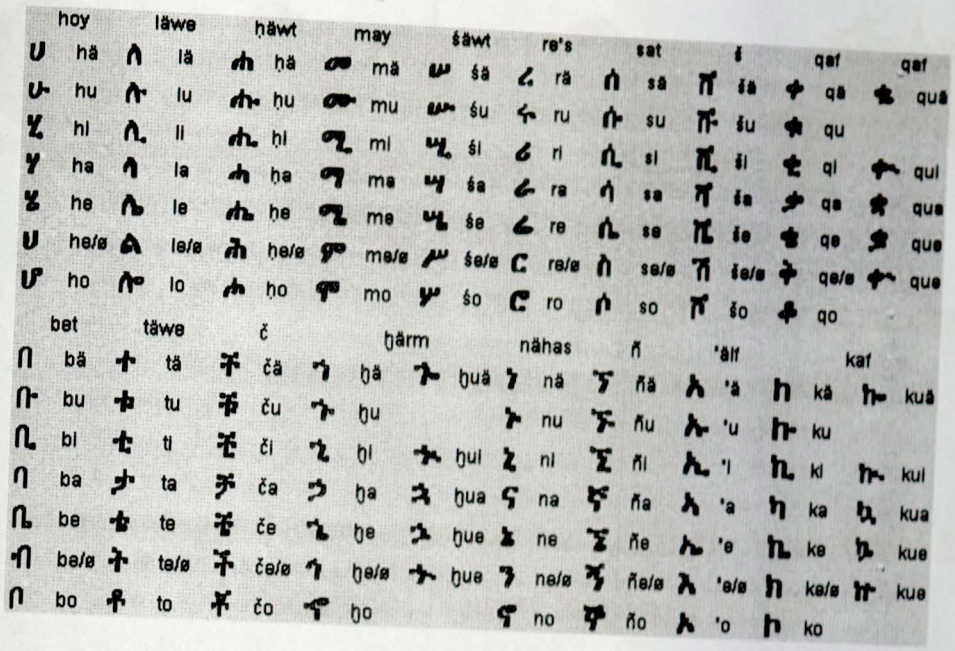
7.8. Character embedding's (Glove)

```
# Character embedings
import itertools
from gensim.models.word2vec import [redacted]
from glove import Corpus, Glove
''' A programm to build a character embedings using Glove/ Python 2.7 / Wende.T '''

data = 'tgt_dataset'
out_file = 'embed_w1.tgt'

sentences = list(itertools.islice(Text8Corpus(data),None))
corpus = Corpus()
corpus.fit(sentences, window=1)
glove = Glove(no_components=100, learning_rate=0.05)
glove.fit(corpus.matrix, epochs=30, no_threads=4, verbose=True)
glove.add_dictionary(corpus.dictionary)
glove.save(out_file,binary=False)
# model.wv.save_word2vec_format('model.txt', binary=False)
# glove.most_similar('man')

```

7.9. Vocabulary Extractor

```
import codecs
''' A programm to generate a vocabulary from a text/ Python 2.7 / Wende.T '''
src = 'train3.src'
dst = 'vocab3.src'
cont = '<unk>' + '\n' + '<s>' + '\n' + '</s>' + '\n'
data = codecs.open(src,'r','utf-8').read()
for line in data.splitlines():
    for voca in line.split():
        if voca not in cont:
            cont += voca + '\n'
fo = codecs.open(dst,'w','utf-8')
fo.write(cont)
fo.close()
```

7.10. Amharic Characters



7.11. Appendix: Url for data and tools

Source code for hornmorpho 2.5

https://github.com/adamsamson/HornMorpho2.5

Source code for Glove embedding's

https://nlp.stanford.edu/projects/glove/

Source Google encoder-decoder framework

https://github.com/google/seq2seq

Installation package for tensorflow 1.11

https://github.com/tensorflow/tensorflow

Installation package for matplotlib 3.0

https://github.com/matplotlib/matplotlib