Addis Ababa University

College of Natural Sciences

SEMANTIC ROLE LABELING FOR AMHARIC TEXT USING DEEP LEARNING

Bemnet Meresa Hailu

A Thesis Submitted to the Department of Computer Science in

Partial Fulfillment of the Degree of masters of Science in

Computer Science

Addis Ababa, Ethiopia

17August 2021

Addis Ababa University

College of Natural Sciences

Bemnet Meresa Hailu

Advisor: Dr. Yaregal Assabie

This is to certify that the thesis prepared by Bemnet Meresa Hailu, titled: *Semantic Role Labeling for Amharic Text Using Deep Learning* and submitted in partial fulfillment of the requirements for the Degree of Masters of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name Signature Date .

Advisor: Dr. Yaregal Assabie .

Examiner: _____

Examiner: _____

ABSTRACT

Semantic Role Labeling (SRL), the task of automatically finding the semantic roles of each argument corresponding to each predicate in a sentence, is one of the essential problems in the research field of Natural Language Processing (NLP). SRL is a shallow semantic analysis task, and an important intermediate step for many NLP applications, such as Question Answering, Machine Translation, Information Extraction and Text Summarization. Feature-based approaches to SRL are based on parsing output, often using lexical resources, and require heavy feature engineering. Errors encountered in the parsing output can also propagate to the SRL output. Neural-based SRL systems, in contrast, can learn the intermediate representations from raw text, bypassing the manual feature extraction task. Recent SRL studies using Deep Learning have shown improved performance over feature-based systems for the English, Chinese and other languages. Amharic exhibits typical Semitic behaviors that pose challenges to the SRL task, such as, rich morphology, and multiple subject-verb-object word orders. In this work, we approach the problem of SRL for the language using deep learning. The input is raw sentence with words represented using a concatenation of word, character, and fastText-level neural word embeddings to capture the morphological, syntactic and semantic information of the words in sentences, and requires no intermediate feature extraction tasks. We have used a bi-directional Recurrent Neural Network (RNN) with Long-Short Term Memory (LSTM) to capture the bi-directional (for argument identification) and long-range (for argument boundary identification), and a conditional random field with viterbi-decoding to implement the SRL system for the language. The system was trained on 8000 instances and tested on 2000 instances, and achieved an accuracy of 94.96% and F-score of 81.2%. We have manually annotated the sentences with their corresponding semantic roles, and future works can consider improving the quality of the data and experiment feature representations using contextual embeddings for improved performance.

Keywords: Semantic role labeler, deep learning, neural word embedding, RNN, LSTM

DEDICATION

This thesis work is dedicated to my beautiful daughters, Eldana and Nael, who has been a constant source of joy in my life. I am truly thankful for having you in my life. I love you, ladies.

ACKNOWLEDGEMENTS

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout this research work.

I would like to express my deep and sincere gratitude to my research advisor, Dr. Yaregal Assabie, for giving me the opportunity to do this research and providing me invaluable guidance throughout this research. He has taught me the methodology to carry out this research and to present the research as clearly as possible. It was a great privilege and honor to work and study under his guidance.

I am much obliged to Mr. Nesredien Suleiman for his endless support and encouragements throughout my stay.

I would like to express my heartfelt thanks to all the Computer Science Department staff for their wisdom and professionalism.

Last but not least, I would like to thank my family and friends for their immense patience and support through this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ACRONYMS AND ABBREVATIONS

Bi-RNN	Bi-directional RNN
CNN	Convolutional Neural Network
CoNLL	Conference on Natural Language Learning
CRF	Conditional Random Field
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
ILP	Integer Linear Programming
IOB	Inside–outside–beginning Tagging
k-NN	k-nearest Neighbor
LM	Language Model
LSTM	Long Short Term Memory
MBL	Memory Based Learning
ML	Machine Learning
MVDM	Modified Value Difference Metric
NLP	Natural Language Processing
POS / PoS	Part of speech tagging
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SRL	Semantic Role Labeling/Labeler
SRN	Simple Recurrent Network
SVMs	Support Vector Machines
TAM	Tense-Aspect-Mood
TiMBLE	Tilburg Memory Based Learner
XOR	Exclusive OR

CHAPTER ONE: INTRODUCTION

1.1 Background

Natural Language Processing (NLP), is a branch of Artificial Intelligence which deals with the interaction between computers and humans using techniques in computer science and linguistics to achieve a human-like language processing for a range of tasks or applications [1]. NLP systems require knowledge of language and linguistic analysis because the rules that dictate the passing of information using natural (human) language are not easy for computers to understand, that is, a text input can be ambiguous, and therefore, multiple, alternative linguistic structures can be built for it [1, 2, 3].

Linguistic analysis techniques in NLP often begin with Phonetics and Phonological analysis, and proceed with Morphology, Syntax, Semantics, Pragmatics, and Discourse analysis [1, 3]. Semantic analysis aims at extracting the meaning of words, phrases and sentences, including the semantic roles of entities mentioned in a sentence and quantification information (such as cardinality, iteration, and dependency) [1, 3]. The tasks involved in semantic analysis are: semantic role labeling, semantic parsing, word sense disambiguation and compositional semantics [1]. The semantic role labeling task is a shallow semantic analysis task that tries to figure out the participants of an event and their relationships in a sentence; the semantic parsing task turns the sentence into its logical form; the word sense disambiguation task tries to figure out what the words in a sentence mean; in compositional semantics, the meaning of a sentence is computed based on the meaning of its parts. Semantic analysis, which determines the meaning of a given sentence and represents that meaning in an appropriate form, is an important component of many NLP applications [1, 2, 3].

Semantic role labeling (SRL) is a sentence-level semantic analysis of text concerned with the characterization of events, such as determining “who” did “what” to “whom”, “where”, “when”, and “how” [4]. Jurafsky and Martin [1] define SRL as the task of automatically finding the semantic roles of each argument of each predicate in a sentence, and argue that in order to accomplish this, the role-bearing constituents in a clause must be identified and their correct

semantic role labels assigned. This identification of participants and their relationships in an event can potentially benefit many NLP applications, such as Question Answering [6], Machine Translation [7, 8], Information Extraction [9], Text Summarization [10], and many more [11].

NLP techniques can use machine learning to derive meaning from natural languages [1, 2, 3]. The machine learning approaches employed in NLP can be: rule-based, memory-based, statistical, and deep learning [1]. Rule-based approaches to NLP focus on pattern-matching with predefined rules. Facts or rules about a language are explicitly represented and linguistic models are developed using explicit rules, in the rule-based approach [1]. Memory-based learning approaches use similarity of new situations to earlier observed situations as their basis for inference [1, 12]. The statistical machine learning approaches employ various mathematical theories for inference [1, 13, 14]. Deep learning provides a very flexible, universal, and learnable framework for representing linguistic information, enabling multi-level automatic feature representation learning [15]. Generalized linguistic models are developed using observed examples of linguistic phenomena in the statistical, memory-based and deep learning approaches [1, 2, 3]. Recently, deep learning techniques have obtained very high performance across many different NLP tasks [2, 16].

SRL systems can be feature-based or neural-based. Feature-based SRL systems are implemented using statistical [13, 14, 17, 18] and memory-based [12, 19, 20] machine learning techniques. Feature-based approaches rely on syntactic features and require heavy feature engineering [13, 21]. Neural-based SRL systems, on the other hand, allow the system to learn the intermediate representations (features). Neural-based SRL systems are implemented using deep learning. Deep learning have been proven to be particularly good at learning intermediate representations [15], which is advantageous in language modeling for NLP tasks [2, 16].

SRL studies using Deep Learning have shown improved performance over the feature-based ones for languages such as English [22, 23, 24] and Chinese [25]. Thus, we hypothesize that the performance of SRL systems for Amharic text can also be improved using neural-based approach. SRL systems are language dependent [1, 5] and we cannot directly apply neural-based SRL studies for other languages to the Amharic language as the linguistic characteristics may differ. The purpose of this study is, therefore, to design and develop an SRL for Amharic text using Deep Learning.

1.2 Motivation

SRL has become a leading task in computational linguistics today, as it is an effective approach to understand underlying meanings associated with word relationships in natural language sentences [1, 4, 5].

Amharic is the official working language of the government of the Federal Democratic Republic of Ethiopia which, according to the *World Population Prospect 2019* of the United Nations [26], has a population estimate of 112 million.

The development of an efficient and effective SRL for Amharic would be used as an input to facilitate the development of other related NLP applications for the language.

All these are sought to be the deriving factor for this thesis proposal.

1.3 Statement of the Problem

There have been several research works on automatic SRL systems for a language or different languages using feature-based and neural-based approaches. The feature-based approaches to SRL are implemented using statistical [13, 14, 17, 18] and memory-based [12, 19, 20] machine learning techniques, often using lexical resources. Feature-based SRL systems rely on syntactic features, and require heavy feature engineering to make a final best prediction - which includes human-designed representations and input features along with weight optimization. However, manually designed features are often over-specified, incomplete, and take a long time to design and validate [13, 21]. Moreover, errors encountered in the intermediate syntactic parsing tasks can potentially propagate to the SRL prediction. Neural-based SRL systems, in contrast, learn the intermediate features from the raw text, bypassing the manual feature extraction task, and are implemented using deep learning technique. Deep learning methods provide flexible learning of intermediate representations from raw inputs, effective end-to-end joint system learning, effective learning methods for using contexts and transferring between tasks, as well as better regularization and optimization methods [15].

SRL systems are language dependent, i.e., an SRL system developed for one language cannot be directly applied for another language because linguistic analysis and representations of languages may differ [1, 5]. Eskedar Yirga [19] has tried to address the lack of an automatic SRL for the

Amharic with memory-based learning (MBL) approach, and has used morphological and syntactic parsers for the intermediate features extraction task. Recent SRL studies using Deep Learning have shown improved performance over the feature-based ones for languages such as English [22, 23, 24] and Chinese [25]. Therefore, we hypothesize that the performance of SRL systems for Amharic text can also be improved using Deep Neural Networks.

1.4 Objectives

General Objective

The overall objective of this study is to design and develop an SRL for Amharic text using Deep Learning.

Specific Objectives

The specific objectives of this study are:

- Review the approaches, methods and techniques used in deep-learning for SRL.
- Collect and prepare digital Amharic text.
- Model a Deep Neural Network for the Amharic SRL using Neural Word Embedding to represent linguistic features of words in Amharic sentences.
- Design and develop a prototype of the proposed model.
- Evaluate the proposed model.

1.5 Methods

The methodology applied in order to achieve the general and specific objectives of this study are presented as follows.

Literature Review

Extensive literature review will be conducted to gain enough understanding of deep learning and their applications in SRL, and the linguistic characteristics of the Amharic language.

Data Collection

Digital Amharic text will be collected from various sources.

Experimentation

For experimental purpose, a prototype of the proposed model will be developed using available open-source tools. The system will be trained, and experiments will be conducted to determine the neural word representations required for the language, and to optimize the model parameters for improved performance.

1.6 Evaluation

Evaluation techniques will focus in determining the neural word representation required for the Amharic SRL to improve the performance. The proposed model will be evaluated using the standard evaluation metrics for semantic role labeling, i.e., computing accuracy precision, recall, and F-measure.

1.7 Scope and Limitations

The scope of this proposal is to design and develop an automatic semantic role labeling (SRL) for machine-editable simple Amharic sentences using deep learning.

This study is limited to the computational resources available to us, and the expertise knowledge in annotating the semantic roles.

1.8 Application of Results

This study is believed to be used as an input in the development of NLP applications for the Amharic language. Some of the possible NLP applications for the language that can benefit from this study are: Question Answering, Machine Translation, Information Extraction, Text Summarization, and many more.

1.9 Organization of the Rest of the Thesis

The remaining part of this Thesis is organized as follows. Chapter two presents a theoretical background about semantic role labeling, neural networks and neural language models, and linguistic features of the Amharic language. Chapter three presents summary of the related previous works on semantic role labeling. The fourth chapter explains the proposed approach of

this study. Chapter five presents the experiments done and the results obtained. Finally, the conclusion and possible future works of the study are presented in Chapter six.

CHAPTER TWO: LITERATURE REVIEW

2.1 Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence which deals with the interaction between computers and humans using techniques in computer science and linguistics to achieve a human-like language processing for a range of tasks or applications [1]. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. NLP systems require knowledge of language and linguistic analysis because the rules that dictate the passing of information using natural language are not easy for computers to understand, that is, a text input can be ambiguous and therefore multiple, alternative linguistic structures can be built for it [1, 2, 3].

Linguistic analysis techniques in NLP often begin with Phonetics and Phonological analysis, which deals with the linguistic sounds of a language, and proceeds with Morphology, Syntax, Semantics, Pragmatics, and Discourse analysis, sequentially [1, 3]. Morphology deals with the meaningful components of words; Syntax analysis acquires knowledge of the structural relationships between words in a sentence; Semantics deals with the meaning of words, phrases and sentences; Pragmatics focuses on the relationship of meaning to the goals and intentions of the speaker; finally, Discourse deals with the meaning of linguistic units larger than a single utterance [1, 3].

NLP techniques rely on machine learning to derive meaning from natural languages [1, 2, 3]. Machine learning in NLP can be supervised, unsupervised, or semi-supervised. In supervised machine learning, NLP systems require well-labeled data (annotated examples) for training. However, unsupervised machine learning techniques allow NLP models to work on their own to discover information. Semi-supervised learning combines unlabeled data with a small labeled training set.

Approaches to NLP can be: rule-based, memory-based, statistical, and deep learning [1]. Facts or rules about a language are explicitly represented and linguistic models are developed using explicit rules, in the rule-based approach [1]. Memory-based learning approach uses similarity of

new situations to earlier observed situations as its basis for inference [1, 12]. The statistical approaches employ various mathematical theories for inference [1, 13, 14]. Deep learning provides a very flexible, universal, and learnable framework for representing linguistic information enabling multi-level automatic feature representation learning [15]. Generalized linguistic models are developed using observed examples of linguistic phenomena in the statistical, memory-based and deep learning approaches [1, 2, 3]. Recently, deep learning studies have obtained very high performance across many different NLP tasks [2, 16].

A natural language processing task may involve in one or more levels of linguistic analysis [1]. Our study focuses on the semantic role labeling task in semantic analysis. Semantic role labeling is a shallow semantic analysis task that tries to figure out the participants of an event and their relationships in a sentence [3]. This task of understanding how participants relate to events—being able to answer the question “Who did what to whom” (and perhaps also “when and where”)—is a central question of natural language understanding, and therefore, a crucial task of many NLP applications.

2.2 Semantic Role Labeling - SRL

Semantic Role Labeling (SRL) is a sentence-level semantic analysis of text concerned with the characterization of events, such as determining “who” did “what” to “whom”, “where”, “when”, and “how” [5]. The predicate of a clause (typically a verb) establishes “what” took place, and other sentence constituents express the participants in the event (such as “who” and “where”), as well as further event properties (such as “when” and “how”) [4]. Jurafsky and Martin [1] define SRL as the task of automatically finding the semantic roles of each argument of each predicate in a sentence, and argue that in order to accomplish this, the role-bearing constituents in a clause must be identified and their correct semantic role labels assigned. Given a sentence, SRL consists of three basic tasks: predicate identification, argument identification, and argument classification [4, 5]. This task of understanding how participants relate to events—being able to answer the question “Who did what to whom” (and perhaps also “when and where”)—is a central question of natural language understanding.

2.2.1 Semantic Roles

Consider the example in [1] about the very mundane goal of understanding text about a purchase of stock by XYZ Corporation:

- XYZ corporation bought the stock.
- They sold the stock to XYZ corporation.
- The stock was bought by XYZ corporation.
- The purchase of the stock by XYZ corporation...
- The stock purchase by XYZ corporation...

This purchasing event and its participants can be described by a wide variety of surface forms: the event can be described by a verb (sold, bought) or a noun (purchase), and XYZ Corp can be the syntactic subject (of bought), the indirect object (of sold), or in a genitive or noun compound relation (with the noun purchase) despite having notionally the same role in all of them.

There is a level of representation that captures the commonality between these sentences: there was a purchase event, the participants were XYZ Corp and some stock, and XYZ Corp was the buyer. These shallow semantic representations, called *semantic roles*, express the role that arguments of a predicate take in the event [5].

In the above sentence representations, the roles of the subjects of the verbs *buy/purchase* and *sell* are *buyer/perchaser* and *seller* respectively. These *deep roles* [1] are specific to each event; *Buying/Purchasing* events have *Buyers/Purchasers*, *Selling* events have *Sellers*.

To be able to answer questions, perform inferences, or do any further kinds of natural language understanding of these events, more knowledge about the semantics of these arguments is required [5]. *Buyers/Purchasers* and *Sellers* have something in common. They are both actors, and they have direct possesive responsibility for their events. Thematic roles [1] are a way to capture this semantic commonality between Buyers/Purchasers and Sellers. The subjects of both these verbs are *agents*. AGENT is the thematic role that represents an abstract idea such as possesional causation. Similarly, the direct objects of both these verbs, the *Bought/PurchasedThing* and *SoldThing*, are both objects that are affected in some way by the action. The semantic role for these participants is *theme*.

Although most thematic roles sets have about a dozen roles, there is no universally agreed-upon set of roles [1].

The main reason computational systems use semantic roles is to act as a shallow meaning representation that can let us make simple inferences that are not possible from the pure surface string of words, or even from the parse tree [5]. Semantic roles, thus, help generalize over different surface realizations of predicate arguments [4, 5]. For example, while the AGENT is often realized as the subject of the sentence, in other cases the THEME can be the subject. Consider the following sentences:

- *John [AGENT] broke the window. [THEME]*
- *John [AGENT] broke the window THEME with a rock. [INSTRUMENT]*
- *The rock [INSTRUMENT] broke the window. [THEME]*
- *The window [THEME] broke.*
- *The window [THEME] was broken by John. [AGENT]*

These examples suggest that *break* has, at least, the possible arguments: AGENT, THEME, and INSTRUMENT.

The set of thematic role arguments taken by a verb is often called the *thematic grid*, *θ-grid*, or *case frame* [1]. Verbs can allow their thematic roles to be realized in various syntactic positions. These multiple argument structure realizations (the fact that break can take AGENT, INSTRUMENT, or THEME as subject, and give can realize its THEME and GOAL in either order) are called *verb alternations* or *diathesis alternations* [1]. Representing meaning at the thematic role level is useful in dealing with complications like diathesis alternations [1]. Yet it has proved quite difficult to come up with a standard set of roles, and equally difficult to produce a formal definition of roles like AGENT, THEME, or INSTRUMENT [5].

These problems have led to alternative semantic role models [1] that use either many fewer or many more roles: The first option is to define generalized semantic roles that abstract over the specific thematic roles. For example, PROTO-AGENT and PROTO-PATIENT are generalized roles that express roughly agent-like and roughly patient-like meanings. These roles are defined, not by necessary and sufficient conditions, but rather by a set of heuristic features that

accompany more agent-like or more patient-like meanings. Thus, the more an argument displays agent-like properties, the greater the likelihood that the argument can be labeled a PROTO-AGENT. The more patient-like the properties, the greater the likelihood that the argument can be labeled a PROTO-PATIENT. The second direction is, instead, to define semantic roles that are specific to a particular verb or a particular group of semantically related verbs or nouns.

2.2.2 Lexical Resources

The two commonly used lexical resources that make use of these alternative versions of semantic roles are PropBank and FrameNet. PropBank uses both proto-roles and verb-specific semantic roles. FrameNet uses semantic roles that are specific to a general semantic idea called a frame. The difference between these two models of semantic roles is that FrameNet employs many frame-specific frame elements as roles, while PropBank uses a smaller number of numbered argument labels that can be interpreted as verb-specific labels, along with the more general argument labels [1]

PropBank

The Proposition Bank, referred to as PropBank, is a resource of sentences annotated with semantic roles. Because of the difficulty of defining a universal set of thematic roles [5], the semantic roles in PropBank are defined with respect to an individual verb sense [27]. Each sense of each verb thus has a specific set of roles, which are given only numbers rather than names: Arg0, Arg1, Arg2, and so on. In general, Arg0 represents the PROTO-AGENT, and Arg1, the PROTO-PATIENT. The semantics of the other roles are less consistent, often being defined specifically for each verb. Nonetheless, there are some generalization: the Arg2 is often the benefactive, instrument, attribute, or end state, the Arg3 is the start point, benefactive, instrument, or attribute, and the Arg4 is the end point. A list of PropBank semantic roles is presented in ANNEX A.

An example of a slightly simplified PropBank entry [27], called a *frame file*, for one sense each of the verbs *agree* would be:

agree.01
Arg0: Agreer

Arg1: Proposition

Arg2: Other entity agreeing

Example: [$_{ArgM-TMP}$ Usually] [$_{Arg0}$ John] agrees [$_{Arg2}$ with Mary] [$_{Arg1}$ on everything].

The PropBank semantic roles can be useful in recovering shallow semantic information about verbal arguments. A PropBank semantic role labeling would allow us to infer the commonality [1] in the event structures of the following three examples, that is, that in each case Big Fruit Co. is the AGENT and the price of bananas is the THEME, despite the differing surface forms.

[$_{Arg0}$ Big Fruit Co.] increased [$_{Arg1}$ the price of bananas].

[$_{Arg1}$ The price of bananas] was increased again [$_{Arg0}$ by Big Fruit Co.]

[$_{Arg1}$ The price of bananas] increased [$_{Arg2}$ 5%].

PropBank also has a number of non-numbered arguments called ArgMs, (ArgMTMP, ArgM-LOC, etc.) which represent modification or adjunct meanings. These are relatively stable across predicates, so are not listed with each frame file. Data labeled with these modifiers can be helpful in training systems to detect temporal, location, or directional modification across predicates [1].

While PropBank focuses on verbs, a related project, NomBank [28] adds annotations to noun predicates. For example, the noun agreement in *Apple's agreement with IBM* [1] would be labeled with Apple as the Arg0 and IBM as the Arg2. This allows semantic role labelers to assign labels to arguments of both verbal and nominal predicates.

FrameNet

While making inferences about the semantic commonalities across different sentences with a verb is useful, it would be even more useful to make such inferences in many more situations, across different verbs, and also between verbs and nouns [29]. For example, we would like to extract the similarity among these three sentences in [1]:

[$_{Arg1}$ The price of bananas] increased [$_{Arg2}$ 5%].

[$_{Arg1}$ The price of bananas] rose [$_{Arg2}$ 5%].

There has been a [$_{Arg2}$ 5%] rise [$_{Arg1}$ in the price of bananas].

Note that the second example uses the different verb rise, and the third example uses the noun rather than the verb rise. The FrameNet project is another SRL project that attempts to address, no matter whether the 5% appears as the object of the verb *increased* or as a nominal modifier of the noun *rise*, *the price of bananas* is what went up, and that *5%* is the amount it went up [1].

Whereas roles in the PropBank project are specific to an individual verb, roles in the FrameNet project are specific to a *frame* [30]. Consider the following set of words:

> *reservation, flight, travel, buy, price, cost, fare, rates, meal, plane*

There are many individual lexical relations of hyponymy, synonymy, and so on between many of the words in this list [1]. The holistic background knowledge that unites these words is called a frame [31].

A frame in FrameNet is a background knowledge structure that defines a set of frame-specific semantic roles, called *frame elements*, and includes a set of predicates that use these roles [1]. The semantic roles (frame elements) in a frame are separated into *core roles*, which are frame specific, and *non-core roles*, which are more like the Arg-M arguments in PropBank, expressing more general properties of time, location, and so on [29]. Each word evokes a frame and profiles some aspect of the frame and its elements. The FrameNet dataset includes a set of frames and frame elements, the lexical units associated with each frame, and a set of labeled example sentences [29]. For example, the *change_position_on_a_scale* frame [1] is defined as follows.

> This frame consists of words that indicate the change of an Item's position on a scale (the Attribute) from a starting point (Initial value) to an end point (Final value).

Example sentences of this frame that includes target words like *rise*, *fall*, and *increase* are:

> [ITEM Food items] rose [ATTRIBUTE in price] [DIFFERENCE by 2%].
>
> [ITEM It] has increased [FINAL STATE to having them 1 day a month].
>
> [ITEM The company shares] fell [FINAL VALUE to less than 1%].

FrameNet also codes relationships between frames [29], allowing frames to inherit from each other, or representing relations between frames like causation, and generalizations among frame elements in different frames can be representing by inheritance as well.

2.2.3 Approaches to Semantic Role Labeling

Semantic role labeling can be feature-based and neural-based. The feature-based ones are implemented with statistical and memory-based approaches to it, and are supervised. While the neural network approaches can be semi-supervised [1].

Feature-based Algorithms

Feature-based algorithms [1, 5] begin by parsing, using broad-coverage parsers to assign a parse to the input string. The parse is then traversed to find all words that are predicates. For each of these predicates, the algorithm examines each node in the parse tree and uses supervised classification to decide the semantic role (if any) it plays for this predicate. Given a labeled training set such as PropBank or FrameNet, a feature vector is extracted for each node, using feature templates. A 1-of-N classifier is then trained to predict a semantic role for each constituent given these features, where N is the number of potential semantic roles plus an extra NONE role for non-role constituents. Any standard classification algorithms can be used. Finally, for each test sentence to be labeled, the classifier is run on each relevant constituent. Algorithm 2.1, taken from Jurafsky and Martin [1], shows the computation procedure for a simplified feature-based semantic role labeling.

Algorithm 2.1: A Simplified Feature-based Semantic Role Labeling

Input: List of words in a sentence
Output: Semantic roles of each word corresponding to each predicate in a sentence in a labeled tree

```
function SEMANTICROLELABEL(words)
    parse ←PARSE(words)
    for each predicate in parse do
        for each node in parse do
            featurevector ← EXTRACTFEATURES (node, predicate, parse)
            CLASSIFYNODE (node, featurevector, parse)
```

Instead of training a single-stage classifier as in the above algorithm, the node-level classification task can be broken down into multiple steps [5], as explained below.

1. Pruning: Since only a small number of the constituents in a sentence are arguments of any given predicate, many systems use simple heuristics to prune unlikely constituents.
2. Identification: a binary classification of each node as an argument to be labeled or a NONE.
3. Classification: a 1-of-N classification of all the constituents that were labeled as arguments by the previous stage

The separation of identification and classification may lead to better use of features (different features may be useful for the two tasks) or to computational efficiency [5].

The assumption that *each argument of a predicate can be labeled independently* is wrong, because there are interactions between arguments that require a more 'global' assignment of labels to constituents. Role labeling systems, thus, often add a fourth step to deal with global consistency across the labels in a sentence [5]. For example, the local classifiers can return a list of possible labels associated with probabilities for each constituent, and a second-pass *Viterbi decoding* or *re-ranking* approach can be used to choose the best consensus label [1]. Integer Linear Programming (ILP) is another common way to choose a solution that conforms best to multiple constraints[1].

Most systems use some generalization of the core set of features introduced by Gildea and Jurafsky [32]. Common basic features templates include: the governing *predicate*, the *phrase type* of the constituent, the *headword* of the constituent computed with standard head rules, the *headword* part of speech of the constituent, the *path* in the parse tree from the constituent to the predicate, the *voice* of the clause in which the constituent appears (active or passive), the binary *linear position* of the constituent with respect to the predicate (either before or after), the *subcategorization* of the predicate, the *named entity type* of the constituent, the *first* or *last word* of the constituent, and etc...

Feauture-based approaches to SRL are based on supervised machine learning, often using the FrameNet and PropBank lexical resources to specify what counts as a predicate, define the set of roles used in the task, and provide training and test sets [1]. They are implemented with the memory-based, or statistical approaches.

Neural Algorithms for SRL

The standard neural algorithm for semantic role labeling is based on the bi-LSTM IOB tagger [1, 22]. With IOB tagging, there is a begin and end tag for each possible role (B-ARG0, I-ARG0; B-ARG1, I-ARG1, and so on), plus an outside tag O. The goal is to compute the highest probability tag sequence $y\hat{}$, given the input sequence of words w: $y\hat{} = argmax_{y \in T} P(y|w)$

The algorithm in [22] maps each input word to pre-trained embeddings, and also associates it with an embedding for a flag (0/1) variable indicating whether that input word is the predicate. These concatenated embeddings are passed through multiple layers of bi-directional LSTM. State-of-the-art algorithms [22] use 3 to 4 layers (6 to 8 total LSTMs), and highway layers are used to connect these layers.

Output from the last bi-LSTM can then be turned into an IOB sequence. Tags can be locally optimized by taking the bi-LSTM output, passing it through a single layer into a softmax for each word that creates a probability distribution over all SRL tags and the most likely tag for word x_i is chosen as t_i, computing for each word essentially: $y_i\hat{} = argmax_{t \in tags} P(t|w_i)$

However, just as feature-based SRL tagging, this local approach to decoding does not exploit the global constraints between tags; a tag I-ARG0, for example, must follow another I-ARG0 or B-ARG0 [1]. There are many ways [22] to take advantage of these global constraints. A conditional random field (CRF) layer can be used instead of a softmax layer on top of the bi-LSTM output, and the Viterbi decoding algorithm can be used to decode from the CRF. An even simpler Viterbi decoding algorithm that may perform equally well and does not require adding CRF complexity to the training process is to start with the simple softmax. The softmax output (the entire probability distribution over tags) for each word is then treated as a lattice and Viterbi decoding can be done through the lattice. The hard IOB constraints can act as the transition probabilities in the Viterbi decoding (thus the transition from state I-ARG0 to I-ARG1 would have probability 0). Alternatively, the training data can be used to learn bi-gram or tri-gram tag transition probabilities as if doing HMM decoding [1].

2.2.4 Evaluation Metrics for Semantic Role Labeling Systems

The standard evaluation for semantic role labeling is to require that each argument label must be assigned to the exactly correct word sequence, and then compute accuracy, precision, recall, and

F-score [1]. To compute these evaluation measures, we will need to count the observed True Positive, True Negative, False Positive and False Negatives of the classification outcomes for each class considered [1], where:

- True Positive (TP) is the number observations that are part of the positive, and are predicted correctly.
- True Negative (TN) is the number of observations that are part of the negative class, and are predicted correctly.
- False Positive (FP), also called a *Type I error*, is the number of observations predicted to be part of the positive class but is actually part of the negative class.
- False Negatives (FN), also called a *Type II error*, is the number of observations predicted to be part of the negative class but is actually part of the positive class.

Accuracy can, thus, be measured as the percentage of inputs in the test set that the classifier correctly labeled:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Precision is the proportion of every observation predicted to be positive that is actually positive, i.e., how many of the items identified were relevant. Models with high precision are pessimistic in that they only predict an observation is of the positive class when they are very certain about it. Formally, precision is:

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

Recall is the proportion of every positive observation that is truly positive. It measures the model's ability to identify an observation of the positive class. Models with high recall are optimistic in that they have a low bar for predicting that an observation is in the positive class:

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

As an evaluation metric, accuracy has some valuable properties, especially its simple intuition. However, better metrics often involve using some balance of precision and recall, i.e., a trade-off

between the optimism and pessimism of our model. F-Score [34] represents a balance between the recall and precision, where the relative contributions of both are equal.

The **F-Measure** (or **F-Score**), which combines the precision and recall to give a single score, is defined to be the harmonic mean of the precision and recall. It is a measure of correctness achieved in positive prediction, i.e., of the observations labeled as positive, how many are actually positive:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

Table 2.1: Confusion Matrix with the Evaluation Metrics

		Actual Class		
		Positive	**Negative**	
Predicted Class	**Positive**	TP: True Positive	FP: False Positive (Type I Error)	Precision: $\frac{TP}{TP + FP}$
	Negative	FN: False Negative (Type II Error)	TN: True Negative	Negative Predictive Value[1]: $\frac{TN}{TN + FN}$
		Recall or Sensitivity: $\frac{TP}{TP + FN}$	Specificity[2]: $\frac{TN}{TN + FP}$	Accuracy: $\frac{TP + TN}{TP + TN + FP + FN}$

To provide an insight into the predictions of an SRL model, a *Confusion Matrix* can also be used. A confusion matrix is an *N* x *N* matrix used for evaluating the performance of a classification model, where *N* is the number of target classes [1]. The matrix compares the actual target values with those predicted by the model, allowing an easy identification of confusion between the target classes. This provides a holistic view of how well the classification model is performing and what kinds of errors (*confusion*) it is making. Table 2.1 shows a summary of the evaluation metrics presented in a confusion matrix, taken from Jurafsky and Martin in [1].

A *Learning Curve* cal also be used to plot the changes in learning performance over time in terms of experience. Reviewing the learning curve of an SRL model during training can be used

[1] Proportion of every observation predicted to be negative that is actually negative.
[2] Proportion of every negative observation that is truly negative.

to diagnose problems with learning, such as an underfit or overfit model, as well as whether the training and validation datasets are suitably representative [36].

2.3 Neural Networks

Neural networks are a fundamental computational tool for language processing, inspired by McCulloch-Pitts neuron [33], a simplified model of the human neuron as a kind of computing element that could be described in terms of propositional logic. A modern neural network, however, is a network of small computing units, each of which takes a vector of input values and produces a single output value through a non-linear function [1].

2.3.1 Basic Structure of Neural Networks

The building block of a neural network is a single computational *unit*, which takes a set of real valued numbers as input, performs some computation on them, and produces an output. A neural unit is takes a weighted sum of its inputs, with one additional term in the sum called a *bias* term [34].

Given a set of inputs $x_1 ... x_n$, a unit has a set of corresponding weights $w_1 ... w_n$ and a bias b, the authors in [34] represent the weighted sum z as:

$$z = \sum_i wixi + b \tag{5}$$

This weighted sum can be represented using vector notation, which is just a list or array of numbers. Thus, the sum z in terms of a weight vector w, a scalar bias b, and an input vector x, can be represented with the convenient dot product:

$$z = w \cdot x + b \tag{6}$$

Here, z is just a real valued number and, instead of using it as a linear function of x, neural units apply a non-linear function f to it [34-36]. The output of this function is referred to as the *activation value* for the unit, a. For a single unit, the activation for the node is the final output of the network, y. So the value y is defined in [34] as:

$$y = a = f(z) \tag{7}$$

The three popular non-linear functions $f()$ are: the *sigmoid*, the t*anh*, and the *rectified linear unit – ReLU* [34].

The *sigmoid* equation is given in [34] as:

$$y = \sigma(z) = \frac{1}{1+e^{-z}} \tag{8}$$

The sigmoid has a number of advantages; it maps the output into the range [0, 1], which is useful in squashing outliers toward 0 or 1. And it's differentiable, which will be handy for learning.

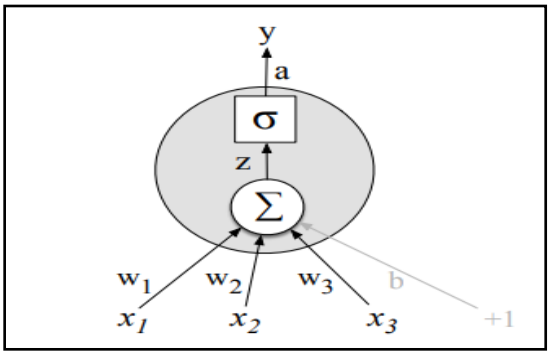


Figure 2.1: A Neural Unit

Figure 2.1, taken from Jurafsky and Martin in [1], shows a neural unit, taking 3 inputs x_1, x_2, and x_3 and computes a weighted sum - multiplying each value by a weight (w_1, w_2, and w_3, respectively), adds them to a bias term b as a weight for an input clamped at +1, and then passes the resulting sum through a sigmoid function producing an output y to result in a number between 0 and 1.

The sigmoid is not commonly used as an activation function. A function that is very similar but almost always better is the *tanh* function; tanh is a variant of the *sigmoid* that ranges from -1 to +1, and is given in [34] as:

$$y = tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \tag{9}$$

The simplest activation function and perhaps the most commonly used, is the *rectified linear unit*, also called the *ReLU*. It's just the same as *x* when *x* is positive, and 0 otherwise [34]:

$$y = max(x, 0) \tag{10}$$

These activation functions have different properties that make them useful for different language applications or network architectures [34, 35]. For example, the rectifier function has nice properties that result from it being very close to linear. In the sigmoid or tanh functions, very high values of *z* result in values of *y* that are saturated, that is, extremely close to 1, which causes

problems for learning. Rectifiers do not have this problem, since the output of values close to 1 also approaches 1 in a nice gentle linear way. By contrast, the tanh function has the nice properties of being smoothly differentiable and mapping outlier values toward the mean.

The power of neural networks comes from combining the neural units into larger networks [34]. One of the demonstrations for the need for multi-layer networks was the proof by Minsky and Papert [37] that a single neural unit cannot compute some very simple functions of its input. The logical functions of XOR, cannot be calculated by a single *perceptron*, a very simple neural unit that has a binary output and does not have a non-linear activation function. A network made up of simple linear units (perceptron) cannot solve the XOR problem. This is because a network formed by many layers of purely linear units can always be reduced, shown to be computationally identical, to a single layer of linear units with appropriate weights [34]. But, it can be calculated by a layered network of units with nonlinear activation functions: using two layers of ReLU-based units, shown in Figure 2.2, taken from the authors in [34].

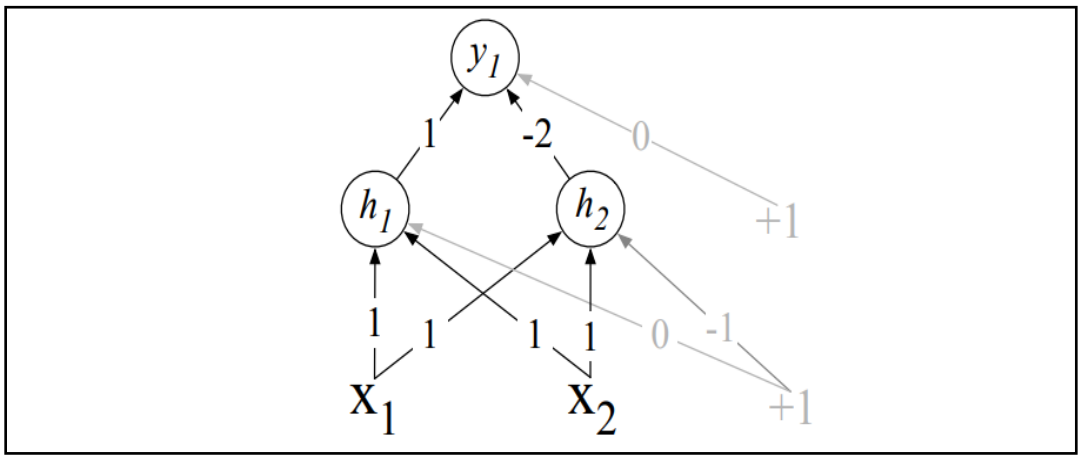


Figure 2.2: XOR Solution

In this figure, the input is being processed by two layers of neural units. The middle layer (called hidden layer, *h*) has two units, and the output layer (*y*) has one unit. A set of weights and biases are shown for each ReLU that correctly computes the XOR function. The hidden layer of the network can be viewed as forming a representation for the input.

The weights are stipulated in the example. However, in real examples,the weights for neural networks are learned automatically using a technique called *error backpropagation* [34-36]. That means the hidden layers will learn to form useful representations. This intuition, that neural

networks can automatically learn useful representations of the input, is one of their key advantages [34].

2.3.2 Types of Neural Networks

There are different types of neural networks, based on architecture and usage. For neural networks to learn in a faster and more efficient way, various neurons are placed in the network in such a way as to maximize the learning of the network for the given problem [34]. This placing of neurons follows a sensible approach and results in an architectural network design with different neurons consuming the output of other neurons, or different functions taking output from other functions in their inputs. If the neurons are placed with connections among them taking the form of cycles, then they form networks such as feedback, recursive, or recurrent neural networks. If, however, the connections between the neurons are acyclic, they form networks such as feedforward neural networks.

Feed-Forward Neural Networks

A feedforward network is a multilayer network in which the units are connected with no cycles; the outputs from units in each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers [34].

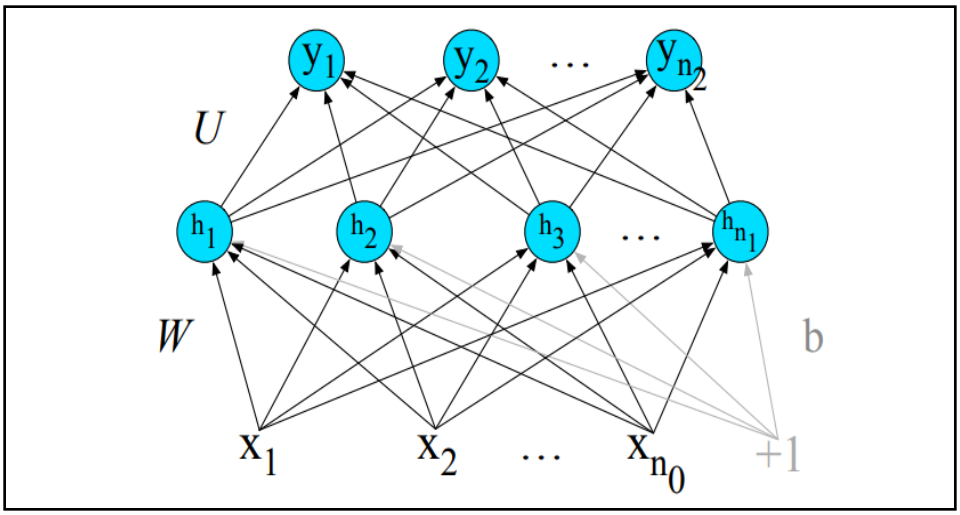


Figure 2.3: A Simple 2-Layer Feedforward Network

Simple feedforward networks have three kinds of nodes: input units, hidden units, and output units. Figure 2.3, taken from Jurafsky and Martin in [1], shows a simple feedforward neural

network. The input units are simply scalar values. The core of the neural network is the hidden layer formed of hidden units, each of which is a neural unit, taking a weighted sum of its inputs and then applying a non-linearity. In the standard architecture, each layer is fully-connected, meaning that each unit in each layer takes as input the outputs from all the units in the previous layer, and there is a link between every pair of units from two adjacent layers. Thus each hidden unit sums over all the input units [34].

The resulting value *h* (for hidden but also for hypothesis) forms a representation of the input. The role of the output layer is to take this new representation *h* and compute a final output. This output could be a real valued number, but in many cases the goal of the network is to make some sort of *classification* decision, such as: sentiment, or multinomial classification [35].

The equations for a feedforward network with a single hidden layer, which takes an input vector *x*, outputs a probability distribution *y*, and is parameterized by weight matrices *W* and *U* and a bias vector *b*, are given in [34] as:

$$\begin{aligned} h &= \sigma(Wx+b) \\ z &= Uh \\ y &= softmax(z) \end{aligned} \tag{11}$$

Algorithm 2.2, taken from the authors in [34], shows the procedure for computing the forward step in an *n*-layer feedforward network, given the input vector $a^{[0]}$:

Algorithm 2.2: Computing the Forward Step in a Feedforward Network

Input: Input vector, n-layers
Output: A probability distribution *y*

function FEEDFORWARD($a^{[0]}$, n)
    for i in 1..n
        $z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]}$
        $a^{[i]} = g^{[i]}(z^{[i]})$
    $\hat{y} = a^{[n]}$
    return $\hat{y}$

The activation functions $g(\cdot)$ are generally different at the final layer. Thus $g^{[2]}$ might be softmax for multinomial classification or sigmoid for binary classification, while ReLU or tanh might be the activation function $g(\cdot)$ at the internal layers [34, 35].

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a neural network that has one or more convolutional layers, which is essentially sliding a filter over the input. Their structure is based on sampling a window or portion of an input, detecting its features, and then using the features to build a representation [34]. Each convolutional layer contains a series of filters known as *convolutional kernels*. The filter is a matrix of integers that are used on a subset of the input values, the same size as the kernel. Each input is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid, in the output channel/feature map. They are used mainly for image processing, classification and segmentation [34, 35].

Recurrent Neural Networks

Language is an inherently temporal phenomenon, it is a sequence that unfolds in time [1]. When dealing with written text we normally process it sequentially, even though we in principle have arbitrary access to all the elements at once. The feedforward neural networks do not have this temporal nature. Such networks employ fixed-size input vectors with associated weights to capture all relevant aspects of an example at once. This makes it difficult to deal with sequences of varying length, and they fail to capture important temporal aspects of language. These models operate by accepting fixed-sized windows of tokens as input; sequences longer than the window size are processed by sliding windows over the input making predictions as they go, with the end result being a sequence of predictions spanning the input. Importantly, the decision made for one window has no impact on later decisions [34]. The sliding window approach is problematic for a number of reasons. First, it shares the primary weakness of Markov approaches in that it limits the context from which information can be extracted; anything outside the context window has no impact on the decision being made. This is an issue since there are many language tasks that require access to information that can be arbitrarily distant from the point at which processing is

happening. Second, the use of windows makes it difficult for networks to learn systematic patterns arising from phenomena like constituency.

A Recurrent Neural Network (RNN) is a class of networks designed to address these challenges by dealing directly with the temporal aspect of language, allowing us to handle variable length inputs without the use of arbitrary fixed-sized windows, and providing the means to capture and exploit the temporal nature of language [1, 34]. A recurrent neural network is any network that contains a cycle within its network connections. That is, any network where the value of a unit is directly, or indirectly, dependent on earlier outputs as an input. While powerful, such networks are difficult to reason about and to train. However, within the general class of recurrent networks there are constrained architectures that have proven to be extremely effective when applied to spoken and written language [35, 36].

Depending on the complexity of their network architechtures, RNNs can be simple or deep.

Simple Recurrent Neural Networks

These networks are useful in their own right and serve as the basis for more complex approaches. Although the basic computation is similar with ordinary feedforward networks, in departure from the window-based approach, sequences are processed by presenting one element at a time to the network.

The hidden layer from the previous time step provides a form of memory, or *context*, that encodes earlier processing and informs the decisions to be made at later points in time. Critically, this architecture does not impose a fixed-length limit on this prior context; the context embodied in the previous hidden layer includes information extending back to the beginning of the sequence [34]. The most significant change lies in the new set of weights, *U*, that connect the hidden layer from the previous time step to the current hidden layer. These weights determine how the network should make use of past context in calculating the output for the current input. As with the other weights in the network, these connections are trained via backpropagation. Figure 2.4 [1], taken from Jurafsky and Martin in [1], illustrates the simple RNN architecture.

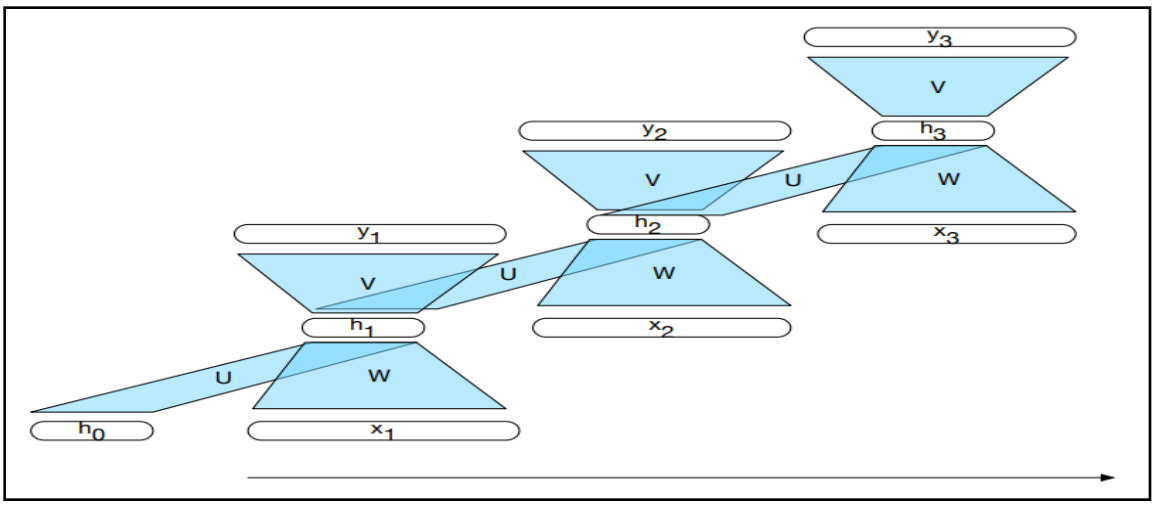


Figure 2.4: A Simple RNN shown Unrolled in Time

Forward inference (mapping a sequence of inputs to a sequence of outputs) in an RNN is nearly the same as feedforward networks. To compute an output y_t for an input x_t , we need the activation value for the hidden layer h_t. To calculate this, we multiply the input x_t with the weight matrix W, and the hidden layer from the previous time step h_{t-1} with the weight matrix U. We add these values together and pass them through a suitable activation function, g, to arrive at the activation value for the current hidden layer, h_t . Once we have the values for the hidden layer, we proceed with the usual computation to generate the output vector, as shown in Algorithm 2.3, taken from the authors in [34].

$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t) \quad (12)$$

In the commonly encountered case of soft classification, computing y_t consists of a softmax computation that provides a normalized probability distribution over the possible output classes [35, 36].

$$y_t = \text{softmax}(Vh_t)$$

The fact that the computation at time t requires the value of the hidden layer from time *t-1* mandates an incremental inference algorithm [34] that proceeds from the start of the sequence to the end. The matrices U, V and W are shared across time, while new values for h and y are calculated with each time step.

Algorithm 2.3: Computing the Forward Step in RRNs

Input: Input sequence *x*, the network
Output: Output sequence *y*

function FORWARDRNN(x, network)

$h_0 \leftarrow 0$

for i←1 to LENGTH(x) do

$h_i \leftarrow g(Uh_{i-1} + Wx_i)$

$y_i \leftarrow f(Vh_i)$

return y

Deep RNNs: Stacked and Bidirectional RNNs

Recurrent networks are quite flexible. By combining the feedforward nature of unrolled computational graphs with vectors as common inputs and outputs, complex networks can be treated as modules that can be combined in creative ways [34]. Two of the more common network architectures used in language processing with RNNs are Stacked RNNs and Bidirectional RNNs.

Stacked RNNs

Stacked RNNs consist of multiple networks where the output of one layer serves as the input to a subsequent layer, depicted in Figure 2.5, taken from the authors in [1].

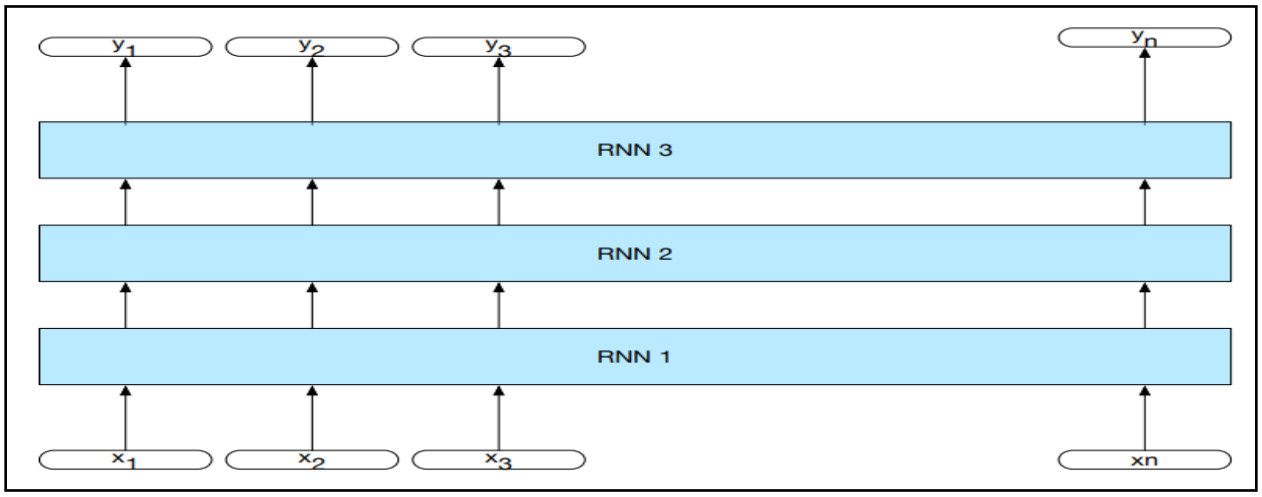


Figure 2.5: A Stacked RNN

Stacked RNNs can outperform single-layer networks. One reason for this success has to do with the network's ability to induce representations at differing levels of abstraction across layers. The initial layers of stacked networks can induce representations that serve as useful abstractions for further layers — representations that might prove difficult to induce in a single RNN [34]. The optimal number of stacked RNNs is specific to each application and to each training set. However, as the number of stacks is increased the training costs rise quickly [35, 36].

Bidirectional RNNs

These networks take advantage of the context to the right of the current input as well. A Bi-RNN consists of two independent RNNs, one where the input is processed from the start to the end, and the other from the end to the start. Combining the forward and backward networks result in a bi-directional RNN [38]. Figure 2.6, taken from Jurafsky and Martin [1], shows architechture of B-RNNs.

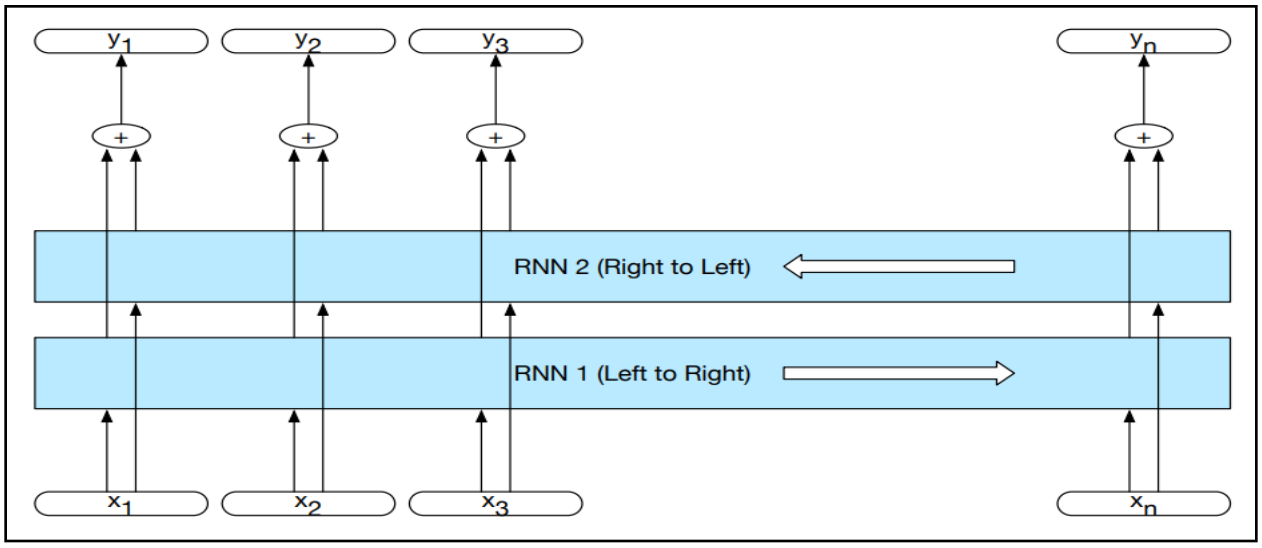


Figure 2.6: A Bi-directional RNN

The forward network can be considered as the context of the network to the left of the current time, given in [38] as:

$$h_t^f = \text{RNN}_{\text{forward}}(x_1^t) \tag{13}$$

Where h_t^f corresponds to the normal hidden state at time *t*, and represents everything the network has gleaned from the sequence to that point.

In the backward network, the hidden state at time *t* now represents information about the sequence to the right of the current input, given in [38] as:

$$h_t^b = \text{RNN}_{\text{backward}}(x_t^n) \tag{14}$$

Here, the hidden state h_t^b represents all the information we have discerned about the sequence from *t* to the end of the sequence. The outputs of the two networks are then combined into a single representation that captures both the left and right contexts of an input at each point in time, given in [38] as:

$$h_t = h_t^f \oplus h_t^b \tag{15}$$

Concatenation is a common approach to combining the two outputs but element-wise summation, multiplication or averaging can also be used [34].

Managing Context in RNNs: Gated RNNs

In practice, it is quite difficult to train RNNs for tasks that require a network to make use of information distant from the current point of processing [34]. Despite having access to the entire preceding sequence, the information encoded in hidden states tends to be fairly local, more relevant to the most recent parts of the input sequence and recent decisions. It is often the case, however, that distant information is critical to many language applications [35].

One reason for the inability of RNNs to carry forward critical information is that the hidden layers, and, by extension, the weights that determine the values in the hidden layer, are being asked to perform two tasks simultaneously: provide information useful for the current decision, and updating and carrying forward information required for future decisions. A second difficulty with training RRNs arises from the need to backpropagate the error signal back through time. The hidden layer at time *t* contributes to the loss at the next time step since it takes part in that calculation. As a result, during the backward pass of training, the hidden layers are subject to repeated multiplications, as determined by the length of the sequence. A frequent result of this process is that the gradients are eventually driven to zero – the so-called *vanishing gradients problem*.

To address these issues, more complex network architectures have been designed to explicitly manage the task of maintaining relevant context over time. More specifically, the network needs

to learn to forget information that is no longer needed and to remember information required for decisions still to come. Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode [34]. These include the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) based networks.

Long Short-Term Memory - LSTM

Long Short-Term Memory (LSTM) networks [39] divide the context management problem into two sub-problems: removing information no longer needed from the context, and adding information likely to be needed for later decision making. LSTMs manage the context by first adding an explicit context layer to the architecture (in addition to the usual recurrent hidden layer), and through the use of specialized neural units that make use of gates to control the flow of information into and out of the units that comprise the network layers. These gates are implemented through the use of additional weights that operate sequentially on the input, and previous hidden layer, and previous context layers.

The gates in an LSTM share a common design pattern; each consists of a feedforward layer, followed by a sigmoid activation function, followed by a pointwise multiplication with the layer being gated [34]. The choice of the sigmoid as the activation function arises from its tendency to push its outputs to either 0 or 1. Combining this with a pointwise multiplication has an effect similar to that of a binary mask. Values in the layer being gated that align with values near 1 in the mask are passed through nearly unchanged; values corresponding to lower values are essentially erased.

The LSTM network architecture explicitly splits the state vector s_i 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). A detailed architecture of an LSTM cell is depicted in Figure 4.5, taken from the authors [36].

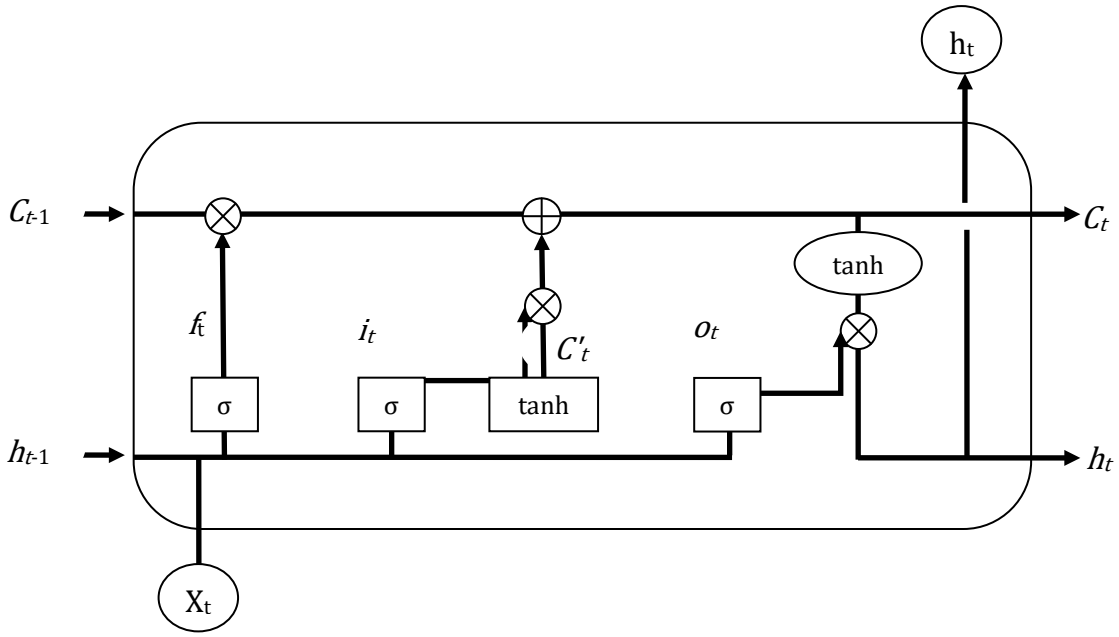


Figure 2.7: A Detailed Architecture of an LSTM Cell

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. The author in [35] represents the LSTM architecture mathematically as:

$$s_t = R_{LSTM}(s_{t-1}, x_t) = [c_t; h_t] \quad (16)$$

$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi}) \quad (17)$$

$$f_t = \sigma\left(x_t W_{xf} + h_{t-1} W_{hf}\right) \quad (18)$$

$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho}) \quad (19)$$

$$z_t = tanh(x_t W_{xz} + h_{t-1} W_{hz}) \quad (20)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot z \quad (21)$$

$$h_t = o_t \odot tanh(c_t) \quad (22)$$

$$y_t = O_{LSTM}(s_t) = h_t \quad (23)$$

where $s_t \in \mathbb{R}^{2.d_h}$, $x_i \in \mathbb{R}^{d_x}$, $c_t, h_t, i_t, f_t, o_t, z_t, \in \mathbb{R}^{d_h}$, $W_{xo} \in \mathbb{R}^{d_x \times d_h}$, and $W_{ho} \in \mathbb{R}^{d_h \times d_h}$.

The state s_t at time t is composed of two vectors, c_t and h_t , where c_t is the memory (context) component and h_t is the hidden state component. There are three gates: i, f, and o, controlling

for input, forget, and output, respectively. The gate values are computed based on linear combinations of the current input x_t and the previous state h_{t-1}, passed through a sigmoid activation function, as in equation 17, 18, and 19. An update candidate z is computed as a linear combination of x_t and h_{t-1}, passed through a tanh activation function (equation 20). The context c_t in equation 21 is then updated: the forget gate controls how much of the previous memory to keep $(f \odot c_{t-1})$, and the input gate controls how much of the proposed update to keep $(i \odot z)$. Finally, the value of h_t (which is also the output y_t) is determined based on the content of the memory c_t, passed through a *tanh* non-linearity and controlled by the output gate (equation 22 and 23). The gating mechanisms allow for gradients related to the memory part c_t to stay high across very long time ranges.

Gated Recurrent Units- GRU

LSTMs introduce a considerable number of additional parameters to recurrent networks [34]. It now have 8 sets of weights to learn (that is, the *U* and *W* for each of the 4 gates within each unit), whereas with simple recurrent units it only had 2. Training these additional parameters imposes a much significantly higher training cost. Gated Recurrent Units (GRUs) [34] ease this burden by dispensing with the use of a separate context vector, and by reducing the number of gates to 2 — a reset gate, *r* and an update gate, *z*. The purpose of the reset gate *r* is to decide which aspects of the previous hidden state are relevant to the current context and what can be ignored. The job of the update gate z is to determine which aspects of this new state will be used directly in the new hidden state and which aspects of the previous state need to be preserved for future use.

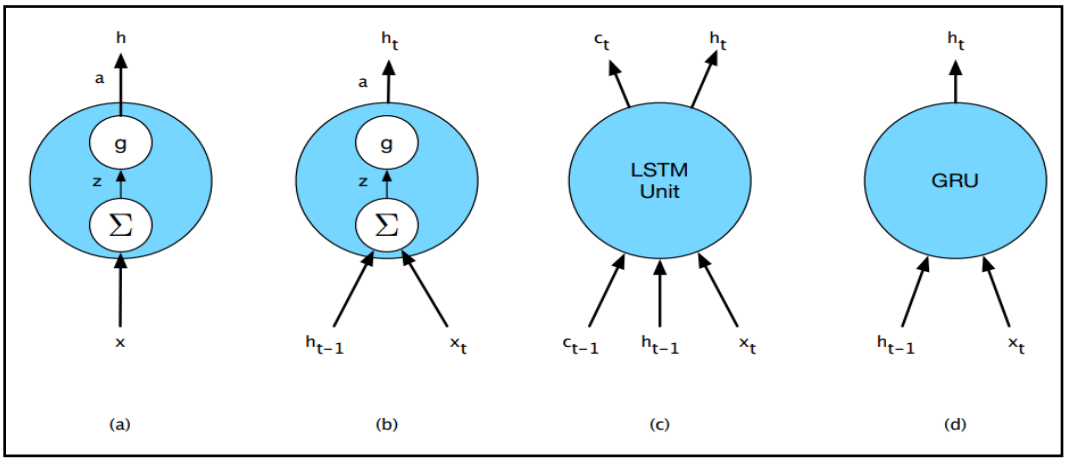


Figure 2.8: Inputs Associated with Basic Neural Units

Figure 2.7, taken from Jurafsky and Martin in [1], illustrates the inputs and outputs associated with basic neural units used in feedforward (a), simple recurrent networks - SRN (b), long short-term memory - LSTM (c) and gate recurrent units - GRU (d).

Applications of Recurrent Neural Networks

Recurrent neural networks have proven to be an effective approach to language modeling, sequence labeling tasks (such as part-of-speech tagging, named entity recognition, structure prediction, semantic role labeling, etc...), as well as sequence classification tasks (such as sentiment analysis and topic classification) [1, 35, 36]. In sequence labeling, the network's task is to assign a label chosen from a small fixed set of labels to each element of a sequence. In the sequence classification task, sequences of text are classified as belonging to one of a small number of categories.

Encoder-Decoder Networks

Encoder-Decoder Networks use one network to create an internal representation of the input, or to "encode" it, and that representation is used as an input for another network to produce the output, or "decode" it [34]. This is useful to go beyond a classification of the input. The final output can be in the same modality, such as, language translation [40], or a different modality, such as, text tagging of an image, based on concepts [35, 36].

The key idea underlying these networks is the use of an *encoder* network that takes an input sequence and creates a contextualized representation of it. This representation is then passed to a *decoder* which generates a task-specific output sequence. The encoder and decoder networks are typically implemented with the same architecture, often using recurrent networks, and allowing the architecture to be trained in an end-to-end fashion for each application [34].

The encoder and the decoder are assumed to have the same internal structure, and the final state of the encoder is the only context available to the decoder as its initial hidden state [34, 40]. The encoder-decoder networks consist of three components [1]. these are:

1. An *encoder* that accepts an input sequence, x_1^n , and generates a corresponding sequence of contextualized representations, h_1^n .

2. A *context* vector, *c*, which is a function of h_1^n, and conveys the essence of the input to the decoder.
3. And a *decoder*, which accepts *c* as input and generates an arbitrary length sequence of hidden states h_1^n, from which a corresponding sequence of output states y_1^m, can be obtained.

The final hidden state inevitably is more focused on the latter parts of input sequence, rather than the input as whole.

To overcome this deficiency, an *attention* mechanism is used to take the entire encoder context into account that dynamically updates during the course of decoding, and that can be embodied in a fixed-size vector [34].

The addition of attention to basic encoder-decoder networks has led to rapid improvement in performance across a wide-range of applications including summarization, sentence simplification, question answering and image captioning [35, 36].

Recursive Neural Networks

In a Recursive Neural Network, a fixed set of weights is recursively applied onto the network structure and is primarily used to discover the hierarchy or structure of the data [34]. Whereas an RNN is a chain, a recursive neural network takes the form of a treelike structure. Such networks have great use in the field of NLP [35, 36], such as to decipher the sentiment of a sentence as the overall sentiment is not dependent on the individual works only, but also on the order in which the words are syntactically grouped in the sentence.

2.3.3 Training Neural Networks

In general, neural networks can be trained in three steps [34]. First it requires a loss function that models the distance between the system output (predicted value) and the gold output (actual value), using the *cross-entropy loss*. Second, to find the parameters that minimize this loss function, the *gradient descent* optimization algorithm can be used. Third, gradient descent requires knowing the gradient of the loss function, the vector that contains the partial derivative of the loss function with respect to each of the parameters. The loss over all those intermediate

layers can be partialed out using the algorithm called *error-backpropagation* or *reverse differentiation*.

Optimization

Optimization in neural networks is a non-convex optimization problem [1], and there are many best practices for successful learning. In neural networks, we need to initialize the weights with small random numbers. It is also helpful to normalize the input values to have 0 mean and unit variance.

Various forms of regularization are used to prevent overfitting [34]. One of the most important is *dropout*: randomly dropping some units and their connections from the network during training [41, 42]. *Tuning* of hyperparameters is also important. The parameters of a neural network are the weights W and biases b; these are learned by gradient descent. The hyperparameters are things that are chosen by the algorithm designer; optimal values are tuned on a devset rather than by gradient descent learning on the training set. Hyperparameters include the learning rate η, the mini-batch size, the model architecture (the number of layers, the number of hidden nodes per layer, the choice of activation functions), and so on.

2.3.4 Vector Semantics and Embeddings

Vectors semantics is the standard way to represent word meaning in NLP, in which the meaning of a word is defined by its distribution in language use, meaning its neighboring words or grammatical environments [1]. The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distributions of word neighbors. Vectors for representing words are called embeddings. A vector is, at heart, just a list or array of numbers. A vector space is a collection of vectors, characterized by their dimension. Vector or distributional models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur. The two most commonly used vector models for representing word meanings are sparse and dense [1].

In sparse models each dimension corresponds to a word in the vocabulary V and cells are functions of co-occurrence counts. The meaning of a word is defined by a simple function of the counts of nearby words, and the word is represented by a long vector with dimensions

corresponding to words in the vocabulary or documents in a collection. The term-document matrix has a row for each word (term) in the vocabulary and a column for each document. The word-context or term-term matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary. Two sparse weightings are common: the tf-idf weighting which weights each cell by its term frequency and inverse document frequency, and PPMI (point-wise positive mutual information) most common for word-context matrices [1].

In dense vectors, instead of vector entries being sparse, mostly-zero counts or functions of counts, the values are real-valued numbers that can be negative. Dense vectors, or embeddings, are more powerful word representation and work better in every NLP task than sparse vectors. *Word2vec* algorithms like skip-gram are a popular way to compute dense embeddings. Skip-gram trains a logistic regression classifier to compute the probability that two words are 'likely to occur nearby in text' **[**43**]**. This probability is computed from the dot product between the embeddings for the two words. Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words. An extension of word2vec, *fastText*, deals with unknown words and sparsity in languages with rich morphology, by using subword models [44]. Each word in fastText is represented as itself plus a bag of constituent n-grams. Other important embedding algorithms include *GloVe*, a method based on ratios of word co-occurrence probabilities [1]. These embeddings are static, meaning that the method learns one fixed embedding for each word in the vocabulary. In contextual embeddings like the popular BERT[3] or ELMO[4] representations, on the other hand, the vector for each word is different in different contexts.

Whether using sparse or dense vectors, word and document similarities are computed by some function of the dot product between vectors. The cosine of two vectors, a normalized dot product, is the most popular such metric [1].

A word embedding is a learned representation for text as real-valued vectors in a predefined vector space where words that have the same meaning have a similar representation. These vectors can capture the grammatical function (*syntax*) and the meaning (*semantics*) of the words

[3] https://github.com/google-research/bert
[4] https://tfhub.dev/google/elmo/

in a sentence, for a list of words defined (called vocabulary) [1]. Word embeddings are capable of capturing context of a word in a sentence, semantic and syntactic similarity, relation with other words, etc. Words with such similarity are assigned similar word embeddings.

2.3.5 Neural Language Models

Neural net-based language models have many advantages [15, 16]. Among these are that neural language models do not need smoothing, they can handle much longer histories, and they can generalize over contexts of similar words. For a training set of a given size, a neural language model has much higher predictive accuracy. Furthermore, neural language models underlie many of the models used for tasks like machine translation, dialog, and language generation. On the other hand, there is a cost for this improved performance: neural net language models are strikingly slower to train [1, 16].

Given a fixed preceding context, the feedforward language model [35] attempts to predict the next word in a sequence. More formally, it computes the conditional probability of the next word in a sequence given the preceding words, $P(w_n|w_1^{n-1})$. The quality of a model is largely dependent on the size of the context and how effectively the model makes use of it. Thus, this sliding-window neural network is constrained by the Markov assumption embodied in the following equation, taken from Jurafsky and Martin in [1].

$$P(w_t|w_1^{n-1}) \approx P(w_t|w_{n-N+1}^{n-1}) \tag{24}$$

That is, anything outside the preceding context of size *N* has no bearing on the computation [1].

Prior context is represented by *embedding*s of the previous words, rather than by exact words, in neural language models allowing them to generalize to unseen data. For an architecture that allows the embeddings to be learned, an extra layer is added to the network, and the error propagated all the way back to the embedding vectors, starting with embeddings having random values and slowly moving toward sensible representations.

With *e* representing the projection layer, formed by concatenating the *n* embeddings for the *n* context vectors, the values of the weight parameters for a feedforward neural language model is given in [1] as:

$$e = (Ex_1, Ex_2, ...Ex_n)$$
$$h = \sigma(We+b)$$
$$z = Uh$$
$$y = \text{softmax}(z) \quad (25)$$

Recurrent neural language models process sequences a word at a time attempting to predict the next word in a sequence by using the current word and the previous hidden state as input [43]. At each forward inference step in a recurrent language model, the network retrieves a word embedding for the current word as input and combines it with the hidden layer from the previous step to compute a new hidden layer. This hidden layer is then used to generate an output layer which is passed through a softmax layer to generate a probability distribution over the entire vocabulary, represented in [1] as:

$$P(w_n|w_1^{n-1}) = y_n$$
$$= softmax(Vh_n) \quad (26)$$

The probability of an entire sequence is then just the product of the probabilities of each item in the sequence, represented in [1] as shown below.

$$\text{P(w1n)} = \prod_{k=1}^{n} P(wk\,|w1k-1)$$
$$= \prod_{k=1}^{n} yk \quad (27)$$

A Conditional Random Fields (CRF) can be combined with recurrent neural network architectures for sequence labeling tasks, such as SRL, to jointly model the label decision by capturing the dependencies across adjacent labels [46].

For $\text{x} = \{x_1, x_2, \dots, x_N\}$ and $\text{y} = \{y_1, y_2, \dots, y_N\}$ representing observed input tokens and corresponding output labels - respectively, the authors in [35] compute a linear-chain CRF's distribution $p(y \mid x)$, as shown below.

$$p(y \mid x) = \frac{1}{Z(x)} \prod_{\text{c}} \emptyset_{\text{c}}\,(\text{y}, \text{x}) \quad (28)$$

where $Z(x)$ is a normalization function computed in [35] as:

$$Z(x) = \sum_k \prod_{\text{c}} \emptyset_{\text{c}}\,(\text{y}_{\text{c}}, \text{x}) \quad (29)$$

Here, the model learns the transition probability of the output labels, $A \in \mathbb{R}^K$, where K is the number of distinct labels including their start and end markers.

The input features of observed tokens $x \in \mathbb{R}^K$ in the clique $\emptyset_c(y, x)$ is the score matrix P_{i,y_i} learned in this layer. All the clique potentials $\emptyset_c$, $1 \leq c \leq C$ are conditioned on input features. The log-linear representation of potential is assumed for the cliques.

For the given sequence of predictions $y = \{y_1, y_2, \ldots, y_N\}$, the probability score S including the start and end tag, y_0 and y_{N+1}, where N is the length of the sequence, is represented in [35] as:

$$S(x, y) = \sum_{i=0}^{N} A_{y_i, y_{i+1}} + \sum_{i=0}^{N} P_{i,y_i} \tag{30}$$

The probability for the sequence y is given in [35] by:

$$p(y \mid x) = \frac{e^{S(x,y)}}{\sum_{y' \in y} e^{S(x,y)}} \tag{31}$$

The objective function is the maximum likelihood of the probability distribution, computed in [35] as:

$$\ln p(y \mid x) = S(x, y) - \ln \sum_{y' \in y} e^{S(x,y)} \tag{32}$$

The final output tag sequence is decided based on the maximum score, represented in [35] as:

$$y' = \text{argmax}_{y' \in y} S(x, y') \tag{33}$$

2.3.6 Training the Neural Language Model

To train the model, that is, to set all the parameters $\theta = E$ (embedding), W (input weight), U (intermediate weight), b (bias), the gradient descent is computed using error backpropagation. Training, thus, not only sets the weights W and U of the network, but also as it predicts upcoming words, it learns the embeddings E for each words that best predict upcoming words [35].

Training proceeds by taking as input a very long text, concatenating all the sentences, starting with random weights, and then iteratively moving through the text predicting each word w_t.

At each word w_t , the cross-entropy (negative log likelihood) loss, is computed in [1] as:

$$L = -\log p(w_t \mid w_{t-1}, \ldots, w_{t-n+1}) \tag{34}$$

The gradient for this loss is then given in [1] as:

$$\theta t + 1 \;=\; \theta t \;-\eta \frac{\partial \,-\log p(wt\,|wt-1,\ldots,wt-n+1)}{\partial\,\theta} \qquad (35)$$

This gradient can be computed in any standard neural network framework [34], which will then backpropagate through *U*, *W*, *b*, *E*.

Training the parameters to minimize loss will result both in an algorithm for language modeling (a word predictor) but also a new set of embeddings *E* that can be used as word representations for other tasks.

2.4 A Summary of the Approches to SRL

Feature-based approaches to SRL often use lexical resources and rely on the syntactic information in identifying the argument, especially, in the very first stage - the pruning stage. A full parse tree information, or a shallow parsing information, which includes part-of-speech tags, chunks and clauses, is required as input in such an SRL system. Feature-based approches to SRL can be useful for languages with large available lexical resouces annotated by professionals - as the errors in parses used can propagate to the SRL prediction.

Neural-based SRL, in contrast, can be used for any language because Neural Networks are capable of learning the intermediate representations present in natural langue sentences – bypassing the manual feature extraction task, which makes them especially useful for less-resourced languages. Neural Networks also provide an effective end-to-end joint system learing, avoiding the need to approach the SRL problem in modular steps (such as prunning, argument identification and argument classification) as in the feature-based approaches to it. Regularization and optimization methods used in Neural Networks are also better compared to the feature-based ones.

2.5 The Amharic Language

Amharic is the official working language of the government of the Federal Democratic Republic of Ethiopia which, according to the *World Population Prospect 2019* of the United Nations [26], has a population estimate of 112 million. Amharic belongs to the well-established Ethiopian Semitic (or Ethio-Semitic, occasionally also African Semitic) branch of South Semitic.

Amharic exhibits typical Semitic behavior [47], in particular the pattern of inflectional and derivational morphology, along with some characteristic Ethiopian Semitic features, such as SOV word order.

2.5.1 Orthography

Amharic is written using a syllabic writing system, one originally developed for the extinct Ethiopian Semitic language Ge'ez and later extended for Amharic and other Ethiopian Semitic languages. Each character of the Ge'ez (or Ethiopic) writing system gets its basic shape from the consonant of the syllable, and the vowel is represented through more or less systematic modifications of these basic shapes [47].

Amharic has 27 consonant phonemes (ሀ, ለ, መ, ረ, ሰ, ሸ, ቀ, በ, ተ, ቸ, ነ, ኘ, አ, ከ, ወ, ዘ, ዠ, የ, ደ, ጀ, ገ, ጠ, ጨ, ጰ, ፀ, ፈ, ፐ), and 7 vowels (አ, ኡ, ኢ, ኣ, ኤ, እ, ኦ). The basic character set consists of one character for each combination of 33 (ሀ, ለ, መ, ረ, ሰ, ሸ, ቀ, በ, ተ, ቸ, ነ, ኘ, አ, ከ, ወ, ዘ, ዠ, የ, ደ, ጀ, ገ, ጠ, ጨ, ጰ, ፀ, ፈ, ፐ, ሐ, ሠ, ኀ, ዐ, ኸ, ጸ) consonants and 7 vowels (አ, ኡ, ኢ, ኣ, ኤ, እ, ኦ). There are also 37 (ሏ, ሗ, ሟ, ሿ, ቧ, ቋ, ቆ, ቧ, ቧ, ኳ, ቷ, ቿ, ኇ, ኗ, ኯ, ኧ, ዅ, ዟ, ዧ, ዷ, ጇ, ጎ, ጧ, ጯ, ፏ, ፗ, ᎀ, ᎁ, ᎂ, ᎃ, ᎄ, ᎅ, ᎆ) further characters representing labialized variants of the consonants followed by particular vowels. The complete system has 268 characters [48].

When the Amharic writing system is used without vocalization diacritics, it does embody two sorts of ambiguity.

First, one of the characters in each consonant set has a dual function: it represents that consonant followed by a particular vowel, and it represents a bare consonant, that is, the consonant as the coda of a syllable. The vowel for these characters is the high central vowel 'እ', the sixth vowel in the set of characters for a given consonant. This ambiguity presents no serious problems because the presence or absence of the vowel 'እ' is normally predictable from the phonetic context [47].

A more serious sort of ambiguity results from the failure of the writing system to indicate consonant gemination. Gemination is a characteristic feature of Amharic, possible with all but two of the consonants and in all positions except initially. For example, አለ - "he said", and አለ - "there is". Gemination has both lexical and grammatical functions and is quite common; most words contain at least one geminated consonant. Therefore syntax must be relied on to disambiguate these words [49].

2.5.2 Morphology

Morphology is the study of words, how they are formed, and their relationship to other words in the same language [47]. It analyzes the structure of words and parts of words, such as stems, root words, prefixes, and suffixes. Words can be derived or inflected. Derivational morphology is concerned with forming new lexemes, that is, words that differ either in syntactic category (part of speech) or in meaning from their bases. The most widespread of techniques for derivation is affixation, the addition of prefixes, suffixes, infixes, circumfixes, and so on, but new words are often formed by other means such as reduplication, internal modification or rearrangement of consonants and vowels, or by subtraction of segments. Inflectional morphology is the study of processes, including affixation and vowel change that distinguish word forms in certain grammatical categories. The prototypical inflectional categories include: tense, case, voice, aspect, person, number, gender, mood, definiteness and others, all of which usually produce different forms of the same word rather than different words. Inflectional morphology differs from derivational morphology or word-formation in that inflection deals with changes made to existing words and derivation deals with the creation of new words.

Derivational Morphology

Lexicon

Verb roots are the basis of much of the lexicon of Amharic [48, 50]. There is also a composite verb category, which is a defining feature of the language. These consist of a word in an invariant form, the composite verb "lexeme", followed immediately by an inflected form of one or another of the verbs አለ - "say", አደረገ - "do, make", or አሰኘ - "cause to feel". A small number of the composite verb lexemes are monomorphemic, underived forms, for example, ቁጭ - "sit", ዝም - "be quiet", but most are derived from verb roots. In combination with አለ, the result is an intransitive verb; in combination with አደረገ or አሰኘ a transitive verb.

Nouns, adjectives, and numerals have similar morphosyntactic properties. Nouns are specified for definiteness, but there is no counterpart to the "status" (construct vs. absolute). Many nouns are derived from verb roots, for example, ነጂ - "driver" from ንድ - "drive", ጥቅስ - "quotation" from ጥቅስ - "quote". Many others are not, for example, ቤት - "house", ሣር - "grass", ሆድ - "stomach". The great majority of adjectives are derived from verb roots, for example, ክፉ - "bad,

evil” from ክፍ - “become bad, evil”, ለፍላፊ - “talkative” from ልፍልፍ - “talk too much”, ብርቱ - “strong” from ብርት - “become strong”.

Amharic has a small number of monomorphemic, underived adverbs and conjunctions, for example, መቼ - “when”, ግን - “but”. However, most expressions with adverbial or conjunctive function consist of nouns, usually with prepositional prefixes and possibly postpositions as well, or subordinate clauses of one sort or another, for example, በግምት - “approximately”, lit. “by guess”. Especially common are clauses with a verb in the gerund form, including composite verbs including a gerund form of the verbs አለ or አደረገ: ቀስ ብሎ - “slowly (3 pers.sing.masc.)”, ዝም ብዬ - “silently, without doing anything (1 pers.sing.)”. Note that the gerund, like all Amharic verbs, agrees with its subject, so these adverbials vary with the main clause subject.

Amharic has approximately nine prepositions. The one-syllable prepositions are treated as prefixes; the two-syllable prepositions may be written as prefixes or as separate words. Prepositions occur with and without a much larger set of postpositions. Examples: ቤቱ - “the house”, እቤቱ - “at/in the house”, ወደ ቤቱ - “to the house”, እቤቱ ዉስጥ - “inside the house”, ከቤቱ ዉስጥ - “from inside the house”, እቤቱ አጠገብ - “next to the house”.

Root and Pattern Processes

Amharic morphology [48] is very complex. Word formation relies primarily on root-and-pattern morphology. As an example, consider the root ስብር, which is the basis of many words having to do with the notion of breaking. Within the verb system, each root appears in four different tense-aspect-mood (TAM) forms and up to ten different forms representing various derivational categories such as causative and reciprocal. Each has a rough meaning that may or may not hold for particular roots. For example, the “passive-reflexive” pattern denotes the passive for a root like ስብር - “break” but the transitive active for a root like ርክብ - “receive”.

Each combination of TAM and derivational category defines a particular pattern template that combines with roots to form a verb stem. For example, the pattern C1:አC2:አC3 represents the imperfect TAM and passive-reflexive derivational category (the C’s indicate the consonant slots, and “:” indicates gemination). In combination with the root ስብር, this pattern yields the stem ስበር - “is broken”.

Nouns and adjectives are formed from a verb root through the application of other patterns. For example, the pattern አC1:አC2አC2አC3 forms a manner noun, አሰባበር - “way of breaking”, and the pattern መአC1C2አC3 forms the infinitive, መስበር - “to break”.

Most of the members of the composite verb lexeme class are also derived from verb roots through root-pattern morphology. For simple three-consonant roots, the main patterns are C1እC2:እC3:, C1አC2አC3:, and C1እC2እC3C2እC3:. Thus from the root ስብር, we have ስብር, ሰበር, and ስብርብር. The meanings of these are described as “intensive”, “attenuated” and “distributive”, respectively [48].

Further complexity results from the possibility of four- and five-consonant roots and from roots which may behave as though they consist of one or two consonants because of the special treatment of certain consonants and because of historical changes. For example, the root meaning “kiss” was originally ስም, but the pharyngeal consonants have been lost in the modern language, resulting in a defective class with only two explicit consonants in each root. For this class, the imperfect passive-reflexive pattern is C1:አC2, for example, ሳም - “is/are kissed”.

Within the three-consonant roots, Amharic has three lexical subclasses, A, B, and C classes, which correspond to different derivational patterns in other Semitic languages [48]. The B class is characterized by the gemination of the second root consonant in nearly all patterns, the C class by the presence of the vowel a between the first and second consonant in all patterns. Membership of a root in one or another of these classes is lexical. For example, the A root ጥብቅ means “be tight”, while the B root ጠበቅ means “wait”. In all, there are roughly 25 classes of roots, including subclasses of the A, B, and C classes and multiple classes for four- and five-consonant roots, each with its own set of patterns.

Other derivational processes are based on suffixation [51]. For example, many adjectives are formed from nouns through the addition of the suffixes -ማ and -አዊ: ዉሃ “water”) ዉሃማ “watery”, አብዮት “revlolution”) አብዮታዊ “revolutionary”.

Inflectional Morphology

Inflectional morphology consists mostly of suffixes, but sometimes of prefixes or circumfixes, and sometimes of pattern changes [48].

Verbs

Amharic verbs inflect for TAM through the various patterns that also convey derivational categories such as passive and reciprocal [48]. The four TAM categories are referred to as perfect(ive), imperfect(ive), jussive-imperative, and gerund(ive) [50]. The perfect and imperfect forms, alone or in combination with various auxiliary verbs, function as indicative main clause verbs and, in combination with various prefixes, as subordinate clause verbs as well. The jussive corresponds to one sort of subjunctive in other languages: ይምጡ - “let them come”. The gerund, also called converb or continuative, is a subordinate form used in combination with a main clause verb.

Verbs in all four TAM categories also inflect for subject person, number, and (in second and third person singular) gender [47]. Each of the TAM categories has its own set of suffixes and/or prefixes for person–number–gender. The affixes for imperfect and jussive-imperative, a combination of prefixes and suffixes, are almost identical. For perfect and gerund, the inflections are all suffixes. As an example, for the simple derivational category, the third person plural forms for the root ሰብር are as follows (the subject agreement affixes are in bold): perfect - ሰበሩ, imperfect - **ይ**ሰብር**ኡ**(ይሰብሩ), jussive - **ይ**ስበር**ኡ**(ይስበሩ), gerund - ሰብር**አው**(ሰብረው). Verbs in all four TAM categories can also take object suffixes, and only one object suffix is possible on a given verb [50]. There is a set of direct object suffixes, and there are two sets of indirect object suffixes, distinguished by the two prepositional suffixes -በ- and -ለ- that they begin with. Examples: ሰበረ**ው** - “he broke it”, ሰበረ**ላት** - “he broke (sth.) for her”. Verbs in the imperfect, perfect, and jussive-imperative (but not gerund) may also take the negative circumfix (main clause indicative verbs) or prefix (subordinate or jussive-imperative verbs): **አ**ልሰበረ**ም** - “he didn’t break”, **አ**ትስበር - “don’t break (sing.masc.)!” Imperfect or perfect verbs may also take the relative prefix የ(ም), used for verbs in relative clauses and some other subordinate clause types: የሰበረ**ው** - “(he) who broke it”, **የ**ም**ት**ሰብረ**ው** - “which you (sing.fem.) break”. Finally, a small set of prepositional and conjunctive prefixes and conjunctive/adverbial suffixes is possible on imperfect and perfect verbs: እንዳትሰብሩት - “so that you (pl.) don’t break it” ቢሰብረ**ውም** - “**even** if he breaks it”.

Counting the verb stem as two morphemes (root and pattern), an Amharic verb [47] may consist of as many as ten morphemes: **ለማያነቡልንም** - “also for they who do not read to us” **ለ-ም-አ-ይ-{ንብብ + ኣC1አC2C3}-ኡ-ል-ን-ም**.

Nominals

Nouns, pronouns, adjectives, and numerals in Amharic have similar morphosyntax [48, 50, 51].

Nouns inflect for number, case, and definiteness. There are two number categories, singular and plural. Plurals are usually formed by the suffix **-ኦች**; broken plurals are rare. The plural suffix is not obligatory when plurality is clear from the context: **ሦስት አስተማሪ/ሦስት አስተማሪዎች** - “three teachers”. When it is used, the plural suffix indicates specificity as well as plurality [48].

The definite article takes the form of a suffix; it has the masculine form **-ኡ** and the feminine form **-ዋ** or **-ኢቱ**. The definite suffix follows the plural suffix if there is one. Amharic has no indefinite article [48].

Amharic subjects are unmarked [50]. Definite, and some indefinite, direct objects take the accusative suffix **-ን**. Oblique cases are signaled by prefixed prepositions, sometimes in combination with postpositions. Example: **መስኮት** - “window”, **ድንጋይ** - “stone”, **ዮሃንስ መስኮቱን በድንጋይ ሰበረው** - “Yohannis broke the window with a stone”. Amharic nouns have no genitive form; instead genitive and possession are indicated by the prepositional prefix **የ**: **የቤቱ ወለል** - “the floor of the house”.

Noun gender can best be described as flexible. While there are strong tendencies, for example, vehicles, countries, and female animals are normally feminine, it is possible to override these tendencies for pragmatic effect. Even male humans may be referred to with feminine pronouns, feminine demonstratives, and feminine verb agreement affixes, when the speaker wishes to convey affection toward the referent or attribute skillfulness or smallness to the referent [51].

Nouns may also take one or another of the same set of conjunctive/adverbial suffixes as verbs: **ቤቱም** - “also the house”.

Demonstrative pronouns and adjectives have singular masculine, singular feminine, and plural forms: **ያ ቤት** - “that house”, **ያቺ ከተማ** - “that city”. Other adjectives do not inflect for gender, but they may have plural forms, especially in headless noun phrases. Adjective plurals, unlike nouns,

are often indicated by reduplication [51], sometimes in combination with the plural suffix **-ኦች**: ትልቅ - “big”, ትልልቅ ቤትኦች(ቤቶች) - “big houses”. It is also possible to use plural adjectives with the singular form of nouns: አዲስ - “new”, ነገር - “thing”, አዳዲስ ነገር -“new things”.

Adjectives and numerals may also take the accusative suffix, the definite suffix, and prepositional prefixes: **ከ**ትልቅ**ኡ**(ከትልቁ) - “from the big (one)”. Possessive adjectives take the form of suffixes on the modified noun [48]. These follow the plural suffix and precede the accusative suffix. Examples: ቤ**ቴ** - “my house”, ቤቶች**አችን**(ቤቶቻችን) - “our houses”, ቤቶች**አቸው**(ቤቶቻቸው) - “their houses”.

2.5.3 Basic Syntactic Structure

Noun Phrase

An Amharic noun phrase has explicit number, case, and definiteness [48, 51]. The accusative suffix appears obligatorily on definite direct objects and optionally on indefinite direct objects. An unusual feature of the language is the placement of the morphemes marking case (either the accusative suffix or one or another prepositional prefix) and definiteness [52]. These are affixed to the noun itself only when it has no modifiers. If the noun has adjective or relative clause modifiers, the morphemes are normally affixed to the first of these. Examples: ቤ**ቱን** - “the house (acc.)”, ትልቁን ቤት - “the big house (acc.)”, የገዛውን ትልቅ ቤት - “the big house (acc.) that he bought”. Headless noun phrases are common [51]. These consist of one or more relative clauses and adjectives. Examples: ትልቁን - “the big one (acc.)”, የገዛውን - “the one (acc.) that he bought”.

Clauses

All Amharic clauses are headed by verbs. The copula, ነው, is a defective verb with only main clause present forms. Its past is filled by the defective verb ነበር, which also serves as the past of the defective verb of existence አለ. In other cases, the copula is replaced by the perfect, imperfect, jussive-imperative, or gerund of either the verb ኖረ - “live” or the verb ሆነ - “become”.

The basic word order of Amharic languages is subject-object-verb (SOV) [51]. The order of subject, object, and oblique arguments of the verb is somewhat flexible [48]. In particular, for pragmatic reasons the subject can follow another argument: የሀንስ መስኮቱን ሰበረው, መስኮቱን የሀንስ ሰበረው - “Yohannis broke the window”.

Verbs agree with their subjects in person, number, and (in second and third person singular) gender. Verbs also agree with definite direct or indirect objects, but not both. There are three sets of object agreement suffixes [48]. This three-way split within the verbal morphology system maps in complex ways onto explicit syntactic arguments of the verb. That is, direct object verb suffixes correspond usually but not always to accusative syntactic arguments, and indirect object verb suffixes correspond usually but not always to explicit prepositional phrases (noun phrases with prepositional prefixes). The -ለ- indirect object suffix agrees with dative and benefactive arguments, while the -በ- suffix agrees with locative, instrumental, and adversative arguments [51]. Examples: ለሂሩት ሰጣት - “he gave (it) to Hirut” (direct object suffix (3 pers.sing.fem.) agrees with explicit syntactic dative), ለሂሩት ሰራላት - “he made (it) for Hirut” (indirect object suffix agrees with explicit syntactic benefactive).

The morphological subject of impersonal verbs [51] is always third person singular masculine, and they take an obligatory direct object agreement suffix that refers to the experiencer: ሂሩት ራበ**አት**(ራባት) - “Hirut is hungry”. The verb of possession behaves similarly; it makes use of the verb of existence አለ with direct object suffixes referring to the possessor. The morphological subject of the verb is the possessed object or objects. ሂሩት ሦስት ወንድሞች አሏት - “Hirut has a brother” (subject agreement 3 pers.plur., object agreement 3 pers.sing.fem.).

Pronoun subjects and pronoun objects are omitted unless they are emphasized [48]. This fact, in combination with the elaborate derivational and inflectional verb morphology, means that sentences consisting of a verb alone or a main verb and an auxiliary verb are not uncommon: አልጠየቅናትም - “we didn’t visit her”, ላፍላላችሁ - “shall I boil (sth.) for you (pl.)?”, አዋደዱን - “they made us like each other”.

Main clause verbs are either in the perfect or a compound imperfect formed from the simple imperfect plus conjugated suffix forms of the defective verb of existence አለ [51]. Subordinate clause verbs are in the perfect, simple imperfect, or gerund. ትፈልጊያለሽ - “you (fem.sing.) want”, ብትፈልጊ - “if you (fem.sing.) want”.

Cleft constructions are very common in Amharic [48]. Cleft constructions for questions are probably more common that non-cleft constructions. In a cleft sentence, the focused argument is placed first, followed by the conjugated copula, followed by other arguments of the original verb,

followed by the verb in relativized form: ምንድነው የሰበረው? - “what did he break?” lit. “what is it that he broke it”.

Relative clauses make use of the relative imperfect or perfect form of the verb: የሰበረው - “that he broke”, የሚሰብረው - “that he breaks”.

Adverbial clauses are usually indicated through the use of prefix conjunctions on the relative form of the verb (in which case the initial የ is dropped) or the bare imperfect: ስለሚሰብረው - “because he breaks it”, ቢሰብረው - “if he breaks it”.

Nominalizations in Amharic take two forms [48]. Either the verb of the nominalized clause appears in the (conjugated) relative form with the prefix conjunction እንደ - “that”, or the verb appears in the (unconjugated) infinitive form, with the verb’s original subject in the form of a possessive suffix. Examples: ምን እንደሚገዛ አላውቅም - “I don’t know what he will buy”; ዝግጁ መሆናቸውን ገለፁ - “they explained that they were ready”, lit. “they explained their being (acc.) ready”.

Amharic permits the chaining of a number of clauses together in a single sentence without explicit conjunctions indicating the relationship between the clauses [48]. The usual interpretation is sequentiality. All verbs but the final one appear in the gerund form. The final verb may be perfect, compound imperfect, jussive, or imperative. All of the gerund forms agree with the subject of the final verb. Example: እቤት ተመልሶ እራት በልቶ ተኛ - “He returned home, ate dinner, and went to bed” lit. “Returning (3 pers.sing.masc.) home, eating (3 pers.sing.masc.) dinner, he went to bed”.

2.5.4 Challenges to Amharic SRL

The most prominent phenomenon of the Amharic languages is the reliance on root-and-pattern paradigms for word formation. The standard account of word formation processes in the language [48] describes words as combinations of two morphemes: a root and a pattern. The root consists of consonants only, by default three, called radicals. The pattern is a combination of vowels and, possibly, consonants too, with ‘slots’ into which the root consonants can be inserted. Words are created by interdigitating roots into patterns: the consonants of the root fill the slots of the pattern, by default in linear order. Other morphological processes in the languages involve affixation and cliticization. Root-and-pattern morphology, a non-concatenative mechanism

unique to Semitic languages, is one of the main challenges for computational processing [47]. Amharic poses a significant challenge to SRL for several reasons [47]. From a syntactic perspective, the following facts are relevant for SRL.

Pro-drop: Amharic tend to drop the subject and mark it morphologically on the verb. For example, ብርቱካን በሉ - '[they] ate_they orange(s)' is pro-dropped where the pronoun 'they' is absent. The explicit rendering of the sentence would be as follows እነሱ ብርቱካን በሉ - 'they ate_they orange(s)'. This is a significant phenomenon for SRL since one of the arguments is missing, and therefore, the system has to either be able to make it explicit or mark a trace with the argument class as well as identify the morpheme marker on the predicate.

Relative free word order: Amharic allows for Verb Subject Object (VSO), Object Verb Subject (OVS), SVO, etc. Example: በላ እዮብ እራቱን - እራቱን በላ እዮብ - እዮብ እራቱን በላ - እዮብ በላ እራቱን - all to mean 'Eyob ate his dinner'. This poses a significant challenge since it has implications in identifying the arguments and their roles, especially when the arguments are proper nouns with no explicit grammatical case marking due to diacritic underspecification especially in the singular cases.

Possessive constructs: Possessive constructs are especially tricky in Amharic as they allow for multiple recursive embeddings as well as allowing for multiple modifiers. These idafa constructs are quite difficult to parse and hence pose a significant challenge to boundary detection for the argument spans. Example: ያለቃዬ ሚስት አጎት የልጅ ልጁ ጓደኟ ጠፋች - my boss's wife's uncle's granddaughter's friend disappeared (lit.) .

These syntactic phenomena play a crucial role in SRL systems. The complexity is magnified with the absence of functional diacritics that mark grammatical case, mood, passivization, definiteness, and agreement. Accordingly, these syntactic characteristics are coupled with the challenge of vowel underspecification to yield a significant obstruction to efficient SRL for the language. SRL requires that there exist quite robust tools for syntactic parsing and vowel restoration for such languages as precursors to SRL [47].

CHAPTER THREE: RELATED WORK

3.1 Introduction

There are several successful SRL models devised for different languages using feature-based and neural-based approaches [1]. Feauture-based approaches to SRL are implemented with the memory-based and statistical machine learning techniques [1]. The neural-based approaches to SRL are implemented using deep learning. In this chapter, we review research works on SRL systems using feature-based and neural-based approaches.

3.2 Feature-based Semantic Role Labeling Systems

The task of semantic role labeling was pioneered by Gildea and Jurafsky [32]. In their work in [4], features based on syntactic constituent trees were identified as most valuable for labeling predicate-argument relationships. The system is based on the *probability distribution* statistical classifier trained on sentences that were hand-annotated with semantic roles by the FrameNet semantic labeling project. Each training sentence is then parsed into a syntactic tree and various lexical and syntactic features were extracted, including the phrase type of each constituent, its grammatical function, and position in the sentence. These features were combined with knowledge of the predicate verb, noun, or adjective, as well as information such as the prior probabilities of various combinations of semantic roles. They used various lexical clustering algorithms to generalize across possible fillers of roles. Test sentences were parsed, annotated with these features, and were then passed through the classifiers. The probability estimation then determines how likely a constituent is to fill each possible role, given the features and the predicate, or target word. Lexical statistics computed on constituent head word were found to be the most important of the features used. The system achieved 82% accuracy in identifying the semantic role of pre-segmented constituents. At the more difficult task of simultaneously segmenting constituents and identifying their semantic role, the system achieved F-Score of 62.9% tested on FrameNet database.

Xue's work [17] takes advantage of availability of the Chinese Proposition Bank and Nombank to develop an SRL system for the Chinese language. The system follows four stages: pruning,

argument detection, argument classification and re-ranking. It accepts a parse tree as an input, both hand-crafted parses and an automatic parser, and then runs argument detection and classification using *maximum entropy* classifier. The classifier is tuned to minimize over-fitting. The output of the argument classification stage, arguments and adjuncts of the same predicate instance (proposition), were chained together with their joint probability and the top K propositions were selected as the re-ranking candidates using a perceptron re-ranker. Experiments were conducted using semantic role labels mapped to hand-crafted parses in the Chinese Treebank, and using a fully automatic Chinese parser that integrates word segmentation, POS-tagging and parsing. Results show that when hand-crafted parses are used, SRL accuracy for Chinese is comparable (F1-Score of 91.3%) to what has been reported for the state-of-the-art SRL systems trained and tested on the English Proposition Bank, even though the Chinese Proposition Bank is significantly smaller in size. When an automatic parser is used, however, the accuracy of the system is significantly lower than the English state-of-the-art (F1-Score of 61.3%).

An SRL system for Modern Standard Arabic that exploits many aspects of the rich morphological features of the language is presented in the work of Diab et al., [18]. It hypothesizes that taking advantage of the interaction between morphology and syntax can improve on a basic SRL system for morphologically rich languages. Kernel methods were used for feature engineering to save time in the design and implementation of features, and rich structured features are combined with traditional attribute-value features derived from English SRL systems. The system is based on a supervised model, and uses *support vector machines* (SVM) algorithm to handle noisy data and large feature sets well. The SVM is used to implement a two step classification approach: argument boundary detection and argument classification. The system is trained and tested using the pilot Arabic Propbank data released as part of the SemEval-2007 data. Given the lack of a reliable Arabic deep syntactic parser, they have used gold standard trees for the Arabic Tree Bank (ATB), and experiments show a global SRL F-score of 82.17%, which significantly improved previously reported results on the same task and dataset.

Abkik and Li used instance-based learning in [20] for the development of an English SRL that performs no explicit generalization but rather extrapolates predictions from the most similar instances in the training data. The authors argue that low-frequency examples in SRL are often

not noise to be abstracted away, but rather correspond to exceptions that require explicit handling. To overcome the large number of low-frequency exceptions in training data which are highly context-specific and are difficult to generalize, they proposed instance-based learning arguing that such learning does not abstract away from specific feature contexts, but rather considers the overall similarity of a test instance to instances in the training data. They used memory-based learning for SRL as a sequence of two classification tasks: joint predicate identification and classification (predicate labeling), followed by joint argument identification and classification (argument labeling). They used the composite features to identify nearest neighbors, that is, instances that share the most similar combination of atomic features, and used a function to assign to each composite of atomic features a discrete distance value, effectively rendering the search for nearest neighbors. A variant of k-nearest neighbors (kNN) classification, called *nearest k-window*, is used with composite features to identify nearest neighbors for SRL. The experiments conducted indicate that high-quality predictions can be derived from a very small number of similar instances. Comparative evaluations experimentally demonstrated that the approach significantly out performs prior state-of-the-art SRL systems for verbal predicates and their roles on both in-domain and out-of-domain data, reaching F1-Score of 89.28% and 79.91%, respectively, using the scoring metric of the CoNLL-2009 shared task.

In Eskedar Yirga's work [19], a seminal effort has been made in developing an automatic SRL for Amharic text using memory-based learning (MBL). The system consists of three components: text pre-processing, training, and labeling. Each word was POS (Part-of-Speech) tagged and the input text was parsed with phrase structure schema using an X-bar theory, in the text-pre-processing task. The system was then trained on a text where segments of texts were already correctly labeled. Finally, it attempts to automatically predict and assign the correct label for every given segment of the text using the information available in the text and the knowledge acquired in training. Overlap and *modified value difference metrics* (MVDM) is used as the classifier algorithm. 240 manually annotated and simple sentences were considered, and PropBank and TiMBLE tools were used to implement the system. Evaluation result showed an F-Score of 79.77% with optimized parameter settings.

3.3 Neural-based Semantic Role Labeling Systems

Early researches on SRL systems were based on syntactic analysis of sentences [5]. Normally, for SRL they estimate a parse tree of a sentence and extract hand-designed features from its syntactic constituents, that is, word and phrase nodes, in the parse tree [14]. These features are then fed into a classifier and trained on a corpus annotated with semantic roles. One drawback of these approaches is that they require hand-designed feature engineering and some level of domain knowledge [13, 14]. Another drawback is that these systems rely on syntactic parsers to solving the SRL problem which is proven to be the major source of the incorrect predictions [1, 5]. SRL researches using deep learning are able to bypass the traditional steps for extracting the intermediate NLP features such as part-of-speech (POS) and syntactic parsing, and avoid hand-crafting the feature templates [22].

Zhou and Xu's work used a deep bi-directional recurrent neural network (RNN) as an end-to-end system for an English SRL in [23]. The system takes only original text information as input feature, without using any syntactic knowledge and avoiding human engineering the feature templates. The work utilizes the bi-directional information by processing the sequence in forward direction with a standard LSTM, first, and taking the output of this LSTM layer by the next LSTM layer as input processed in reverse direction. These two standard LSTM layers are composed to a pair of LSTM. Then they stack LSTM layers pair after pair to obtain the deep LSTM model. The input features are processed by the 8 layers of LSTM bi-directionally with conditional random field (CRF) model put on top for tag sequence prediction. Experimental analysis on the system shows that the model used is better at handling longer sentences than statistical and memory-based models. The latent variables of the model can also implicitly capture the syntactic structure of a sentence. The model achieved an F-score of 81.07% on CoNLL-2005 shared task and 81.27% on CoNLL-2012 shared task, both outperforming the previous systems based on parsing results and feature engineering, which heavily rely on the linguistic knowledge from expert.

A Chinese SRL is presented in the work of Wang et al. [25] using deep learning. Prior Chinese SRL researches heavily relied on feature engineering and fail to model the long range dependencies in a sentence. This work, however, uses bi-directional recurrent networks with long-short-term-memory to capture both bi-directional and long range dependencies in a

sentence with minimum feature engineering. They have used the current word, the current POS tag, the predicate, left and right words, left and right POS tags and the distance to the predicate as the features embedded and concatenated to get the word representation feature vector in their work. The dependencies in a sentence were captured from both directions with bi-directional RRN, and long-range dependencies were well modeled with LSTM architecture. Given a sentence, their model architecture first gets representation for each word to be labeled. A bidirectional RNN-LSTM layer is then used to combine the local information of a word and its contextual information from both directions. After a linear transformation, a score corresponding to a kind of semantic role label is produced for each word to be labeled. Compared to previous works on Chinese SRL, the system achieved a significant improvement with experiments on Chinese PropBank (CPB) resulting to an F-1Score of 77.09%.

A novel deep architecture for an English SRL based on stacked attentive representations is presented in the work of Park [24]. The model is based on the self-attention mechanism for sentence representation, arguing that it can derive the intra-connections among every word within a sentence, and, therefore, is a proper method to capture the associations for SRL. The recurrent updates in RNNs, which require long training time over a large data set, were also removed by providing selective connections among attentive representations. By stacking the self-attentions with the selective connection module, the model can capture the hierarchical information over the attentive representations - which are built on the self-attention modules, while training. The accuracy gains were improved by capturing the hierarchical information using the connection module, and the model can be parallelized to avoid the repetitive updates in RNNs. As a result, experimental results on the model, has reduced the training time by 62% from the baseline, and that the model performs better in accuracy compared to the state-of-the-art prior studies. The model achieved an F-score of 86.6% and 83.6% on the CoNLL-2005 and CoNLL-2012 shared tasks, respectively.

3.4 Summary

There are several SRL systems developed using different approaches for different languages. In the previous sections, we have reviewed some of the different approaches on SRL systems for the English, Chinese, Arabic and Amharic languages. Table 3.1 summarizes our review.

Table 3.1: Summary of Related Work on SRL Systems

Language	Author	Approach	Technique used to Extract Features	Classifier Model	Experimental Results	Remark from Experimental Analysis
English	Jurafsky & Gildea [32]	Statistical Machine Learning	Input sentence is parsed and various lexical and syntactic features were extracted	Probability Estimation	F-score of 62.9%, tested on the FrameNet database	Lexical statistics computed on constituent head word were found to be most important of the features used.
	Abkik & Li [20]	Memory-based Leaning	Input sentence is parsed and features such as headword, voice, lemma and path from constituent to predicate were extracted	k-nearest Neighbor	F-Score of 89.28% for in-domain and 79.91% for out-of-domain data, tested on CoNLL-2009	High quality prediction from very small number of similar instances.
	Zhou & Xu [23]	Deep Learning	Input is Raw text, and features were extracted (learned) using neural word embedding	Bi-RNN with LSTM and CRF	F-Score of 81.07% on CoNLL-2005 and 81.27% on CoNLL-2012	Latent variables of the model can capture the syntax, and the Model is better at handling longer sentences than statistical and MBL.
	Park [24]	Deep Learning	Input is Raw text, and features were extracted (learned) using neural word embedding	Self-attention Mechanism	F-Score of 86.6% on the CoNLL-2005 and 83.6% CoNLL-2012	Improved accuracy by capturing Hierarchical information, and Recurrent updates were removed by providing selective connections among attentive representations.
Chinese	Xue [17]	Statistical Machine Learning	Input sentence is parsed using hand-crafted and automatic parsers, different features were extracted	Maximum Entropy	F-Score of 61.3% automatic, & 91.3% for hand-crafted parsers, on CPB	Increased accuracy when hand-crafted parsers are used, but significantly lower when using automatic parsers.
	Wang et al. [25]	Deep Learning	Input is sentences with minimum feature engineering, the features were then embedded	Bi-RNN with LSTM	F-Score of 77.09% tested on Chinese PropBank	Bi-LSTM can capture bi-directional and Long-range dependencies.
Arabic	Diab et al. [18]	Statistical Machine Learning	Sentence was parsed and kernel methods were used to extract features	Support-Vector-Machine (SVM)	F-Score of 82.17%, tested on Arabic PropBank	Hypothesizes that the interaction between Morphology and Syntax can improve SRL performance for morphologically rich languages.
Amharic	Eskedar Nega [19]	Memory-based Leaning	POS tagged each word, and parsed input text with phrase structure schema to extract different features	Overlap & MVDM	F-Score of 79.77%, tested on 240 manually annotated sentences	Integrating the classifier with different parameter settings resulted in increased performance.

Feature-based SRL systems require heavy feature engineering, and manually designed features are often over-specified, incomplete, and take a long time to design and validate. Feature-based approaches to SRL rely on the syntactic information [13, 14] in identifying the argument, and use a full parse tree information, or a shallow parsing information, which includes part-of-speech tags, chunks and clauses, is required as input in such an SRL system. However, using syntactic

parsers have been proven to be a majour source of incorrect predictions in such systems. Moreover, optimization and regularization methods used in feature-based SRL systems are difficult [12, 13, 19]. In contrast, SRL systems using Deep Learning can easily learn important features or representations from just the raw input – avoiding the use of syntactic parsers and the need to manually design and extract features [22]. SRL systems using Deep Learning can, therefore, bypass the intermediate feature extraction task [23]. Argument identification and Argument boundary identification tasks can be modeled in Deep Learning using a Bi-LSTM network [25] – which has been proven to be better in handling longer sentences than the statistical and memory-based learning techniques in feature-based approach to the SRL system [23]. In addition, Deep Learning provides an effective end-to-end joint system learning, and better regularization and optimization methods [15, 16], which are advantageous to the SRL problem.

Amharic is a morphologically-rich language, making the feature extraction task required in the feature-based SRL for the language very complex and prone to errors because it requires morphological and syntactic parsers [19], and errors encountered in the parsers can propagate to the SRL prediction task. Moreover, feature-based SRL system often use huge collection lexical resources annotated by professionals - to avoid errors in parses, and Amharic can be considered as a less-resourced language with regards to the availability of semantically annotated lexical resources by professionals. A neural-based SRL for the language can bypass the feature-extraction task by learning the feature representation present in natural language sentences from just the raw input, and also avoid the possible errors in parsing that can possibly propagate to the SRL prediction. For example, the semantic similarity of words embeddings can be represented with word embeddings, the morphological information of words using character embeddings, and the syntactic structure using a Bi-LSTM network. Therefore, this study will fill the gap and address the issue of semantic role labeling for Amharic text using Deep Learning.

CHAPTER FOUR: DESIGN FOR THE AMHARIC SRL

4.1 Introduction

In this chapter, we discuss about the overall design of the proposed semantic role labeler for Amharic text. First, we illustrate the generalized model of the proposed system, and then we describe each component. The parameter settings, training, optimization and predictions of the model are also discussed.

4.2 The Model for the Proposed Amharic SRL

The generalized model of our proposed solution for the semantic role labeler for Amharic text using deep learning is presented in Figure 4.1. The model is adapted from the work of Zhou and Xu [23], with the difference in the approach to the word representation component. In this component we have used concatenated word representations to capture the morphological, syntactic and semantic information of words in Amharic sentences, which can be considered as a different approach to the task.

Semantic role labeling is considered as a task of sequence labeling, which assigns a label for each word in the sequence. To identify the boundary information of semantic roles, we have adopted the IOB tagging schema for the labels. For sequence labeling, especially for the problem of SRL, it is important to capture dependencies in the sequence. This is because the semantic role label for a word not only relies on its local information, but also is determined by long-range dependencies from other words surrounding the word [1]. We have used RNN for its ability to capture the contextual information [1], which is crucial to the SRL task. Moreover, we have enriched the basic RNN model with bidirectional LSTM RNN to model both bidirectional (for argument identification) and long range dependencies (for argument boundary detection) of the words in a sentence [23, 25].

Given a sentence along with its semantic role labels in IOB, we first apply preprocessing task to extract the text from its associated labels, remove unwanted characters, tokenize both, encode and finally pad them to have a fixed length. Next we get representation for each word to be

labeled. Then a bidirectional LSTM RNN layer is designed to combine the local information of a word and its contextual information from both directions. With a fully connected layer to form more complex features, finally, a linear output layer, which is a CRF layer, on top of the network is deployed for final prediction to yield a probability distribution over possible output labels. With this model, an output vector whose each dimension is a score corresponding to semantic role label is given for each word to be labeled.

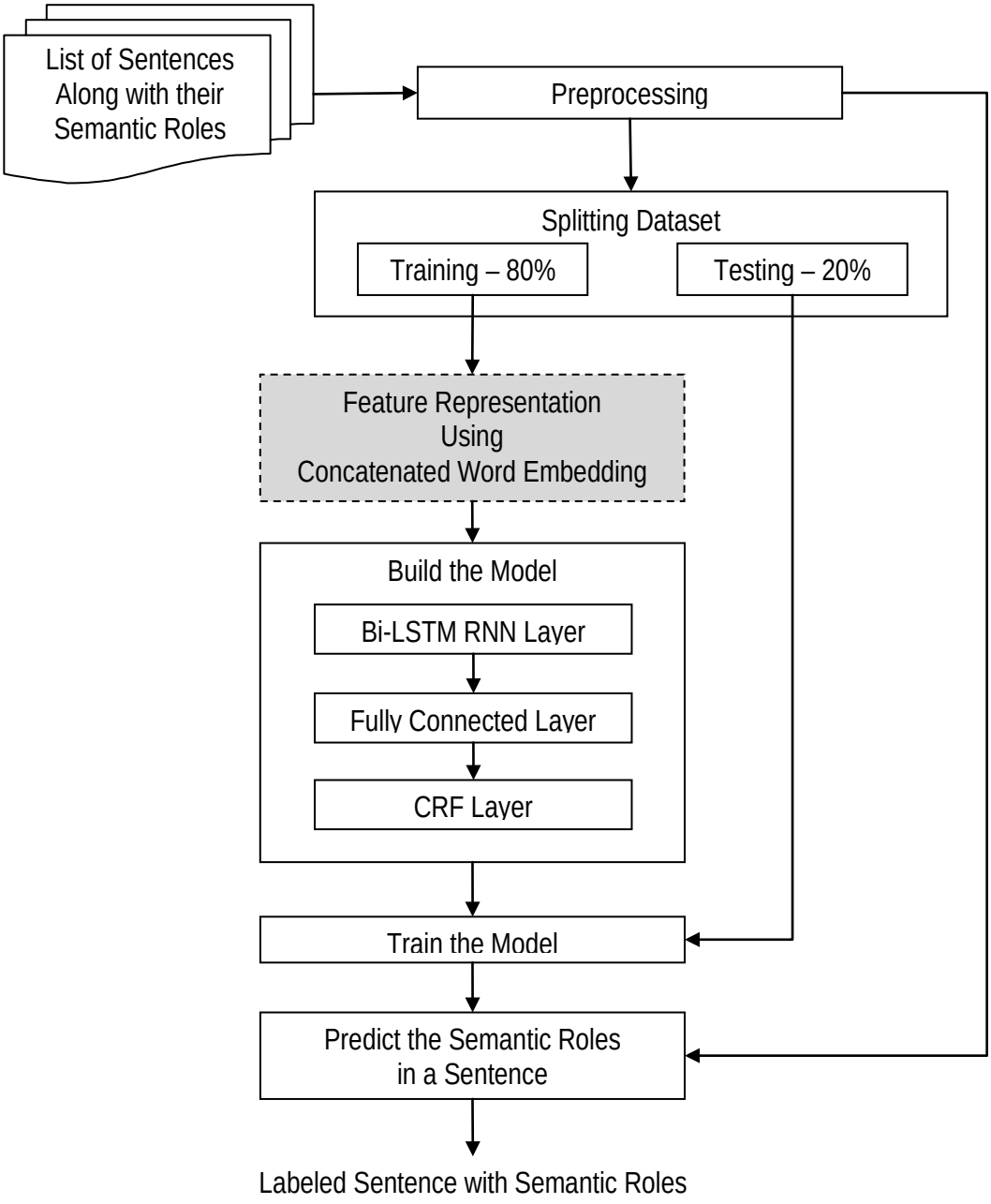


Figure 4.1: General Model of the Proposed Solution

4.2.1 Sentence

List of sentences annotated with a tab-separated semantic roles are provided to the model in a text file for training. We have used a whitespace as a word separator in a sentence. The annotated roles follow PropBank semantic roles in IOB tagging schema.

4.2.2 Preprocessing

Text preprocessing is often the first step in the pipeline of a Natural Language Processing system, with potential impact in its final performance. It transforms the text into a more digestible form so that the machine learning algorithms can perform better. The quality of the text plays an important role in the performance of the model. If the input text is not preprocessed, the model may behave unexpectedly due to inconsistent data, and performance may be affected.

The preprocessing component of our model takes the input sentences and applies cleaning, normalization, tokenization, encoding and padding sub-tasks to each sentence, as depicted in Figure 4.2.

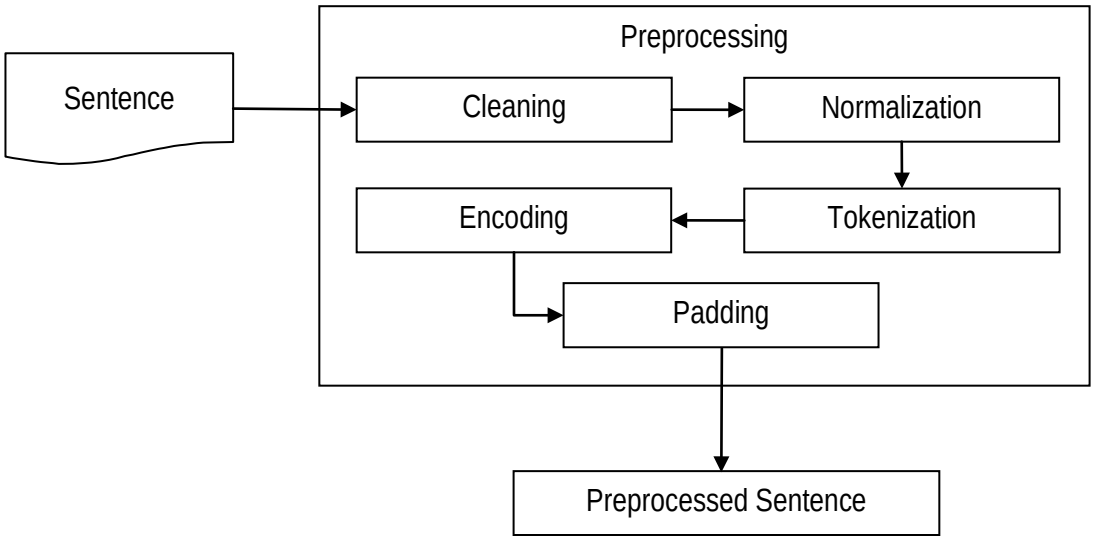


Figure 4.2: Subtasks of the Preprocessing Component

Cleaning

Cleaning consists of getting rid of the less useful parts of text through stop-word removal, dealing with capitalization, special characters and other details. For Amharic text, however,

capitalization and stop-words will not be an applicable. We have used this sub-task to strictly remove non-Geez characters and all punctuations from the text, except for the Amharic full-stop. Arabic numerals are not removed from the text as they are used in Amharic sentences.

Normalization

Text normalization is the process of transforming a text into a canonical (standard) form. Text normalization puts all words on equal footing, and allows processing to proceed uniformly. Normalizing text can mean performing a number of tasks, but for our case we have limited it to separating hyphenated words, correcting misspelled words, and removing double white-spaces between words or character repetitions (noise) having no Amharic meaning. For example, we have represented "ሥነ-ጥበብ" as two different words “ሥነ” and “ጥበብ”, separated with a single white space. In addition, we have separated the Amharic full-stop character (።) from the last word in the sentence attached to it with a single whitespace.

Tokenization

Tokenization is the process of splitting strings of text into smaller pieces, or “tokens”. Larger chunks of text can be tokenized into sentences, and sentences can be tokenized into words, etc. In our case, we have tokenized the sentences into words. We have used the whitespace between words to separate the words. We have also treated the Amharic full-stop character (።) as a separate “token”.

Encoding

Unlike humans, computers cannot understand raw text. Raw text needs to be encoded to numeric values. Text encoding is the process of converting text into numerical representation. In this text preprocessing sub-task of our model, we have transformed the words in a sentence into word, character and *fastText*-level encodings, and the IOB tags of the semantic role labels into one-hot encoding.

Word-level Encoding: List of unique words, numerals and special characters used in Amharic sentences, are sorted and indexed with integer numbers, starting from zero, to constitute the word vocabulary. The index number 0 is reserved for the padding. The index number 1 is also reserved for words that are not in the vocabulary - considered as *Unknown*. Every word in a sentence is represented by its integer index in the words vocabulary. In this regard, the sentence “አበራ

ትምህርት ጨረሰ።” has the word-level encoding [[9388], [7716], [20040], [20560]], each number representing index of the word in the words vocabulary. The procedure for the word-level encoding is given in Algorithm 4.1.

Algorithm 4.1: Word-level Encoding of a Sentence

Input: Tokenized sentence list and indexed words vocabulary
Output: List of sentences encoded with word-level encoding

```
Function WORD_LEVEL_ENCODING(tokenized_Sentences_List,indexed_Words_Vocabulary)
    sentence_List ← INITIALIZE(sentence_List)
    for each sentence in tokenized_Sentences_List do
        words_List ← INITIALIZE(words_List)
        for each word in sentence do
            if word is in indexed_Words_Vocabulary do
                index ← GET_INDEX(word, indexed_Words_Vocabulary)
            else
                index ← GET_INDEX('Unknown', indexed_Words_Vocabulary)
            words_List ← APPEND(index)
        sentence_List ← APPEND(words_List)
    return sentence_List
```

Character-level Encoding: All the unique Geez characters, including punctuations and numerals, and Arabic numerals are indexed with integer numbers starting from zero to constitute the character vocabulary. The index number 0 is reserved for the padding, and the index number 1 is reserved for characters that are not in the vocabulary - considered as Unknown. Every character in a word is represented by its integer index in the characters vocabulary, and every word in a sentence is represented by its character-level encoding. Therefore, the sentence “አበራ ትምህርት ጨረሰ።” has the character-level encoding of [[137, 81, 45], [102, 31, 8, 47, 102], [246, 42, 50], [328]], each number representing index of the character in the characters vocabulary, and each series of characters representing the word in the sentence. The procedure for the character-level encoding is given in Algorithm 4.2.

FastText-level Encoding: Words in the sentences are encoded using the fastText embedding. Although the fastText is a dense vector model and can represent word information in a sentence using character n-grams along with window-size, we are using this as a fist-level representation of the words in sentences, or *encoding*, because we want to capture the morphological representation along with the syntactic interaction of the words in our embedding component. We have used *fastText-level* encoding for the words from a tokenized sentence passed to as a

first-level representation and capture the morphological structures of the words through character n-grams. The procedure for the *fastText*-level encoding is given in Algorithm 4.3.

Algorithm 4.2: Character-level Encoding of the Words in a Sentence

Input: Tokenized sentence list and indexed characters vocabulary
Output: List of sentences encoded with character-level encoding

```
Function CHARACTER_LEVEL_ENCODING(tokenized_Sentences_List,indexed_Characters_Vocabulary)
    sentence_List ← INITIALIZE(sentence_List)
    for each sentence in tokenized_Sentences_List do
        words_List ← INITIALIZE(words_List)
        for each word in sentence do
            characters _List ← INITIALIZE(characters _List)
            for each character in word do
                if character is in indexed_Characters_Vocabulary do
                    index ← GET_INDEX(character, indexed_Characters_Vocabulary)
                else
                    index ← GET_INDEX('Unknown', indexed_Characters_Vocabulary)
                characters_List ← APPEND(index)
            words_List ← APPEND(characters_List)
        sentence_List ← APPEND(words_List)
    return sentence_List
```

Algorithm 4.3: FastText-level Encoding of the Words in a Sentence

Input: Tokenized sentence list, vector size, window size, minimum word count, minimum char n-gram and maximum character n-gram
Output: List of sentences encoded with fastText-level encoding

```
Function FASTTEXT_LEVEL_ENCODING(tokenized_Sentences_List, vector_Size, window_Size,
                                 minimum_Word_Count, minimum_Char_n_Gram,
                                 maximum_Char_n_Gram)
    fastText_Embedding ← FastText(vector_Size, window_Size, minimum_Word_Count,
                                  minimum_Char_n_Gram, maximum_Char_n_Gram)
    fastText_Embedding ← BUILD_VOCABULARY(tokenized_Sentences_List)
    fastText_Embedding ← TRAIN(tokenized_Sentences_List, tokenized_Sentences_List_Length,
                               epochs)
    encoded_Sentence_List ← INITIALIZE(encoded_Sentence_List)
    for each sentence in tokenized_Sentences_List do
        encoded_Sentence ← INITIALIZE(encoded_Sentence)
        for each word in sentence do
            encoded_Word ← INITIALIZE(encoded_Word)
            encoded_Word ← Get_Embedding(word, fastText_Embedding)
            encoded_Sentence ← APPEND(encoded_Word)
        encoded_Sentence_List ← APPEND(encoded_Sentence)
    return encoded_Sentence_List
```

One-hot Encoding: For categorical variables where no ordinal relationship exists, integer encoding allows the model to assume a natural ordering between categories, resulting in poor performance. One-hot encoding is a process by which categorical variables are converted into a form that could be provided to machine learning algorithms to do a better job in prediction. We have used one-hot encoding for the semantic role labels. The labels are first encoded in integer (indexed) and then converted to binary (one-hot) encoding. The index number 0 is also reserved for the padding. With this encoding, for the sentence "አበራ ትምህርት ጨረሰ።", having a tag sequence of "B-A0 B-A1 B-V O", the one-hot encoding is shown below.

[[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], ⟶ B-A0
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], ⟶ B-A1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], ⟶ B-V
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]. ⟶ O

The procedure for encoding the tag series into one-hot encoding is presented in Algorithm 4.3.

Algorithm 4.4: One-Hot-Encoding of the Semantic Roles of the Words in a Sentence

Input: Tokenized tags list and indexed tags vocabulary
Output: List of tags encoded with one-hot encoding

```
Function ONE_HOT_ENCODING(tokenized_Tags_List, indexed_Tags_Vocabulary)
        tags_In_ONE_HOT_MATRIX ← TAG_INDEX_TO_BINARY(indexed_Tags_Vocabulary)
        tags_List ← INITIALIZE(tags_List)
        for each tags in tokenized_Tags_List do
                tag_List ← INITIALIZE(tag_List)
                for each tag in tags do
                        index ← GET_INDEX(TAG, tags_In_ONE_HOT_MATRIX)
                        tag_List ← APPEND(index)
                tags_List ← APPEND(index)
        return tags_List
```

Padding

In a real-world sequential machine learning task, the entries of a dataset can be of arbitrary length. In our case, the number of tokens in a sentence varies between sentences. Since our model implementation can only take in sentence entries of a fixed length, we need to either cut up or fill up each sentence in the dataset with some dummy element to an identical length. This preprocessing procedure is often named as 'padding'. In this work, we adjust all of the sentences

to the length of the longest sentence (the sentence with the largest number of tokens). Sentences that are shorter than this length will be filled with zero(s) in the gap. After encoding the sentences with word, character and fastText-level encodings, and their corresponding semantic role label tags with one-hot encodings, we have padded the sentences to have the same length that corresponds to the sentence with the maximum number of words by filling the gap with zero(s).

4.2.3 Concatenated Word Embedding

Experimental analysis from the work of Diab et al., [18] shows that taking advantage of the interaction between morphology and syntax can improve the performance of a basic SRL system for morphologically rich languages. The work of Zhou and Xu [23] reports that latent variables of the Bi-LSTM RNN network can capture the syntactic structure of the words in a sentence. Wang et al., [25] use a Bi-LSTM RNN network to capture the bi-directional and long-range dependencies of the words in a sentence, which is important to the SRL task.

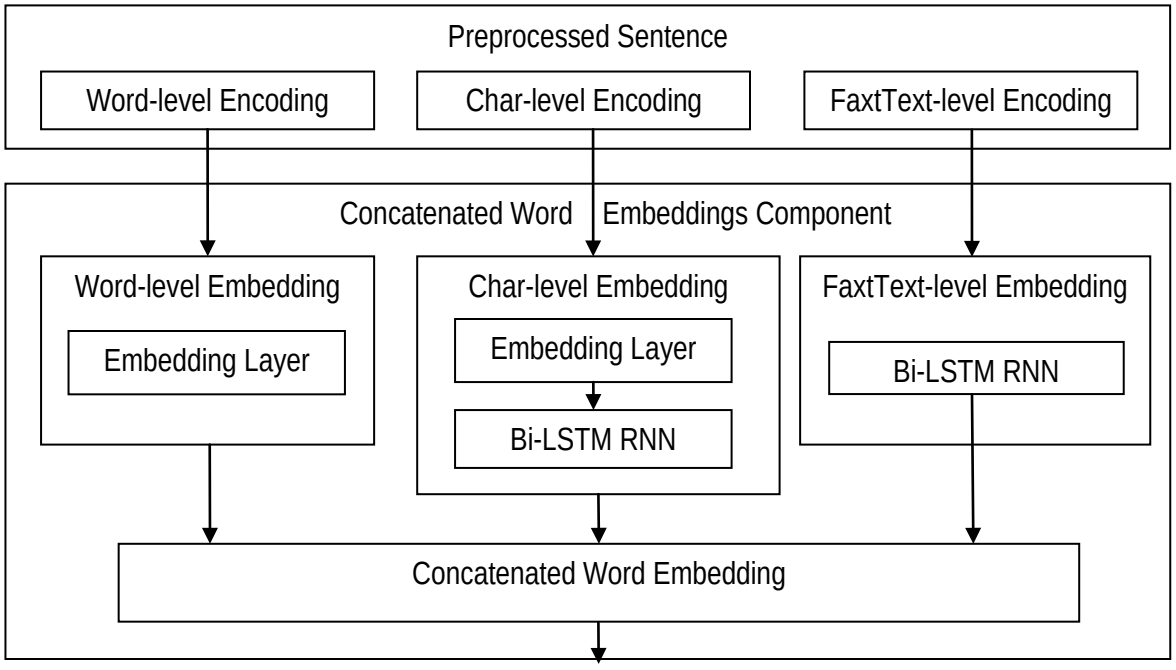


Figure 4.3: Subtasks of the Concatenated Word Embeddings Component

Amharic is a morphologically rich language, and SRL systems for the language need to capture the morphological, syntactic and semantic information of the words in a sentence. Amharic,

therefore, requires a character-based embedding to better capture the morphological information of words, which is crucial in the SRL task for the language. The morphological and semantic information of words can be modeled with character and word-level neural embeddings, respectively, and the syntactic and context information of a word can be modeled with a Bi-LSTM RNN network. Moreover, the interaction between the morphology and syntax of a word can be captured by passing the character-level embeddings thorough a Bi-LSTM network.

In this work, we have used a concatenation of word, character, and *fastText*-level neural embeddings to capture the morphological, syntactic and semantic information of the words in sentences. Figure 4.3 shows in detail the neural word embeddings used in this work. The semantic information of words can be captured with the word-level embeddings, and the morphological information of a word can be captured with character-level embedding supplemented with fastText embedding to capture the context to some degree. By feeding the output of the character and *fastText*-level neural embeddings to a Bi-LSTM RNN network, each, we can model the syntactic representations of the words in sentences and also the relationship between the morphology and syntax of a word.

Word-level Embedding

In this sub-task of our model, preprocessed sentences encoded in word-level are passed through a standard word embedding layer. Word-level embeddings are represented by column vectors in an embedding matrix $W^{\text{wrd}} \in \mathbb{R}^{\text{d}^{\text{wrd}} \times |V^{\text{wrd}}|}$. Each column $W_i^{wrd} \in \mathbb{R}^{\text{d}^{\text{wrd}}}$ corresponds to the word-level embedding of the i^{th} word in the words vocabulary. A word w is transformed into its word-level embedding r^{wrd} by using the matrix-vector product:

$$r^{\text{wrd}} = W^{\text{wrd}} v^{\text{wrd}} \tag{32}$$

where v^{wrd} is a vector of size of the words-vocabulary $|V^{\text{wrd}}|$ which has value 1 at index w and zero in all other positions. The matrix W^{wrd} is a parameter to be learned, and the dimension of the word-level embedding d^{wrd} is a hyper-parameter to be tuned. Algorithm 4.5 shows the procedure for the word-level embedding in our model.

For a sentence with a word sequence $\{W_1, W_2, \ldots, W_N\}$, the word-level embedding of the sentence, S_w , is given as:

$$S_w = \{r_1^{wrd}, r_2^{wrd}, \dots, r_N^{wrd}\} \quad (33)$$

Algorithm 4.5: Word-Level Embedding of the Words in a Sentence

Input: Word-level encoding, words vocabulary size, embedding vector size and maximum words in a sentence
Output: List of sentences embedded with word-level embedding

```
Function WORD_LEVEL_EMBEDDING(word_Level_Encoding, words_Vocabulary_Size,
                              embedding_Vector_Size, maximum_Words_In_A_Sentence)
    weigth_Matrix ← INITIALIZE(words_Vocabulary_Size, embedding_Vector_Size)
    Embedded_Sentence_List ← INITIALIZE()
    for each sentence in word_Level_Encoding do
        Embedded_Sentence ← INITIALIZE()
        for each word in sentence do
            embedded_Word ← EMBED(weigth_Matrix[word])
            Embedded_Sentence ← APPEND(embedded_Word)
        weigth_Matrix ← UPDATE(weigth_Matrix)
        Embedded_Sentence_List ← APPEND(Embedded_Sentence)
    return Embedded_Sentence_List
```

Here, for the sentence "ያዕቆብ ዱላዉን ለዮናስ አቀበለው።", encoded in word-level encoding as [[474452], [2], [22959], [241215], [525169]], the word-level embedding of the words in the sentence represented in a 5-dimension vector is given as follows.

[[-0.117, -0.039, -0.077, 0.0477, 0.031], ⟶ ያዕቆብ
[-0.039, 0.005, -0.036, -0.014, -0.042], ⟶ ዱላዉን
[0.001, -0.039, -0.044, 0.033, 0.043], ⟶ ለዮናስ
[-0.030, -0.008, -0.032, 0.023, 0.032], ⟶ አቀበለው
[-0.134, -0.233, -0.294, 0.023, 0.109]] ⟶ ።

With the word-level embedding we can be able to capture the semantic similarity of words, such as "ሠው" and "ሰው" (see ANNEX E).

Character-level Embedding

Word-level embeddings can only handle words observed in the vocabulary. It deals with out-of-vocabulary (OOV) words by simply assigning them some random vector values which, if not remedied, would end up confusing our model. Having a character embedding, every single word's vector can be formed even it is out-of-vocabulary word. Another benefit is that it good fits for misspelled words, and handles infrequent words better than word embeddings. But, most

importantly, intra-word information - information about word morphology and shape, which is extremely useful especially when dealing with morphologically rich languages - like Amharic, can also be learned (represented) in character-level embeddings.

Here, preprocessed sentences encoded in character-level encoding are passed through a standard time-distributed embedding layer, which are then passed to a time-distributed Bi-LSTM RNN layer. This way the embedding can represent not only intra-word information, but also its context in a sentence by capturing the morphological and syntactic interaction of the word. Given a word w composed of M characters $\{C_1, C_2, \dots, C_M\}$, we first transform each character C into a character embedding r^{chr}. Character embeddings are encoded by column vectors in the embedding matrix $W^{\text{chr}} \in \mathbb{R}^{\text{d}^{\text{chr}} \times |V^{\text{chr}}|}$. For a character C, its embedding r^{chr} is obtained by the matrix-vector product:

$$r^{chr} = W^{chr} v^{chr} \tag{34}$$

where v^{chr} is a vector of size $|V^{chr}|$ which has value 1 at index C and zero in all other positions. The matrix W^{chr} is a parameter to be learned, and the dimension of the character-level embedding d^{chr} is a hyper-parameter to be tuned.

For the sentence "ያዕቆብ ዱላዉን ለዮናስ አቀበለው።", encoded with character-level encoding in a 15 characters in a word with padding is shown below.

[[193, 172, 72, 86, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], ⟶ ያዕቆብ
[199, 13, 161, 126, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], ⟶ ዱላዉን
[10, 196, 124, 55, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], ⟶ ለዮናስ
[137, 66, 81, 10, 165, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], ⟶ አቀበለው
[328, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]] ⟶ ።

The character-level-embeddings for the character "ያ" in a 20-dimension vector is shown below.

[0.016, -0.0436, -0.018, 0.1188, -0.131, 0.036, 0.045, -0.127, -0.011, 0.039, -0.111, 0.066, 0.0167, -0.209, 0.080, -0.079, 0.090, 0.234, 0.043, -0.190]

The character-level-embeddings for the character "ዮ" in a 20-dimension vector is shown below.

[0.030, 0.015, 0.025, 0.084, 0.024, 0.032, 0.112, 0.022, -0.004, 0.070, 0.048, -0.113, 0.097, 0.033, 0.072, 0.006, -0.031, -0.204, -0.047, -0.052]

The character-level embedding for the sentence "ያዕቆብ ዱላዉን ለዮናስ አቀበለው።" before passing through a Bi-LSTM layer is shown below.

```
[[[0.016, -0.043, -0.018, ..., 0.234, 0.043, -0.190],
  [-0.001, 0.087, -0.083,..., -0.188, -0.289,  0.066],
  [0.010, 0.251, 0.171, ..., 0.016, 0.003, -0.047],
                                                ...,
  [0.036, -0.024, 0.003, ..., -0.006, 0.049,  0.036],
  [0.036, -0.024, 0.003, ..., -0.006, 0.049,  0.036],
  [0.036, -0.024, 0.003, ..., -0.006, 0.049,  0.036],
                                                ...,
  [0.036, -0.024, 0.003, ..., -0.006, 0.049,  0.036],
  [0.036, -0.024, 0.003, ..., -0.006, 0.049,  0.036],
  [0.036, -0.024, 0.003, ..., -0.006, 0.049,  0.036]]],
```

These character-level embeddings are fed to a time-distributed Bi-LSTM RNN to capture the context of the characters in words in a sentence, as well as the interaction between the morphological and syntactic information of the words in a sentence.

Algorithm 4.6: Character-Level Embedding of the Words in a Sentence

Input: Character-level encoding, character vocabulary size, embedding vector size, maximum characters in a word and maximum words in a sentence
Output: List of sentences embedded with character-level embedding

```
Function CHARACTER_LEVEL_EMBEDDING(character_Level_Encoding, characters_Vocabulary_Size,
                                   embedding_Vector_Size, maximum_Characters_In_A_Word,
                                   maximum_Words_In_A_Sentence)
    weigth_Matrix ← INITIALIZE(characters_Vocabulary_Size, embedding_Vector_Size)
    Embedded_Sentence_List ← INITIALIZE()
    for each sentence in character_Level_Encoding do
        Embedded_Sentence ← INITIALIZE()
        for each word in sentence do
            embedded_Word ← INITIALIZE()
            for each character in word do
                embedded_character ← EMBED(weigth_Matrix[character])
                embedded_Word ← APPEND(embedded_character)
                weigth_Matrix ← UPDATE(weigth_Matrix)
            Embedded_Sentence ← APPEND(embedded_Word)
        Embedded_Sentences ← APPEND(Embedded_Sentence)
    Embedded_Sentence_List ← BI_LSTM_RNN(Embedded_Sentence_List)
    return Embedded_Sentence_List
```

The character embedded sentence, S_{chr}, for the words $\{W_1, W_2, \dots, W_N\}$, each having M characters and passing through a Bi-LSTM RNN layer, is given as:

$$S_{chr} = BiLSTMRNN\left(\left[r_{11}^{chr}, r_{12}^{chr}, \dots, r_{1M}^{chr}\right], \left[r_{21}^{chr}, r_{22}^{chr}, \dots, r_{2M}^{chr}\right], \dots, \left[r_{N1}^{chr}, r_{N2}^{chr}, \dots, r_{NM}^{chr}\right]\right) \quad (35)$$

The number of hidden units and the activation of the Bi-LSTM RNN layer are hyper-parameter of the layer to be tuned. Algorithm 4.6 shows the procedure for the character-level embedding in our model.

The character-level embedding for the sentence "ያዕቆብ ዱላዉን ለዮናስ አቀበለው።" after passing through a time-distributed Bi-LSTM RNN layer with 64 hidden units and a *tanh* activation function is shown below.

```
[[[-0.011, 0.043, -0.076, ..., 0.005, 0.020, -0.114],
  [-0.001, 0.024, -0.042, ..., 0.484 , 0.102,  0.138],
  [0.162, -0.363, -0.014, ..., -0.420, -0.085,  0.284]],
                                                ...,
  [[-0.125, 0.072, 0.103, ..., 0.0409, 0.037, -0.007],
  [-0.125, 0.0722, 0.103, ..., 0.0409, 0.037, -0.007],
  [-0.1253, 0.072, 0.1023, ..., 0.049, 0.037, -0.007]]]
```

In this regard, the character embedding for the character "ከ" in the sentence "**ከ**በደና አበበ **ከ**ለንደን መጡ።" is treated differently. The character "ከ" with the word "**ከ**በደና" reflects an *Agent* (Arg1), while the character "ከ" in the word "**ከ**ለንደን" reflects the *Direction* (ArgM-Dir) the agents came from (see ANNEX E).

Since we are capturing the context of the characters for the words in the sentence with the time-distributed Bi-LSTM RNN layer, the character "**ን**" with the word "ከለንደ**ን**" in the sentence above, which could have been a *Patient* modifier (Arg1) in other word sequences, such as "እዮብ አበበ**ን** መታው።" or "እዮብ ያዕቆብ**ን** ወደ ከተማ ላከው።", is recognizing "ከለንደ**ን**" as the direction the Agents "**ከ**በደና አበበ" came from (see ANNEX E).

fastText-level Embedding

The morphological structure of a word carries important information about the meaning of the word. Word embeddings like *Word2Vec*, which train a unique word embedding for every individual word, do not take the morphological structure of a word into account [43]. The morphological structure of a word is especially significant for morphologically rich languages (like Amharic) in which a single word can have a large number of forms, each of which might occur rarely, thus making it hard to train good word embeddings. *fastText* [44], an extension to

Word2Vec, attempts to solve this by treating each word as an aggregation of its sub-words, which are taken to be the character n-grams of the word. The vector for a word is simply taken to be the sum of all vectors of its component character n-grams. *fastText* can also obtain vectors even for out-of-vocabulary (OOV) words, by summing up vectors for its component character n-grams, provided at least one of the character n-grams was present in the training data [53]. Our model uses *fastText* encoding for the words from a tokenized sentence passed. This helps our model capture the morphological structures of the words through character n-grams. These *fastText* encoded words in a sentence are then passed through a Bi-LSTM RNN to capture the bi-directional and long-range dependencies of the words in a sentence. Algorithm 4.7 shows the procedure for the fastText-level embedding in our model.

For a sentence represented in *fastText*-level encoding with a word sequence $\{W_{1fasText}, W_{2fasText}, \dots, W_{NfasText}\}$, the *fastText* embedding of the sentence, $S_{fasText}$, passing through a Bi-LSTM RNN layer is given as:

$$S_{fasText} = BiLSTMRNN\left(W_{1fasText}, W_{2fasText}, \dots, W_{NfasText}\right) \quad (36)$$

Algorithm 4.7: FastText-Level Embedding of the Words in a Sentence

Input: FastText-level encoding
Output: List of sentences embedded with fastText-level embedding

```
Function FASTTEXT_LEVE_EMBEDDING(fastText_Level_Encoding)
        Embedded_Sentence_List ← INITIALIZE()
        Embedded_Sentence_List ← BI_LSTM_RNN(fastText_Level_Encoding)
        return Embedded_Sentence_List
```

Here, the vector size, the minimum count to consider a word, the minimum and maximum number of character n-grams, the maximum distance between the current and predicted word within a sentence, the training algorithm, and the number of training epochs are hyper-parameters of the *fastText* encoding, and the number of hidden units and the activation function are also hyper-parameter of the Bi-LSTM RNN layer in our *fastText*-level embedding, all to be tuned.

The fastText-level embedding for the sentence "ያዕቆብ ዱላዉን ለዮናስ አቀበለዉ።" trained for 10 epochs with vector dimension of 5, minimum word count of 1, window size of 3 and, minimum and maximum character n-grams of 1 and 6, respectively, is shown below.

[[-0.837, 2.036, 2.944, -0.324, 1.474], ⟶ ያዕቆብ
[-1.581, 0.912, 2.714, -0.027, 1.580], ⟶ ዲላዉን
[-1.338, 2.785, 3.988, -0.233, 1.975], ⟶ ለዮናስ
[-0.562, 3.077, 4.268, -1.024, 2.382], ⟶ አቀበለው
[-1.919, 2.701, 3.265, 1.620, 0.615]] ⟶ ።

For the above sentence, the output of the fastText-level embedding of the words in the sentence, fed to a bi-LSTM RNN network with 32 hidden units and a *tanh* activation function to capture the bi-directional and long-range dependencies of the words in the sentence is represented below.

```
[[[-0.025, -0.011, 0.063, ..., 0.057, -0.062,  0.049],
  [0.028, -0.057, 0.208, ..., -0.040, 0.132,  0.082],
  [-0.117, -0.066, 0.047, ..., 0.087, 0.109,  0.011],
                                              ...,
  [-0.057, -0.025, -0.058, ..., -0.004, -0.002, -0.05],
  [-0.058, -0.026, -0.058, ..., -0.000, -0.004, -0.032],
  [-0.058, -0.026, -0.059, ..., 0.001, -0.004, -0.014 ]]]
```

Concatenated Word Embedding

The Concatenated Word Embeddings component of our model combines (joins) the information in all these three levels of embeddings for the words in a sentence through concatenation along the sentence length, and passes them to the next component in our model architecture - the Bi-LSTM RNN component. This helps our model to better capture the syntactic and semantic similarity of words, intra-word information, and relations with other words in a sentence. In addition, it helps our model to deal with out-of-vocabulary (OOV) words better.

The concatenated word embedding W_{emb} for a word W in our model is, given as:

$$W_{emb} = W_{wrd} \oplus W_{chr} \oplus W_{fasText} \tag{37}$$

where W_{wrd} is the word-level embedding given in equation 32, W_{chr} the character-level embedding given in equation 34, and $W_{fasText}$ is the fastText embedding of the word.

For a sentence with a word sequence $\{W_1, W_2, \dots, W_N\}$, the concatenated sentence embedding S_{emb} for the sentence is, thus, $\{W_{emb\,1}, W_{emb\,2}, \dots, W_{embN}\}$, which can also be represented as:

$$S_{emb} = S_{wrd} \oplus S_{chr} \oplus S_{fasText} \tag{38}$$

where S_{wrd} is the word-level embedding, S_{chr} the character-level embedding, and $S_{fasText}$ is the fastText embedding of the sentence, computed in equation 33, 35 and 36, respectively. Algorithm 4.8 shows the procedure for the concatenated word embedding in our model.

For the sentence "ያዕቆብ ዱላዉን ለዮናስ አቀበለው።", the concatenated word embedding is shown below.

[[[-0.117, -0.034, -0.077, ..., 0.057, -0.063 , 0.049],
[-0.034, 0.005, -0.036, ..., -0.040, 0.132, 0.082],
[0.001, -0.039, -0.044, ..., 0.087 , 0.109, 0.011],
...,
[0.070, 0.093, 0.010, ..., -0.004, -0.002, -0.050],
[0.070, 0.093, 0.010, ..., -0.000, -0.004 , -0.032],
[0.070, 0.093, 0.010, ..., 0.001, 0.004, -0.014]]]

With this embedding, we can be able to represent the morphological, syntactic and semantic information of words in Amharic sentences. Algorithm 4.8 shows the procedure for the concatenated word embedding in our model.

Algorithm 4.8: Concatenated Word Embedding of the Words in a Sentence

Input: List of sentences embedded with word-level embedding, character-level embedding and fastText-level embedding
Output: List of sentences embedded with concatenated word embedding

```
Function CONCATENATED_WORD_EMBEDDING(word_Level_Embedding, character_Level_Embedding,
                                     fastText_Level_Embedding,)
    Embedded_Sentence_List ← INITIALIZE()
    Embedded_Sentence_List ← CONCATENATE(word_Level_Embedding, character_Level_Embedding,
                                         fastText_Level_Embedding,)
    return Embedded_Sentence_List
```

4.2.4 Bi-LSTM RNN Layer

A bi-directional recurrent neural network (Bi-RNN) consists of two independent RNNs, one where the input is processed from the start to the end, and the other from the end to the start. The two networks are then combined into a single representation [38]. This helps Bidirectional RNNs to capture both the left and right contexts of an input at each point in time [25]. When processing

a sequence of words, both past and future inputs are known for a given time, allowing to effectively utilize the features in both right and left directions [1]. However, information encoded in hidden states of Bi-RNN networks tends to be fairly local, more relevant to the most recent parts of the input sequence and recent decisions [1]. Our model augments the Bidirectional RNN component with LSTM network to manage the context critical to distant information, thereby learning long-range dependencies [25]. This also helps for our model to capture the syntactic structure of words in a sentence. Figure 4.4 shows in detail the Bi-LSTM RNN component of our model, adapted from Jurafsky and Martin in [1].

Given a sequence of input vectors $\{x_1, x_2, \ldots, x_N\}$, the Bi-LSTM RNN computes a context representation vector h_t for each input x_t. Here, the input is passed to the forward and backward LSTMs to capture both left and right context of the word. The final representation of a word h_t is obtained by concatenating the right context h_t^f and left context h_t^b, $h_t = h_t^f \oplus h_t^b$. Each input x_t in our model is represented as W_{emb} from equation 37, and the output vector h_t is then passed as an input to the next component of our model, the fully connected layer.

The weight and bias parameters are learned in our model, and the number of hidden units and activation function are hyper-parameters that can be tuned for optimized performance.

The BI-LSTM RNN layer can capture the bi-directional and long-range dependencies of the words in a sentence [25]. For the SRL sequence labeling task, the bi-directional dependencies in the Bi-LSTM RNN network can serve as the basis for argument identification [1]. For example, the semantic role label of the word "**ሠው**" in the sentences "**ሠው** ክፉ ነው።" and "ክፉ **ሠው** ነው።" depends on the availability of an argument to the right of it. So, the semantic role label of the word "**ሠው**" in the sentence "**ሠው** ክፉ ነው።" is An *AGENT*, and in the sentence "ክፉ **ሠው** ነው።" is a *THEME*, as shown below.

[ሠው]Arg0 [ክፉ]Arg1 [ነው]Pred [።]O

[ክፉ ሠው]Arg1 [ነው]Pred [።]O

In an SRL sequence labeling task, the long-range dependencies captured through the LSTM in the Bi-LSTM RNN network can serve in argument boundary identification [1]. For example, the word sequence "**ጥር ፮ ቀን ፩፱፻፭፮ ዓ/ም**" in the sentence "እሱ **ጥር ፮ ቀን ፩፱፻፭፮ ዓ/ም** በመኪና አደጋ ሞተ ።",

expresses the temporal context of the arguments, or *argument boundaries*, for the verb-predicate "ሞተ", and is captured through the LSTM of the Bi-LSTM RNN network, as shown below.

[አሱ]Arg1 [ጥር ፷ ቀን ፲፱፻፷ ዓ/ም]ArgM-TMP [በመኪና አደጋ]ArgM-MNR [ሞተ]Pred [።]O

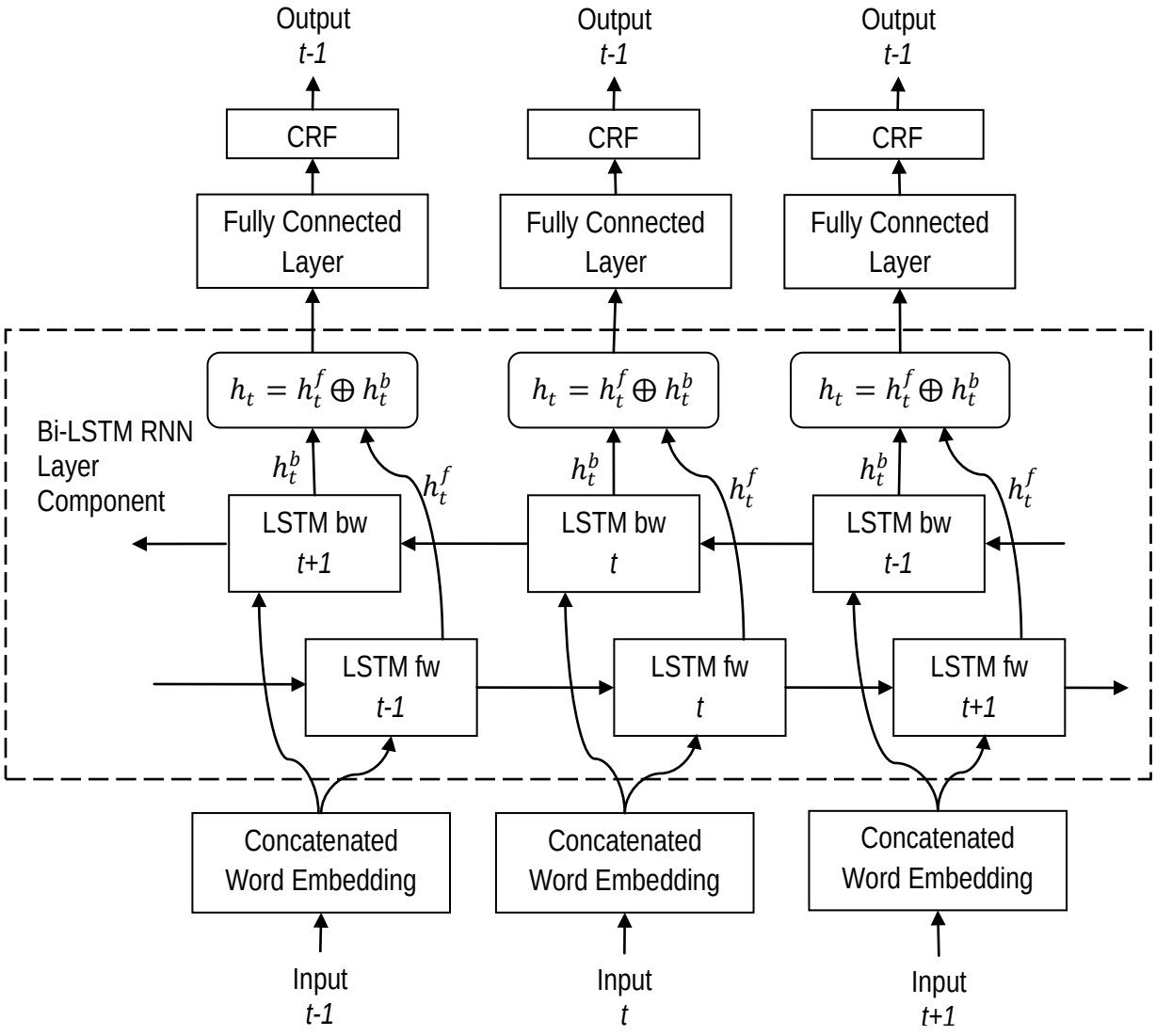


Figure 4.4: The Bi-LSTM RNN Layer Component of the Model

4.2.5 Fully Connected Layer

We have complemented the Bi-LSTM RNN layer with a fully connected layer on top of it. Fully connected layers are capable of learning any kind of dependencies present in dataset [36]. This enables our model to learn more complex features, or dependencies, present in the input sentence. This layer takes its input from the Bi-LSTM RNN layer component of our model, and

produces a vector h_t, which is then passed to the next CRF layer component in our model. The weight and bias parameters for this layer are learned by the model, and the number of hidden units and the activations function are hyper-parameters of this layer that can be tuned.

We have accounted for the morphological, syntactic and semantic features with the concatenated word embeddings and the Bi-LSTM RNN layer, but the Fully Connected Layer component can learn dependencies present between the semantic roles, themselves. For example, consider the sentence "እሱ ከ 3 ቀን በታላ ተወለደ።" where we made a spelling error "ታ" instead of "ኋ" with the word "**በታላ**". The semantic role of the word sequence "ከ 3 ቀን", which could have been a temporal argument for the verb-predicate "*ተወለደ*", seems to depend on the semantic role of the word "በታላ", as shown below.

[እሱ]Arg1 [ከ 3 ቀን]ArgM-DIR [በታላ]ArgM-MNR [*ተወለደ*]Pred [።]O

Here, the Fully Connected Layer is learning the dependency between the semantic roles *ArgM-DIR* and *ArgM-MNR* from the dataset.

However, if we correct the spelling error, the semantic role for word sequence "ከ 3 ቀን" along with the word "በኋላ" maintains its temporal nature in the sentence "እሱ ከ 3 ቀን በኋላ ተወለደ።", as shown below.

[እሱ]Arg1 [ከ 3 ቀን በኋላ]ArgM-TMP [*ተወለደ*]Pred [።]O

4.2.6 CRF Layer

A Conditional Random Fields (CRF), when combined with neural architectures for sequence labeling tasks, can jointly model the label decision by capturing the dependencies across adjacent labels [46]. Our model uses a Linear Chain CRF layer to simultaneously produce a score matrix *P* for a given sequence, and effectively model the soft IOB constraints by incorporating dependencies across the output labels. For example, an "I" tag must follow another "I" or "B" of the same class, and a "B" tag must not follow an "I" or another "B" tag of the same class.

This CRF layer, of size equal to the number of distinct labels, is placed on top of the fully connected layer component of our model. The output of the fully connected layer component, h_t, is then projected into this layer. This layer produces a score matrix P, where $P_{i,j}$ represents the score of j^{th} tag of i^{th} input token. This score matrix is then passed to the objective function in

this layer to maximize the probability of the correct score for the sequence with viterbi-decoding. The final output tag sequence y' is decided based on the maximum score, given the input sequence of word inputs x: $y' = \text{argmax}_{y' \in y} S(x, y')$.

4.3 Training

Training a neural network requires finding a loss function that models the distance between the predicted labels and the true labels, finding parameters that minimize this loss function - through gradient decent optimization algorithm, and finally finding the vector that contains the partial partial derivative of the loss function with respect to each of the parameters (gradient of the loss) through error backpropagation or reverse differentiation algorithm. To train an RNN network requires to create the unrolled computation graph for a given sentence - in which the same parameters are shared across many parts of the computaion and additional input is added at various layers, adds a loss node to the unrolled graph, and then uses the backpropagation algorithm to compute the gradients with respect to that loss.

The trainig process of our model proceeds by taking as input the encoded list of sentences and their associated roles fed into the model in a tab-separated text file, generates the word embeddings and combines them, and starting with random weights and then iteratively moving through the sequence predicting each role in the sequence. Our model uses conditional random field (CRF) loss for the loss function, and Adamax gradient decent algorithm to minimize the loss function. The Adam algorithm is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments [54]. A stochastic gradient descent method is the most efficient method for training neural networks than gradient decent and batch gradient descent because it is straight forward to implement, computationally efficient, requires less memory, is invariant to diagonal rescale of the gradients, well suited for problems with large dataset and parameters, appropriate for problems with noisy or sparse gradients, and hyperparameters have intuitive interpretation and typically require little tuning [1]. Adamax is a variant of Adam based on the infinity norm. Our model uses Adamax for its superior performance in models where embeddings are used.

The learning rate and the decay rate of the optimization algorithm, and the batch size (number of samples per gradient update), and number of epochs (number of iterations) of the training are hyperparameters of our model that can be tuned for optimized performance.

4.4 Model Optimization

Training a deep neural network that can generalize well to new data is a challenging problem. A model with too little capacity cannot learn the problem, whereas a model with too much capacity can learn it too well and overfit the training dataset. Both cases result in a model that does not generalize well. Overfitting occurs when a model fits exactly against its training data [41]. When this happens, the algorithm unfortunately cannot perform accurately against unseen data, defeating its purpose. Various regularization methods are used to prevent neural models from overfitting.

Our model uses the dropout technique, which randomly drops neurons from the network during training to prevent complex co-adaptations on training data [42]. Specifically, our model uses *spatial1D* dropout for the concatenated word embeddings - where it drops entire 1D feature maps instead of individual elements, and a regular dropout for the Bi-LSTM RNN layer outputs. The dropout rate (fraction of the input units to drop) is a hyperparameter to be tuned in both. We have also used *earlystopping* mechanism to monitor the model performance on a validation set and stop training when performance degrades.

4.5 Prediction

SRL is classification problem. It requires identifying the predicate and its constituents in a sentence, and assigning the correct semantic roles to the constituents of the predicate. Our model performs all these subtasks in SRL, all at once in a unified manner - taking advantage of end-to-end learning in deep learning.

Prediction in our model proceeds by accepting a sentence as input and preprocessing it, first. The sentence is cleaned, normalized, tokenized, encoded and finally padded accordingly, and then passed to the prediction process. The CRF layer component of our model then produces a score matrix of the semantic roles for each word in the sentence, decoded with viterbi algorithm to implement the soft IOB transition constraints. This score matrix is then passed through an

argmax function to yield the semantic roles with the highest scores for each word in the sentence, outputting the predicted sentence, the sentence with its words annotated with PropBank semantic roles.

CHAPTER FIVE: EXPERIMENT AND RESULTS

5.1 Introduction

In this chapter, we discuss about the corpus used for experiment, the experimentation environment, the experimental scenarios considered and the evaluation metrics used, and report the results obtained.

5.2 The Corpus

Amharic sentences were collected from various sources to help our model generalize better for out-of-domain data. These sources include the Amharic Bible[5], Addis Zemen Gazette, The Ethiopian Reporter Newspaper[6], Ethiopian Federal Negarit Gazette[7], and from corpus collected for Amharic-English machine translation[8], Amharic Pos-tagging[9], Amharic spell corrector[10], and list of Ethiopic names[11] in the *github*. The entire collected corpus is encoded in UTF[12]. 10,000 sentences, each containing a maximum of 32 *tokens* (31 words and the '።' character), are selected for the experiment from the sources, and were manually annotated with their semantic roles. 23 labels are collected from the considered sentences for the experiment, which contain 10 semantic roles in IOB tagging schema, the 'O' tag, the predicate 'V' tag and the padding '<PAD>' tag (see ANNEX B). Each word in the sentences are then annotated with their corresponding labels (see ANNEX C), and stored in a tab-separated text file. The word vocabulary contains unique words extracted from the considered sources, which amount to 526,370. This helps our model to better represent words out of the considered sentences when predicting new sentences. The character vocabulary contains 365 unique Geez and other characters that can be used in Amharic sentences. We have used most of the Geez characters to avoid poor character representations for

[5] https://www.wordproject.org/bibles/verses/amharic/index.htm
[6] https://www.ethiopianreporter.com/
[7] https://chilot.me/
[8] https://github.com/adtsegaye/Amharic-English-Machine-Translation-Corpus
[9] https://github.com/maobedkova/AmharicCorpus
[10] https://github.com/Yididya/amharic_spell_corrector
[11] https://github.com/Simonbelete/ethiopic-names
[12] UTF (Unicode Transformation Format) is a character encoding capable of encoding valid character code points of Unicode

out-of vocabulary characters. Simple Amharic sentences in the basic Amharic subject-object-verb (SOV) word order (see Section 2.4.3), and containing only one verb, are considered for the experiment.

5.3 Experimentation Environment

This section presents the host and development environment used, and the implementation of the model.

5.3.1 Host Computer

The host computer used for the experiments in this study is a *Lenovo* laptop, Intel(R) Core™ i7-4510u model 2.00GHz speed and x64-based CPU processor, with 8GB RAM in a 64-bit Windows 7 operating system.

5.3.2 Development Tools

We have used different open source data science tools in developing the prototype of the proposed system. We have chosen these tools because they are freely available and have plenty and active community support, or contributors. We have used Keras with Tensforflow as the backend to implement the layers and the model. Keras[13] is a deep learning API written in Python that runs on top of TensorFlow, developed with a focus on enabling fast experimentation through its user friendliness, modularity, and extensibility. TensorFlow[14] is an end-to-end open source platform for machine learning by Google Inc. It has comprehensive tools, libraries, and community resources to easily build and deploy ML-powered applications. We have implemented the CRF Layer component of our model with the CRF layer from Keras-contrib[15], which is a repository for Keras contributed modules with additional layers, activations, loss functions, optimizers, etc. which are not yet available within Keras itself. The CRF layer in this extension is an implementation of linear chain conditional random field (CRF). We have used the *FastText* embedding from Gensim[16], which is a free open-source Python library for

[13] https://keras.io/
[14] https://www.tensorflow.org/
[15] https://github.com/keras-team/keras-contrib
[16] https://pypi.org/project/gensim/

representing documents (sentence, phrase, word…) as semantic vectors, designed to process raw digital text using unsupervised machine learning algorithms, such as Word2Vec, Doc2Vec and FastText. We have used Anaconda[17], a Python distribution that comes preinstalled with lots of useful Python libraries for data science, which is shipped with Jupyter[18] Notebook, NumPy[19], Pandas[20], Matplotlib[21], and etc. We have used the Jupyter Notebook to write and run our codes, the Numpy to manipulate multidimensional arrays, the Pandas to importing the data and apply some primitive preprocessing, the Matplotlib with Seaborn[22] to visualize the model performance. Finally, we have computed the Accuracy, Precision, Recall and F-Score of the model using Seqeval[23], Sklearn[24] and sklearn-crfsuite[25], which are Python frameworks for sequence labeling evaluation.

5.3.3 Building The Model

The 10,000 Amharic sentences collected, along with their semantic roles in IOB tagging schema are stored, tab-separated, in a text file. List of unique Amharic words, extracted from the various sources considered, is stored in a separate text file. The list of labels considered is also kept in a separate text file. The preprocessing of the sentences and their associated labels is done using Python, Pandas and Numpy.

Once the preprocessing is done, the model is implemented using Keras with the TensorFlow as the backend. The word-level and character level embeddings are built with keras standard *Embedding* layer. The word-level embedding takes the unique words in the words vocabulary (526,372) as its input dimension and the maximum number of words in a sentence (32) as its input length, and represents them in an *N*-dimension vector specified by the user. The word-level embedding for a word input is then looked up from the produced embedding matrix by taking a sentence as an input. The words vocabulary includes a place-holder 'Unknown' for words not in the vocabulary, and 'PAD' for padding sentences with less than the maximum number of words

[17] https://www.anaconda.com/
[18] https://jupyter.org/
[19] https://numpy.org/
[20] https://pandas.pydata.org/
[21] https://matplotlib.org/
[22] https://seaborn.pydata.org/
[23] https://github.com/chakki-works/seqeval
[24] https://scikit-learn.org/stable/
[25] https://pypi.org/project/sklearn-crfsuite/

considered in a sentence (32). The character-level embedding takes as its input dimension the characters vocabulary (367) and the maximum number of characters in a word (15) as its input length, and represents them in an *N*-dimension vector specified by the user, which is then passed to a *TimeDistributed* Keras layer. The output of this is also passed through a *TimeDistributed Bidirectional LSTM* keras layer. The character-level embedding for a word is looked up from the produced embedding matrix by taking a sentence as an input. The characters vocabulary includes a place-holder 'Unknown' for characters not in the vocabulary, and 'PAD' for padding words with less than the considered maximum number of characters in a word (15).

The fastText-level embedding takes as an input the tokenized list of sentences, builds the word vocabulary, and embeds them with Gensim's *fastText* embedding of a vector-size, word window size, minimum count of words to consider, and minimum and maximum character n-grams specified by the user. Sentences with words less than the maximum number of words in a sentence considered (32) are padded with *zero*. The output of the fastText-level embedding for the words in a sentence is then passed through a keras' *Bidirectional LSTM* layer.

The output of the word-level, character-level and fastText-level embeddings for words in a sentence are the concatenated through a keras' *Concatenate* layer in the axis that corresponds to the maximum number of words in a sentence considered (32), which is then passed to a *SpatialDropout1D* keras layer to avoid the model from overfitting. This output is then passed to the next Bi-LSTM RNN component of our model as an input.

The Bi-LSTM RNN component of the model is implemented using keras' *Bidirectional LSTM* layer. The output of this layer is passed to a keras' *Dropout* layer to avoid the model from overfitting. The result is then passed to the next Fully Connected layer component of the model.

The Fully Connected layer component is implemented using keras' *Dense* layer which is the passed through a *TimeDistributed* keras layer.

Finally, the CRF layer component of the model, which takes the output of the Fully Connected layer as its input, is implemented using keras' *CRF* layer.

5.3.4 Model Parameters and Training

The dataset is split into training and testing as 80-20 ratio, respectively. Out of the 10,000 manually annotated sentences with their semantic roles considered, the model is trained on 8,000 sentences (80%), and tested on the rest 2,000 sentences (20%).

The training of fastText-level embedding for the words in the sentences is done independent from the model. The word-level embeddings for the words in the words vocabulary and character level embeddings for the words in the sentences, however, are done along with the model.

The *crf-loss* of the keras' *CRF* losses is used as the loss function optimized with a variant of the Adam's stochastic gradient decent algorithm - called *Adamax* optimizer in the keras optimizer functions. The training is run on a number of *batch-size* and iterations (*epochs*), monitored with keras' *EarlyStopping* mechanism to avoid model overfitting. TensorFlow's add-on *TensorBoard* is used to log the model parameters for the training iterations. Finally, the model is saved at each iteration with keras' *ModelCheckpoint*.

Parameter Settings of the Model

The parameter settings of our model are described as follows: The dimensions of the word-level embeddings are set to 5, and the character-level-embeddings are set to 20. Increasing the dimension of the embeddings, would mean finding similarities between embeddings and would result in model overfitting.

The character-level embeddings pass through a *TimeDistributed Bidirectional LSTM* layer of 128 hidden units with *tanh* as the activation function. We have set the hidden units to 128 because we want to represent all the possible morphological and syntactic interaction of a word in a sentence, and still maintain a clear distinction between the representations. A small number of hidden units resulted in overlapped representations and increasing the number of hidden units resulted in poor representation of the interactions. We have used the *tanh* activation function for its superior performance in representing both high and low class instances.

The fastText-level embeddings have a vector dimension of 5 with window size of 3, minimum word count of 1, and a minimum and maximum character n-gram of 1 and 6, respectively. We have used a small vector dimension (5) to avoid non-existent similarity between words which

would result in poor representation. As the window size increases, fastText embeddings can suffer in representing the morphology of a word with long words-context, so we have used the window size to be just 3. The maximum characters a word can have in our sentences is 15, and we have used the minimum character n-gram as 1 and the maximum as 6 (maximum char-gram *fastText* can compute is 6), to compute a words vector based on its character n-grams.

We have trained the fastText-level embeddings for 10 iterations, for the embeddings not to overfit but learn the vector representations. The output of the fastText-level embeddings are also fed to a *Bidirectional LSTM* layer of 32 hidden units with the *tanh* activation function. Since the morphological and syntactic (context) information is captured to some degree with the *fastText* vector representations, we have used a small number of hidden units (32) to avoid model overfitting.

All the word-level, character-level and fastText-level embeddings are then concatenated in the axis that corresponds to the sentence length, and fed to a *SpacialDroput1D* with a keep probability of 0.8 to avoid model overfitting.

The result is fed to the *Bidirectional LSTM* layer of 32 hidden units and with a *tanh* activation function. A *Dropout*, with keep probability of 0.8 is applied to avoid model overfitting. Here, we have only used 32 hidden units to represent the bi-directional (argument identification) and long-range (argument boundary detection) as we have already accounted the semantic similarity of words, the morphological information of a word, and the morphological and syntactic interaction of a word. So, increasing the number of hidden units would result in model overfitting.

The output of the Bi-LSTM RNN Layer component of our model is fed to the *Dense* layer which has 64 hidden units and no activation function. Here, we want to capture any dependencies present in the dataset, maintaining the size coming from the Bi-LSTM RNN Layer component, which is 64, and the linearity of the data with no activation function.

The output of the *Dense* layer component is passed to the *CRF* layer, with the dimension equal to the tags in IOB schema (23). We have adopted the *Adamax* as the parameter optimizer with a learning rate set to 0.001 and the constant for numerical stability (epsilon) set at $\epsilon = 1e^{-7}$. To avoid exploding gradients problem, we clipped the norm of gradients with a threshold of 1.0.

We have trained the model for 50 iterations (ephocs) with mini-batches of size of 4, monitoring the loss at each iteration with *EarlyStopping* set at 5 as its patience.

We have conducted experiments on the model with different parameter settings for model optimization, and have taken the best-model with the optimized parameters. The F-Score for the model with different parameter settings is presented in ANNEX D.

The parameter settings of the Concatenated Word, the Bi-LSTM Layer and the Fully Connected Layer components of the model are presented in Table 5.1.

Table 5.1: Parameter Settings of the Model Components

Component	Embedding Level	Embedding Vector Size	Bi-LSTM RNN Network			Dropout	
			Layers	Hidden Units	Activation Function	Type	Keep Probability
Concatenated Word Embedding	Word-level	5	-	-	-	Spacial-1D	0.8
	Character-level	20	1	128	tanh		
	FastText-level	5	1	32	tanh		
Bi-LSTM RNN Layer			1	32	tanh	Regular	0.8
Fully Connected Layer			1	64	NONE	-	-

5.4 Evaluation

The evaluation of the proposed approach aims at answering the research question: *what kind of neural word embedding is required to capture the morphological, syntactic and semantic information of the words in Amharic sentences for an SRL system using deep learning?*

5.4.1 Experimental Scenario

To examine the hypothesis of this study, four experimental scenarios were conducted to determine the type of neural word embeddings required to represent the morphological, syntactic and semantic information about words in Amharic sentences that can replace manually designed features, and thus, improve the performance an SRL system for the language.

We hypothesize that the semantic similarity of words can be represented with the word-level embedding, the morphological information of a word and the interaction between the morphology and syntax of a word cab be represented with the character level embedding and feeding the output to a Bi-LSTM RNN network, respectively. By complementing this with the

fastText-level embedding, we can capture morphological information of words by computing the character n-grams and the context of a word by computing its relation to the words surrounding it using the character n-gram and window-size features of the *fastText* embedding, respectively. Feeding this output to a Bi-LSTM network can capture the morphological and syntactic interactions of a word. The Bi-LSTM Layer component of the model can capture the bi-directional (argument identification) and the long-range (argument boundary detection) dependencies of words in a sentence [1, 23, 25]. Any other kind of dependencies present in the dataset, e.g., the dependencies between the semantic role labels, can be represented by the Fully Connected Layer component of the model.

To check our hypothesis, we have conducted experiments using the below four scenarios.

1. In the first scenario, we have considered word-level embeddings only. For this scenario, the words in the words vocabulary were embedded with the standard keras' *Embedding* layer of input length equal to the maximum number of words considered in a sentence (32). These word embeddings were then passed to the Bi-LSTM RNN component of the model, and trained.
2. In the second scenario, we have considered character-level embeddings only. The characters in the characters vocabulary were also embedded using the standard keras' *Embedding* layer of input length equal to the maximum number of characters considered in a word (15), and the maximum number of words considered in a sentence (32). The output of this is fed to a keras' *TimeDistributed* layer, which itself is passed to a keras' *TimeDistributed Bidirectional LSTM* layer to capture the bidirectional and long-range dependencies of the characters for words in a sentence. The output of these character-level embeddings were then passed to the Bi-LSTM RNN component of the model, and trained.
3. In the third scenario, the word-level embeddings were complemented with character-level embeddings, to see for any improvements in performance. Both character and word-level embeddings were concatenated and then passed to the Bi-LSTM RNN component of the model, and trained.
4. Finally, both the word and character-level embeddings were supplemented with fastText-level embeddings, to check for any improvements in performance. The output of fastText-level embeddings was fed to a keras' *Bidirectional LSTM* layer to capture the bidirectional and long-range dependencies of the fastText embedded words in a sentence. All the three

embeddings, the word, character and fastText-level, were then concatenated and fed to the Bi-LSTM RNN component of the model, and trained.

The results for the above experimental scenarios are reported on Table 5.2.

5.4.2 Evaluation Metrics

The standard evaluation for semantic role labeling is to require that each argument label must be assigned to the exactly correct word sequence, and then compute accuracy, precision, recall, and F-score [1]. We have computed the Accuracy, Precision, Recall and F-Score of the model. We have computed a *Confusion Matrix* of the model to provide us a holistic view of how well our classification model is performing and what kinds of errors (*confusion*) it is making. We have also used a *Learning Curve* to plot the changes in learning performance over time in terms of experience., and diagnose problems with learning, such as an underfit or overfit model, as well as whether the training and validation datasets are suitably representative.

5.5 Test Result

With the model setup discussed in Section 5.3.3, the model parameters and training discussed in Section 5.3.4 and the experimental scenarios we have set in Section 5.4.1, we have obtained the test results presented on Table 5.2.

Table 5.2: Accuracy, and Average Precision, Recall, and F-score of the Model

Embedding Used in Model	Accuracy	Precision	Recall	F-Score
Word Embedding	91.73	72.2	62.3	66.9
Character Embedding	93.95	78.6	75.3	76.9
Word + Character Embedding	94.67	81.6	78.7	80.1
Word + Character + fastText Embedding	**94.96**	**82.8**	**79.7**	**81.2**

The word-level embeddings can capture the semantic similarity between words. The character-level embeddings can capture the morphological information of words, and the interaction between the morphology and syntax of a word can by feeding the output to a Bi-LSTM RNN network.

From Table 5.2, we can observe that the model with only the word-level embedding has achieved lower results than the other models setups considered. This indicates that the word-

level embedding alone can struggle in representing the features of the words in Amharic sentences. This could be due to the fact that Amharic is a morphologically-rich language, and requires a character level embedding to better capture the morphological features, which is evident from the results achieved in the model with the character-level embeddings only. The models with the concatenated word and character-level embeddings, and word, character and *fastText*-level embeddings achieved comparable results. This could account to the similarity in the nature of the embeddings used in both the character-level and *fastText*-level; both are character-based representations of words. The model with the concatenated word, character and fastText-level embedding, however, achieved higher results which may contribute to the window-size and character n-gram features of the *fastText* embedding. We can observe that using the concatenated word, character and *fastText*-level embeddings resulted better performance in capturing the morphological, syntactic and semantic information about words in Amharic sentences.

Table 5.3: Summary of the Average Precision, Recall and F-score for each Class

Class	Precision	Recall	F-Score	Support
ARG0	0.781	0.824	0.802	1366
ARG1	0.639	0.645	0.642	1986
ARG2	0.595	0.372	0.458	328
ARG3	0.000	0.000	0.000	1
ARG4	0.000	0.000	0.000	3
ARGM-DIR	0.760	0.447	0.563	170
ARGM-LOC	0.444	0.371	0.405	140
ARGM-MNR	0.554	0.435	0.487	416
ARGM-PRP	0.544	0.339	0.417	127
ARGM-TMP	0.414	0.215	0.283	135
PREDICATE	1.000	1.000	1.000	2000
<PAD>	1.000	1.000	1.000	2000
Micro avg	**0.828**	**0.797**	**0.812**	8671
Macro avg	0.561	0.471	0.505	8671
Weighted avg	0.816	0.797	0.803	8671

For the model with the best result on Table 5.2, we have presented a summary of the precision, recall and F-score for each class and each sequence item (each class in IOB) on Table 5.3 and Table 5.4, respectively.

Some observations can be made regarding the results shown on Table 5.3, and Table 5.4.

- The result reported for labels with low number of instances is relatively low. For example, the F-score for both ARG3 and ARG4 reported in both tables is 0.00. This can be contributed in part to the low number of training examples with these labels in the training dataset.
- The label 'PREDICATE' has an F-score of 1.000. This is probably due to the position of the predicate (the word-order) considered in all the sentences in the dataset, i.e. the last word before the full-stop character (SOV), which makes identification of the predicate relatively easy.

Table 5.4: Summary of the Average Precision, Recall and F-score for each Class in IOB

Class	Precision	Recall	F-Score	Support
<PAD>	1.000	1.000	1.000	50112
B-ARG0	0.831	0.877	0.853	1366
B-ARG1	0.696	0.703	0.700	1986
B-ARG2	0.639	0.399	0.492	328
B-ARG3	0.000	0.000	0.000	1
B-ARG4	0.000	0.000	0.000	3
B-ARGM-DIR	0.790	0.465	0.585	170
B-ARGM-LOC	0.504	0.421	0.459	140
B-ARGM-MNR	0.590	0.464	0.520	416
B-ARGM-PRP	0.570	0.354	0.437	127
B-ARGM-TMP	0.471	0.244	0.322	135
I-ARG0	0.672	0.615	0.642	680
I-ARG1	0.664	0.880	0.757	2884
I-ARG2	0.347	0.169	0.227	302
I-ARG3	0.000	0.000	0.000	1
I-ARG4	0.000	0.000	0.000	6
I-ARGM-DIR	0.741	0.413	0.531	242
I-ARGM-LOC	0.382	0.380	0.381	213
I-ARGM-MNR	0.533	0.367	0.435	436
I-ARGM-PRP	0.604	0.431	0.503	255
I-ARGM-TMP	0.592	0.360	0.448	197
O	1.000	1.000	1.000	2000
PREDICATE	1.000	1.000	1.000	2000
Accuracy			**0.950**	64000
Macro avg	0.549	0.458	0.491	64000
Weighted avg	**0.948**	**0.959**	**0.947**	64000

Table 5.5: Confusion Matrix for the Main Classes

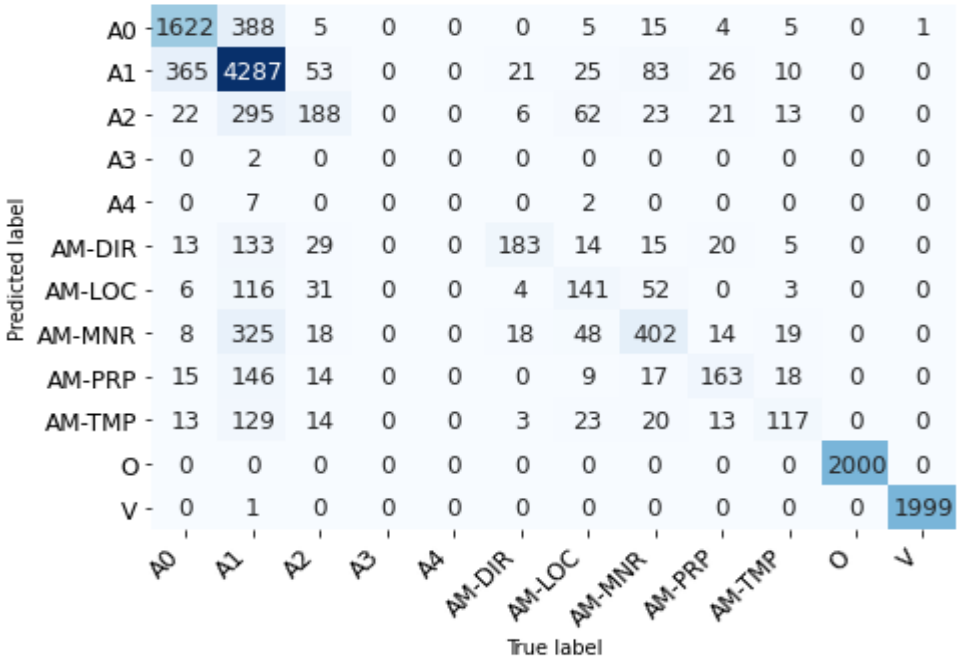
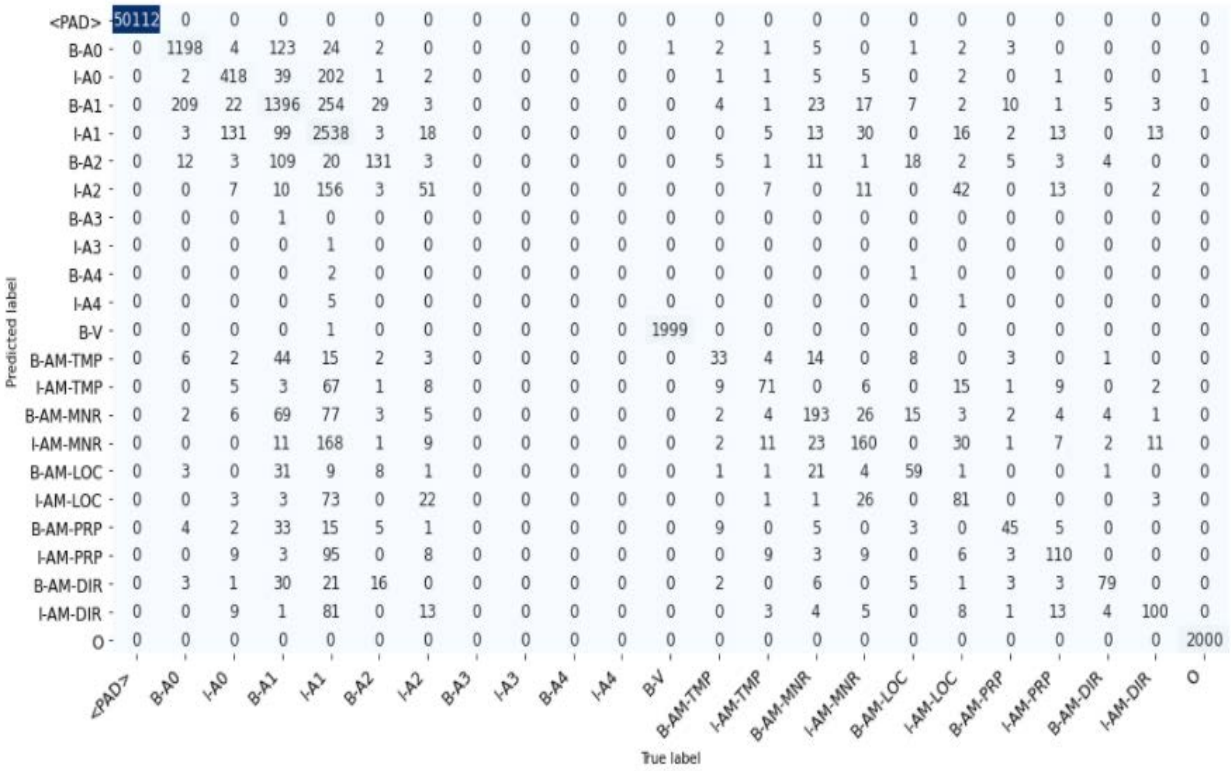


Table 5.6: Confusion Matrix for all the Classes in IOB



We have also computed both the class-based and IOB-based confusion matrix for the best performing model to see the correctly classified and misclassified instances, which are presented on Table 5.5 and 5.6, respectively. In both tables, the off-diagonal results reported represent misclassified instances of the associated class. These misclassifications (confusions) are probably due to the low number of instances of the classes in confusion. Another reason could account to the lack of expertise in annotating the sentences with the correct semantic roles.

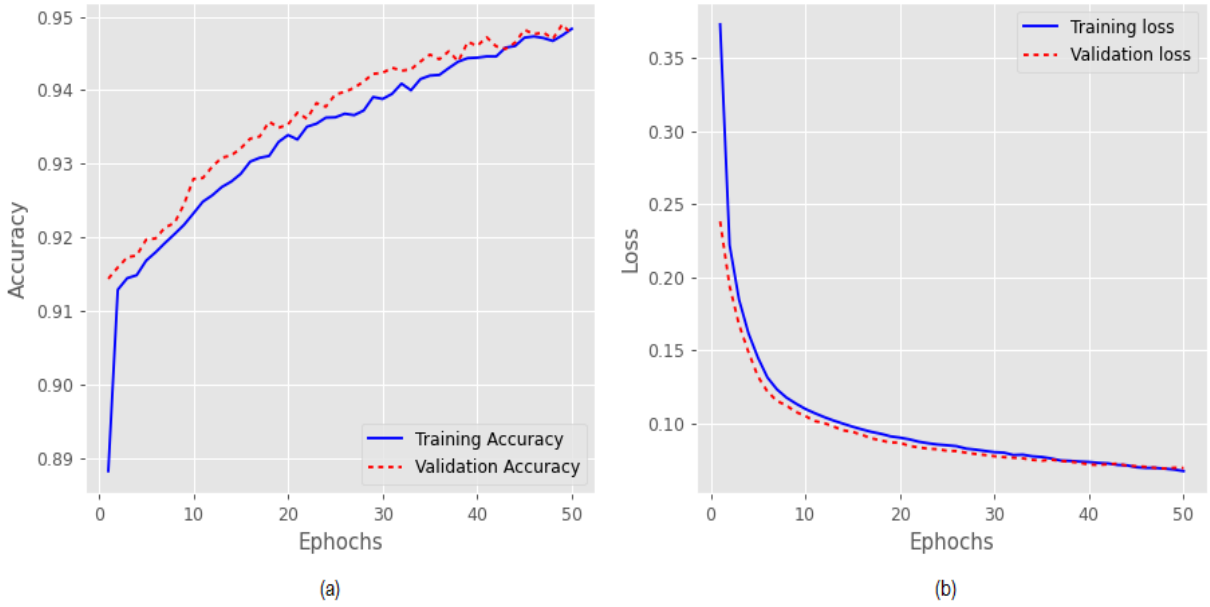


Figure 5.1: Learning Curve of the Model

The learning curve for the best performing model from Table 5.2 is depicted on Figure 5.1. Figure 5.1(a) shows the training and validation accuracy of the model. From this figure, we can observe that the model performance is growing over time, which means the model is improving with experience, i.e., it is learning. We can see that the validation accuracy does not dip below the training accuracy, indicating the model is a good-fit. We also see it grows rapidly at the beginning, and eventually grows over time it before reaches a plateau, which indicates that it still requires mode data to learn more. From the figure in Figure 5.1(b), which show the training and validation loss of the model, we can observe that the model is able to obtain sufficiently lower error value (loss) on the training set to a point of stability, with a minimal gap between the final training and validation losses. The validation loss shows steady movements around the training loss with minimal gap, indicating both the training and validation dataset are representative to

each other, i.e., the training dataset provides sufficient information to learn the problem, and the validation dataset also provides sufficient information to evaluate the ability of the model to generalize. We can observe that the model is a good-fit.

5.6 Discussion

The results reported here provide a first insight into the possibilities semantic role labeling for Amharic text based on deep learning. The experiments were carried out using keras API with TensorFlow as the backend. Four experimental scenarios were conducted to determine the type of neural word embeddings required to represent the morphological, syntactic and semantic information about words in Amharic sentences that can replace manually designed features, and thus, improve the performance of SRL systems for the language using deep learning. We have used word-level embedding only, character-level embedding only, concatenated word and character-level embeddings, and a concatenation of word, character and fastText-level embeddings for the words in the sentences. In all the scenarios, the output of the embeddings is fed to a Bi-LSTM RNN to capture the bi-directional and long-range dependencies of the words in the sentences. Finally, we have used a CRF with viterbi-decoding to produce a score matrix and maximized the likelihood of the probability distribution by applying soft IOB constraint. We have evaluated the model in all the four scenarios using standard SRL evaluation metrics: accuracy, precision and recall. For the model with the best result, we have also used confusion matrix to evaluate the performance of the classification, and a learning curve to interpret the model performance. The model with all the concatenated word, character and fastText-level embeddings, achieved a higher performance with 94.96 % accuracy and an F-score of 81.2%. The learning curve of the model, having minimal generalization gap and representative dataset, can be interpreted as a good-fit.

Amharic is a morphologically rich language, and requires a word and character-level neural word embedding in an SRL system for the language using deep learning for an improved performance bypassing the feature extraction task. We have used a concatenation of word, character and fastText-level word embeddings, and have been able to capture the semantic similarities of words using word-level embedding, the morphological information of words using the character and fastText-level embeddings. We have passed the character and fastText-level embedding output through Bi-LSTM RNN networks to capture the syntactic information of

words, their context in sentences, and the interaction between the morphology and syntax of a word in a sentence. We have presented sample predictions from the model that achieved best results in ANNEX E.

Our proposed system is not compared with other SRL systems because the data format, data size, and annotations used may all vary.

CHAPTER SIX: CONCLUSION AND FUTURE WORK

6.1 Conclusion

This study was conducted to develop SRL for simple Amharic sentences using deep learning. We have used a concatenation of word, character and fastText-level neural word embeddings to learn features of the words in sentences - avoiding manual feature extraction task, a Bi-LSTM RNN network to capture the context of the words in the sentences, and a CRF with viterbi-decoding to produce a score matrix of the roles for each word in a sentence. The system achieved an accuracy of 94.96% and F-score of 81.2%.

We can draw a number of conclusion form our experimental observations of the system. First, Amharic SRL systems using deep learning require word and character-level neural word embeddings to represent the features about words in Amharic sentences that are important to the SRL task, so as to avoid the need to manually designed features. Second, combining word, character and fastText-level neural embeddings, to represent the morphological, syntactic and semantic information about words in Amharic sentences, can significantly enhance the performance of the system. Thirdly, SRL systems using deep neural networks can result in better performance given a good quality input dataset having larger number of instances with balanced class distribution.

6.2 Contributions of the Study

The main contributions of the study are listed below.

- An SRL study for Amharic text using Deep Learning is presented.
- The general neural word embedding architecture for Amharic text using concatenated word and character-level embeddings to capture the morphological, syntactic and semantic information of words in Amharic sentences.
- The system has implemented semantic role labeler for simple Amharic sentences using deep learning by avoiding the manual feature extraction task.

- The 10, 000 sentences manually annotated with their corresponding semantic roles.

6.3 Future Work

Based on the experiments conducted and results obtained in Chapter 5, the following potential extensions that can enhance the performance, scope, and outcomes of the Amharic SRL are suggested as future works.

- Classification tasks using deep learning perform well on a reasonable corpus size with well-balanced class distribution. This study has used a medium size corpus with unbalanced class distribution, and future works can consider improving the quality of the corpus for an improved performance.
- The choice of the word embeddings used in this study is constrained to the limited computational resources that were available to us. Future works can experiment this study with other contextualized neural word embeddings methods, such as ElMo and BERT, in a higher capacity computational environment.
- The sentences used in this study lack expert-based semantic role annotations. Future works can experiment this study with professionally annotated semantic roles.
- Simple Amharic sentences are targeted in this study. Future works can extend this study to constitute complex Amharic sentences.

REFERENCES

[1] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Third Edition, Prentice Hall, 2019.

[2] D. Khurana, A. Koli, K. Khatter and S. Singh, "Natural Language Processing: State of The Art, Current Trends and Challenges", white paper, 2017, retrieved from https://arxiv.org/abs/1708.05148, Last access on December 1, 2020.

[3] A. Jain, G. Kulkarini and V. Shah, "Natural Language Processing", *International Journal of Computer Science and Engineering*, Vol. 6, Issue-1, 2018, pp. 161-167.

[4] D. Gildea and D. Jurafsky, "Automatic Labeling of Semantic Roles", *Association for Computational Linguistics*, Vol. 28, Issue-3, 2002, pp. 245–288.

[5] L. Marquez, X. Carreras, K.C. Litkowsky and S. Stevenson, "Semantic Role Labeling: An Introduction to the Special Issue", *Association for Computational Linguistics*, Vol. 34, No. 2, 2008, pp, 145-159.

[6] L. He, M. Lewis and L. Zettlemoyer, "Question-Answer Driven Semantic Role Labeling: Using Natural Language to Annotate Natural Language", in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, September 2015.

[7] D. Liu and D. Gildea, "Semantic Role Features for Machine Translation", in *Proceedings of the 23rd International Conference on Computational Linguistics*, Beijing, China, August 2010.

[8] W. Aziz, M. Rios and L. Specia, "Shallow Semantic Trees for SMT", in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, Edinburgh, Scotland, July 2011.

[9] J. Christensen, Mausam, S. Soderland and O. Etzioni, "An Analysis of Open Information Extraction Based on Semantic Role Labeling", in *Proceedings of the 6th International Conference on Knowledge Capture*, Banff Alberta, Canada, June 2011.

[10] A. Khan, N. Salim and Y. J. Kumar, "A Framework for Multi-Document Abstractive Summarization Based on Semantic Role Labeling", *Applied Soft Computing*, Vol. 30, 2015, pp. 737–747.

[11] A. H. Osman, N. Salim, M. S. Binwahlan, R. Atleeb and A. Aboubieda, “An Improved Plagiarism Detection Scheme Based on Semantic Role Labeling”, *Applied Soft Computing*, Vol. 12, Issue 5, 2012, pp. 1493-1502.

[12] B. Kouchnir, “A Memory-based Approach for Semantic Role Labeling”, in *Proceedings of the Eighth Conference on Computational Natural Language Learning,* Boston, Massachusetts, USA, May 2004.

[13] V. Punyakanok, D. Roth and W. Yih, “The Importance of Syntactic Parsing and Inference in Semantic Role Labeling”, *Association for Computational Linguistics*, Vol. 34, No. 2, 2008, pp. 258-287.

[14] K. Hacioglu, “Semantic Role Labeling Using Dependency Trees”, in p*roceedings of the 20th International Conference on Computational Linguistics*, Geneva, Switzerland, August 2004.

[15] R. Collobert and J. Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning”, in *Proceedings of the 25th International Conference On Machine Learning*, Helsinki, Finland, July 2008.

[16] T. Young, D. Hazarika, S. Poria and E. Cambria, “Recent Trends in Deep Learning Based Natural Language Processing“, *IEEE Computer Intelligence Magazine,* Vol. 13, 2018, pp. 55–75.

[17] N. Xue, “Labeling Chinese Predicates with Semantic Roles”, *Computational Linguistics: MIT Press Journals*, Vol. 34, No. 2, 2008, pp. 225-255.

[18] M. Diab, A. Moschitti and D. Pighin, “Semantic Role Labeling Systems for Arabic using Kernel Methods”, in *Proceedings of Association for Computational Linguistics*, Columbus, Ohio, USA, June 2008.

[19] Eskedar Yirga Tadesse, “Semantic Role labeler for Amharic Text Using Memory-Based Learning”, Unpublished Masters Thesis, Department of Computer Science, Addis Ababa University, 2017.

[20] A. Abkik and Y. Li, “K-SRL: Instance-based Learning for Semantic Role Labeling”, in *Proceedings of the 26th International Conference on Computational Linguistics*, Osaka, Japan, December 2016.

[21] P. Cunningham, B. Smyth and T. Veale, “On the Limitations of Memory Based Reasoning”, in *Proceedings of the 2nd European Workshop on Case-Based Reasoning*, Paris, France, November 1994.

[22] L. He, K. Lee, M. Lewis and L. Zettlemoyer, “Deep Semantic Role Labeling: What Works and What’s Next”, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017.

[23] J. Zhou and W. Xu, “End-to-End Learning of Semantic Role Labeling Using Recurrent Neural Network”, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, Beijing, China, July 2015.

[24] J. Park, “Selectively Connected Self-Attentions for Semantic Role Labeling”, *Applied Sciences: Computing and Artificial Intelligence*, Vol. 9, Issue 8, 2019, pp. 1716-1716.

[25] Z. Wang, T. Jiang, B. Chang and Z. Sui, “Chinese Semantic Role Labeling with Bidirectional Recurrent Neural Networks”, in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, September 2015.

[26] United Nations Department of Economic and Social Affairs: *Population Dynamics*, “World Population Prospects 2019”, online report, 28 August 2019, retrieved from https://population.un.org/wpp/Publications/Files/WPP2019_Release-Note-rev1.pdf/, Last access on February 19, 2020.

[27] M. Palmer, D. Gildea and P. Kingsbury, “The Proposition Bank: An Annotated Corpus of Semantic Roles”, *Association for Computational Linguistics*, 2005, pp. 71-106.

[28] A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman, “The NomBank Project: An Interim Report”, in *Proceedings of the NAACL/HLT Workshop: Frontiers in Corpus Annotation*, Boston, Massachusetts, USA, May 2004.

[29] J. Ruppenhofer, M. Ellsworth, Petruck M., C. Johnson, C. Baker and J. Scheffezyk, *FrameNet II: Extended Theory and Practice*, Berkley, 2016.

[30] C. Baker, C. Filmore and J. Lowe, “The Berkeley FrameNet Project”, in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, Montreal, Canada, August 1998.

[31] C. Filmore, “Frames and the Semantics of Understanding”, *Quaderni di Semantica*, 1985, Vol. 6, No. 2, pp. 222–254.

[32] D. Gildea and D. Jurafsky, “Automatic Labeling of Semantic Roles”, In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, October 2000.

[33] W. MucCulloch and W. Pitts, “A Logical Calculus of the Ideas Immanent In Nervous Activity”, *Bulletin of Mathematical Biophysics*, 1943, Vol.5, pp. 115-133.

[34] I. Godfellow, Y. Bengio and A. Courville, *Deep Learning*, The MIT Press, 2016.

[35] Y. Goldberg, *Neural Network Methods for Natural Language Processing: Synthesis Lectures on Human Language Technologies*, Morgan and Claypool, 2017.

[36] P. Goyal, S. Pande and K. Jain, *Deep Learning for Natural Language Processing: Creating Neural Networks with Python*, Apress, 2018.

[37] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, 1969.

[38] M. Schuster and K. Paliwal, “Bidirectional Recurrent Neural Networks”, *IEEE Transactions on Signal Processing*, 1997, Vol. 45, No. 11, pp. 2673–2681.

[39] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory”, *Neural Computation*, 1997, Vol. 9, No. 8, pp. 1735– 1780.

[40] K. Cho, B. van Merrienboer, C. Gulchere, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”, in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, October 2014.

[41] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Improving Neural Networks by Preventing Co-adaptation of Feature Detectors”, white paper, 2012, retrieved from https://arxiv.org/abs/1207.0580 , Last access on December 12, 2020.

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Journal of Machine Learning Research (JMLR)*, 2014, Vol. 15, No. 56, pp. 1929-1958.

[43] T. Mikolov, K. Chren, G. Corrodo and J. Dean, “Efficient Estimation of Word Representations In Vector Space”, in *Proceedings of the International Conference on Learning Representations (ICLR)*, Scottsdale, Arizona, USA, May 2013.

[44] A. Joulin, E. Grave, P. Bojanowski and T. Mikolov, “Bag of Tricks for Efficient Text Classification”, *In proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2017, Vol. 2, No. 12, pp. 427-431.

[45] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky and S. Khudanpur, “Recurrent Neural Network Based Language Model”, in *Proceedings of the 11th Annual Conference of the International Speech Communication Association - INTERSPEECH*, Makahari, Chiba, Japan, September 2010.

[46] M. E. Peters, W. Ammar, C. Bhagavatula and R Power, “Semi-supervised Sequence Tagging with Bidirectional Language Models”**,** white paper, 2017, retrieved from https://arxiv.org/abs/1705.00108, Last accessed on June 30, 2021.

[47] I. Zitouni, *Natural Language Processing of Semitic Languages*, Springer, Heidelberg, Berlin, 2014.

[48] R. Hertzron, *The Semitic Languages*, Routledge, London/New York, 1997.

[49] Tadesse Anberbir, M. Gasser, T. Takara and K. D. Yoon, “Grapheme-to-Phoneme Conversion for Amharic Text-to-Speech System”, in *Proceedings of the Conference on Human Language Technology for Development (HLDT)*, Alexandria, Egypt, May 2011.

[50] W. Leslau, *Introductory Grammar of Amharic*, Wiesbaden, Germany, 2000.

[51] E. Lipinski, *Semitic Languages*, *Outline of a Comparative Grammar*, Peeters, Leuven, Belguim, 1997.

[52] M. Gasser, “A Dependency Grammar for Amharic”, in *Proceedings of the workshop on language resources(LR) and human language technologies(HLT) for Semitic languages*, Valtetta, Malta, May 2010.

[53] V. Zolotov and D. Kung, “Analysis and Optimization of fastText Linear Text Classifier”**,** white paper, 2017, retrieved from https://arxiv.org/abs/1702.05531, Last accessed on June 30, 2021.

[54] D. P. Kingma, J. L. Ba, “Adam: A Method for Stochastic Optimization”**,** white paper, 2017, retrieved from, https://arxiv.org/abs/1412.6980, Last accessed on June 30, 2021.

ANNEXES

ANNEX A: Description of the PropBank Semantic Roles

No.	Semantic Role	Description
1	ARG0	Agent, Operator, Causer
2	ARG1	Patient, Theme
3	ARG2	Instrument, Experiencer, Beneficiary, Attribute
4	ARG3	Starting Point, Beneficiary, Attribute
5	ARG4	Ending Point
6	ARGM-COM	Comitative
7	ARGM-LOC	Locative
8	ARGM-DIR	Directional
9	ARGM-GOL	Goal
10	ARGM-MNR	Manner
11	ARGM-TMP	Temporal
12	ARGM-EXT	Extent
13	ARGM-REC	Reciprocals
14	ARGM-PRD	Secondary Predicate
15	ARGM-PRP	Purpose
16	ARGM-CAU	Cause
17	ARGM-DIS	Discourse
18	ARGM-ADV	Adverbial
19	ARGM-ADJ	Adjectival
20	ARGM-MOD	Modal
21	ARGM-NEG	Negation
22	ARGM-DSP	Direct Speech
23	ARGM-LVB	Light Verb
24	ARGM-CXN	Construction

ANNEX B: List of Tags Considered in the System

No.	Tag
1	<PAD>
2	B-ARG0
3	B-ARG1
4	B-ARG2
5	B-ARG3
6	B-ARG4
7	B-ARGM-DIR
8	B-ARGM-LOC
9	B-ARGM-MNR
10	B-ARGM-PRP
11	B-ARGM-TMP
12	I-ARG0
13	I-ARG1
14	I-ARG2
15	I-ARG3
16	I-ARG4
17	I-ARGM-DIR
18	I-ARGM-LOC
19	I-ARGM-MNR
20	I-ARGM-PRP
21	I-ARGM-TMP
22	O
23	PREDICATE

ANNEX C: Sample Sentences Annotated with Semantic Roles

1. ፍርድ ሁሉ ፍትሐዊ ተብሎ አይጠቀስም። B-A1 I-A1 B-AM-MNR I-AM-MNR B-V O
2. እናንተ የአገልጋዩን ስም አታጥፋ። B-A0 B-A2 B-A1 B-V O
3. ቅንጅትና ቅንብሩ ለእኔ ለጊዜው የማይፈታ እንቆቅልሽ ሆነብኝ። B-A0 I-A0 B-A2 B-A1 I-A1 I-A1 B-V O
4. እነሱ በቤቴ ውስጥ ይህን አድርገዋል። B-A0 B-AM-LOC I-AM-LOC B-A1 B-V O
5. እኔ እናንተን አስቀድሜ አስጠንቅቄያችኋለሁ። B-A0 B-A1 B-AM-MNR B-V O
6. ተስፋዬ ወደተቀረቀሩ የመቃብር በሮች ይወርዳል። B-A1 B-AM-DIR I-AM-DIR I-AM-DIR B-V O
7. እናንተ ገርነትንና ትዕግሥትን ልበሱ። B-A0 B-A1 I-A1 B-V O
8. እኚሁ አዲስ ፕሬዚዳንት መፈንቅለ አመራሩን ለመፈፀም ከፍተኛ ግፊት ማድረጋቸውን ውስጥ አዋቂ ምንጮች ይገልጻሉ። B-A1 I-A1 I-A1 I-A1 I-A1 I-A1 I-A1 I-A1 I-A1 B-A0 I-A0 I-A0 B-V O
9. እኤአ ከ1983 ዓ/ም ወዲህ በግምት 13 ሚሊዮን ህዝብ ለሞት ተዳርጓል። B-AM-TMP I-AM-TMP I-AM-TMP I-AM-TMP B-A1 I-A1 I-A1 I-A1 B-AM-DIR B-V O
10. አታላይ ባወራ ቁጥር ይዋሻል። B-A1 B-AM-TMP I-AM-TMP B-V O
11. እውነተኛ ጥበብ በንዳና ላይ ትጮኻለች። B-A1 I-A1 B-AM-LOC I-AM-LOC B-V O
12. አንተ እየፈራረሰች ነውና ስንጥቆቿን ጠግን። B-A0 B-AM-PRP I-AM-PRP B-A1 B-V O
13. እያንዳንዱ ዛፍ በፍሬው ይታወቃል። B-A1 I-A1 B-AM-MNR B-V O
14. የአልኘስ መዝናኛ ከተማ እንቅስቃሴ በድንገት ቆሟል። B-A1 I-A1 I-A1 I-A1 B-AM-MNR B-V O
15. ከ30 በላይ ቁስለኞች መኖራቸውን ፍልስጤማውያን ቅዳሜ ዕለት ገልጸዋል። B-A1 I-A1 I-A1 I-A1 B-A0 B-AM-TMP I-AM-TMP B-V O
16. ሚስተር ቡሽ ቅዳሜ ማታ ወደ ዋሽንግተን ተመልሰዋል። B-A1 I-A1 B-AM-TMP I-AM-TMP B-AM-DIR I-AM-DIR B-V O
17. ሚኬሌም ለታዳጊ ለወጣትና ለብሔራዊ ተሰልፏል። B-A1 B-A2 I-A2 I-A2 B-V O
18. ሰኞ መጋቢት 28 ዘሪሁን የተባለው እኔን ቢሮው አስጠራኝ። B-AM-TMP I-AM-TMP I-AM-TMP B-A0 I-A0 B-A1 B-AM-DIR B-V O
19. ብርሃንና ሰላም ኢትዮጵያ ቡናን ከጥሎ ማለፍ ዋንጫ ውጪ አድርጎታል። B-A0 I-A0 B-A2 I-A2 B-AM-DIR I-AM-DIR I-AM-DIR B-A1 B-V O
20. አራፋት ሲዳከሙ እንደሃማስ ያሉ ተራማጅ ቡድኖች ጥንካሬ አግኝተዋል። B-AM-TMP I-AM-TMP B-A0 I-A0 I-A0 I-A0 B-A1 B-V O

ANNEX D: Model Result with Different Parameter Settings

Model No.	Concatenated Word Embedding Component							Bi-LSTM RNN Layer			Dense Layer		F-Score
	Word-level Embedding	Character-level Embedding			fastText-level Embedding								
	Vector Size	Vector Size	Bi-LSTM RNN		Vector Size	Bi-LSTM RNN							
			Units	Activation		Units	Activation	Layers	Units	Activation	Units	Activation	
Using Different Vector Dimensions for the Word, Character and fastText-level Embeddings													
1	100	100	128	tanh	100	32	tanh	1	32	tanh	64	NONE	67.4
2	50	50	128	tanh	50	32	tanh	1	32	tanh	64	NONE	70.2
3	32	32	128	tanh	32	32	tanh	1	32	tanh	64	NONE	72.8
4	16	16	128	tanh	16	32	tanh	1	32	tanh	64	NONE	75.7
5	**5**	**20**	**128**	**tanh**	**5**	**32**	**tanh**	1	**32**	**tanh**	**64**	**NONE**	**81.2**
Using Different Bi-LSTM RNN Units for the Character-level Embedding													
6	5	20	256	tanh	5	32	tanh	1	32	tanh	64	NONE	73.8
7	5	20	512	tanh	5	32	tanh	1	32	tanh	64	NONE	71.4
8	5	20	64	tanh	5	32	tanh	1	32	tanh	64	NONE	78.3
9	5	20	32	tanh	5	32	tanh	1	32	tanh	64	NONE	75.7
Using Different Bi-LSTM RNN fastText-level Embedding													
10	5	20	128	tanh	5	64	tanh	1	32	tanh	64	NONE	78.7
11	5	20	128	tanh	5	128	tanh	1	32	tanh	64	NONE	74.4
12	5	20	128	tanh	5	256	tanh	1	32	tanh	64	NONE	71.1
13	5	20	128	tanh	5	512	tanh	1	32	tanh	64	NONE	67.8
14	5	20	128	tanh	5	16	tanh	1	32	tanh	64	NONE	77.8
Using Different Units for the Bi-LSTM RNN Layer Component													
15	5	20	128	tanh	5	32	tanh	1	64	tanh	64	NONE	78.5
16	5	20	128	tanh	5	32	tanh	1	128	tanh	64	NONE	74.6
17	5	20	128	tanh	5	32	tanh	1	256	tanh	64	NONE	71.3
18	5	20	128	tanh	5	32	tanh	1	512	tanh	64	NONE	67.5
19	5	20	128	tanh	5	32	tanh	1	1024	tanh	64	NONE	65.7
Using Multiple Layers for the Bi-LSTM RNN Layer Component													
20	5	20	128	tanh	5	32	tanh	2	32	tanh	64	NONE	75.5
21	5	20	128	tanh	5	32	tanh	3	32	tanh	64	NONE	71.3
22	5	20	128	tanh	5	32	tanh	4	32	tanh	64	NONE	67.5
Using Different Units for the Fully Connected (Dense) Layer Component													
23	5	20	128	tanh	5	32	tanh	1	64	tanh	128	NONE	78.4
24	5	20	128	tanh	5	32	tanh	1	64	tanh	256	NONE	77.5
25	5	20	128	tanh	5	32	tanh	1	64	tanh	512	NONE	72.3
26	5	20	128	tanh	5	32	tanh	1	64	tanh	32	NONE	79.2
27	5	20	128	tanh	5	32	tanh	1	64	tanh	16	NONE	75.6
Using Different Activation Functions													
28	5	20	128	sigmoid	5	32	sigmoid	1	32	sigmoid	64	NONE	73.2
29	5	20	128	relu	5	32	relu	1	32	relu	64	NONE	65.4
30	5	20	128	tanh	5	32	tanh	1	32	tanh	64	tanh	78.6
31	5	20	128	tanh	5	32	tanh	1	32	tanh	64	sigmoid	74.7
32	5	20	128	tanh	5	32	tanh	1	32	tanh	64	relu	71.3

*All the models are run for 50 iterations (ephocs) with mini-batches of size of 4, monitoring the loss at each iteration with *EarlyStopping* set at 5 as its patience, and with Adamax optimizer set at a learning rate of 0.001, the constant for numerical stability (epsilon) set at $\epsilon = 1e^{-7}$, and the norm of gradients clipped with a threshold of 1.0.

ANNEX E: Sample Predictions of the Model

1. [አበበ]Arg1 [ወደ ስራ]ArgM-DIR [ሄደ]Pred [።]O
2. [አበበ]Arg1 [ጉድጓድ ውስጥ]ArgM-LOC [ገባ]Pred [።]O
3. [አበበ]Arg1 [ወደቀ]Pred [።]O
4. [አበበ]Arg1 [እየሮጠ]ArgM-MNR [መጣ]Pred [።]O
5. [አበበ]Arg1 [እየሳቀ]ArgM-MNR [መጣ]Pred [።]O
6. [እዮብ]Arg0 [አበበን]Arg1 [መታው]Pred [።]O
7. [ከበደና አበበ]Arg1 [ከለንደን]ArgM-DIR [መጡ]Pred [።]O
8. [እኔ]Arg0 [ጥሩ ሠው]Arg1 [ነኝ]Pred [።]O
9. [እናንተ]Arg0 [ወሬ]Arg1 [አትስሙ]Pred [።]O
10. [ሠው]Arg0 [ክፉ]Arg1 [ነው]Pred [።]O
11. [ክፉ ሠው]Arg1 [ነው]Pred [።]O
12. [ሰው]Arg0 [ክፉ]Arg1 [ነው]Pred [።]O
13. [ያዕቆብ]Arg0 [አቡሸን]Arg1 [በድንጋይ]ArgM-MNR [መታው]Pred [።]O
14. [ያዕቆብ]Arg0 [አቡሸን]Arg1 [በድንጋይ]ArgM-MNR [ቀጠቀጠው]Pred [።]O
15. [ያዕቆብ]Arg0 [አቡሸን]Arg1 [በድንጋይ ቀጠቀጦ]ArgM-MNR [ገደለው]Pred [።]O
16. [ያዕቆብ]A1 [አቡሸ ጋር]AM-MNR [ተደባደበ]V [።]O
17. [እሱ]Arg1 [ከ 3 ቀን በኋላ]ArgM-TMP [ተወለደ]Pred [።]O
18. [እሱ]Arg1 [ከ 3 ቀን]ArgM-DIR [በኃላ]ArgM-MNR [ተወለደ]Pred [።]O
19. [እሱ]Arg1 [ጥር ፭ ቀን ፩፱፱፭ ዓ/ም]ArgM-TMP [በመኪና አደጋ]ArgM-MNR [ሞተ]Pred [።]O
20. [ያዕቆብ]Arg0 [ዱላዉን]Arg1 [ለዮናስ]Arg2 [አቀበለው]Pred [።]O
21. [የጥበብ መጀመርያ]Arg0 [እግዚአብሔርን መፍራት]Arg1 [ነው]Pred [።]O

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by:

Name: _____

Signature: _____

Date: _____