TRANSITION BASED DEPENDENCY PARSER FOR AMHARIC LANGUAGE USING DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Research shows that attempts done to apply existing dependency parser on morphologically rich languages including Amharic shows a poor performance. In this study, a dependency parser for Amharic language is implemented using arc-eager transition system and LSTM network. The study introduced another way of building labeled dependency structure by using a separate network model to predict dependency relation. This helps the number of classes to decrease from 2n+2 into n, where n is the number of relationship types in the language and increases the number of examples for each class in the dataset. In addition to that, the treebank for training and evaluation is constructed in such away that, morphological features are separately treated as a standalone word to have their own part of speech tag and dependency relation with other words in the sentence. This helps the system to capture morphological richness of the language. The proposed system is evaluated and achieves 91.54% and 81.4% unlabeled and labeled attachment score respectively.

1 INTRODUCTION

Amharic is one of the Ethiopian languages, grouped under Semitic branch of Afro-Asiatic language. Amharic serves as the official working language of the Federal Democratic Republic of Ethiopia and it is the second most spoken Semitic language in the world after Arabic with a total speakers of around 22 million, as per the census of 2007 (contributors, 2018). Even though many works have been done to develop language-processing tools, they are mainly concentrated on English, European and East-Asian languages (Reut Tsarfaty & Nivre, 2013). In spite of large number of speakers, Amharic is one of the under-resourced languages that needs many linguistic tools to be developed in general and dependency parser in particular. Researches like Maltparser attempted to develop universal dependency parser that can be used to parse sentences from different languages without making language specific modification to the system (NIVRE J., 2007). Even though it is language independent dependency parser, few languages are incorporated in to Maltparser, such as English, Dutch, Turkish and Italian (NIVRE J., 2007). However, parsing in morphologically rich languages is different in which, dependency relation exists between not only orthographic words (space-delimited tokens) but also relation within the word itself (Reut Tsarfaty & Nivre, 2013). For example, in Amharic, an orthographic word combines some syntactic words into one compact string. These words can be function words such as prepositions, conjunctions and articles. This makes an orthographic word functioning as a phrase, clause or sentence (Binyam Ephrem Seyoum, 2018). Yoav Goldberg (2009) used MST (maximum spanning tree) and Maltparser dependency parsers on Hebrew treebank. As the authors noted in their study, both systems were poor in representing morphological richness of the language. Later, Yuval marton (2012) presented a mechanism to improve the performance of maltparser for parsing Arabic sentence by adding lexical, inflectional and part of speech tags. However, the system is unable to show significant improvement relative to the original parser. Gasser (2010) tried to develop a dependency grammar for Amharic using XDG (extensible dependency grammar). Unfortunatly thier work was not tested on corpus data. This aim of this study is to design and develop dependency parser for Amharic language that

uses the rule of arc-eager transition system. The main contributions of this paper are: (i) designing a neural network model with LSTM that gives a good accuracy, (ii) introducing a separate model for building labeled dependency structure, (iii) developing a dependency parser that can produce unlabeled and labeled dependency structures for Amharic sentences.

2 Dependency Parsing

Dependency parsing is one of the techniques for analyzing syntactic structure of a sentence. It represents the head-dependent relationship between words in a sentence classified by functional categories or relationship labels (Rangra, 2015). Dependency parsing provides a clear predicate argument structure, which is useful in many NLP applications that make use of syntactic parsing (Nivre, 2010). As shown in figure 1, the direct link from the head-word

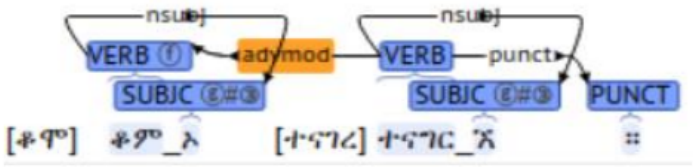

Figure 1: Dependency structure of an Amharic sentence (adopted from Biniyam et.al)

to the dependent-word makes dependency parsing easily identify which word modifies or compliments which other word in a sentence. This property makes it suitable for languages with flexible word order (Nivre, 2010).

3 Long-short term memory in natural language processing

Recurrent neural network was created in the 1980's but has just been recently gained popularity (Kanchan M. Tarwani, 2017). Unlike feedforward neural networks, recurrent neural networks (RNN) have a cyclic or recursive connection. This cyclic connection allows information to be passed from one-step of the network to the next, makes it more powerful for modeling inputs of sequences. Due to this behavior, the network has been useful in natural language processing tasks in general(Kanchan M. Tarwani, 2017). Although the connection of networks in a recursive way is useful to model sequences, it brought a problem of long-term dependency.

LSTMs are explicitly designed to avoid the long-term dependency problem. All RNNs have the form of chain of repeating modules of neural network connected by single 'tanh' layer. When we come to LSTM, it also has this repeating neural network connected in a different way, LSTM has four layers interacting in a very special way, instead of having a single 'tanh' layer. Because of this, LSTM have the ability to remove or add information to the cell state by the help of carefully regulated by structures called gates.

4 Methodology

The proposed Amharic dependency parser consists of two phases, which are training or learning phase and parsing phase, see figure 2.

In the first phase, two network models are trained on Amhraic treebank to predict arc-eager transition actions and to predict relationship type that exists between head-word and dependent-word. The second phase is parsing phase in which the trained deep learning models are used for constructing a dependency structure to an Amharic sentence.

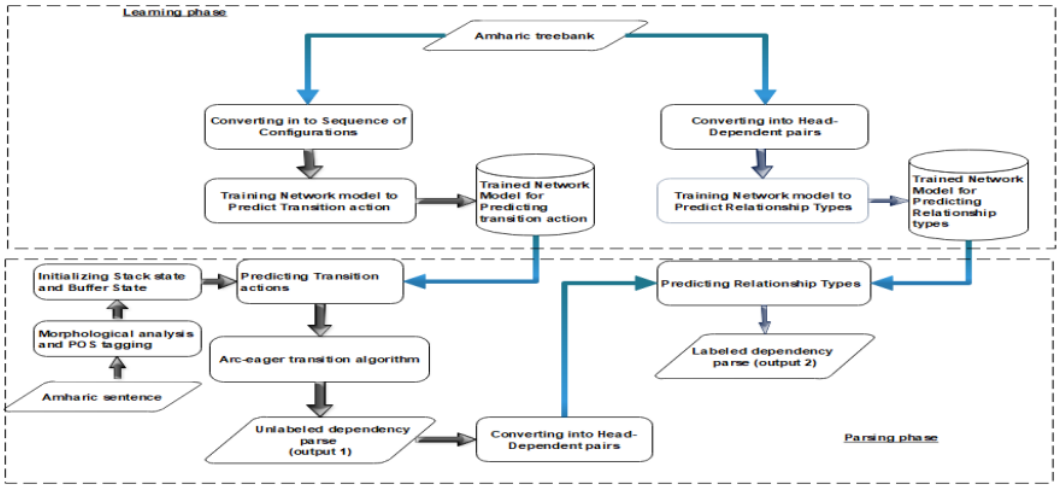

Figure 2: Architecture of the system

4.1 Learnign transition actions

In this step, a network model, see figure 3, is trained on Amharic treebank for it to learn the pattern of arc-eager transition actions.

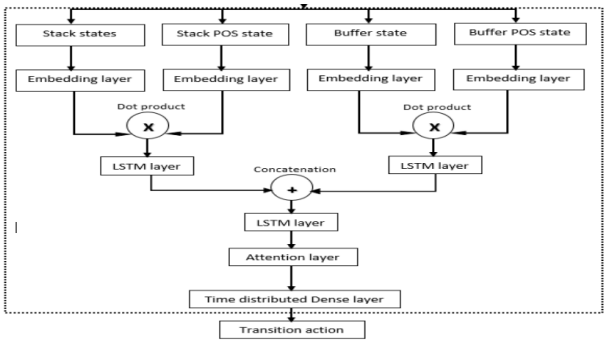

Figure 3: Network model of learning transition action

Here, we designed an algorithm, see Appendix A, to convert Amharic treebank into sequence of arc-eager transition configurations. A sample configuration is shown in table 1. These configurations are used as input to train the network model for predicting one of the four (shift, left-arc, right-arc and reduce) arc-eager transition actions.

4.2 Learning relationship type

Previous studies on transition based dependency parsing have shown the construction of both unlabeled and labeled dependency structures in one process (NIVRE J., 2007)(D. Chen, 2014)(Dyer, 2015). Which means they build labeled dependency structure in a similar way with unlabeled structure. This makes number of transitions to grow from four to 2(n)+2, where n is the number of relationship types in the language. In other words, each left-arc and right-arc have n alternatives. Due to this, the number of examples for each class in the dataset become smaller. The problem gets severe for languages with small treebank like Amharic. In addition to that, as we observed in the experiment, the relationship type that holds between a dependent-word and head-word is not dependent on the transition configuration rather it is dependent on the POS tag of the words. For instance, the relationship type between **‘ካብር’** and **‘ኡ’** in the sentence **“ካብር-ኡ-ተገደል-ኧ።”**

Table 1: Sample instances of transition configuration.

Stack state	Stack POS state	Buffer state	Buffer POS state	Transition
[root]	[root]	[ሳህን, ኡ, ን, ወረወር, ኡ ,ት,::]	[NOUN, DET, ACC, VERB, SUBJC, OBJC,PUNCT]	Shift

Table 2: Sample inputs for relationship type prediction. (with 'expl' relation)

dependent-word	POS tag	head-word	POS tag	Relation
አበበ	PROPN	በል	VERB	nsubj
በሶ	NOUN	በል	VERB	obj
በል	VERB	ROOT	ROOT	root
አ	SUBJC	በል	VERB	expl

is 'det' because of the part of speech tag of the word 'ኡ' is DET. To overcome this problem, we designed a separate network model, see figure 5, to train relationship type that exist between dependent-word and head-word. The network model accepts a pair of dependent-word and head-word along with thier POS tag, see Table 2.

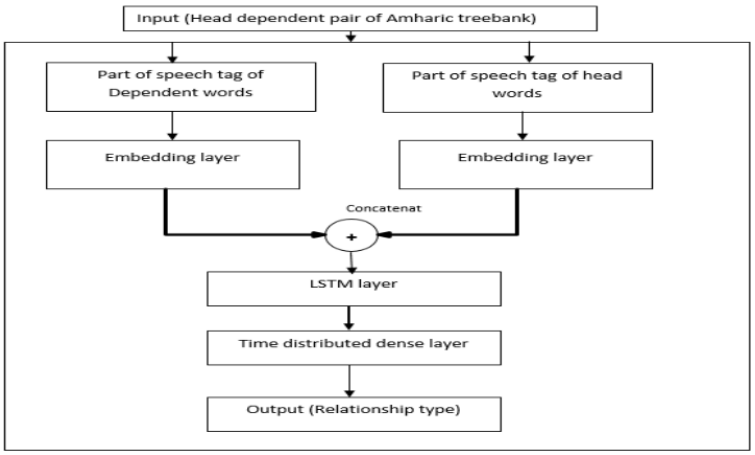

Figure 4: Network model for learning relationship type

4.3 Parsing phase

The purpose of this phase is to construct a dependency relation for a given new sentence by the help of trained network models. The steps followed in parsing phase are depicted in figure 2. The inputs for system are Amharic sentences, which are morphologically analyzed and tagged with POS, see Appendix B. The system uses this information to predict sequence of transition action and builds unlabeled dependency structre using arc-eager algorithm. After that, it uses the unlabeled dependency tree

to predict the dependency relation between the dependent-word and head-word in the sentence and produces labeled dependency tree.

5 Experimental results

5.1 Datasets

For the experiment we used a treebank with 1574 sentences among them 500 sentences are constructed by the researcher with the help of linguistic experts at Jimma University and the rest are collected from Binyam Ephrem Seyoum (2018). All of the sentences are collected from fictions and other types of novel books for the sake of relative structural correctness.

5.2 Evaluation of transition action prediction

When we convert the treebank to transition configuration we get 26,242 configurations, among this we used 70% (with 1,102 sentences having 17,888 configurations) for training and 30% (with 472 sentences having 8,353 configurations) for testing. The dataset has 1475 unique words. We used additional key words, such as –OOV- to indicate out of vocabulary words; -PAD- to indicate padding (zero) and 'root' to indicate the root word. Due to this, we got 1478 unique words. This makes the embedding layer for stack state and buffer state to have a dimension of 1478. On the other hand, the dataset has 26 unique part of speech tags including '-OOV-', '-PAD-' and 'root'. This makes the embedding layer for POS state of the stack and buffer to have 26 dimensions. For the LSTM layer, we used 256 output dimensions and 128 dimensions for attention layer. For the output layer, we used four output dimensions for shif, left-arc, right-arc and reduce. We used Adam optimization technique with learning rate 0.01 and batch-size 256. In addition, the training iterates for 50 epochs. Using this experiment the model correctly predicts 94.7% transitions.

5.3 Evaluation of relationship type prediction

This experiment also used similar data to the pervious experiment converted into list of head-dependent pairs. When the dataset is converted in to head dependent pairs, we got 15,534 head-dependent pairs. Among this, 70% (with 10,874 head dependent pairs) are used for training the model and 30% (with 4,660 head-dependent pairs) are used for testing the model. We used the embedding layer and LSTM layers with 256-output dimensions. We used 36 relationship types including 'root' (to indicate the root word) and '-PAD-' (to indicate padded zero) for the output dimension. Adam optimization is used to train the network model with 0.01 learning rate, 100 epochs and 256 batch-size. Out of 4660 head-dependent pairs, the model correctly predicts a dependency relation for 81.4% head-dependent pairs.

5.4 Attachment score

Unlabeled attachment score is the accuracy of the system to attach the correct head-word with the dependent word without considering the relationship type. On the other hand, labeled attachment score is the accuracy of the system to attach the correct head-word to the correct dependent-word and giving the correct relationship type. The parser is evaluated on 4,660 head dependent pairs, which are 30% of the total dataset and scores 91.54% accuracy for unlabeled attachment and 81.4% accuracy for labeled attachment.

5.5 Comparison of the proposed system with MaltParser

Performance comparision is made between the proposed system and the latest version of maltparser (maltparser 1.9.1). As shown in table 3, the proposed system out

Table 3: Comparision between Maltparser and the proposed system.

Parser	Unlabeled Attachement Score	Labeled Attachement Score
Maltparser	84.2%	69.1%
The proposed system	91.54%	84.1%

performes maltparser both in unlabled and labeled attachement scores by 7.3% and 15% respectively.

6 Conclusion

In this paper, we implemented a dependency parser system for Amharic language. The parser is developed based on arc-eager transition action and for predicting relationship types. The introduction of the second newtwork model is to increase the number of examples for each class (relationship types) from the tree bank and increase the accuracy of labeled attchement score.

The system is evalaluated on Amharic treebank and results 91.54% and 82.4% for unlabeled and labeled attchement score respectively. From the experiment, we noted that, the system can perform better even by increasing the size of the treebank.

References

Baye Yimam Mekonnen Binyam Ephrem Seyoum, Yusuke Miyao. Universal dependencies for amharic,. LRCE, 2018.

Wikipedia contributors. Amharic, 2018. URL https://en.wikipedia.org/w/index.php?title=Amharic&oldid=870444041.

C. Manning D. Chen. A fast and accurate dependency parser using neural networks. In Association for Computational Linguistics, 2014.

Ballesteros M. Ling W. Matthews A. Smith N.A Dyer, C. Transition-based dependency parsing with stack long short-term memory. In Association for Computational Linguistics, 2015.

M. Gasser. a dependency grammar for amharic. School of Informatics and Computing Indiana University, 2010.

Swathi Edem Kanchan M. Tarwani. Survey on recurrent neural network in natural language processing. International Journal of Engineering Trends and Technology, 2017.

J. Nivre. Dependency parsing. Language and Linguistics Compass, 2010.

NILSSON J. CHANEV A. ERYİGİT G. KÜBLER S. MARSI E. NIVRE J., HALL J. Maltparser: A language-independent system for data-driven dependency parsing. In Natural Language Engineering, 2007.

R. Rangra. Basic parsing techniques in natural language processing. International Journal of Advances in Computer Science and Technology, 2015.

Sandra Kübler Reut Tsarfaty, Djamé Seddah and Joakim Nivre. Parsing morphologically rich languages. 2013.

Michael Elhadad Yoav Goldberg. Hebrew dependency parsing: Initial results. In ssociation for Computational Linguistics, 2009.

Owen Ranbow Yuval marton, Nizar Habash. Dependency parsing of modern standard arabic with lexical and inflectional features. In ssociation for Computational Linguistics, 2012.

A Algorithm for converting treebank into sequence of arc-eager transition configuration

```
Algorithm convertTreebankToTransitionSequences(treebank)
   totalStackState, totalBufferState, transitionSequence=[], [], []
   FOR EACH sentence in treebank DO
      // initilizing stack state and buffer state
      stackState=[“root”]
      bufferState=[sentence]
       transition=0
       WHILE bufferState is not empty Do
          totalStackstate.ADD(stackState)
          totalBufferState.ADD(bufferState)
          IF stackState(TOP) is head (bufferState(FIRST)) THEN
             transition=RIGHTARC
             stackState.PUSH(bufferState(FIRST))
             REMOVE(bufferState(FIRST))
          ELSE IF bufferState(FIRST) is head(stackState(TOP)) THEN
             transition=LEFTARC
             stackState.POP()
          ELSE IF stackState(TOP) has no dependent at bufferState THEN
             transition=REDUCE
             stackState.POP()
          ELSE
             transition=SHIFT
             stackState.PUSH(bufferState(FIRST))
             REMOVE(bufferState(FIRST))
          END IF
          transitionSequence.ADD(transition)
       END WHILE
    END FOR
END algorithm
```

B Sample input for the parsing system

Index	Word	POS tag
1	ልጅ	NOUN
2	ኡ	DET
3	የ	PART
4	ገደል	VERB
5	ኧ	SUBJC
6	ው	PRON
7	እባብ	NOUN
8	ትልቅ	ADJ
9	ን	AUX
10	ኧው	PRON
11	።	PUNCT