TRANSITION BASED DEPENDENCY PARSER FOR AMHARIC LANGUAGE USING DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Researches 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 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 in that it 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). On the other hand, 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 languages. Later, Yuval marton (2012) presented a mechanism to improve the performance maltparser for parsing Arabic sentence by adding lexical, inflectional and part of speech. However, the system unable to show significant improvement relative to the original parser. Gasser (2010) ried to develop a dependency grammar for Amharic using XDG (extensible dependency grammar). The limitation of this work is that it does not consider the morphology of the language and the system is not tested on corpus dat. 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 contribution 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 structure 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). The advantage of dependency parsing is that it provides a clear predicate argument structure, which is useful in many NLP applications that makes use of syntactic parsing (Nivre, 2010). As shown in figure

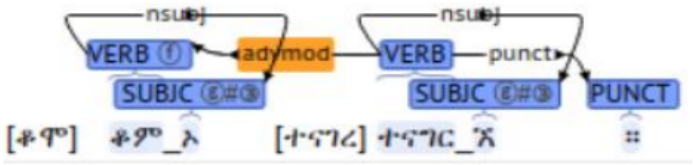

Figure 1: Dependency structure of an Amharic sentence (adopted from Biniyam et.al)

1, the direct link from the headword to the dependent word makes dependency parsing easily identify which word modifies or compliments which word in a sentence. This property makes it suitable for languages with flexible word order (Nivre, 2010).Approaches to perform dependency parsing are broadly grouped into grammar based and data driven approaches. In grammar-based approach, constructing dependency structure rely on explicitly defined formal grammar. In such approaches, parsing is defined as the analysis of a sentence with respect to the given grammar (Nivre, 2010). Data driven approaches for dependency parsing uses a statistical method to build the dependency structure. In such approaches, the parser is trained on labeled dataset to learn the pattern of dependency relation, latter it uses its memory to parse new sentences. Data driven dependency parser is further grouped in to transition based and graph based approaches. In transition-based dependency parsing, the parser uses transition systems to build a dependency structure. In this method, statistical model is trained on sequences of transition configurations or states for predicting transition action for new configuration (Nivre, 2005). Then the parser uses the predicted sequence of transition actions as a guideline to build a dependency structure for a given sentence (Nivre, 2005).

3 Amharic language

Amharic is a Semitic language spoken in Ethiopia. It is the second wide mostly spoken Semitic language in the world next to Arabic (Martha Yifru, 2009). Amharic exhibits a root-pattern morphological phenomenon, in which the root can be combined with a particular prefix or suffix to create a single grammatical form or another stem (Martha Yifru, 2009)(Demeke & Getachew, 2016). An Amharic orthographic word can act as a full-fledged sentence(Binyam Ephrem Seyoum, 2018)(Gasser, 2010). For example, as shown in figure 2, **አልመጣችም** (she did not come) can be segmented in to (**አል-መጣ-ች-ም**). From this, we can understand that clitics in an Amharic sentence can have their own POS and can act as a separate token with a dependency relation with their host word.

4 Long-short term memory in natural language processing

Recurrent neural network was created in the 1980's but have just been recently gained popularity (kanchan m. tarwani, 2017). Unlike feedforward neural networks, recurrent

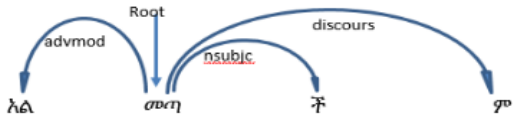

Figure 2: Dependency relation in an Amharic sentence

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. RNN might be able to connect previous information to the present state, unfortunately, as the gap grows RNN become unable to learn to connect the information (Hochreiter, 1997). In other words, the conventional RNN error signals flowing backward in time to either explode or vanish. The first problem makes the weights of the network to oscillate and the later one makes the network unable to learn at all. These problems are called vanishing and exploding gradient problems. Long-short-term memory (LSTM) is presented by Hochreiter (1997) to overcome the problem of long-term dependency of RNN.

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.

5 Methodology

The proposed Amharic dependency parser consists of two phases, which are training or learning phase and parsing phase, see figure 3.

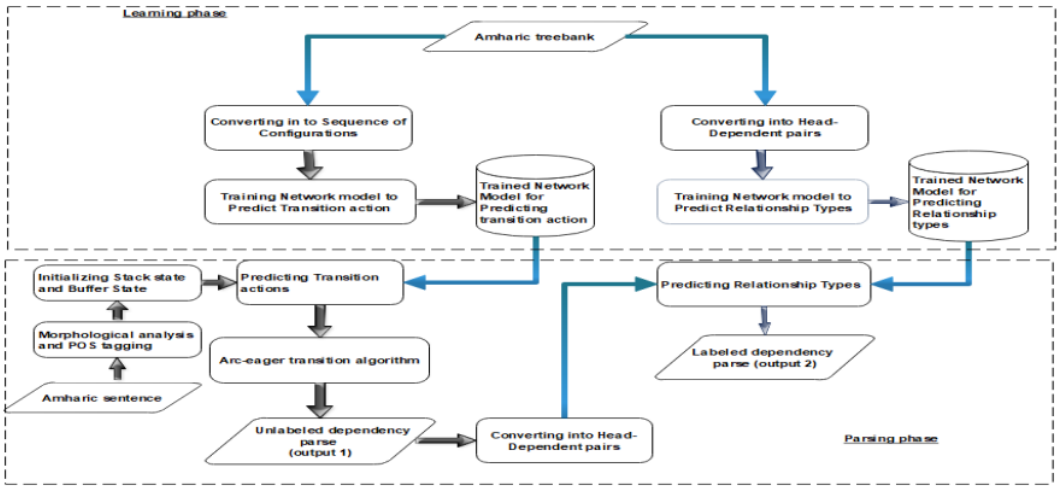

Figure 3: Architecture of the system

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

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

5.1 Learnign transition actions

In this step, a network model, see figure 4, is trained on Amharic treebank to learn the pattern of arc-eager transition actions.

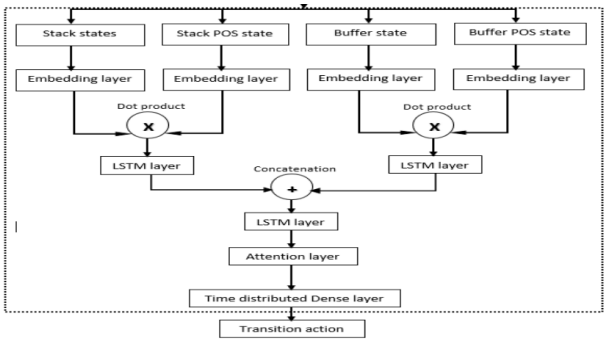

Figure 4: Network model of learning transition action

For doing this we desgined an algorithm that converts Amharic treebank into sequence of arc-eager transition configurations, see Appendix A. A single configuration have five elements (stack state, stack part of speech tag state, buffer state, buffer part of speech tag state, and transition action)m, see Table 1. Each inputs then embedded to make a dense representation. That means words that are similar or used in similar grammatical places will have a closer numerical value.

After each inputs are embedded, related inputs (stack of words state with stack of words POS state and buffer of words state with buffer of words POS state) are scalar multiplied using equation 1 and equation 2.

$$sWPi = sWi \,.\, sTi \quad \ldots\ldots\ldots\ldots \quad (1)$$

$$bWP = bWi \,.\, bTi \quad \ldots\ldots\ldots\ldots \quad (2)$$

Where sWi is a word in the stack, sTi is the corresponding part of speech tag of the word in the stack, bWi is a word in the buffer and bTi is the corresponding part of speech tag of the word in the buffer. Before concatenating the stack state and buffer

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

Dependentword	POS tag	Headword	POS tag	Relation
አበበ	PROPN	በል	VERB	nsubj
በሶ	NOUN	በል	VERB	obj
በል	VERB	ROOT	ROOT	root
አ	SUBJC	በል	VERB	expl

state, we used LSTM layer to make information to be connected from one-step to the next in the stack and buffer states.

f t = sigmoid (w f . [h t-1 , sWpt] + b f)
C t = (f t * c t-1) + (i t * c t ’)
I t = sigmoid (w i . [h t-1 , sWpt] +b i)
C t ’=tanh (w c . [h t-1 , sWpt]+ b c)
O t = sigmoid (W o . [h t-1 , sWpi] +b o)
h t = O t * tanh(C t)

Where wf, wi, wc and wo are the weight matrix for each layer, bf, bi, bc and bo are the bias for each layer; sWpt is the input value (dot product of word and corresponding part of speech tag) at time t and ht-1 is the previous output of the LSTM layer. The outputs of the two LSTM layers (LSTM layer for stack state and LSTM layer for buffer state) are then concatenated and passes through the last LSTM layer. This helps the information to be further connected from stake state to buffer state. Since arc-eager transition system keeps the word at stack before all its dependents at the buffer are seen, the transfer of information from stack state to buffer state helps the network to incorporate this phenomenon. The final layer of the network model is fully connected dense layer. It is applied to each time step of the outputs from the previous layer to decide the final output. The output of the network is one of the arc-eager transition actions (shift, left-arc, right-arc or reduce).

5.2 Learning relationship type

Previous studies on transition based dependency parsing constructs both unlabeled and labeled dependency structure In one process (NIVRE J., 2007)(D. Chen, 2014)(Dyer, 2015). That means they use 2n+2 classes during training the classifier, where ‘n’ is the total number of relationship types in the language. 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. In other words, each left-arc and right-arc have n alternatives, where n is number of relationship types in the language. This method hhas a problem in that the number of examples for each class fromhe dataset to be smaller. The problem becomes severe if the size of the dataset is samll. Besides that, as we observed in the experiment, the relationship type that holds between a dependent word and headword is not dependent on the transition configuration. That is because, transition configurations only give an information about which action to select among the four transition actions. On the other hand, the type of dependency relationship can be determined by using the part of speech tag of the dependent and headword. For example, the relationship type between ‘ነብር’ and ‘ኡ’ in the sentence ‘‘ነብር-ኡ-ተገደል-ኧ።’’ is ‘det’ because of the part of speech tag of the word ‘ኡ’ is DET. To ovvercome this problem we designed a separate network model, see figure 5, to train relationship type that exist between dependentword and headword. The network model accepts a pair of dependentword and headword along with thier POS tag, see Table 2.

The relationship between the dependent word ‘አ’ and the headword ‘በል’ is ‘expl’. The relationship type ‘expl’ is used if the subject is explicitly indicated in the sentence, in our case, for instance the word ‘አበበ’ in table 2. But if the subject is not stated

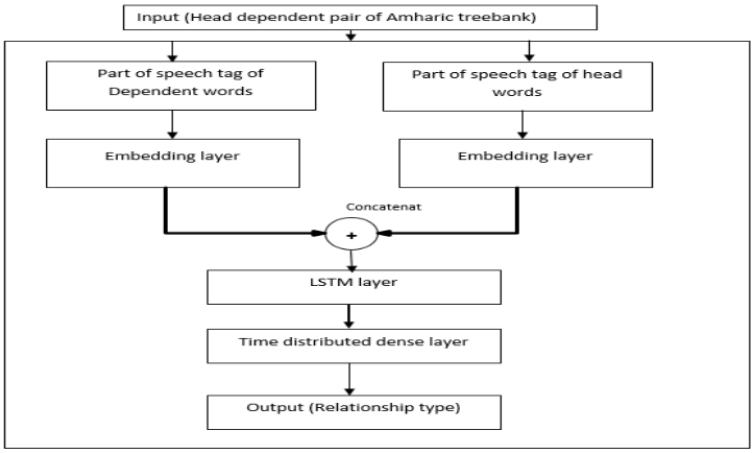

Figure 5: Network model for learning relationship type

Table 3: Sample inputs for relationship type prediction

Dependentword	POS tag	Headword	POS tag	Relation
በሶ	NOUN	በል	VERB	obj
በል	VERB	ROOT	ROOT	root
ኦ	SUBJC	በል	VERB	nsubj

the subject marker will be used as a subject, see table 3. To handle we used LSTM network, which have an ability to pass information to the next state.

5.3 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 4. The inputs for system are Amharic sentences, which are morphologically analyzed and tagged with POS. The system uses thiese informationto 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 dependentword and headword in the sentence and produces labeled dependency tree.

6 Experimental result

6.1 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.

6.2 Evaluation of transition action prediction

When we convert the treebank to transition configuration we got 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

Table 4: 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%

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.

6.3 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 256-output dimension. 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 learning rate 0.01, 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.

6.4 Attachment score

Unlabeled attachment score is the accuracy of the system to attach the correct headword 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 headword 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.

6.5 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 4, the proposed system out performes maltparser.

7 Discussion

The system is evaluated using unlabeled attachment score (UAS) and labeled attachment score (LAS). From the experiment, the system correctly constructs 91.54% and 81.4% unlabeled and labeled attachments respectively. We also made a comparison between latest version of maltparser and the proposed system significantly outfits it.

The challenge observed during this study was the accuracy of the labeled attachment score. From experiments, we observed that the amount of dataset used in the ex-

periment has a contribution to the problem. Since the number of relationship types, which are 36, are greater relative to the number of transitions, which are four, labeled dependency parse needs more number of dataset to improve its accuracy and label with the performance of unlabeled attachement score.

8 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 increas 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. From the experiment, we noted that, the system can perform even better 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.

G. A. Demeke and M. Getachew. manual annotation of amharic news items with part of speech tags and its challenges. ELRC working papers, 2016.

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.

Jürgen Hochreiter, Sepp Schmidhuber. Long short-term memory. Neural computation, 1997.

swathi edem kanchan m. tarwani. Survey on recurrent neural network in natural language processing. International Journal of Engineering Trends and Technology, 2017.

Wolfgang Menzel Martha Yifru. Amharic part-of-speech tagger for factored language modeling. In International conference RANLP, 2009.

J. Nivre. Two strategies for text parsing. SKY Journal of Linguistis, 2005.

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)
   totaStackState, totalBuffeState, 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
```