Jurnal Linguistik Komputasional

JLK
Vol 3, No 1
Maret 2020

Dense Word Representation Utilization in Indonesian Dependency Parsing

Arief Rahman, Ayu Purwarianti

School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Jl. Ganesha No. 10, Bandung 40132
[1]23516008@std.stei.itb.ac.id
[2]ayu@stei.itb.ac.id

***Abstract*— Available Indonesian dependency parsers can be considered worse than other languages' parsers that have been researched thoroughly. Currently, Indonesia dependency parsers can't reliably parse sentences with gerund(s) and/or ellipsis correctly. This is because of the sparse feature representation that causes difficulty in parsing these types of sentences. In this research, dense representation is proposed for Indonesian dependency parser. The use of dense word representation may allow better generalization and gives more information regarding the words to be parsed, which allows a more accurate parsing. The scope of the dependency parsing in this research is limited to well-formed Indonesian sentences, using the local transition-based parsing. Based on our experiments, we found that using word embedding instead of sparse word representation increases parsing accuracy significantly.**

***Keywords*—dependency parsing, neural network, word embedding, Indonesian parser, gerund, elipsis**

I. Introduction

Dependency parsing is an important syntactic task in natural language processing. Dependency parsing is an NLP task used to determine the syntactical structures of the tokens/words in a sentence. The task has recently seen a significant development, especially with the development of the Universal Dependency corpus [1], which allows under-resourced languages to be researched with little difficulty.

Even so, recent studies in Indonesian dependency parsers has not seen much traction. Most of the studies have not used the state-of-the-art parsing technique. Nizami & Purwarianti [2] used a modified graph-based algorithm. Kuncoro [3] used an ensemble method using transition-based parsing. This line of work is then continued by Rahman & Purwarianti [4].

All parsers created by the studies described before, however, have not been able to parse several types of sentences. The first type is sentence containing gerund (a verb that acts as a noun). This type of sentence is difficult to be parsed because the parser often mistakes it as the root of the sentence. The second type is sentence with ellipsis (deletion of repetitive phrase in connected clauses). The parser often has difficulty on parsing the ellipsis clauses.

Rahman & Purwarianti [4] hypothesized that the reason for this parsing error is because of the feature representation. The parsers used by Rahman & Purwarianti, MaltParser and MSTParser, use a one-hot, sparse representation. This representation most of the time has a large feature space and often difficult to generalize because of its discrete property. The one-hot representation also does not contain any context information, which is needed for syntactical tasks like parsing.

In this paper, we propose in using dense representation for Indonesian dependency parser. Dense representation, also commonly known as distributed representation, can represent a feature from one-hot representation with much less feature space and uses real numbers, which are easier to be generalized by machine learning algorithm. The dense representation also inherently contains the contextual information of the feature, which is significantly needed for dependency parsing.

This paper is structured as follows. Section 2 describes the background and relevant works related to dependency parsing, dense representation (particularly word embedding), and dependency parser that uses dense representation. Section 3 describes our proposed dependency parser that uses dense representation. Section 4 describes the dataset used for training the embedding and the dependency parser. Section 5 describes the experiment results and our analysis regarding the experiment results. We limit the scope of dependency parsing to greedy transition-based parsing for well-formed Indonesian sentences.

II. Background & Related Works

A. Dependency Parsing

Dependency parsing is a syntactical parsing task that determines how each token modifies another token in a sentence. The result of this is a dependency graph, which is represented as a directed graph. Each vertex represents a token in the sentence, while each edge represents a dependency relation from the head token to modifier token. Fig. 1 shows an example of dependency graph for Indonesian sentence *Dugaan itu tidak meleset* (That conjecture does not miss).

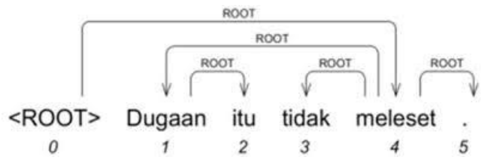

Fig. 1. Kiperwasser & Goldberg neural network parsing architecture

We focus the scope of our study on greedy transition-based parsers [5]. A transition-based parser uses a state machine to parse sentences. The parser used a sequence of transitions to move the parsing state into a state that shows a complete dependency tree. A transition-based parser uses a transition system, which consists of the following four elements:

1. A list of possible states. The most-used representation for a parsing state is a list (*buffer*) containing unprocessed tokens (*B*), a stack containing currently processed tokens (*S*), and a list containing dependency relations created (*R*). A dependency relation is a triple (*x*, *y*, *r*), where *x* is the child token, *y* is the parent token, and *r* is the dependency relation label.
2. A list of possible transitions. A transition can change the current parsing state, which can add or remove tokens from a list (mostly on *B* or *S*), move tokens from one list to another (from *B* to *S*), and/or add dependency relations (on *R*).
3. A start state, which consists of *B* containing the sentence tokens and special dummy token ROOT, empty *S*, and empty *R*.
4. An end state, which consists of empty *B*, *S* containing only ROOT, and non-empty *R*.

Some of the most used transition systems are the arc-eager [6] and arc-hybrid [7] transition system. Our work uses the arc-hybrid transition system, which has three possible transitions to use:

1. SHIFT $(b_0|B, S, R)$
   $= (B, S|b_0, R)$
2. LEFT$_{\text{rel}}$ $(b_0|B, S|s_1|s_0, R)$
   $= (b_0|B, S|s_1, R \cup \{b_0, s_0, rel\})$
3. RIGHT$_{\text{rel}}$ $(B, S|s_1|s_0, R)$
   $= (B, S|s_1, R \cup \{s_1, s_0, rel\})$

A parsing using the transition-based parser is done by the following steps. First, a start state is created based on the input sentence. After that, at each state, the best transition is determined by using an oracle. The oracle is modelled as a parameterized function that can determine the score for a transition on a parsing state. The oracle is trained using a dependency treebank that has been transformed into a list of states and transitions. The parsing ends when a terminal state is reached. This type of parsing is called greedy parsing because it only considers the best transition for each state. There's another type of transition-based parsing called beam-search transition-based parsing [8], which we will not discuss in this paper.

Algorithm 1 Greedy transition-based parsing
```
1. Input: sentence $s = w_1, w_2, \ldots, w_n$, POS tags $p = p_1, p_2, \ldots, p_n$, oracle $O$ that accepts two inputs: state $c$ and next transition $t$, and outputs a score
2. $c \leftarrow$ INITIAL_STATE($s$)
3. while not TERMINAL_STATE($c$) do
4.     $t_{best} \leftarrow \arg\max_{t \in \text{LEGAL\_STATE}(c)} O(c, t)$
5.     $c \leftarrow$ CHANGE_STATE($c$, $t_{best}$)
6. return TREE($c$)
```

B. Dense Word Representation

Dense representation, also called distributed representation or embedding, is currently one of the most popular feature representations. It is a dense, low-dimensional, and real-valued vector that represents a feature's syntactical properties. A dense representation has two main advantages over one-hot representation for being easier to generalize and faster to process. The dense representation also has contextual information regarding the feature itself, because they are most often trained using a neural network, like word2vec [9], [10]. From this point onward, we will call the dense word representation as word embedding.

We use three types of word embedding in this paper: word2vec [9], [10], wang2vec [11], and fasttext [12]. word2vec trains the word embedding using a neural network that has a log-linear complexity, which allows fast training. It uses no non-linear hidden layer and shared projection layer for all words. Mikolov proposed two architectures for efficient word embedding learning: CBOW (Continuous Bag of Words) and skip n-gram. The CBOW training trains a word embedding by predicting a word (focus word) given the n words around it (context words). The skip-gram training does the opposite by predicting the words around the focus word, given the focus word itself. wang2vec does the same thing as word2vec, with additional positional information embedded during the prediction. This approach allows training a word embedding that is more attuned for syntactical tasks. fasttext is also a derivative of word2vec. However, it also trains a sub-word embedding for each sub-word present in a word. This allows a better coverage of words, which reduces the probability of a word not having a word embedding (OOV, out-of-vocabulary).

C. Dependency Parsing with Word Embedding

There are many ways to incorporate word embedding into dependency parsing. The most common way to do this is by representing the word itself using the word embedding. In this paper, we use an architecture provided by Kiperwasser & Goldberg [13] as our main reference.

Kiperwasser & Goldberg uses a Bi-LSTM to represent a feature vector for each token in the sentence (represented by x_i in Fig. 2). x_i is the concatenation of the token vector $e(w_i)$ and the token's POS vector $e(p_i)$:

$$x_i = e(w_i) \circ e(p_i)$$

Both $e(w_i)$ and $e(p_i)$ are trained along with the parser, but $e(w_i)$ in particular can be initialized with a pre-trained word embedding, which will be fine-tuned while training the parser.

Afterwards, the forward and backward vector for the current token (Vf_i and Vb_i respectively) are created by using the Bi-LSTM network. Both Vf_i and Vb_i are then concatenated to create the final feature vector to represent the token (denoted by V_i in Fig. 2).

$$\begin{aligned} Vf_i &= \mathrm{LSTM}^f(x_{i:n}, i) \\ Vb_i &= \mathrm{LSTM}^b(x_{i:n}, i) \\ V_i &= Vf_i \circ Vb_i \end{aligned}$$

After that, for the current sentence, the feature vectors present for each token is used to represent the state input in a transition-based parser in a separate, multi-layer perceptron (MLP) neural network. Kiperwasser & Goldberg uses the top three tokens on *S* and the first token of *B* to represent the parsing state (Example in Fig. 2 shows the first *the*, *jumped*, and *over* as the top three tokens in *S*, while the second *the* is the top token in *B*). The neural network has one hidden layer with tanh activation function and a softmax denoting the probability of each transition.

$$\begin{aligned} O(c, t) &= \mathrm{MLP}(f_c)[t] \\ \mathrm{MLP}(f_c) &= \mathrm{softmax}(W_2 \,.\, \tanh(W_1 \,.\, f_c + b_1) + b_2) \\ f_c &= Vs_1 \circ Vs_1 \circ Vs_0 \circ Vb_0 \end{aligned}$$

W_1, W_2, b_1, and b_2 are the parameters of the MLP.

The parser uses greedy transition-based parsing with arc-hybrid transition system, with the MLP in the neural network acting as the dynamic oracle. Fig. 2 shows the architecture of the neural network parser.

III. Proposed Architecture for Indonesian Dependency Parser

We extend Kiperwasser & Goldberg's architecture [13] by adding a word embedding based on a Bi-LSTM that accepts a sequence of character embedding as the input. One of the main problems in Kiperwasser & Goldberg's architecture is the lack of OOV handling when representing a word using the Bi-LSTM feature. To overcome this problem, we used another way to represent an OOV word with the word embedding, by using another Bi-LSTM.

The character-based Bi-LSTM accepts a sequence of characters present in the token. Each character in the sequence is represented by a character embedding. The embedding for the token is then created by concatenating the final forward and the backward vector of the character-based Bi-LSTM (*xf* and *xb* respectively). Given an *n*-characters token *w* consisting of characters $c_1, \ldots, c_n$, we can associate each character with character embedding $e(c_i)$, which will then be used on the character-based Bi-LSTM in order to get the token embedding x_w. Like its word counterpart, the character embeddings are also trained along with the parser.

$$\begin{aligned} xf &= \mathrm{CharLSTM}^f(e(c)_{i:n}, n) \\ xb &= \mathrm{CharLSTM}^b(e(c)_{i:n}, n) \\ x_w &= xf \circ xb \end{aligned}$$

In this study, we limit the character-based Bi-LSTM to be used only for OOV words. There are two reasons for this limitation. First, it allows the character-based Bi-LSTM to be attuned particularly to words that rarely appear in a sentence. Second, it allows the word to adapt to the structure of the regular word embedding, especially if the regular word embeddings are initialized from a pre-trained embedding. Fig. 3 shows the architecture for the character embedding.

IV. Dataset

There are two types of dataset used in this study. The first one is used to train the word embedding, while the second one is used to train the dependency parser. The next two sections describe these datasets further.

A. Word Embedding Corpus

To train the word embedding, we use the pre-processed corpus consisting of sentences from the following sources.

1) Wikipedia dump (updated February 1, 2018), containing 5,015,997 sentences.

2) List of sentences from Universal Dependencies 2.1 (only the train and dev sentences), containing 5401 sentences.

3) List of example sentences taken from Alwi's Indonesian grammar book [15], totaling in 2349 sentences.

Below are the pre-processing steps we did to the corpus. These pre-processing steps are necessary to reduce the word vocabulary and get a word embedding matrix with a better representative quality

1) Change all the letters to lowercase.

2) Tokenize the sentences into separate words using NLTK's punkt model. Further tokenization is done by separating the words according to the punctuation marks (e.g. "*paru-paru*" becomes "*paru*", "-", and "*paru*").

3) Change all numerals into special token "_NUM_", after the tokenization in step (2) is finished.

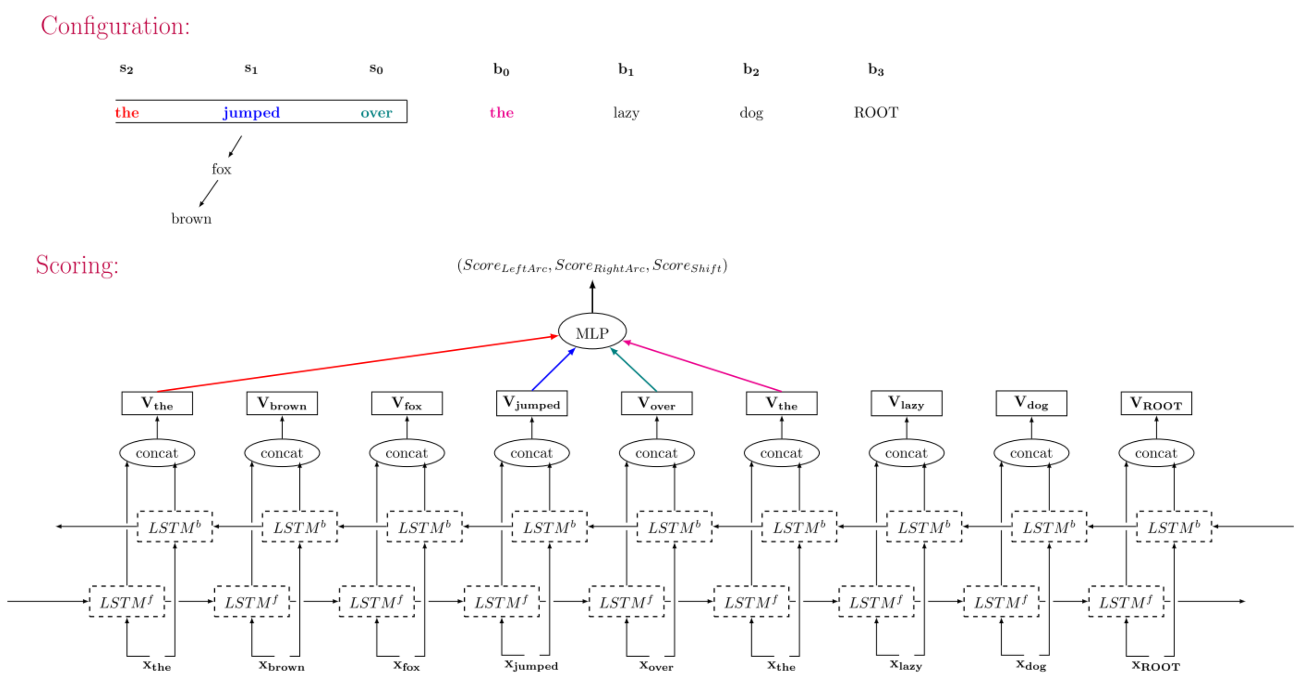

Fig. 2. Kiperwasser & Goldberg neural network parsing architecture

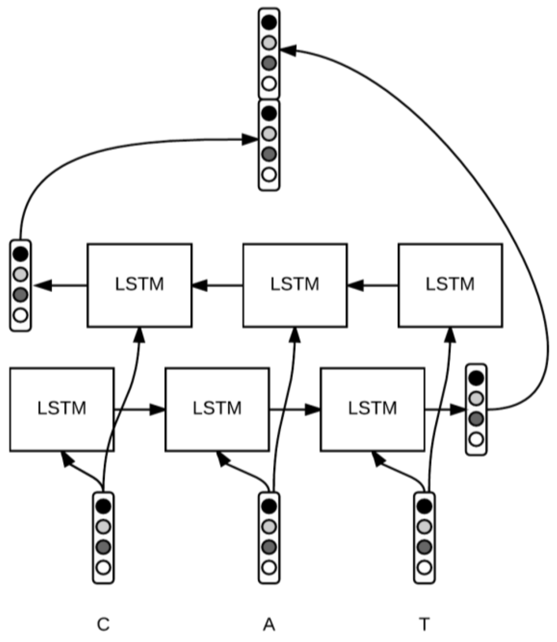

Fig. 3. Character embedding architecture

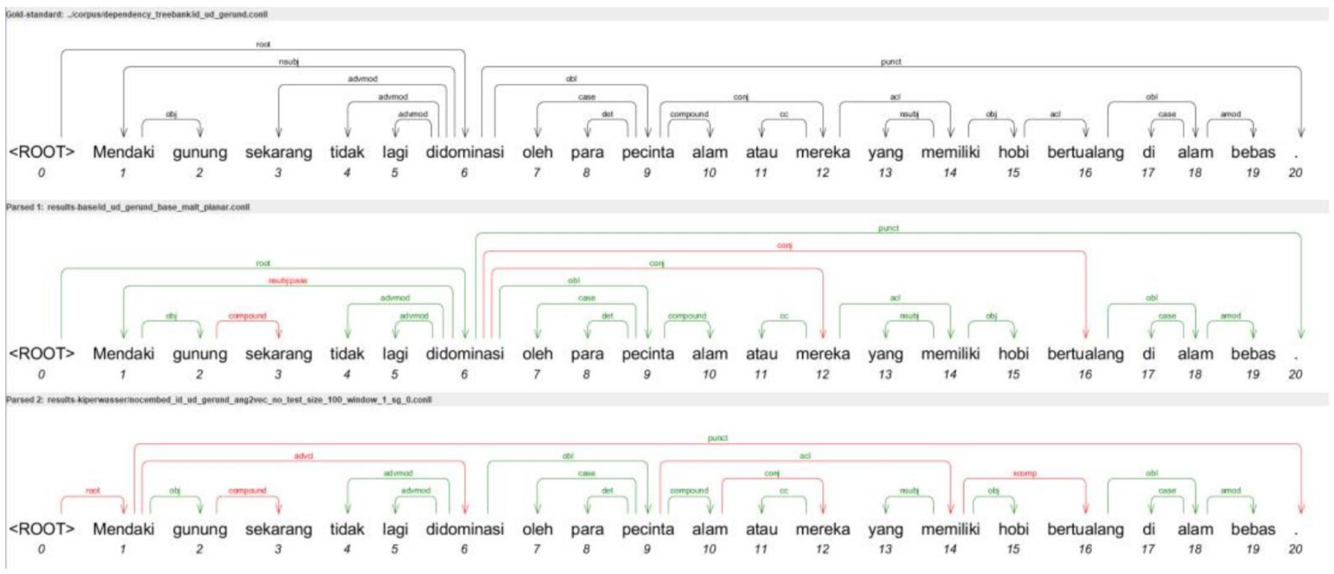
Gold-standard: ../corpus/dependency_treebank/id_ud_gerund.conll
Parsed 1: results-base/id_ud_gerund_base_malt_planar.conll
Parsed 2: results-kiperwasser/nocembed_id_ud_gerund_ang2vec_no_test_size_100_window_1_sg_0.conll
<ROOT> Mendaki gunung sekarang tidak lagi didominasi oleh para pecinta alam atau mereka yang memiliki hobi bertualang di alam bebas .
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
root
nsubj
advmod
obj
obl
case
det
compound
conj
cc
acl
amod
punct
advcl
xcomp

Fig. 4. (a) Gold-standard dependency parsing for sentence *Mendaki gunung sekarang tidak lagi didominasi oleh para pecinta alam atau mereka yang memiliki hobi bertualang di alam bebas* (T/L: Mountain-climbing is not dominated by nature lovers or those who have a hobby of venturing to the wild nature anymore.) (b) Baseline parsing result of the same sentence (c) Parsing result of the same sentence by our proposed approach

B. *Dependency Corpus*

We used the edited and filtered Universal Dependencies 2.1 corpus to train the dependency parsers. We used the train part of the dependency corpus as the training data, and the merged dev and test part of the dependency corpus as the testing data. We also used a manually annotated corpus containing 50 sentences with ellipsis (focused on the anaphoric and cataphoric ellipsis) and 50 sentences with gerund(s) for a separate experiment scenario.

We retagged the POS of each token present in all corpora using our POS tagger, which uses INACL POS scheme. After that, each sentence is reparsed using UDPipe, which uses the exact same dependency corpus (including the dev and test data) as the training data.

V. Experiments

A. *Experiment Scenarios*

We conducted several experiments with various parsing models based on the word embedding used to train the model. We created 36 different models based on the following parameters:

1) The training algorithm: word2vec, wang2vec, and fasttext

2) Training scheme: CBOW and skipgram

3) Context window size: 1, 2, and 5

4) Use character-based embedding for OOV words: yes and no

We also created baseline models based on Rahman & Purwarianti's list of baseline parsers, along with the ensemble model created by them [4]. Below are the baseline parsers used in this study.

1) MaltParser Nivre (arc-eager and arc-standard)

2) MaltParser Covington (projective and non-projective)

3) MaltParser Stack (eager, lazy, and projective)

4) MaltParser Planar

5) MaltParser 2-Planar

6) MSTParser Eisner

7) MSTParser Chu-Liu-Edmonds

8) Ensemble model using Eisner, Chu-Liu-Edmonds, and Planar algorithm

Finally, we compared the performance of the baseline and the neural network parsers on sentences containing ellipsis and gerund. We used the best baseline parser and the best neural network parser in this final scenario. For the sentences with ellipsis, we checked overall results of the parser, while for the sentences with gerund(s), we only check the dependencies between the predicate of the sentence and the gerund word itself.

There are two evaluation metrics used in this study: UAS (Unlabeled Assignment Score) and LAS (Labeled Assignment Score). UAS score is determined by the number of tokens that have the correct parent. LAS score determined by the number of correct parent and dependency label the tokens have.

B. *Experimental Results*

Table I shows the baseline parsers results. We can see that the Planar algorithm from MaltParser produced the parser with the best UAS and LAS (76.91% and 67.80% respectively). This contrasts with Rahman's results [4], which states the graph-based parsing algorithm as the best parser for Indonesian language. However, after checking the errors in the graph-based parser, we found that most of the errors were because of the incorrect POS tags assigned to some words in sentences with more than 1 predicate.

TABEL I
BASELINE PARSER RESULT

No	Baseline Parser	Score	
		UAS	*LAS*
1	MaltParser Nivre arc-eager	76.87%	66.71%
2	MaltParser Nivre arc-standard	74.92%	66.95%
3	MaltParser Covington non-projective	72.64%	65.17%
4	MaltParser Covington projective	72.64%	65.55%
5	MaltParser Stack projective	74.13%	64.36%
6	MaltParser Stack eager	75.00%	67.33%
7	MaltParser Stack lazy	74.51%	67.02%
8	MaltParser Planar	**76.91%**	**67.80%**
9	MaltParser 2-Planar	76.31%	67.73%
10	MSTParser Eisner	74.18%	37.23%
11	MSTParser Chu-Liu-Edmonds	74.02%	35.61%
12	Ensemble Eisner – Chu-Liu-Edmonds - Planar	74.02%	63.45%

We can also see from Table I that all parsers from MSTParser have a low LAS score. We deduced that the low LAS score is caused by MSTParser's first-order feature, which only considers the candidate head and dependent. The lack of context features in the first-order graph-based parser causes a considerably worse performance in labeling dependency than the transition-based parser, which is much richer in feature representation.

Table II shows the neural network parsers results. We can see that the usage of word embedding allows significantly better parsing results, with 5% increase. The usage of word embedding prevents most common mistakes that tend to happen in non-word embedding parsers. Even with several POS tagger errors in the corpora, the neural network parser is still able to achieve a better UAS and LAS scores consistently.

Unfortunately, we couldn't determine how the character-based embedding contribute to parsing accuracy. The neural network parsers that use the character-based embedding perform better and worse at the same time than the baseline parsers, especially on OOV words. The difference in performance is affected by two factors. The first one is the addition of character-based embedding also changes how much the other weights are updated. The second one is how the character-embedding replaces the pre-trained embedding. The pre-trained embedding is created by using the projection matrix (which is actually the weights coming from the input word), while the character-based embedding is created by concatenating two output vectors created by the character-based Bi-LSTM. The representation difference causes the character-based embedding to be treated as noise.

Table III shows the parsing results for sentences with ellipsis and gerund. Using the neural network parser with word embedding increases the parsing accuracy for both types of ellipsis. A surprising result, however, can be seen on the gerund result. In this case, the neural network parser performs considerably worse than the baseline parser. In most cases, the neural network parser almost consistently assigned the gerund as the root of the sentence, especially if the gerund shows before the root verb.

Fig. 4 shows an example of a case where the gerund is parsed incorrectly. In this example, *Mendaki* from *Mendaki gunung* is a gerund and should have created a dependency triple (*Mendaki*, *didominasi*, *csubj*). Instead, our parser created a dependency triple (*Mendaki*, *ROOT*, *root*). Another wrong parsing example (dependency trees not shown in this paper) is *Menurutnya, bersenandung menenangkan jiwa* (According to him, humming soothes your soul). In this case, our parser made a dependency triple (*bersenandung*, *ROOT*, *root*) while it should have been (*menenangkan*, *ROOT*, *root*).

From this result, we can conclude that adding the word embedding does not solve the gerund parsing problem. We found out that the problem lies on the dependency labels. Our current dataset treats all subjects, regardless of their form, as *nsubj* (nominal subject), while they should have been treated as *csubj* (clausal subject). Since there are too few sentences that have gerund(s) in the dataset, it caused difficulties in recognizing a gerund as a nominal instead of as a predicate. The POS tag of gerunds may also contribute to the parsing difficulties, due to it also having a verb POS tag, which is often attributed as the predicate of the sentence. The lack of verb forms that specialize as a gerund (like VBG in Penn Treebank POS tags) makes it difficult for Indonesian parsers to differentiate the gerunds and the predicates.

VI. CONCLUSIONS

We have modified the available Indonesian dependency corpus by using standard INACL POS Tags and adding 50 sentences with gerund and ellipsis. We also created a neural network dependency parser that uses word embedding as one of the main features. We expanded the neural network parser based on Kiperwasser & Goldberg's architecture by adding a character-based embedding.

TABLE I. NEURAL NETWORK PARSERS RESULTS

No	Embedding	CBOW / Skipgram	Window size	Score without character embedding		Score with character embedding	
				UAS	*LAS*	*UAS*	*LAS*
1	word2vec	CBOW	1	81.89%	75.72%	**82.65%**	**76.51%**
2		Skipgram	1	81.81%	75.71%	82.45%	76.06%
3		CBOW	2	82.09%	75.95%	82.17%	75.96%
4		Skipgram	2	82.26%	75.78%	81.94%	75.72%
5		CBOW	5	82.41%	75.93%	82.30%	75.92%
6		Skipgram	5	82.08%	75.81%	82.02%	75.74%
7	wang2vec	CBOW	1	81.89%	75.61%	82.11%	75.97%
8		Skipgram	1	**82.56%**	76.04%	81.54%	75.26%
9		CBOW	2	82.32%	**76.18%**	82.30%	75.91%
10		Skipgram	2	81.90%	75.80%	81.96%	75.85%
11		CBOW	5	82.16%	75.89%	82.13%	75.81%
12		Skipgram	5	82.30%	76.01%	81.98%	75.76%
13	fasttext	CBOW	1	82.33%	76.00%	82.06%	75.71%
14		Skipgram	1	81.58%	75.37%	82.11%	75.83%
15		CBOW	2	81.92%	75.59%	82.08%	75.85%
16		Skipgram	2	82.21%	75.70%	82.28%	76.04%
17		CBOW	5	82.14%	75.71%	82.18%	76.00%
18		Skipgram	5	82.02%	75.82%	82.19%	76.05%

TABLE II. PARSER RESULTS ON SENTENCES WITH ELLIPSIS AND GERUND

No	Parser	Anaphora		Cataphora		Gerund
		UAS	*LAS*	*UAS*	*LAS*	*UAS*
1	MaltParser Planar	81.55%	74.36%	77.93%	73.81%	24%
2	Kiperwasser & Goldberg, without character embedding, wang2vec, window 1, CBOW	84.22%	80.22%	84.79%	80.99%	12%

References

[1] J. Nivre et al., "Universal Dependencies v1: A Multilingual Treebank Collection," Proc. 10th Int. Conf. Lang. Resour. Eval. (LREC 2016), pp. 1659–1666, 2016.

[2] M. Nizami and A. Purwarianti, "Modification of Chu-Liu/Edmonds algorithm and MIRA learning algorithm for dependency parser on Indonesian language," Proc. - 2017 Int. Conf. Adv. Informatics Concepts, Theory Appl. ICAICTA 2017, 2017.

[3] A. Kuncoro, "Pemanfaatan Pengurai Dependensi Ensemble dan Teknik Self-Learning untuk Meningkatkan Akurasi Pengurai Bahasa Indonesia," 2013.

[4] A. Rahman and A. Purwarianti, "Ensemble Technique Utilization for Indonesian Dependency Parser," 31st Pacific Asia Conf. Lang. Inf. Comput., 2017.

[5] J. Nivre, J. Hall, and J. Nilsson, "MaltParser: A data-driven parser-generator for dependency parsing," Lr. 2006, vol. 6, pp. 2216–2219, 2006.

[6] J. Nivre, "An efficient algorithm for projective dependency parsing," Proc. 8th Int. Work. Parsing Technol. IWPT, pp. 149–160, 2003.

[7] M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta, "Dynamic programming algorithms for transition-based dependency parsers," ACL-HLT 2011 - Proc. 49th Annu. Meet. Assoc. Comput. Linguist. Hum. Lang. Technol., vol. 1, no. 1974, pp. 673–682, 2011.

[8] R. Johansson and P. Nugues, "Investigating multilingual dependency parsing," Proc. Tenth Conf. Comput. Nat. Lang. Learn. (CoNLL-X 2006), no. June, pp. 206–210, 2006.

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," arXiv Prepr. arXiv1301.3781, pp. 1–12, 2013.

[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," pp. 1–9, 2013.

[11] W. Ling, C. Dyer, A. W. Black, and I. Trancoso, "Two/Too Simple Adaptations of Word2Vec for Syntax Problems," Proc. 2015 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol., pp. 1299–1304, 2015.

[12] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," 2016.

[13] E. Kiperwasser and Y. Goldberg, "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations," Acl, vol. 4, pp. 313–327, 2016.

[14] G. Genthial, "Sequence Tagging with Tensorflow." [Online]. Available: https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html. [Accessed: 14-Oct-2018].

[15] H. Alwi, H. Lapoliwa, A. M. Moeliono, and S. Dardjowidjojo, Tata Bahasa Baku bahasa Indonesia, 3rd ed. Balai Pustaka, 2000.