Neural Computing and Applications (2024) 36:2999–3016
https://doi.org/10.1007/s00521-023-09223-3

ORIGINAL ARTICLE

Integrating graph embedding and neural models for improving transition-based dependency parsing

Phuong Le-Hong[1] · Erik Cambria[2]

Received: 9 February 2023 / Accepted: 26 October 2023 / Published online: 27 November 2023
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2023

Abstract
This paper introduces an effective method for improving dependency parsing which is based on a graph embedding model. The model helps extract local and global connectivity patterns between tokens. This method allows neural network models to perform better on dependency parsing benchmarks. We propose to incorporate node embeddings trained by a graph embedding algorithm into a bidirectional recurrent neural network scheme. The new model outperforms a baseline reference using a state-of-the-art method on three dependency treebanks for both low-resource and high-resource natural languages, namely Indonesian, Vietnamese and English. We also show that the popular pretraining technique of BERT would not pick up on the same kind of signal as graph embeddings. The new parser together with all trained models is made available under an open-source license, facilitating community engagement and advancement of natural language processing research for two low-resource languages with around 300 million users worldwide in total.

Keywords Dependency parsing · recurrent neural networks · transformers · transition-based parsing · English · Indonesian · Vietnamese

1 Introduction

Dependency structures have played an important role in analyzing natural languages, and syntactic dependency parsing has recently gained popularity in the natural language processing (NLP) community. The conference on computational natural language learning (CoNLL) featured four shared tasks which devoted to multilingual dependency parsing in 2006, 2007, 2017 and 2018. In 2006, participants trained a single parser on data from 13 different languages [5]. In 2007, the shared task continued to explore data-driven methods for multilingual dependency parsing but added the problem of domain adaptation and partly different languages [35]. Ten years later, the 2017 CoNLL shared task focused on learning dependency parsers that can work in real-world setting, starting from raw text, and that can work on many typologically different languages [48]. The 2018 CoNLL shared task continued this effort but added a focus on morphological analysis as well as data from new languages. The two recent shared tasks have been made possible by the Universal Dependencies (UD) initiative which has developed treebanks for 50+ languages with cross-linguistically consistent annotation and recover ability of the original raw texts [34, 38]. These shared tasks are major milestones for parsing research which have enabled a comparison not only of parsing and learning methods, but also of the performance that can be achieved for different languages.

Practically, all data-driven methods that have been proposed for dependency parsing in recent years use statistical machine learning models, in particular neural network models, and can be described as either transition-based or graph-based methods [15]. Regardless of the methods, a crucial step in parser design is choosing the right feature function for the underlying statistical classifier [6]. Kiperwasser and Goldberg [18] proposed a simple and effective method for dependency parsing which is based on bidirectional long short-term memory neural

✉ Phuong Le-Hong
phuonglh@vnu.edu.vn

Erik Cambria
cambria@ntu.edu.sg

[1] Vietnam National University, Hanoi, Vietnam

[2] School of Computer Science and Engineering, NTU, Singapore, Singapore

networks. Despite its simplicity, resulting parsers match or surpass the state-of-the-art performances on English and Chinese dependency parsing.

The dependency structure of a natural language is essentially a graph whose nodes are tokens and arcs are dependency relations between tokens. This observation allows us to come up with an idea of connecting two lines of research namely dependency parsing and knowledge base modeling. A knowledge base is a relational graph of meaningful entities (nodes) and relationships (arcs). Knowledge graph embedding methods aim to model relationships between nodes and arcs by interpreting them as operations on the low-dimensional embeddings of the entities.

The main purpose of this research is to test the usefulness of knowledge graph modeling methods in improving dependency parsing. We introduce a method that integrates node embeddings learned by a graph embedding algorithm into an effective dependency parsing scheme. The resulting augmented feature function allows the model to achieve better parsing results on multiple languages. We perform multilingual evaluation to test generality of our proposed method, which helps improve dependency parsing of both low-resource and high-resource languages.

In this paper, we make the following main contributions:

- We present a method for extracting useful distributed features which are learned from dependency graphs by a graph embedding algorithm.
- We present a method which integrates graph embedding features into a state-of-the-art bidirectional recurrent neural network scheme.
- We demonstrate the effectiveness of the proposed method in English, Indonesian and Vietnamese. Experimental results on three standard dependency treebanks of these languages all show a significant improvement of parsing scores.
- We show that the popular pretraining technique of BERT would not pick up on the same kind of signal as graph embeddings.
- Our code and experimental results are publicly available on GitHub. The code is implemented in the Julia programming language.[1]

The remainder of this paper is structured as follows: Sect. 2 presents background for our work; Sect. 3 presents related work, focusing on recent research efforts on dependency parsing; Sect. 4 describes the baseline model, a graph embedding algorithm and the proposed integrated method; Sect. 5 presents experimental results on three dependency treebanks for Indonesian, Vietnamese and English; finally, we provide concluding remarks and discuss future work in Sect. 6.

2 Preliminaries

2.1 Syntactic structures

Constituency structure and dependency structure are two types of syntactic representation of a natural language sentence. While a constituency structure represents a nesting of multi-word constituents, a dependency structure represents dependencies between individual words of a sentence. The syntactic dependency represents the fact that the presence of a word is licensed by another word which is its governor. In a typed dependency analysis, grammatical labels are added to the dependencies to mark their grammatical relations, for example *subject* or *indirect object*. Figure 1 shows a constituency structure and its corresponding dependency structure. The syntactical tags are those from the Penn treebank project, and the dependency labels are from the Universal Dependency project.[2]

2.2 Dependency parsing

Dependency parsing is a fundamental task of NLP which aims to find the most probable dependency structure of a given sentence. Modern methods to dependency parsing can be roughly classified into two categories, namely transition-based and graph-based parsing [21]. Transition-based parsers treat parsing as a sequence prediction problem where a transition sequence from an initial configuration to some terminal configuration is predicted, which derives a target dependency parse tree [36, 37]. The most important component of a transition-based parser is a transition classifier which is trained to score the possible transitions at each step and guide the parsing process. Graph-based parsers treat parsing as a search-based structured prediction problem in which the goal is learning a scoring function over dependency trees such that the correct tree has the largest probability [30, 31].

The focus of this paper is on improving transition-based dependency parsing. For practical applications, the speed of transition-based parsers have been more appealing than graph-based parsers. For example, spaCy[3]—a widely used industry tool, implements a transition-based dependency parser component for its processing pipeline. We present transition-based dependency parsing in more detail in the following paragraphs.

[1] https://github.com/phuonglh/jvl/, under the *VLP/aep* module.

[2] https://universaldependencies.org/.

[3] https://spacy.io/api/dependencyparser.

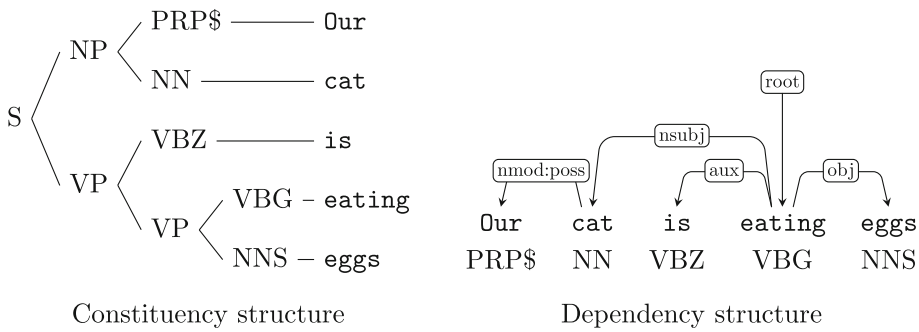

Fig. 1 Constituency and dependency structure of an English sentence

Table 1 Four transition types of the arc-eager parsing algorithm

Name	Operation	Precondition
LEFT-ARC	$(\sigma\|u, v\|\beta, A) \Rightarrow (\sigma, v\|\beta, A \cup \{(v, u)\})$	$\nexists k : (k, u) \in A$
RGHT-ARC	$(\sigma\|u, v\|\beta, A) \Rightarrow (\sigma\|u\|v, \beta, A \cup \{(u, v)\})$	$\nexists k : (k, v) \in A$
REDUCE	$(\sigma\|u, \beta, A) \Rightarrow (\sigma, \beta, A)$	$\exists v : (v, u) \in A$
SHIFT	$(\sigma, v\|\beta, A) \Rightarrow (\sigma\|v, \beta, A)$	$\beta \neq \emptyset$

As stated above, transition-based dependency parsing aims to predict a transition sequence from an initial configuration to some terminal configuration. It uses a classifier to predict the correct transition based on features extracted from the configuration at each step of the derivation process. In this paper, we examine only greedy parsing where the classifier always selects the best decision at each local stage. This greedy method tends to perform slightly worse than the search-based methods because of subsequent error propagation. An alternative method is beam search which may give better result but it is slower.

We use the arc-eager algorithm, which is a transition-based dependency parsing algorithm [33]. In an arc-eager system, a *configuration* $c = (\sigma, \beta, A)$ consists of *a stack* σ, *a buffer* β and *a set of dependency arcs* A. The initial configuration for a sentence $s = w_1, w_2, \ldots, w_n$ is $\sigma = [\text{ROOT}], \beta = [w_1, \ldots, w_n]$ and $A = \emptyset$. A configuration c is terminal if the buffer is empty and the stack contains a single element ROOT. We use the notation $v|\beta$ to indicate that the first element of the buffer is the word v; the notation $\sigma|u$ to indicate that the top element of the stack is the word u; and $A_c = \{(x, y)\}$ where x, y are words of a sentence being parsed to indicate the set of dependency arcs of a configuration c.

The arc-eager parsing algorithm defines four types of transitions as shown in Table 1. In the labeled version of parsing, there are in total $|\mathcal{T}| = 2N_l + 2$ transitions where N_l is the number of different arc labels. The preconditions of the four transition types are explained as follows:

- The precondition of LEFT-ARC $u \leftarrow v$ is that there does not exist any arc coming to u; in other words, u has not been dependent on another word. After this transition, the parsing of u is done and popped from the stack.
- The precondition of RIGHT-ARC $u \rightarrow v$ is that there does not exist any arc coming to v. After this transition, the word v is pushed onto the stack to consider next word. Note that there can be multiple arcs coming out of u.
- The REDUCE transition pops the stack and presupposes that the top element has already been attached to its head in a previous RIGHT-ARC transition.
- The SHIFT transition extracts the first element of the buffer and pushes it onto the stack. This transition requires that the buffer is nonempty.

As an example from the Universal Dependency English Web Treebank, the sentence "*Google is a nice search engine.*" contains seven tokens. An artificial token ROOT is inserted, serving as the unique root of its graph; this is a standard addition that simplifies both theoretical definitions and computational implementations. Figure 2 shows the dependency graph of this sentence. Note that the root word

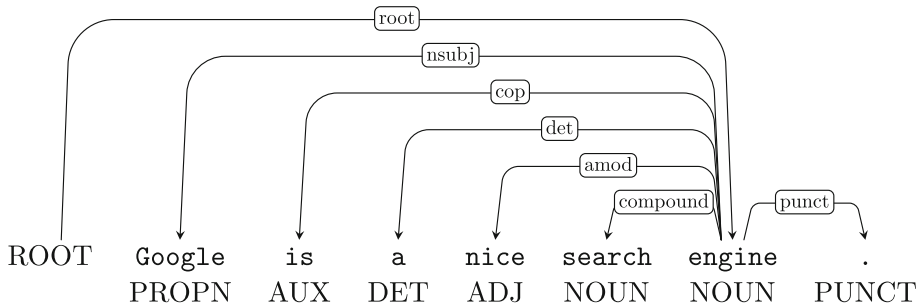

Fig. 2 Dependency graph of the English sentence "*Google is a nice search engine.*". The annotation labels, including part-of-speech tags and dependency labels, are defined consistently across multiple languages by the Universal Dependency project

of this sentence is "*engine*". Table 2 illustrates an example of one transition sequence which guides the parser from the initial configuration to a terminal one.

It has been shown that the arc-eager parsing algorithm can derive any projective dependency tree G for an input sentence s and doing so always adds arc as early as possible, i.e., an arc between u and v will be added as soon as they are at the stack/buffer top, respectively, or not at all. The essential step in the greedy parsing process is to predict a correct transition. A statistical classifier is used to predict the most probable transition with highest probability. In this setting, the prediction function is defined as $f : \mathcal{C} \rightarrow \mathcal{T}$, where $\mathcal{C}$ is a set of all possible configurations and $\mathcal{T}$ is the set of all possible transitions:

$$\mathcal{T} = \{\text{LEFT} - \text{ARC}, \text{RIGHT} - \text{ARC}, \text{REDUCE}, \text{SHIFT}\}$$

The greedy transition-based parsing algorithm is as follows.

Algorithm 1: Greedy transition-based parsing

Data: sentence s with words $w_1, w_2, \ldots, w_n$ and their corresponding parts-of-speech $p_1, p_2, \ldots, p_n$; parameterized classifier $f_\theta(c)$
Result: a set of dependency arcs

```
c ← initial(s);
repeat
    t̂ ← f_θ(c);
    c ← t̂(c);
until (terminal(c));
return tree(c)
```

Table 2 An example of arc-eager dependency parsing. We use SH, LA, RA and RE symbols to refer to shift, left-arc, right-arc and reduce transitions, respectively

Transition/A	Stack (σ)	Buffer (β)
	[ROOT]	[Google is a nice search engine .]
SH	[ROOT Google]	[is a nice search engine .]
SH	[ROOT Google is]	[a nice search engine .]
SH	[ROOT Google is a]	[nice search engine .]
SH	[ROOT Google is a nice]	[search engine .]
SH	[ROOT Google is a nice search]	[engine .]
LA-compound	[ROOT Google is a nice]	[engine .]
engine → search		
LA-amod	[ROOT Google is a]	[engine .]
engine → nice		
LA-det	[ROOT Google is]	[engine .]
engine → a		
LA-cop	[ROOT Google]	[engine .]
engine → is		
LA-nsubj	[ROOT]	[engine .]
engine → Google		
RA-root	[ROOT engine]	[.]
ROOT → engine		
RA-punct	[ROOT .]	[]
engine → .		
RE	[ROOT]	[]

Given a sentence s of n words and their corresponding parts-of-speech, the parser is initialized with the configuration c (line 1). A statistical classifier parameterized with θ predicts the most likely transition $\widehat{t}$ for c (line 3). The transition $\widehat{t}$ is applied to the configuration, resulting in a new parser configuration (line 4). The parsing process terminates when reaching a final configuration (line 5). The parse tree, represented by a set of dependency arcs, is then extracted and returned (line 6).

The algorithm guarantees termination after at most $2n$ transitions for a sentence of length n, which means the time complexity is $O(n)$ given that transitions can be performed in constant time. Furthermore, the dependency graph given at termination is guaranteed to be acyclic and projective [29].

The statistical classifier relies on a feature function $\phi(c)$ to represent the configuration c as a vector. This vector is then fed to a scoring function which assigns scores to (c, t) pairs, from which the highest scoring transition $\widehat{t}$ is chosen. Recent works usually use a nonlinear function, namely a multi-layer perceptron, i.e., feed-forward neural network model, with one hidden layer to estimate scores for each feature vector $\phi(c)$. These scores are then passed to a softmax layer to select $\widehat{t}$.

A crucial step in parser design is choosing the right feature function $\phi(c)$ for the underlying statistical classifier. In the next subsection, we give a brief review of feature representations methods in NLP in general and in dependency parsing in particular.

2.3 Feature representations

There are numerous methods to represent features. They can be classified into three main methods. The first and foremost simple method is using the *one-hot encoding* in that each feature is represented by its index in a prebuilt feature vocabulary. In the second method, each feature is represented by a vector that contains other features occurring with it in a context. This method is called the *distributional semantics representation* [2]. Although the two representation methods have been successfully applied in numerous NLP tasks, they share the same two inherent drawbacks: high dimensionality and feature sparseness. Many NLP systems use millions of indicator features, and from a statistical perspective, their corresponding parameters are poorly estimated because there are insufficient data to correctly estimate such parameters. For this reason, special techniques such as dimensionality reduction or feature selection are often required when using one-hot or distributional representations. The third method represents each feature by a dense, low-dimensional and real-valued vector. This is called *distributed feature representation* or feature embedding. Representing features in a continuous vector space has a long and rich history in natural language processing; all methods depend on the *distributional hypothesis* [14] which states that words that occur in the same contexts share semantic meaning. The different methods following this hypothesis can be divided into two approaches: count-based methods (e.g., latent semantic analysis) and predictive methods (e.g., neural probabilistic language models). While count-based methods compute the statistics of how often some words co-occur with its neighbor words in a large text corpus and then embed these counts into a low-dimensional and continuous vector for each word, predictive methods directly predict a word from its neighbors in terms of low-dimensional embedding vectors.

With the rise of deep learning methods, distributed feature representations have been shown to be advantageous in many NLP tasks. Each dimension of the embedding represents a latent feature which hopefully captures useful syntactic and semantic similarities [42]. The dimensionality is fixed for all the words in the vocabulary, and this dimensionality is much smaller than the vocabulary size.[4]

3 Related work

3.1 Recent research efforts on dependency parsing

In the past, before neural network models came into prominent, good parsers had relied on linear models over handcrafted feature functions. In particular, a feature function $\phi(c)$ for transition-based parsing usually looks at core components, for example "token on top of stack", "the first token on buffer", or "leftmost child of the second-to-top token on the stack", etc. Such a feature function requires a predefined set of feature templates, where each template instantiates a binary indicator function over a conjunction of core elements, resulting in features of the form "word on top of stack is X and part-of-speech of first word on buffer is Y". An example of well-designed feature set includes about 20 core components and 72 feature templates [49]. A feature extraction method for transition-based parsing usually looks at information such as the word identity and part-of-speech tags of a fixed number of tokens on top of the stack, a fixed number of tokens on top of the buffer, the leftmost and rightmost modifiers of tokens on the stack and the length of the spans covered by the tokens on the stack.

[4] In practice, the dimensionality is usually in the range of a dozen to one thousand.

The design of the feature function is a major challenge in parser design. In recent years, many researchers have attempted to come up with a good feature extraction method which is not only efficient but also reduces the required manual effort. Most of recent efforts have used distributed representations which are generated by neural network models. In particular, a low-rank tensor representation to automatically find good feature combinations was proposed by Lei et al. [25]. In another work at the same time, Chen and Manning [7] suggested encoding each core feature as a dense low-dimensional vector, and the vectors are then concatenated and fed into a multi-layer perceptron which can potentially capture diverse feature combinations. A larger number of core features are used including 18 word vectors, 18 part-of-speech vectors and 12 dependency labels vectors.

Another line of recent works does not focus on reducing the effort in hand-crafting effective feature combinations but tackling the feature-engineering problem by designing novel neural network models capable of encoding the parser state, including its subtrees as vectors, which are then fed to nonlinear classifiers. Dyer et al. [11] encode the entire stack and buffer as a stack long short-term memory, where each stack element is itself based on a compositional representation of parse trees. Le and Zuidema [22] encoded each tree node as two compositional representations capturing the inside and outside structures around the node and feed the representations into a reranker. Zhu et al. [51] used a similar reranking approach based on recursive convolutional neural networks. Kiperwasser and Goldberg [17] presented an easy-first parser based on a novel hierarchical-LSTM tree encoding. Finally, Kiperwasser and Goldberg [18] encoded each sentence token by bidirectional LSTMs and feature vectors are constructed by concatenating a few BiLSTM vectors, resulting a simple and effective scheme for accurate dependency parsing. Despite this simplicity, the parsers yielded very competitive parsing accuracies on English and Chinese.

A different line of related research is learning dependency-based word embeddings. Levy and Goldberg [26] generalized the skip-gram model, performing experiments with dependency-based contexts, and showed that they produce different embeddings. The dependency-based embeddings are less topical and exhibit more functional similarity than the skip-gram embeddings. Ling et al. [27] proposed an extension to the continuous bag-of-words model which adds an attention model that considers contextual words differently depending on the word type and its relative position to the predicted word. Word embeddings learned by this method were shown to improve some semantically and syntactically oriented tasks.

In the last five years, neural parsing, like most of NLP, has shifted from static representations of each word type to deep contextualized word representations. Such methods allow encoding words with respect to the sentential context in which they appear, producing sentence-level, dynamic vectors as representations instead of static vectors. These representations are typically produced by BiLSTM or Transformers [43]. Two most popular deep contextualized embedding models are ELMo [40] and BERT [9]. Both models have been used for dependency parsing [12, 28, 50]. In particular, Kondratyuk and Straka [20] leveraged a multilingual BERT self-attention model pretrained on 104 languages to improve parsing scores without requiring any recurrent or language-specific components.

Most of recent methods focus on high-resource and well-studied languages such as English and Chinese which already have existing datasets available and are accessible to the research community. However, most languages suffer from limited data collection and low awareness of published data and tools for research. Two of the languages which suffer from this resource scarcity problem are Indonesian and Vietnamese.

3.2 Indonesian dependency parsing

Indonesian (sometimes also referred to as Bahasa Indonesian) is the official language of Indonesia. It is an Austronesian language that has been used as a lingua franca in the multilingual Indonesian archipelago for years. Indonesia is the fourth most populous nation in the world—of which the majority speak Indonesian, which makes it one of the most widely spoken languages in the world [41]. According to a recent statistic, Indonesian is also the fourth largest language used over the Internet, with around 196 million users.[5] Despite a large amount of Indonesian data available over the Internet, the advancement of NLP research in Indonesian is slow-moving. This problem occurs because available datasets are scattered, with a lack of documentation and minimal community engagement. Moreover, many existing studies in Indonesian NLP do not provide codes and test splits, making it impossible to reproduce results [45]. The first-ever Indonesian natural language understanding benchmark named IndoNLU is published in 2020 by Wilie et al. [45]. This benchmark includes 12 diverse tasks, ranging from single sentence classification to pair-sentences sequence labeling with different levels of complexity. The authors also provide a set of Indonesian pretrained models (IndoBERT) trained on a large Indonesian dataset (Indo4B). However, this benchmark does not have dependency parsing, an important NLP task that we are concerned with in this work. The

[5] https://www.internetworldstats.com/stats3.htm.

first work on Indonesian dependency parsing is that of Green et al. [13], which reports an ongoing work on the development of an Indonesian dependency treebank as well as the first full implementation of a dependency parser for Indonesian. Their ensemble support vector machine parser achieved an average unlabeled attachment score of about 60%. This evaluation result was performed on a quite small corpus of 100 annotated sentences, consisting of 2705 tokens.

The method and results of Indonesian dependency parsing presented in our work are a necessary development of dependency parsing for Indonesian. The experimental results are obtained on a standard training/development and test split of a universal dependency treebank for Indonesian, which establish a new benchmark score for Indonesian dependency parsing.

3.3 Vietnamese dependency parsing

Vietnamese is among the top 20 most spoken languages [39], with around 100 million users. Vietnamese belongs to the Viet–Muong group of the Mon–Khmer branch, which in turn belongs to the Austro–Asiatic language family. Vietnamese is also similar to languages in the Tai family. The Vietnamese vocabulary features a large number of Sino-Vietnamese words which are derived from Chinese [1, 24].

Compared to Indonesian language processing, Vietnamese speech and language processing has been paid more attention since the last 15 years, especially by the VLSP community.[6] The VLSP Consortium regroups all academic and industrial research teams involved in Vietnamese language and speech processing. The very first kick-off meeting of this community was in 2005, organized at the Institute of Information Technology, Vietnam Academy of Science and Technology. Since 2012, the VLSP Consortium has organized a series of workshops, in conjunction with big international conferences organized in Vietnam. Until 2021, seven events have taken place since then with different forms of activities such as technical reports, activity reports, discussion panel and shared tasks on VLSP. In particular, at VLSP 2019 workshop, the first shared task on Vietnamese dependency parsing was proposed in order to promote the development of dependency parsers for Vietnamese. A dependency corpus in the universal dependency format of 4,000 Vietnamese sentences was developed and released for this shared task.

The earliest experimental result on Vietnamese dependency parsing was published in 2012 by [23] where the authors conducted a syntactic analysis of Vietnamese sentences using an automatically extracted lexicalized tree-adjoining grammar. The derivation trees permit extracting dependency relations between lexical units of an input sentence. The best results they obtained are 73.21% of unlabeled attachment score for dependency accuracy on a test corpus, which is also automatically extracted from a constituency treebank. The first dependency treebank for Vietnamese was published in 2013 in which 3000 sentences are automatically converted and manually edited [32]. This corpus was later revised and converted into the universal dependency format and made publicly available in 2016.[7] It is included for the CoNLL shared task "Multilingual Parsing from Raw Text to Universal Dependencies" containing 48 dependency labels for Vietnamese based on the Stanford dependency labels set [47, 48].

4 Methods

4.1 Baseline model

Our baseline transition classification model is a three-layer feed-forward neural network model which includes a feature embedding layer, a dense layer with the sigmoid activation function and a softmax layer. We use a simple feature function which looks at the following information at each parser configuration:

- The word identity, shape, lemma, part-of-speech tag and universal parts-of-speech tag of the top token on the stack;
- The same five features of the first token on the buffer;
- The same five features of the second-to-top token on the stack;
- The same five features of the second token on the buffer;

This feature function extracts 20 discrete features in total for each parser configuration and they are fed to the feature embedding layer of the model. Each feature is embedded into an e-dimensional vector by the underlying embedding matrix of size $e \times |V|$ of the embedding layer where $|V|$ is the size of the feature set extracted from a training corpus. We investigate two different methods for combining embedding vectors, resulting two variants of the baseline method:

1. continuous bag-of-feature model (BOF): the 20 feature embedding vectors are summed to represent the embedding vector of the configuration, that is

[6] Vietnamese Language and Speech Processing, http://vlsp.org.vn/.

[7] https://github.com/UniversalDependencies/UD_Vietnamese-VTB/.

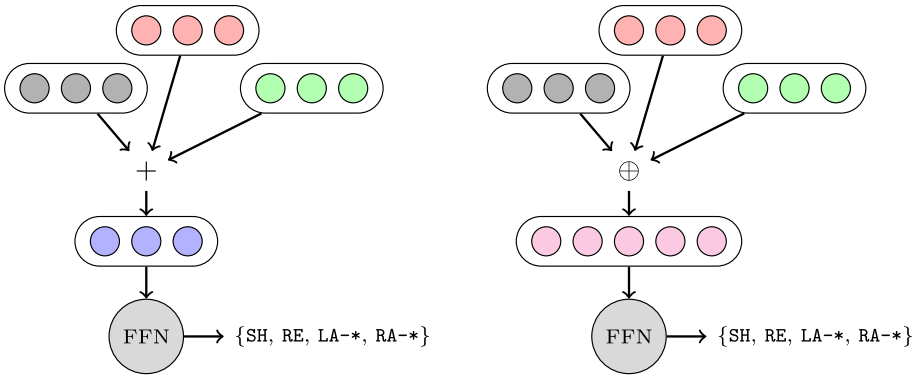

Fig. 3 Two variants of the baseline model. The left figure depicts the continuous bag-of-features model (BOF) where feature embeddings are added (+) before being fed to a feed-forward network (FFN) model. The right figure depicts the sequence-of-feature model (SOF) where features are concatenated (⊕). The FFN has one hidden layer and one softmax layer to perform labeled transition classification

$$\phi_B(c) = \sum_{j=1}^{20} \vec{v}_{f_j},$$

where $\vec{v}_f$ is the embedding vector of the feature f. The feature vector $\phi_B(c)$ has e dimensions as its summands.

2. concatenated sequence-of-feature model (SOF): the 20 feature embedding vectors are concatenated to represent the embedding vector of the configuration, that is

$$\phi_S(c) = \vec{v}_{f_1} \oplus \vec{v}_{f_2} \oplus \ldots \oplus \vec{v}_{f_{20}}.$$

In this representation, the feature vector $\phi_S(c)$ has $20 * e$ dimensions.

It is worth noting that we do not use pretrained embeddings. All the feature embeddings are trained together with the models. Figure 3 depicts an illustration of two models BOF and SOF. The FFN has one hidden layer of h units and one softmax layer to perform labeled transition classification. The quantities e and h are hyperparameters of the models and their best values are selected using a development dataset.

4.2 RNN-based model

The feature function of the baseline model embeds all features into the same real-valued vector space of e dimensions. As usual embedding models in NLP, it makes more sense to use different embedding matrices for words, shapes and part-of-speech tags. In this improved model, we represent each word as a w-dimensional vector in $\mathbb{R}^w$ and the full word embedding matrix is $\mathbf{W} \in \mathbb{R}^{w \times N_w}$ where N_w is the dictionary size. Each shape or word form of a word is represented by a s-dimensional vector and the full shape embedding matrix is $\mathbf{S} \in \mathbb{R}^{s \times N_s}$ where N_s is the size of the shape dictionary.[8] Similarly, each part-of-speech tag (PoS) is represented by a p-dimensional vector and the full PoS embedding matrix is $\mathbf{P} \in \mathbb{R}^{p \times N_p}$ where N_p is the PoS dictionary size.

For a sentence of n tokens $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, each token $\mathbf{x}_j$ is represented by a triple (w_j, s_j, p_j) where w_j, s_j and p_j are integer indices of the word, shape and part-of-speech of token $\mathbf{x}_j$, respectively, for all $j = 1, 2, \ldots, n$. The whole sentence is thus represented by an integer matrix of size $3 \times n$. After being fed to an embedding layer, each token is encoded as the concatenated vector of the corresponding word, shape and PoS embedding vectors:

$$\vec{\mathbf{x}}_j = \vec{v}(w_j) \oplus \vec{v}(s_j) \oplus \vec{v}(p_j).$$

The dimension of $\vec{\mathbf{x}}_j$ is $w + s + p$, and these hyperparameters are optimized on a development set.

We adopt the method proposed by Kiperwasser and Goldberg [18] which replaces the handcrafted feature functions in favor of minimally defined feature functions which make use of automatically learned bidirectional recurrent neural network representations to capture contextual information of each token in an input sentence. The RNN uses either bidirectional GRU or LSTM units and introduces context by representing each input token $\mathbf{x}_j$ as its BiRNN vector

$$\vec{v}_j = \texttt{BiRNN}([\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots, \vec{\mathbf{x}}_n], j).$$

Each parsing configuration $c = (\sigma, \beta, A)$ consists of a stack σ, a buffer β and a set A of dependency arcs. Both the stack and the buffer contains integer indices pointing to sentence tokens. Our feature function is the concatenated vectors of the top two tokens on the stack and the top two tokens on the buffer.[9] More precisely, for a configuration $c = (\cdots|\sigma_2|\sigma_1, \beta_1|\beta_2|\cdots, A)$, the feature function is defined as

[8] The shape dictionary of a word includes a dozen of different word forms such as number, date, allcaps, url....

[9] Kiperwasser and Goldberg [18] select the top three tokens on the stack and the first token on the buffer. They use the arc-hybrid system instead of the arc-eager system as in our work.

$$\phi_R(c) = \vec{v}_{\sigma_2} \oplus \vec{v}_{\sigma_1} \oplus \vec{v}_{\beta_1} \oplus \vec{v}_{\beta_2},$$

where v_j is the contextual vector encoded by the BiRNN as specified above. This selection of relevant tokens aligns well with the simple feature function used in the baseline model, which makes experimental results of different models comparable. Figure 4 shows an illustration of the arc-eager transition-based parsing model.

It is important to note that, in this model, the whole pipeline is trained in an end-to-end fashion, where the embedding layer (the word, shape and PoS embedding matrices), the BiRNN layer(s) and the FFN are trained jointly. When parsing a sentence, the pipeline iteratively computes the scores for all possible transitions, predicts and selects the best transition until the final configuration is reached. The pipeline is fully supervised: all the feature embeddings are trained together with the model without using pretrained word embeddings.

many areas such as social network analysis, recommender systems or knowledge bases.

The main idea of graph embedding is to extract local or global connectivity patterns between entities and then use these patterns to perform prediction and generalize the observed relationship between a specific entity and all others. We hypothesize that this idea can be useful to improve the generalization ability of a dependency parser. If a graph embedding model can capture latent relationships between words of a sentence and that information is integrated properly into a feature function then it would help improve the accuracy of a statistical classifier, resulting in a more accurate dependency parser.

Algorithm 2: TransE for learning dependency embeddings

Data: training set $S = \{\ell(h,t)\}$, vocabulary $\mathcal{V}$, dependency set $\mathcal{L}$, margin γ, learning rate α, embedding dimension k;
Result: word embedding matrix $\mathbb{R}^{k\times|\mathcal{V}|}$, label embedding matrix $\mathbb{R}^{k\times|\mathcal{L}|}$

$\vec{w} \leftarrow \texttt{uniform}\left(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}}\right), \forall w \in \mathcal{V}$;
$\vec{\ell} \leftarrow \texttt{uniform}\left(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}}\right), \forall \ell \in \mathcal{L}$;
$\vec{\ell} \leftarrow \vec{\ell}/\|\vec{\ell}\|, \forall \ell \in \mathcal{L}$;
for $t \leftarrow 1$ *to* M **do**
  $\vec{w} \leftarrow \vec{w}/\|\vec{w}\|, \forall w \in \mathcal{V}$;
  $S_b \leftarrow \texttt{sample}(S, b)$;
  $T_b \leftarrow \emptyset$;
  for $\ell(h,t) \in S_b$ **do**
    $\ell(h',t') \leftarrow \texttt{corrupt}(\ell(h,t))$;
    $T_b \leftarrow T_b \cup \{(\ell(h,t), \ell(h',t'))\}$;
  $J = \sum_{(\ell(h,t),\ell(h',t'))\in T_b} \left[\gamma + d(\vec{h} + \vec{\ell}, \vec{t}) - d(\vec{h'} + \vec{\ell}, \vec{t'})\right]_+$;
  $\vec{\ell} \leftarrow \vec{\ell} - \alpha * \frac{\partial J}{\partial \vec{\ell}}$;
  $\vec{w} \leftarrow \vec{w} - \alpha * \frac{\partial J}{\partial \vec{w}}$;
return $\{\vec{w} \in \mathbb{R}^k, \forall w \in \mathcal{V}\}$, $\{\vec{\ell} \in \mathbb{R}^k, \forall \ell \in \mathcal{L}\}$

4.3 RNN and graph embedding model

One recent problem in learning from multi-relational data is graph embedding. We are concerned with directed graphs whose nodes correspond to entities and arcs of the form (*head*, *label*, *tail*), each of which indicates that there exists a labeled relationship between entities *head* and *tail*. Models of multi-relational data play an important role in

We apply the TransE embedding method which was proposed by Bordes et al. [4]. This method is simple but powerful, outperformed state-of-the-art methods in link prediction on knowledge base benchmarks. Since then, there were other graph embedding approaches which are more complicated such as graph transformers [44] but we do not investigate them in this work. TransE is an energy-based model which learns relationships by interpreting them as translations operating on the low-dimensional

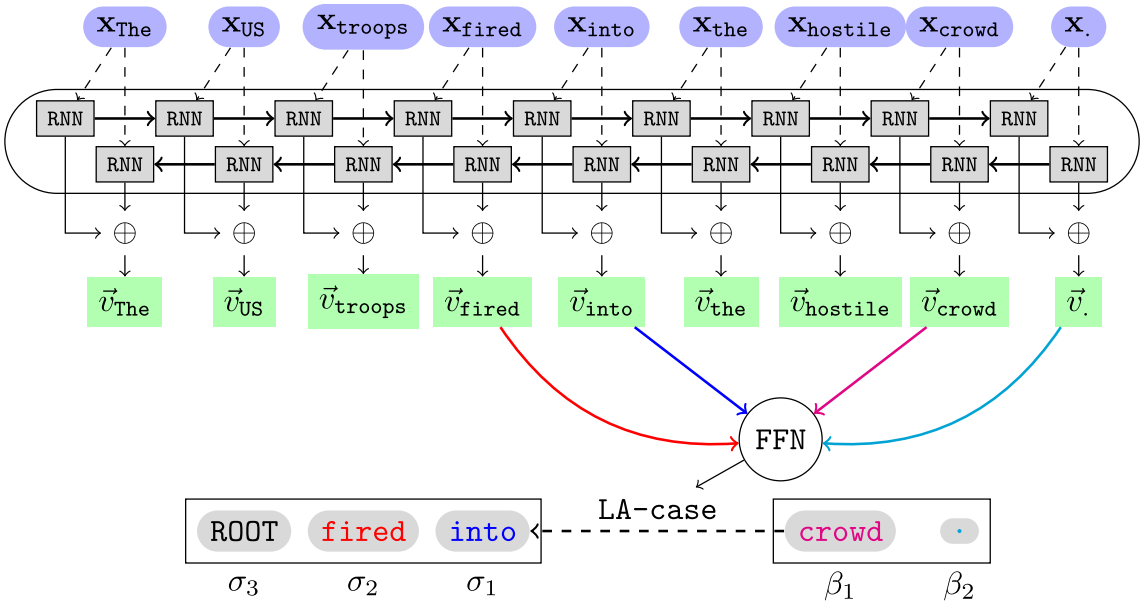

Fig. 4 Illustration of the neural model of the arc-eager transition-based parser. The configuration (stack σ and buffer β) is depicted at bottom. Each transition is scored by a two-layer feed-forward network (FFN). The embeddings of the tokens are learned by a bidirectional recurrent neural network (BiGRU or BiLSTM) and fed to the FFN. The colors of the four selected tokens correspond to colors of the FFN inputs. Each $\mathbf{x}_j$ is a concatenation of a word, a shape and a part-of-speech embedding vector. The FFN and the entire pipeline are optimized jointly to predict the best transition at each parsing step

embeddings of the entities. In this model, if a relationship (h, ℓ, t) holds, where h and t are two entities and ℓ is a relationship between them, then the embedding of the tail entity t should be close to the embedding of the head entity h plus some vector that depends on the relationship ℓ. In the context of dependency parsing, h is a head word, t is a dependent word of h and ℓ is a typed dependency between h and t. The TransE model learns the corresponding embedding vectors in the vector space $\mathbb{R}^k$ such that

$$\vec{h} + \vec{\ell} \approx \vec{t}$$

when there exists the typed dependency $\ell(h, t)$. In other words, if $\ell(h, t)$ holds, $\vec{t}$ should be a nearest neighbor of $\vec{h} + \vec{\ell}$, while $\vec{h} + \vec{\ell}$ should be far away from $\vec{t}$ otherwise. Suppose that $d(\vec{u}, \vec{v})$ is a dissimilarity measure between two real-valued vectors $\vec{u}$ and $\vec{v}$, to learn such embeddings, we minimize a margin-based ranking criterion over the training set:

$$J = \sum_{\ell(h,t) \in S} \sum_{\ell(h',t') \in S'} \left[\gamma + d(\vec{h} + \vec{\ell}, \vec{t}) - d(\vec{h'} + \vec{\ell}, \vec{t'}) \right]_+$$

where $[a]_+$ denotes the positive part of a, $\gamma > 0$ is a margin hyperparameter, S is the set of correct triplets extracted from the training set and S' is the set of corrupted triplets, constructed from S as follows:

$$S' = \{\ell(h', t) | h' \in \mathcal{V}\} \cup \{\ell(h, t') | t' \in \mathcal{V}\},$$

where $\mathcal{V}$ is the vocabulary. S' is composed of training dependencies with either the head or dependent replaced by a random word, but not both at the same time. That loss function favors lower values for observed dependencies in a dependency treebank than for unobserved ones and is thus a natural implementation of the intended criterion. It is worth noting that in this model, for a given word, its embedding vector is the same when it appears as the head or as the dependent.

Algorithm 2 describes the detailed procedure for training word and dependency embeddings using the TransE method. Given a training set of labeled dependencies $S = \{\ell(h, t)\}$ where $h, t \in \mathcal{V}$ are head and dependent words and $\ell \in \mathcal{L}$ is a dependency label, the algorithm estimates word and label embedding matrices as follows. The embedding vectors are initialized using a uniform distribution depending on the desired embedding dimension k (lines 1–2). The dependency embeddings are first normalized (line 3). The training procedure is performed in M iterations (line 4). In each iteration, a mini-batch of size b is sampled from the training set S (line 6) and a mini-dataset T_b is constructed by including corrupted samples into S_b, where the corrupted triples are built as described above (line 7–10). A loss function J is computed with respect to the embedding vectors on the mini-dataset T_b (line 11) and minimized using a gradient-based optimization method, for example the stochastic gradient descent method (lines 12–13).

We run the Algorithm 2 on the training dependency graphs of a dependency treebank to get a word embedding matrix. Each word is represented by a real-valued vector of k dimensions, $\vec{w} \in \mathbb{R}^k$. These vectors are then concatenated with the corresponding vectors generated by a BiRNN before being fed into a neural network transition classifier. Thus, these pretrained embeddings are fine-tuned together with the parameters of the model. Figure 5 illustrates our proposed model.

5 Experiments

5.1 Datasets

We evaluate our proposed models on English, Indonesian and Vietnamese dependency treebanks which are publicly available from the Universal Dependency (UD) website. UD is a project that seeks to develop cross-linguistically consistent treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning and parsing research from a language typology perspective. UD is an open community effort with over 300 contributors producing nearly 200 treebanks in over 100 languages. This is a valuable linguistic resource for evaluating dependency parsers in a multilingual context.

There exist three Indonesian dependency treebanks in UD version 2.7. We experimented with the Indonesian GSD treebank in this work. This is the largest corpus among the three treebanks available for this language.[10] This corpus contains 5,593 sentences and 121,923 tokens. This corpus uses 16 universal part-of-speech tags and 31 universal dependency relations. There is only one Vietnamese dependency treebank, the Vietnamese VTB. This corpus contains 3,000 sentences and 43,754 tokens, using 14 universal part-of-speech tags and 29 universal dependency relations. There are a dozen of dependency treebanks for English. We select the largest corpus—the English Web Treebank (EWT) for experimentation in this work. This corpus comprises 251,725 words and 16,062 sentences, taken from five genres of web media: weblogs, newsgroups, emails, reviews and Yahoo! answers. Table 3 shows some statistics of the datasets used in our experiments.

5.2 Details of the training algorithms

The training objective is to set the score of correct transitions above the scores of incorrect ones. We use the common cross-entropy loss to minimize the objective function. The scores corresponding to possible transitions for a configuration computed by the FFN are normalized by the softmax function to produce a probability vector. A cross-entropy loss is then used to minimize an objective function. The model learns to minimize the loss over the training data. This correlates with maximizing the number of correct transitions in the predicted outputs.

Since the pipeline model is trained in an end-to-end fashion, the gradients of the entire network, including the FFN, the BiRNN and the embedding matrices for words, shapes and part-of-speech tags with respect to the sum of the losses are computed using the backpropagation algorithm. We perform multiple training epochs, using early stopping—the training process is stopped when the development accuracy does not increase after three consecutive epochs.

The maximal sequence length of each sentence is set to 40 tokens. The models are all trained by the Adam optimizer [16] with default parameters.[11] The batch size is set to 32. On each dataset, we run a set of experiments with different recurrent architecture (BiGRU or BiLSTM), with different number of recurrent layers, different number of hidden units in each recurrent layer and different number of units in the dense layer of the FFN. Each experiment is run five times, and its results are averaged and taken as the final result. We report the accuracy on the training set, development set and test set.

We train the TransE algorithm for learning dependency embeddings in $M = 100$ iterations, producing $k = 16$ dimensions for word and label embeddings. We selected the margin γ among $\{1, 2, 10\}$ using LAS on the development corpus. The graph embeddings are fixed, i.e., they are not fine-tuned when training the parser. Once the embeddings are trained, they are concatenated into the core word features of parsing configurations, as illustrated in Fig. 5.

5.3 Evaluation metrics

The standard evaluation metrics of dependency parsing are *unlabeled attachment score* (UAS) and *labeled attachment score* (LAS). These scores measure the percentage of nodes with correctly assigned reference to head node, without or with the label (type) of the relation, respectively. Note that the attachment of all nodes including punctuation is evaluated. Since the performance of a transition-based parser depends largely on its underlying transition classifier, the transition classification accuracy is also reported.

[10] The GSD treebank is about five times larger than the PUD or CSUI treebanks.

[11] All models are implemented in the Julia programming language using the https://fluxml.ai library.

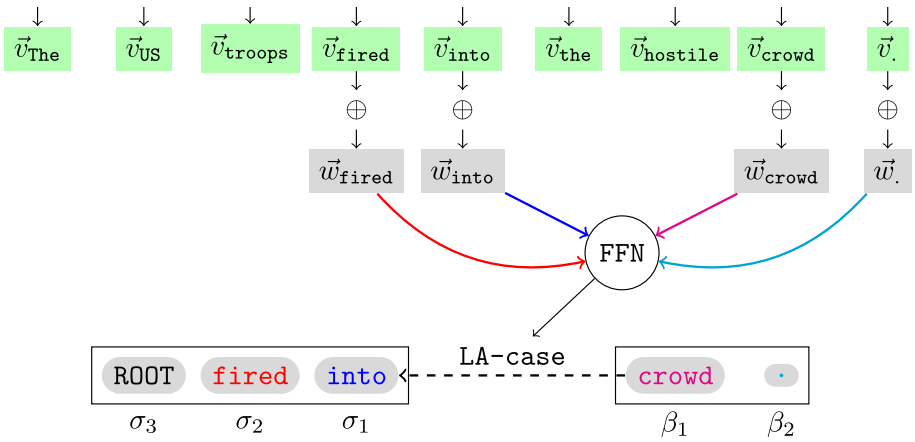

Fig. 5 Illustration of an improved version of the model in Fig. 4 where word embeddings trained by a graph embedding model are integrated into core features of a parsing configuration. $\vec{v}$ vectors are generated by a bidirectional recurrent network as shown in Fig. 4, while $\vec{w}$ vectors are generated by the TransE graph embedding model. The corresponding $\vec{v}$ and $\vec{w}$ vectors are concatenated before being fed into a neural network classifier

Table 3 Some statistics of treebanks used in this study

Statistics	Ind. GSD	Vie. VTB	Eng. EWT
Total sentences	5,593	3,000	16,062
Training	4,477	1,400	12,543
Development	559	800	2,002
Test	557	800	2077
Total tokens	121,923	43,754	251,725
Universal PoS tags	16	14	17
Universal relations	27	29	37

5.4 Results

In the first set of experiments, we evaluate the baseline model on the Indonesian and Vietnamese dependency treebank. Two variants, BOF and SOF, are evaluated. We selected the embedding dimension e among $\{25, 50, 75\}$ for the BOF model, among $\{5, 10, 15, 20\}$ for the SOF model,[12] and the hidden dimension h of the feed-forward model among $\{32, 64, 128, 256, 300\}$ on the development set of each dataset. Table 4 shows the scores of the baseline model on the Indonesian treebank. The best UAS and LAS of the BOF variant on the development set is 54.00% and 43.86%, respectively, achieved at the optimal configuration $e = 75, h = 128$. At this configuration, the UAS and LAS on the test set are 54.50% and 44.76%, respectively. The SOF model variant is better than the BOF model, for which the best test UAS and LAS are 55.97% and 47.00%, respectively, achieved at the optimal parameters $e = 20, h = 300$, where the best scores on the development set are 56.88% UAS and 47.30% LAS. As shown in Table 5, on the Vietnamese treebank, the BOF model attains the best development UAS and LAS of 48.53% and 36.93%, respectively, at the optimal configurations $e = 50, h = 300$, at which the test UAS and LAS are 46.20% and 35.64%, respectively. The best development scores of the SOF variant are 45.46% and 36.83% at the optimal configuration $e = 15, h = 32$, where the test scores of the SOF variant are 43.19% and 34.88%, significantly worse than those of the BOF variant.

In the second set of experiments, we evaluate the accuracy of the RNN-based model on the same datasets. As presented in Sect. 4.2, the important hyperparameters of this model include word embedding dimension w, universal part-of-speech embedding dimension p, shape embedding dimension s, recurrent dimension r (i.e., the output dimension of the recurrent layer) and hidden dimension h (i.e., the output dimension of the FNN dense layer). Following the work of Kiperwasser and Goldberg [18] which achieved the state-of-the-art result on English dependency parsing,[13] we fix $p = 25$. The shape embedding s is chosen as 4 experimentally.[14] Since the size of both Indonesian and Vietnamese dataset is much smaller than that of the English treebank, we vary the word embedding dimension w in the set $\{50, 100\}$, the recurrent dimension r in the set $\{16, 32, 64, 128, 150, 200, 256\}$ and the hidden units h in the set $\{64, 128, 256\}$ to investigate their effect to the accuracy. Two variants of the recurrent unit are evaluated, including unidirectional gated recurrent

[12] Recall that in the SOF variant, 20 embedding vectors of individual features are concatenated, resulting in an embedding dimension of $20 * e$.

[13] Kiperwasser and Goldberg [18] evaluated their model on the English Penn Treebank corpus.

[14] This small number makes sense due to a small number of 12 different possible word shapes.

Table 4 Performance of the baseline model on the training and development set of the Indonesian dependency treebank. The first two columns shows feature embedding dimensions and the number of hidden units of the FNN. The next two columns show the transition classifier accuracy on the training and development set. The next two columns show unlabeled attachment scores. The last two columns show labeled attachment scores. The first half of the table shows scores of the BOF model, while the second half shows scores of the SOF model

		Classifier Accuracy		**UAS**		LAS	
e	*h*	Train	Dev	Train	Dev	Train	Dev
BOF Model							
25	32	80.95	62.23	64.78	50.62	58.55	42.27
25	64	79.22	61.65	65.09	51.62	58.46	42.95
25	128	80.54	62.29	66.30	52.32	59.82	43.65
25	256	80.88	61.66	66.69	52.37	60.45	43.35
25	300	80.84	61.88	66.49	52.36	60.24	43.52
50	32	79.50	60.47	65.62	51.35	58.70	42.40
50	64	80.88	61.86	66.51	52.53	60.11	43.79
50	128	79.93	61.79	66.11	52.36	59.49	43.64
50	256	80.99	61.22	67.35	53.24	61.12	43.84
50	300	81.04	61.21	67.33	52.72	60.98	43.35
75	32	77.06	59.13	65.92	52.43	56.99	41.31
75	64	77.39	59.90	66.49	53.30	57.89	42.65
75	128	81.06	61.65	68.16	**54.00**	60.62	**43.86**
75	256	78.99	60.14	67.35	53.17	60.07	43.29
75	300	79.59	60.41	67.30	52.86	60.27	42.95
SOF Model							
5	32	79.44	62.82	64.68	51.73	57.79	43.45
5	64	81.53	62.83	67.01	53.36	59.87	44.65
5	128	81.87	63.44	67.87	54.25	60.80	45.43
5	256	79.84	63.12	67.11	54.99	59.63	45.71
5	300	81.57	62.86	68.55	55.11	61.50	45.85
10	32	85.11	63.12	70.41	55.18	63.86	45.84
10	64	82.44	63.39	69.57	55.63	62.13	46.14
10	128	86.05	63.31	71.36	55.47	65.14	45.72
10	256	80.10	63.11	68.98	55.52	61.75	46.04
10	300	85.58	63.40	71.36	55.77	65.14	46.04
15	32	84.89	63.55	71.07	55.69	64.25	46.23
15	64	86.79	63.82	71.95	56.29	65.80	46.85
15	128	83.87	63.32	70.49	55.61	63.74	46.27
15	256	85.83	63.20	71.58	55.26	65.53	45.82
15	300	81.27	63.11	69.59	56.17	62.52	46.59
20	32	85.99	63.48	71.52	55.43	64.85	46.19
20	64	88.60	63.30	73.48	55.74	67.69	46.54
20	128	87.68	63.06	73.14	55.85	67.36	46.19
20	256	90.11	63.15	74.89	56.71	70.04	46.73
20	300	86.62	63.20	73.18	**56.88**	67.43	**47.30**

The bold font shows the best score on each column

Table 5 Performance of the baseline model on the training and development set of the Vietnamese dependency treebank. Column names are similar to those in Table 4. In general, the BOF model outperforms the SOF model on this dataset

		Classifier Accuracy		UAS		LAS	
e	*h*	Train	Dev	Train	Dev	Train	Dev
BOF Model							
25	32	85.08	57.27	68.96	45.04	62.63	35.07
25	64	85.83	58.21	71.35	46.80	65.70	37.00
25	128	81.68	56.68	70.46	47.22	64.10	36.86
25	256	82.11	56.95	70.51	46.92	63.82	36.38
25	300	83.27	56.53	71.39	47.17	65.13	36.40
50	32	83.07	56.83	70.10	45.92	63.85	35.82
50	64	81.18	56.49	70.19	46.08	63.91	36.23
50	128	83.85	57.73	71.63	47.37	65.47	37.43
50	256	84.78	56.94	72.78	48.12	65.55	36.83
50	300	83.08	56.40	72.62	**48.53**	64.53	**36.93**
75	32	75.25	53.88	64.68	45.35	52.46	32.78
75	64	81.83	56.05	69.12	47.26	60.91	35.95
75	128	81.58	57.35	69.71	47.04	62.68	36.87
75	256	79.36	56.13	69.65	47.21	62.34	36.79
75	300	81.11	56.44	70.71	48.00	62.51	36.73
SOF Model							
5	32	84.81	55.37	64.70	36.60	58.58	29.82
5	64	83.42	55.58	66.75	39.82	60.22	32.19
5	128	85.01	54.24	68.83	41.23	62.80	33.43
5	256	80.63	55.93	67.34	43.24	60.44	34.68
5	300	82.24	55.35	68.42	44.07	61.66	34.96
10	32	84.96	55.36	70.52	44.06	63.48	35.39
10	64	83.54	56.23	69.65	44.79	62.68	36.36
10	128	85.94	55.49	71.39	44.27	65.17	36.02
10	256	84.14	55.91	70.98	44.01	64.50	35.44
10	300	82.33	54.93	70.21	44.06	63.52	35.59
15	32	84.74	55.62	71.68	**45.46**	64.51	**36.83**
15	64	85.81	56.39	71.90	44.94	65.29	36.62
15	128	85.31	56.06	71.64	44.94	65.52	36.50
15	256	86.67	54.60	72.72	43.89	67.03	34.91
15	300	82.85	54.75	71.34	44.19	64.62	35.29
20	32	85.70	55.81	72.61	45.19	65.67	36.33
20	64	86.19	56.27	73.07	45.26	66.40	36.76
20	128	85.67	55.90	72.70	44.73	66.69	36.09
20	256	82.80	55.45	71.06	44.49	64.28	35.82

The bold font shows the best score on each column

unit (GRU) and bidirectional GRU (BiGRU). We use one recurrent layer in all experiments.

Table 6 shows the performance of the RNN-based model on the test sets of the two languages with respect to their optimal hyperparameters tuned on their corresponding development sets. We see that the best scores are obtained

by a small word embedding dimension w of 50, probably due to the size of these treebanks. For Indonesian, the best test scores of the unidirectional GRU are 60.67% of UAS and 52.05% of LAS when the recurrent dimension r is 128 and the hidden dimension h is 50. With the bidirectional GRU, the best test scores are 60.29% of UAS and 52.57% of LAS when $r = 32$ and $h = 128$. It is interesting to see that the BiGRU architecture is 0.52% of LAS point better than that of the GRU architecture; but it is 0.38% of UAS worse than that of the GRU architecture. We project that a treatment of both forward and backward direction by the BiGRU helps improve label prediction between a head and a dependent token in a sequence. The best performance of the bidirectional GRU model on the Vietnamese test set is achieved with $w = 50$, $r = 256$ and $h = 64$, where UAS is 52.35% and LAS is 46.93%. This model is significantly better than the baseline model with an improvement of about 11.3% of absolute point of LAS.

In the third set of experiments, we compare the performance of integrated models with the RNN-based models to see the effectiveness of graph embeddings. Table 7 displays the results on the datasets for the two compared methods. The integration of TransE embeddings helps improve the UAS and LAS scores. On the Indonesian test set, the UAS and LAS gains are about 0.2 and 0.44 of absolute points in average; the best test UAS and LAS are 60.74% and 52.35%, respectively. On the Vietnamese test set, these gains are about 0.73 for UAS and 0.77 of LAS in average; the best test UAS and LAS are 52.06% and 47.24%, respectively.

In the fourth set of experiments, we compare the performance of the methods on the English corpus. Since the English corpus is much larger than that of the Indonesian and Vietnamese datasets, to reduce running time when searching for the optimal hyperparameters, we opt to vary only the recurrent dimension, fixing the word embedding dimension at $w = 100$, part-of-speech embedding dimension at $p = 25$, shape embedding dimension at $s = 4$ and the FFN hidden dimension at $h = 256$.[15] The recurrent dimensions are varied in the set $\{128, 150, 256, 300\}$. Again, two recurrent architectures are compared, either using unidirectional GRU or bidirectional GRU. Figure 6 shows the evaluation results on the development dataset. We see that bidirectional models are better than unidirectional one. The best UAS and LAS of RNN-based models are 60.55% and 56.52%, respectively, when the recurrent dimension is 300. Using these optimal hyperparameters, the UAS and LAS scores on the test dataset are 60.43% and 56.33%, respectively.

In the fifth set of experiments, we evaluate the effectiveness of graph embeddings by running the integrated model on the English EWT corpus. Figure 7 shows the development performance of the integrated models where graph embeddings are incorporated into unidirectional and bidirectional GRU models. We observe that integrated RNN-based models outperform the previous models significantly. Their best UAS and LAS are 61.60% and 57.83%, respectively, obtained by using 256 bidirectional GRU units. The standard deviation of the GRU model is 0.00999 and that of the GRU+ model is 0.00421. We test the statistical significance of the results by performing a paired sample t test with $\alpha = 5\%$. The test result confirms that the differences between LAS scores of the two models are statistically significant, where the two-sided p-value is less than 10^{-7}.[16] Using these optimal hyperparameters, the UAS and LAS scores on the test dataset are 61.58% and 57.81%, respectively, which are significantly better than the previous results.

In summary, these experimental results have confirmed the research hypothesis—integration of embeddings learned from dependency graphs help improve dependency parsing for multiple languages. The maximal scores and gains on the three treebanks are summarized in Table 8.

Table 6 Performance of the RNN-based model on the Indonesian and Vietnamese test set obtained at their optimal hyperparameters which are tuned on the corresponding development sets

Language	w	r	h	**GRU**		BiGRU	
				UAS	LAS	UAS	LAS
Indonesian	50	32	128			60.29	52.57
	50	128	128	60.67	52.05		
Vietnamese	50	150	64			52.35	46.93
	50	256	64	51.43	46.55		

In these experiments, we fix the part-of-speech embedding dimension at $p = 25$ and the shape embedding dimension at $s = 4$

Table 7 Test performance of the two methods on the datasets at their best parameters tuned on the development datasets. GRU+ is the GRU model with graph embedding integrated.

Language	GRU		GRU+		Gain	
	UAS	LAS	UAS	LAS	Δ_u	Δ_l
Indonesian	60.67	52.05	60.74	52.35	0.07	0.30
Vietnamese	51.43	46.55	52.06	47.24	0.63	0.69

Δ_u and Δ_l are UAS and LAS gains, respectively

[15] These embedding dimensions have been tuned by Kiperwasser and Goldberg [18].

[16] We use the package HypothesisTests of Julia to perform the statistical tests.

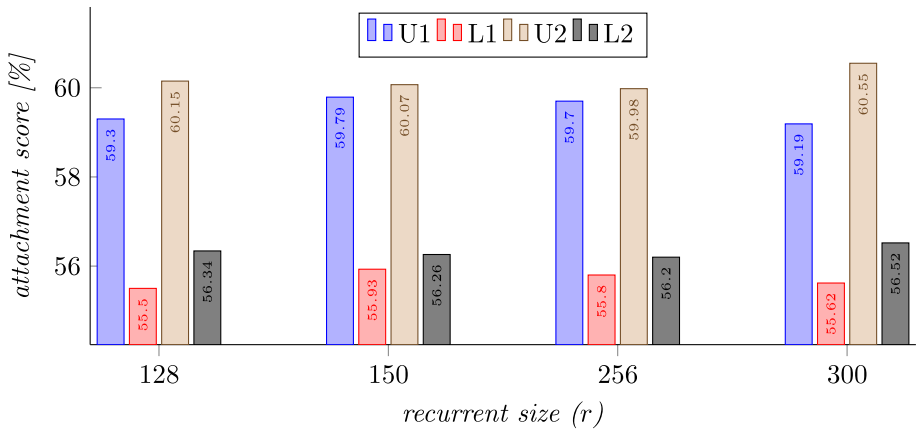

Fig. 6 Performance of the recurrent models on the English development set. The *x*-axis shows recurrent sizes and the *y*-axis shows UAS scores and LAS scores using either unidirectional GRU or bidirectional GRU architecture. U1 and L1 are UAS and LAS of unidirectional GRUs, and U2 and L2 are UAS and LAS of bidirectional GRUs

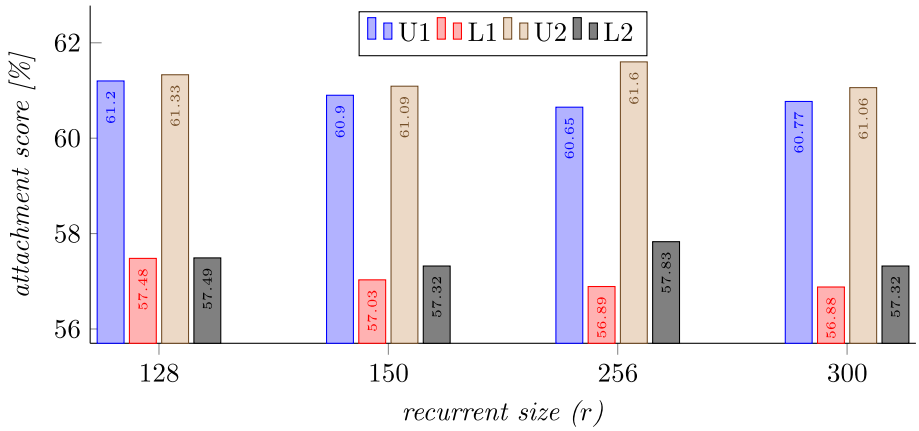

Fig. 7 Performance of the graph embedding integrated models on the English development set. Labels are interpreted as in Fig. 6

5.5 Pretrained embedding integration

In the previous experiments, we intended not to use pretrained embeddings such as ELMo [40] or BERT [9] in order to compare architectures only. As presented in subsection 3.1, pretrained models have come to dominate the NLP field during the last few years. A natural research question arises is "*Are graph embeddings and pretrained embeddings complementary? Or, do they encode redundant information?* In other words, we would like to answer the question whether or not recent pretrained models would pick up the same kind of signal as graph embeddings. We performed an additional experiment to investigate this problem on the English EWT corpus.

In this experiment, we replace the bidirectional recurrent network component of our model in Fig. 4 with a pretrained BERT for English.[17] Since BERT employs a bidirectional Transformers [43] which has the benefit of learning potential dependencies between words directly, overcoming a main drawback of recurrent models which often struggle to learn long-range dependencies [19]. For a token $\mathbf{x}_j$ in sentence S, BERT's input representation is composed by summing a word embedding $\mathbf{w}_j$, a position embedding $\mathbf{i}_j$ and a wordpiece embedding $\mathbf{p}_j$, i.e., $\vec{v}(j) = \mathbf{w}_j + \mathbf{i}_j + \mathbf{p}_j$. Each $\mathbf{x}_j$ is passed to an L-layer bidirectional transformer, which is trained with a masked language model objective. In our experiment, we adopt the uncased model of the transformer whose number of layers $L = 12$, number of hidden units $H = 768$ and $A = 12$ self-attention heads. The BERT embeddings are then passed to a FFN of 256 dimensions, as in the GRU experiments.

Interestingly, this pretrained BERT model has the best UAS and LAS of 59.25% and 56.49%, respectively, on the test set in five runs, which are in between the scores of the GRU and GRU+ models. Table 9 shows a comparison of the three models on the English treebank. This experimental result shows that the popular pretraining technique would not pick up on the same kind of signal as graph embeddings. It also suggests a potential method for improving parsing results by integrating graph embeddings into a BERT-based model in future work.

[17] More precisely, we use the model *bert-uncased_L-12_H-768_A-12* which is publicly available.

Table 8 Maximal scores and gains on the three treebanks

Language	UAS	LAS	max Δ_u	max Δ_l
English	61.58	57.81	1.15	1.48
Indonesian	60.74	52.35	0.35	0.45
Vietnamese	52.06	47.24	1.14	1.14

Table 9 Comparison of three architectures on the English EWT treebank

Model	UAS	LAS
GRU	60.43	56.33
GRU+ (graph embeddings)	**61.58**	**57.81**
BERT (pretraining)	59.25	56.49

The pretrained BERT-based model is slightly better than the supervised RNN model in terms of LAS but is worse than the graph embedding model

Table 10 Comparison of existing dependency parsing systems for Indonesian and Vietnamese

Language	LAS	System
Indonesian	79.19	[48]
	80.05	[48]
Vietnamese	47.51	[47]
	55.22	[47]

5.6 Discussion

In the CoNLL-2017 shared task on multilingual dependency parsing, the best test LAS achieved on the Indonesian and Vietnamese treebank by any parser is 79.19% and 47.51%, respectively [48]. The best performing system for Indonesian parsing builds on a deep biaffine graph-based neural dependency parser, which uses a well-tuned LSTM network to produce vector representations for each word and then uses those vector representations in biaffine classifiers to predict the head token of each dependent and the class of the resulting edge [10]. In order to address the rare word problem, it is included a character-based word representation that uses an LSTM to produce embeddings from sequences of characters. The best system for Vietnamese parsing takes an ensemble approach by blending multiple instances of three parsers with different architectures, including one graph-based parser trained with the perceptron, one transition-based beam search parser also trained with the perceptron and one greedy transition-based parser trained with neural networks [3]. One year later, in the CoNLL-2018 shared task, the best test LAS scores for Indonesian and Vietnamese were pushed to 80.05% and 55.22%, respectively. The average test scores for Indonesian and Vietnamese are 73.05% and 40.40%, respectively [47]. To our knowledge, these scores are the state-of-the-art dependency parsing results for these two low-resource languages. Table 10 shows a comparison of existing parsing systems.

In this work, we do not aim to build a sophisticated method for obtaining the best scores for dependency parsing but to develop a new method for improving a state-of-the-art parsing paradigm, using only a simple feature set. The proposed method is also simple to implement, language independent and it has been proven efficient on three dependency treebanks of different sizes and characteristics.

6 Conclusion

In this paper, we presented a method that extracts and integrates dependency graph embeddings into a state-of-the-art dependency parsing method. We evaluated the proposed method on three benchmarks for English, Indonesian and Vietnamese. Extensive experiments showed the effectiveness of our method where the learned graph embeddings help improve both the unlabeled attachment score and labeled attachment score by a clear margin over the baseline feature set and method. In addition, we also show that recent pretraining techniques such as BERT would not pick up on the same kind of signal as graph embeddings for English.

In a future work, we will investigate other representations of the dependency trees, which may take into account their complete internal structures, not only their individual nodes as in the current work. We will also investigate the embeddings of whole sentences and their corresponding dependency derivation under the same model formulation. We plan to devise more qualitative and quantitative studies of the syntactic structure embeddings as well as their applications in various NLP tasks. Syntactic structure embeddings learned from a grammar formalism such as the lexicalized tree-adjoining grammar would be a valuable source for improving dependency parsers [8]. We will also investigate methods for combining tree embeddings into recent self-supervised learning methods such as BERT [9], XLNet [46] or ELMo [40] rather than recurrent neural network based models. Finally, we will perform experiments on other languages to investigate the effectiveness of our models in a multilingual context.

Acknowledgements This study is supported by Vingroup Innovation Foundation (VINIF) in project code VINIF.2020.DA14.

Data availability The datasets generated during and/or analyzed during the current study are available in the Universal Dependencies repository: https://universaldependencies.org.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Informed Consent Informed consent was not required as no human or animals were involved.

Human and animal rights This article does not contain any studies with human or animal subjects performed by any of the authors.

References

1. Alves M (1999) What's so Chinese about Vietnamese? In: Proceedings of the Ninth Annual Meeting of the Southeast Asian Linguistics Society, pp 221–224, University of California, Berkeley, USA
2. Baroni M, Lenci A (2010) Distributional memory: a general framework for corpus-based semantics. Comput Linguist 36(4):673–721
3. Björkelund A, Falenska A, Yu X, and Kuhn J (2017) Ims at the conll 2017 ud shared task: Crfs and perceptrons meet neural networks. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pp 40–51, Vancouver, Canada. Association for Computational Linguistics
4. Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O (2013) Translating embeddings for modeling multi-relational data. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) Advances in Neural Information Processing Systems, vol 26. Curran Associates Inc, pp 1–9
5. Buchholz S and Marsi E (2006) CoNLL-X shared task on multilingual dependency parsing. In: Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X), pp 149–164, New York City. Association for Computational Linguistics
6. Cavallari S, Cambria E, Cai H, Chang K, Zheng V (2019) Embedding both finite and infinite communities on graph. IEEE Comput Intell Mag 14(3):39–50
7. Chen D. and Manning C (2014) A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp 740–750, Doha, Qatar. Association for Computational Linguistics
8. Dang HV and Le-Hong P (2021) A combined syntactic-semantic embedding model based on lexicalized tree-adjoining grammar. Comput Speech Lang 68
9. Devlin J, Chang M-W, Lee K, and Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL, pages 1–16, Minnesota, USA
10. Dozat T, Qi P, and Manning CD (2017) Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pp 20–30, Vancouver, Canada. Association for Computational Linguistics
11. Dyer C, Ballesteros M, Ling W, Matthews A, and Smith NA (2015) Transition-based dependency parsing with stack long short-term memory. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp 334–343, Beijing, China. Association for Computational Linguistics
12. Fernandez Astudillo R, Ballesteros M, Naseem T, Blodgett A, and Florian R (2020) Transition-based parsing with stack-transformers. In Findings of the Association for Computational Linguistics: EMNLP 2020, pp 1001–1007, Online. Association for Computational Linguistics
13. Green N, Larasati SD, and Zabokrtsky Z (2012) Indonesian dependency treebank: annotation and parsing. In: Proceedings of the 26th Pacific Asia Conference on Language, Information, and Computation, pp 137–145, Bali, Indonesia. Faculty of Computer Science, Universitas Indonesia
14. Harris ZS (1954) Distributional structure. Word 10(2–3):146–162
15. Ji S, Pan S, Cambria E, Marttinen P, Yu PS (2022) A survey on knowledge graphs: representation, acquisition and applications. IEEE Trans Neural Netw Learn Syst 33(10):494–514
16. Kingma DP and Ba J (2015) Adam: a method for stochastic optimization. In: Bengio Y and LeCun Y, eds, Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, pp 1–15, San Diego, CA, USA
17. Kiperwasser E, Goldberg Y (2016) Easy-first dependency parsing with hierarchical tree LSTMs. Trans Assoc Comput Linguist 4:445–461
18. Kiperwasser E, Goldberg Y (2016) Simple and accurate dependency parsing using bidirectional LSTM feature representations. Trans Assoc Comput Linguist 4:313–327
19. Kolen JF and Kremer SC (2001) Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp 237–243. IEEE
20. Kondratyuk D. and Straka M (2019) 75 languages, 1 model: parsing Universal Dependencies universally. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp 2779–2795, Hong Kong, China. Association for Computational Linguistics
21. Kübler S, McDonald R, and Nivre J (2009) Dependency parsing. Morgan & Claypool Publishers
22. Le P and Zuidema W (2014) The inside-outside recursive neural network model for dependency parsing. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp 729–739, Doha, Qatar. Association for Computational Linguistics
23. Le-Hong P, Nguyen TMH, and Azim R (2012) Vietnamese parsing with an automatically extracted tree-adjoining grammar. In: Proceedings of the IEEE RIVF, pp 91–96, HCMC, Vietnam
24. Le-Hong P, Roussanaly A, Nguyen T-M-H (2015) A syntactic component for Vietnamese language processing. J Lang Modell 3(1):145–184
25. Lei T, Xin Y, Zhang Y, Barzilay R, and Jaakkola T (2014) Low-rank tensors for scoring dependency structures. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp 1381–1391, Baltimore, Maryland. Association for Computational Linguistics
26. Levy O and Goldberg Y (2014) Dependency-based word embeddings. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp 302–308, Baltimore, Maryland. Association for Computational Linguistics
27. Ling W, Tsvetkov Y, Amir S, Fermandez R, Dyer C, Black AW, Trancoso I, and Lin C-C (2015) Not all contexts are created equal: better word representations with variable attention. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp 1367–1372, Lisbon, Portugal. Association for Computational Linguistics

28. Liu J and Zhang Y (2017) Encoder-decoder shift-reduce syntactic parsing. In: Proceedings of the 15th International Conference on Parsing Technologies, pp 105–114, Pisa, Italy. Association for Computational Linguistics
29. McDonald R, Nivre J (2011) Analyzing and integrating dependency parsers. Comput Linguist 37(1):197–230
30. McDonald R and Pereira F (2006) Online learning of approximate dependency parsing algorithms. In: Proceedings of EACL, pp 81–88, Trento, Italy
31. McDonald R, Pereira F, Ribarov K, and Hajic J (2005) Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of HLT-EMNLP, pp 522–530, Vancouver, Canada
32. Nguyen TL, Ha ML, Nguyen VH, Nguyen TMH, and Le-Hong P (2013) Building a treebank for Vietnamese dependency parsing. In The 10th IEEE RIVF, pp 147–151, Hanoi, Vietnam. IEEE
33. Nivre J (2003) An efficient algorithm for projective dependency parsing. In: Proceedings of the Eighth International Conference on Parsing Technologies, pp 149–160, Nancy, France
34. Nivre J, de Marneffe M-C, Ginter F, Goldberg Y, Hajič J, Manning CD, McDonald R, Petrov S, Pyysalo S, Silveira N, Tsarfaty R, and Zeman D (2016) Universal Dependencies v1: a multilingual treebank collection. In: Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16), pp 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA)
35. Nivre J, Hall J, Kübler S, McDonald R, Nilsson J, Riedel S, and Yuret D (2007) The CoNLL 2007 shared task on dependency parsing. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp 915–932, Prague, Czech Republic. Association for Computational Linguistics
36. Nivre J and McDonald R (2008) Integrating graph-based and transition-based dependency parsers. In: Proceedings of ACL-08, pp 950–958, Columbus, Ohio, USA. ACL
37. Nivre J and Scholz M (2004) Deterministic dependency parsing of English text. In: Proceedings of COLING 2004, pp 1–7, Geneva, Switzerland
38. Nivre JEA (2018) Universal dependencies 2.2. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University
39. Lewis MP, Simons GF, Fennig CD (eds) (2014) Ethnologue: languages of the World, 17th edn. SIL International, Dallas, Texas, USA
40. Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, and Zettlemoyer L (2018) Deep contextualized word representations. In: Proceedings of NAACL, pp 1–15, Louisiana, USA
41. Sneddon JN (2004) The Indonesian language: its history and role in modern society. UNSW Press
42. Turian J, Ratinov L, and Bengio Y (2010) Word representations: a simple and general method for semi-supervised learning. In: Proceedings of ACL, pp 384–394, Uppsala, Sweden
43. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in Neural Information Processing Systems, vol 30. Curran Associates Inc
44. Velickovic P, Cucurull G, Casanova A, Romero A, Lio P, and Bengio Y (2018) Graph attention networks. In: Proceedings of the Sixth International Conference on Learning Representations (ICLR), pp 1–12, Vancouver, Canada
45. Wilie B, Vincentio K, Winata GI, Cahyawijaya S, Li X, Lim ZY, Soleman S, Mahendra R, Fung P, Bahar S, and Purwarianti A (2020) IndoNLU: benchmark and resources for evaluating Indonesian natural language understanding. In: Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing. Association for Computational Linguistics
46. Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov R, and Le QV (2019) XLNet: generalized autoregressive pretraining for language understanding. In: Proceedings of NeurIPS, pp 5754–5764
47. Zeman D, Hajič J, Popel M, Potthast M, Straka M, Ginter F, Nivre, J, and Petrov S (2018) CoNLL 2018 shared task: multilingual parsing from raw text to Universal Dependencies. In: Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pp 1–21, Brussels, Belgium. Association for Computational Linguistics
48. Zeman D, Popel M, Straka M, Hajič J, Nivre J, Ginter F, Luotolahti J, Pyysalo S, Petrov S, Potthast M, Tyers F, Badmaeva E, Gokirmak M, Nedoluzhko A, Cinková S, Hajič jr J, Hlaváčová J, Kettnerová V, Urešová Z, Kanerva J, Ojala S, Missilä A, Manning CD, Schuster S, Reddy S, Taji D, Habash N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H, de Paiva, V., Droganova, K., Martínez Alonso, H., Çöltekin, Ç., Sulubacak, U., Uszkoreit, H, Macketanz, V, Burchardt A, Harris K, Marheinecke K, Rehm G, Kayadelen T, Attia M, Elkahky A, Yu Z, Pitler E, Lertpradit S, Mandl M, Kirchner J, Alcalde HF, Strnadová J, Banerjee E, Manurung R, Stella A, Shimada A, Kwak S, Mendonça G, Lando T, Nitisaroj R, and Li J (2017) CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pp 1–19, Vancouver, Canada. Association for Computational Linguistics
49. Zhang Y and Nivre J (2011) Transition-based dependency parsing with rich non-local features. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp 188–193, Portland, Oregon, USA. Association for Computational Linguistics
50. Zhang Z, Liu S, Li M, Zhou M, and Chen E (2017) Stack-based multi-layer attention for transition-based dependency parsing. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp 1677–1682, Copenhagen, Denmark. Association for Computational Linguistics
51. Zhu C, Qiu X, Chen X, and Huang X (2015) A re-ranking model for dependency parser with recursive convolutional neural network. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp 1159–1168, Beijing, China. Association for Computational Linguistics

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.