A non-projective greedy dependency parser with bidirectional LSTMs

David Vilares
Universidade da Coruña
LyS Group
Departamento de Computación
Campus de Elviña s/n, 15071
A Coruña, Spain
david.vilares@udc.es

Carlos Gómez-Rodríguez
Universidade da Coruña
FASTPARSE Lab, LyS Group
Departamento de Computación
Campus de A Elviña s/n, 15071
A Coruña, Spain
carlos.gomez@udc.es

Abstract

The LyS-FASTPARSE team presents BIST-COVINGTON, a neural implementation of the Covington (2001) algorithm for non-projective dependency parsing. The bidirectional LSTM approach by Kiperwasser and Goldberg (2016) is used to train a greedy parser with a dynamic oracle to mitigate error propagation. The model participated in the *CoNLL 2017 UD Shared Task*. In spite of not using any ensemble methods and using the baseline segmentation and PoS tagging, the parser obtained good results on both macro-average LAS and UAS in the *big treebanks* category (55 languages), ranking 7th out of 33 teams. In the *all treebanks* category (LAS and UAS) we ranked 16th and 12th. The gap between the *all* and *big* categories is mainly due to the poor performance on four parallel PUD treebanks, suggesting that some 'suffixed' treebanks (e.g. Spanish-AnCora) perform poorly on cross-treebank settings, which does not occur with the corresponding 'unsuffixed' treebank (e.g. Spanish). By changing that, we obtain the 11th best LAS among all runs (official and unofficial). The code is made available at https://github.com/CoNLL-UD-2017/LyS-FASTPARSE

1 Introduction

Dependency parsing is one of the core structured prediction tasks researched by computational linguists, due to the potential advantages that obtaining the syntactic structure of a text has in many natural language processing applications, such as machine translation (Miceli-Barone and Attardi, 2015; Xiao et al., 2016), sentiment analysis (Socher et al., 2013; Vilares et al., 2017) or information extraction (Yu et al., 2015).

The goal of a dependency parser is to analyze the syntactic structure of sentences in one or several human languages by obtaining their analyses in the form of dependency trees. Let $w = [w_1, w_2, ..., w_{|w|}]$ be an input sentence, a *dependency tree* for w is an edge-labeled directed tree $T = (V, E)$ where $V = \{0, 1, 2, \ldots, |w|\}$ is the set of nodes and $E = V \times D \times V$ is the set of labeled arcs. Each arc, of the form (i, d, j), corresponds to a syntactic *dependency* between the words w_i and w_j; where i is the index of the *head* word, j is the index of the *child* word and d is the *dependency type* representing the kind of syntactic relation between them.[1] We will write $i \xrightarrow{d} j$ as shorthand for $(i, d, j) \in E$ and we will omit the dependency types when they are not relevant.

A dependency tree is said to be non-projective if it contains two arcs $i \rightarrow j$ and $k \rightarrow l$ where $min(i, j) < min(k, l) < max(i, j) < max(k, l)$, i.e., if there is any pair of arcs that cross when they are drawn over the sentence, as shown in Figure 1. Unrestricted non-projective parsing allows more accurate syntactic representations than projective parsing, but it comes at a higher computational cost, as there is more flexibility in how the tree can be arranged so that more operations are usually needed to explore the much larger search space.

Non-projective transition-based parsing has been actively explored in the last decade (Nivre and Nilsson, 2005; Attardi, 2006; Nivre, 2008, 2009; Gómez-Rodríguez and Nivre, 2010; Gómez-Rodríguez et al., 2014). The success of neural networks and word embeddings for pro-

[1]Following common practice, we are using node 0 as a dummy root node that acts as the head of the syntactic root(s) of the sentence.

Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 152–162,
Vancouver, Canada, August 3-4, 2017. © 2017 Association for Computational Linguistics

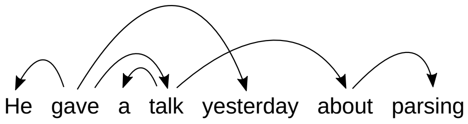

Figure 1: A non-projective dependency tree

jective dependency parsing (Chen and Manning, 2014) also encouraged research on neural non-projective models (Straka et al., 2016). However, to the best of our knowledge, no neural implementation is available of unrestricted non-projective transition-based parsing with a dynamic oracle. Here, we present such an implementation for the Covington (2001) algorithm using bidirectional long short-term memory networks (LSTM) (Hochreiter and Schmidhuber, 1997), which is the main contribution of this paper.

The system is evaluated at the *CoNLL 2017 UD Shared Task*: *end-to-end multilingual parsing using Universal Dependencies* (Zeman et al., 2017). The goal is to obtain a Universal Dependencies v2.0 representation (Nivre et al., 2016) of a collection of raw texts in different languages.

2 End-to-end multilingual parsing

Given a raw text, we: (1) segment and tokenize sentences and words, (2) apply part-of-speech (PoS) tagging over them and (3) obtain the dependency structure for each sentence.

2.1 Segmentation and PoS tagging

For these two steps we relied on the output provided by UDpipe v1.1 (Straka et al., 2016), which was provided as a baseline model for the shared task.

2.2 The BIST-COVINGTON parser

BIST-COVINGTON is built on the top of three core ideas: a non-projective transition-based parsing algorithm (Covington, 2001; Nivre, 2008), a neural scoring model with bidirectional long short-term memory networks as feature extractors that feed a multilayer perceptron (Kiperwasser and Goldberg, 2016), and a dynamic oracle to mitigate error propagation (Gómez-Rodríguez and Fernández-González, 2015).

2.2.1 The Covington (2001) algorithm

The idea of Covington's algorithm is quite intuitive: any pair of words w_i, w_j in w have a chance to be connected, so we need to consider all such pairs to determine the type of relation that exists between them (i.e. $i \xrightarrow{d} j$, $j \xrightarrow{d} i$ or $\texttt{none}$). One pair (i, j) is compared at a time. We will be referring to the indexes i and j as the *focus words*. It is straightforward to conclude that the theoretical complexity of the algorithm is $\mathcal{O}(|w|^2)$.

Covington's algorithm can be easily implemented as a transition system (Nivre, 2008). The set of transitions used in BIST-COVINGTON and their preconditions is specified in Table 1. Each transition corresponds to a parsing configuration represented as a 4-tuple $c = (\lambda_1, \lambda_2, \beta, A)$, such that:

- λ_1, λ_2 are two lists storing the words that have been already processed in previous steps. λ_1 contains the already processed words for which the parser still has not decided, in the current state, the type of relation with respect to the focus word j, located at the top of β. λ_2 contains the already processed words for which the parser has already determined the type of relation with respect to j in the current step.

- β contains the words to be processed.

- A contains the set of arcs already created.

Given a sentence w the parser starts at an initial configuration $c_s = ([0], [], [1, ..., |w|], \{\})$ and will apply valid transitions until reaching a final configuration c_f such that $c_f = (\lambda_1, \lambda_2, [], A)$. Figure 2 illustrates an intermediate parsing configuration for our introductory example.

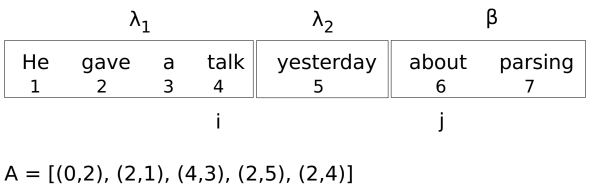

Figure 2: A parsing configuration for our introductory example just before creating a non-projective RIGHT ARC *talk* $\rightarrow$ *about*.

2.2.2 A dynamic oracle for Covington's algorithm (Gómez-Rodríguez and Fernández-González, 2015)

Given a gold dependency tree, τ_g, and a parser configuration c, we can define a loss function

Transitions		
LEFT ARC	$(\lambda_1\|i, \lambda_2, j\|\beta, A)$	$(\lambda_1, i\|\lambda_2, j\|\beta, A \cup \{(j, d, i)\})$
RIGHT ARC	$(\lambda_1\|i, \lambda_2, j\|\beta, A)$	$(\lambda_1, i\|\lambda_2, j\|\beta, A \cup \{(i, d, j)\})$
SHIFT	$(\lambda_1, \lambda_2, i\|\beta, A)$	$(\lambda_1 \cdot \lambda_2\|i, [], \beta, A)$
NO-ARC	$(\lambda_1\|i, \lambda_2, \beta, A)$	$(\lambda_1 \cdot i\|\lambda_2, \beta, A)$
Preconditions		
LEFT ARC	$i > 0$ *and* $\nexists(k \rightarrow i) \in A$ *and* $\nexists(i \rightarrow ... \rightarrow j)$	
RIGHT ARC	$\nexists(k \rightarrow j) \in A$ *and* $\nexists(j \rightarrow ... \rightarrow i)$	
NO-ARC	$i > 0$	

Table 1: Set of transitions for BIST-COVINGTON as described in Nivre (2008). $a \rightarrow ... \rightarrow b$ indicates there is a path in the dependency tree that allows to reach b from a

$\mathcal{L}(c, \tau_g)$ that determines the minimum number of missed arcs of τ_g across the possible outputs (A) of final configurations that can be reached from c, i.e., the least possible number of errors with respect to τ_g that we can obtain from c. A static (traditional) oracle is only defined on canonical transition sequences that lead to the gold tree, so that $\mathcal{L}(c, \tau_g) = 0$ at every step during the training phase. However, during the test phase such training strategy might end up in serious error propagation, as it is difficult for the parser to recover from wrong configurations that it has never seen, resulting from suboptimal transitions that increase loss. A dynamic oracle (Goldberg and Nivre, 2012) explores such wrong configurations during the training phase to overcome this issue. Instead of always picking the optimal transition during training, the parser moves with probability x to an erroneous (loss-increasing) configuration, namely the one with the highest score among those that increase loss.

To compute $\mathcal{L}$ for non-projective trees we used the approach proposed by Gómez-Rodríguez and Fernández-González (2015, Algorithm 1). This dynamic oracle can be computed in $\mathcal{O}(|w|)$ although the current implementation in BIST-COVINGTON is $\mathcal{O}(|w|^3)$. To choose the dependency type corresponding to the selected transition (in case it is a LEFT or RIGHT ARC), we look at the gold treebank.

2.2.3 The BIST-parsers (Kiperwasser and Goldberg, 2016)

The original set of BIST-parsers is composed of a projective transition-based model using the arc-hybrid algorithm (Kuhlmann et al., 2011) and a graph-based model inspired in Eisner (1996). They both rely on bidirectional LSTM's (BILSTM's). We kept the main architecture of the arc-hybrid BIST-parser and changed the parsing algorithm to that described in §2.2.1 and §2.2.2. We encourage the reader to consult Kiperwasser and Goldberg (2016) for a detailed explanation of their architecture, but we now try to give a quick overview of its use as the core part of BIST-COVINGTON.[2]

In contrast to traditional parsers (Nivre et al., 2006; Martins et al., 2010; Rasooli and Tetreault, 2015), BIST-parsers rely on embeddings as inputs instead of on discrete events (co-occurrences of words, tags, features, etc.). Embeddings are low-dimensional vectors that provide a continuous representation of a linguistic unit (word, PoS tag, etc.) based on its context (Mikolov et al., 2013).

Let $\mathbf{w}$=$[\mathbf{w}_1, ..., \mathbf{w}_{|w|}]$ be a list of word embeddings for a sentence, let $\mathbf{u}$=$[\mathbf{u}_1, ..., \mathbf{u}_{|w|}]$ be the corresponding list of universal PoS tag embeddings, $\mathbf{t}$=$[\mathbf{t}_1, ..., \mathbf{t}_{|w|}]$ the list of specific PoS tag embeddings, $\mathbf{f}$=$[\mathbf{f}_1, ..., \mathbf{f}_{|w|}]$ the list of morphological features ("feats" column in the Universal Dependencies data format) and $\mathbf{e}$=$[\mathbf{e}_1, ..., \mathbf{e}_{|w|}]$ a list of external word embeddings; an input $\mathbf{x}_i$ for a word w_i to BIST-COVINGTON is defined as:[3]

$$\mathbf{x}_i = \mathbf{w}_i \circ \mathbf{u}_i \circ \mathbf{t}_i \circ \mathbf{f}_i \circ \mathbf{e}_i$$

where $\circ$ is the concatenation operator.

Let LSTM($\mathbf{x}$) be an abstraction of a standard long short-term memory network that processes the sequence $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_{|\mathbf{x}|}]$, then a BILSTM encoding of its ith element, BILSTM($\mathbf{x}, i$) can be

[2] Including some additional capabilities that we included especially for BIST-COVINGTON.

[3] It might turn out that for some treebank/language some of this information is not available, in which case the unavailable elements are considered as empty lists.

defined as:

$$\text{BILSTM}(\mathbf{x}, i) = \text{LSTM}(\mathbf{x}_{1:i}) \circ \text{LSTM}(\mathbf{x}_{|\mathbf{x}|:i})$$

In the case of multilayer BILSTM'S (BIST-parsers allow it), given n layers, the output of the BILSTM_m is fed as input to BILSTM_{m+1}. From the BILSTM network we take a hidden vector $\mathbf{h}$, which can contain the output hidden vectors for: the x leftmost words in β, the rightmost y of λ_1, and the z leftmost and v rightmost words in λ_2.

The hidden vector $\mathbf{h}$ is used to feed a multilayer perceptron with one hidden layer and four output neurons that predicts which transition to take. The output is computed as $W_2 \cdot tanh(W \cdot \mathbf{h} + b) + b_2$, where W, W_2, b and b_2 correspond to the weight matrices and bias vectors of the hidden and output layer of the perceptron. Similarly, BIST-parsers (including BIST-COVINGTON) use a second perceptron with one hidden layer to predict the dependency type. In this case the output layer corresponds to the number of dependency types in the training set.

2.3 Postprocessing

BIST-COVINGTON as it is allows parses with multiple roots, i.e., with several nodes assigned as children of the dummy root. This was not allowed however by the task organizers, as it is enforced by Universal Dependencies that only one word per sentence must depend on the dummy root. To overcome this, the output is postprocessed according to Algorithm 1. Basically, we look for the first verb rooted at 0, or for the first word whose head is 0 if there is no verb, and reassign all other words to the selected term:

Algorithm 1 Multiple to single node root

```
procedure TO_SINGLE(V, E)
    ▷ Get the nodes rooted at zero (those whose head has to be reassigned)
    RO ← []
    for i in V do
        if head(i) = 0 then
            append(RO, i)
    ▷ We select the first verb linked to the dummy root to remove multiple roots
    if len(RO) > 1 then
        closest_head ← RO[0]
        for r0 in RO do
            if utag(r0) = VERB then
                closest_head ← r0
                break
    ▷ Reassign the head of the invalid nodes (rooted to the dummy root) to closest_head
        for r0 in RO do
            if r0 ≠ closest_head then
                head(r0) ← closest_head
```

3 Experiments

We here describe the official treebanks used in the shared task (§3.1), the general setup used to train the models (§3.2) and some exceptions to said general setup that were applied to special cases (§3.3). We also discuss the experimental results obtained by our system in the shared task (§3.4).

3.1 CoNLL 2017 treebanks

3.1.1 Training/development splits

60 treebanks from 45 languages were released to train the models, based on Universal Dependencies 2.0 (Nivre et al., 2017a). Most of them already contained official training and development splits. A few others lacked a development set. For these, we applied a training/dev random split (80/20) over the original training set. All development sets were only used to evaluate and tune the trained models. *No development set* was used to train any of the runs, as specified in the task guidelines.

Additionally, four *surprise languages* (truly low resource languages), were considered by the organization for evaluation: Buryat, Kurmanji, North Sami and Upper Sorbian. For these, the organizers only released a tiny sample set consisting of very few sentences annotated according to the UD guidelines.

3.1.2 Test splits

The organizers provided a test split for each of the treebanks released in the training phase, including the surprise languages. Additionally, they provided test sets corresponding to 14 parallel treebanks in different languages translated from a unique source. All of these test sets (Nivre et al., 2017b) were hidden from the participating teams until the shared task had ended. Using the TIRA environment (Potthast et al., 2014) provided for the shared task, participants could execute runs on them, but not see the outputs or the results.

3.2 General setup

We used the gold training treebanks to train the parsing models. We trained one model per treebank. No predicted training treebank (predicted universal and/or specific tags and morphological features) was used for training, except for the case of Portuguese (see §3.3.1).

Embeddings: Word embeddings are set to size 100 and universal tag embeddings to 25. Language-specific tag and morphological feature embeddings are used and set to size 25, if they are available for the treebank at hand. Using external word embeddings seems to be beneficial to improve parsing performance (Kiperwasser and Goldberg, 2016), but it also makes models take more time and especially much more memory to train. The external word embeddings used in this work (the ones pretrained by the *CoNLL 2017 UD Shared Task* organizers[4]) are of size 100. Due to lack of enough computational resources, we only had time to train 38 models (mainly corresponding to the smallest treebanks) including this information. Models trained with external word embeddings are marked in Table 3 with $\star$.

Parameters: Adam is used as optimizer (Kingma and Ba, 2014). Models were trained for up to 30 epochs, except for the two smallest training sets (Kazakh and Uyghur), where models were trained for up to 100 epochs. The size of the output of the stacked BILSTM was set to 512. For very large treebanks (e.g. Czech or Russian-SyntagRus) or treebanks where sentences are very long (e.g. Arabic), we set it to 256, also to counteract the lack of physical resources to finish the task on time. These models are marked in Table 3 with $\bullet$. The number of BILSTM layers is set to 2. To choose a transition, BIST-COVINGTON looks at the embeddings of: the first word in β, the rightmost three words in λ_1, and the leftmost and rightmost word in λ_2 (i.e., following the notation in Section 2.2.3, we set $x = 1$, $y = 3$, $z = 1$ and $v = 1$).

Other relevant features of the setup: Aggressive exploration is applied to the dynamic oracle, as in the original arc-hybrid BIST-PARSER.

3.3 Special cases

For some treebanks, we followed a different strategy due to various issues. We enumerate the changes below:

3.3.1 The Portuguese model

Surprisingly, the model trained on the Portuguese treebank suffered a significant loss with respect to the UDpipe baseline when parsing the full predicted (segmentation and tagging) development file. We first hypothesized this was due to a low accuracy on predicting the "feats" column in comparison to other languages, as they are pretty sparse. To try to overcome this, we trained a model without considering them, but it did not solve the problem. Our second option was to train a Portuguese model on its predicted training treebank.[5] Additionally, despite being a relative large treebank, we included external word embeddings to boost performance. This helped us to obtain a performance similar to that reported by UDpipe.

3.3.2 Surprise languages

As training an accurate parser with so little data might be a hard task , especially in the case of *data-hungry* deep learning models, we used other training treebanks for this purpose. We built a set of parsers inspired on the approach presented by Vilares et al. (2016), who find that training a multilingual model on merged harmonized treebanks might actually have a positive impact on parsing the corresponding monolingual treebank. In this particular case, we are assuming that a trained model over multilingual treebanks might be able to capture similar treebank structures for unseen languages.

In particular, we: (1) ran every trained monolingual model on the sample sets, (2) for each surprise language, we chose the top three languages where the corresponding models obtained the best performance and (3) trained a parser taking the first 2 000 sentences of the training sets corresponding to such languages and merging them.

Thus, we did not use the provided sample data for training, but only as a development set to choose suitable source languages for our cross-lingual approach.

3.3.3 Parallel (PUD) treebanks

The only information our models knew about the parallel treebanks during the testing phase was the language in which they were written. To parse these languages we follow a simplistic approach, using the models we had already trained on the provided training corpora: (1) if there is only one model trained on the same language we take that model, (2) else if there is more than one model trained on that language, we take the one trained over the largest treebank (in number of sentences),

[4] http://hdl.handle.net/11234/1-1989

[5] We used the predicted tokenization and tagging provided by UDpipe.

otherwise (3) we parse the PUD treebank using the English model.[6]

3.4 Results

Official and unofficial results for our model and for the rest of participants on the test set can be found at the task website: http://universaldependencies.org/conll17/results.html, but in this section we detail the results obtained by BIST-COVINGTON.

3.4.1 Results on *small* and *big* treebanks categories

Table 2 shows the performance on the test sets for the treebanks where an official training set was released.

In Table 3 we summarize our results on the development sets for those treebanks that provided an official one. Although not shown for brevity and clarity reasons, it is easy to check for the reader that BIST-COVINGTON outperformed the baseline UDpipe[7] for all these treebanks on the gold configuration (gold segmentation, gold tags). The same is true, except for Chinese (-0.69 decrease in LAS) and Portuguese (-0.09), in the fully predicted configuration (end-to-end parsing). It is easy to conclude from the table that including external word embeddings has a positive effect in most of the treebanks we had time to try. This is especially true when performing end-to-end parsing, where only for three languages (English-LinES, Gothic and Old Church Slavonic) a negative effect was observed.[8]

Table 4 shows the top three selected languages for each surprise treebank, the performance of the monolingual and multilingual (merged) models on them on the sample set (used as dev set), and also shows the performance of the multilingual models in the official test sets.

Table 5 shows our performance on the PUD treebanks (test sets). There are 4 PUD treebanks for which we obtained a poor performance: Spanish, Finnish, Portuguese and Russian. Average LAS loss with respect to the top system in the cor-

Treebank	LAS
Ancient_Greek -PROIEL	67.85_{8}
Ancient_Greek	59.83_{6}
Arabic	66.54_{10}
Basque	73.27_{5}
Bulgarian	85.76_{6}
Catalan	85.37_{18}
Chinese	56.76_{2}
Croatian	77.91_{11}
Czech-CAC	82.71_{16}
Czech-CLTT	68.92_{23}
Czech	83.77_{11}
Danish	75.27_{11}
Dutch-LassySmall	82.49_{6}
Dutch	71.89_{7}
English-LinES	73.47_{13}
English-ParTUT	74.50_{12}
English	76.00_{14}
Estonian	61.79_{7}
Finnish-FTB	76.80_{7}
Finnish	76.11_{8}
French-ParTUT	72.09_{25}
French-Sequoia	77.77_{23}
French	79.86_{20}
Galician-TreeGal	65.42_{17}
Galician	79.24_{12}
German	68.35_{22}
Gothic	62.07_{7}
Greek	81.43_{6}
Hebrew	59.28_{9}
Hindi	86.88_{15}
Hungarian	66.00_{9}
Indonesian	72.94_{23}
Irish	58.05_{22}
Italian	85.60_{16}
Japanese	72.68_{17}
Kazakh	16.20_{26}
Korean	63.85_{14}
Latin-ITTB	79.58_{7}
Latin-PROIEL	61.45_{7}
Latin	48.92_{7}
Latvian	63.05_{7}
Norwegian-Bokmaal	84.49_{8}
Norwegian-Nynorsk	83.10_{7}
Old_Church_Slavonic	67.21_{4}
Persian	77.68_{17}
Polish	82.09_{7}
Portuguese-BR	86.74_{9}
Portuguese	80.91_{19}
Romanian	80.58_{11}
Russian-SynTagRus	87.55_{9}
Russian	76.98_{8}
Slovak	76.47_{6}
Slovenian-SST	43.80_{21}
Slovenian	82.92_{7}
Spanish-AnCora	86.83_{7}
Spanish	83.24_{8}
Swedish-LinES	75.04_{10}
Swedish	77.33_{13}
Turkish	57.22_{5}
Ukrainian	61.21_{15}
Urdu	78.31_{9}
Uyghur	27.92_{23}
Vietnamese	38.33_{12}

Table 2: BIST-COVINGTON results on the test sets, for those treebanks from which a training set was provided (*small* and *big* treebanks categories)

[6] This latter case should and did never happen, as the task organizers specified in advance that the parallel treebanks would correspond to languages with existing treebanks, but we included it as a fallback mechanism.

[7] http://universaldependencies.org/conll17/baseline.html

[8] Due to not so rich embeddings and/or the model finishing earlier than expected during training. See §5.

Treebank	Gold treebank LAS no E	Gold treebank LAS E	Predicted treebank LAS no E	Predicted treebank LAS E
Ancient_Greek -PROIEL	81.44	N/A	70.5	N/A
Ancient_Greek*	71.01	**71.31**	60.41	**61.25**
Arabic$^{\star\bullet}$	79.12	**79.71**	64.37	**65.62**
Basque*	81.53	**82.06**	72.00	**73.42**
Bulgarian*	89.88	**90.46**	84.33	**85.30**
Catalan$^{\bullet}$	90.63	N/A	87.21	N/A
Chinese	80.34	N/A	55.31	N/A
Croatian*	**83.86**	83.64	78.04	**78.74**
Czech-CAC$^{\bullet}$	88.64	N/A	84.93	N/A
Czech-CLTT$^{\bullet}$	82.28	N/A	68.03	N/A
Czech$^{\bullet}$	90.70	N/A	85.47	N/A
Danish*	83.85	**85.78**	74.92	**76.94**
Dutch-LassySmall*	86.59	**86.65**	76.78	**77.50**
Dutch	86.82	N/A	76.47	N/A
English-LinES*	**83.74**	83.05	**76.48**	76.44
English-ParTUT*	84.15	**84.60**	76.24	**77.07**
English	**88.02**	N/A	76.7	N/A
Estonian*	79.26	**80.21**	61.09	**62.80**
Finnish-FTB	**89.00**	N/A	76.43	N/A
Finnish	86.51	N/A	76.96	N/A
French-Sequoia	89.14	N/A	81.79	N/A
French$^{\bullet}$	89.86	N/A	85.8	N/A
Galician$^{\star\bullet}$	84.22	82.58	80.17	79.03
German	87.63	N/A	73.61	N/A
Gothic*	80.82	**81.17**	**60.84**	60.82
Greek*	86.03	**86.37**	79.74	**80.05**
Hebrew$^{\star\bullet}$	**85.26**	85.13	62.18	**62.39**
Hindi	93.42	N/A	87.41	N/A
Hungarian*	80.84	**81.30**	69.16	**70.43**
Indonesian	80.39	N/A	74.91	N/A
Italian-ParTUT*	86.20	**86.83**	78.90	**79.56**
Italian	90.30	N/A	86.05	N/A
Japanese*	**96.48**	96.46	73.99	**74.20**
Korean	68.66	N/A	60.18	N/A
Latin-ITTB	84.21	N/A	72.22	N/A
Latin-PROIEL	79.37	N/A	61.98	N/A
Latvian*	**77.25**	76.55	63.12	**63.62**
Norwegian-Bokmaal	91.45	N/A	85.13	N/A
Norwegian-Nynorsk	91.06	N/A	83.38	N/A
Old_Church_Slavonic*	**84.59**	84.52	**66.93**	66.66
Persian$^{\star\bullet}$	86.85	N/A	80.44	81.45
Polish*	91.04	**91.25**	81.43	**82.18**
Portuguese-BR$^{\bullet}$	90.91	N/A	86.41	N/A
Portuguese$^{\star\bullet}$	**94.94**	93.09	79.3	**84.00**
Romanian*	**85.08**	84.44	80.97	**81.01**
Russian-SynTagRus$^{\bullet}$	91.91	N/A	88.29	N/A
Russian*	85.12	**86.07**	78.02	**79.09**
Slovak*	87.61	**88.39**	75.59	**77.35**
Slovenian*	92.28	**93.14**	82.48	**84.15**
Spanish-AnCora$^{\bullet}$	90.50	N/A	86.21	N/A
Spanish$^{\bullet}$	87.90	N/A	84.25	N/A
Swedish-LinES*	84.23	**84.44**	76.39	**76.86**
Swedish*	84.88	**85.03**	76.41	**76.64**
Turkish*	61.66	**64.46**	55.05	**57.60**
Urdu*	**87.63**	87.50	77.43	**77.49**
Vietnamese*	72.21	**72.58**	42.27	**42.94**

Table 3: BIST-COVINGTON results on the dev set, for those treebanks that have an official dev set (all treebanks except French-ParTUT, Irish, Galician-TreeGal, Kazakh, Slovenian-SST, Kazakh, Uyghur and Ukrainian). $\star$ indicates the model was also trained with external word embeddings (E). $\bullet$ indicates the BILSTM output dimension was 256. The performance of some models is likely to be improved, as its training finished earlier than expected due to lack of time to finish it or memory issues (see also §5)

responding treebank was 32.47, which implied a LAS loss up to 1.60 points in the official global ranking. We hypothesized that taking the model

Surprise language	Top 3 treebanks	Sample set *Monolingual*	Sample set *Multilingual*	Test set *Multi*
Buryat	Hindi German Korean	36.60 32.68 27.45	43.14	28.65_{5}
Kurmanji	Romanian Czech Slovenian	38.84 37.19 31.40	39.26	32.08_{16}
North Sami	Estonian Finnish Finnish-FTB	45.38 40.82 40.14	57.14	32.58_{14}
Upper Sorbian	Slovenian Slovak Bulgarian	65.22 64.78 61.09	70.65	52.50_{15}

Table 4: LAS on the surprise languages sample sets for: (1) top 3 best performing monolingual models for which there is an official training treebank and (2) a multilingual model trained on the first 2 000 sentences of each of such treebanks. For the multilingual models, the last column shows its performance on the test sets (subscripts indicate our ranking in that language)

trained on the largest treebank of the same language was the safest option to parse PUD texts, but in retrospective this clearly was not the optimal choice. Those four PUD treebanks were parsed with models trained on Universal Dependencies (UD) treebanks whose official name has a *suffix* (i.e. Spanish-Ancora, Finnish-FTB, Portuguese-BR and Russian-SyntagRus), which were larger than the unsuffixed UD treebank. However, we think such a poor performance surpasses what can be reasonably expected from an universal treebank written in the same language. From Table 5 it is reasonable to conclude that such suffixed treebanks parse more than poorly on cross-treebank settings, in comparison to the model trained on the unsuffixed treebank (rightmost column). We wonder if this can be an indicator of those treebanks sharing universal dependency types, but diverging in terms of syntactic structures, which caused the low LAS scores in those cases.

A possible contributing factor to this could be that the annotators of the parallel treebanks used guidelines from the unsuffixed treebanks, or automatic output trained on them, as a starting point from the annotation process. At the point of writing we cannot confirm whether this is the case, as documentation for the PUD treebanks is not yet publicly available.

PUD treebank	Trained on largest treebank (official)	LAS	Trained on uns. treebank (unofficial)	LAS
Arabic	Arabic	45.12_{11}	=	
Czech	Czech	80.13_{10}	=	
German	German	66.29_{19}	=	
English	English	78.79_{16}	=	
Spanish	Spanish-Ancora	$53.73_{30}\downarrow$	Spanish	78.90
Finnish	Finnish-FTB	$40.66_{28}\downarrow$	Finnish	80.70
French	French	73.15_{23}	=	
Hindi	Hindi	51.15_{13}	=	
Italian	Italian	83.84_{15}	=	
Japanese	Japanese	76.09_{18}	=	
Portuguese	Portuguese-BR	$54.75_{27}\downarrow$	Portuguese	72.84
Russian	Russian-SyntagRus	$44.69_{31}\downarrow$	Russian	70.00
Swedish	Swedish	69.60_{17}	=	
Turkish	Turkish	34.96_{4}	=	

Table 5: LAS/UAS performance on the PUD treebanks (test sets). The $\downarrow$ symbol indicates a drastic gap in performance with respect the average performance of BIST-COVINGTON. We show how parsing the PUD treebank with a model trained on the corresponding unsuffixed treebank clearly improves the LAS accuracy.

4 Discussion

BIST-COVINGTON worked very well on languages where official training/development sets were available, what the organizers named *big treebanks* (55 treebanks), category where we ranked 7th out of 33 systems, both for LAS and UAS metrics, in spite of not using any ensemble method and not performing custom tokenization, segmentation or tagging.

More in detail, we ranked in the top ten LAS for 35 languages, where 32 belong to the category of *big treebanks*: Arabic (10th), Bulgarian (6th), Buryat (5th), Czech-PUD (10th), Old Church Slavonic (4th), Greek (6th), Spanish (8th), Spanish-Ancora (7th), Estonian (7th), Basque (5th), Finnish (8th), Finnish-ftb (7th), Gothic (7th), Ancient_Greek (6th), Ancient_Greek-PROIEL (8th), Hebrew (9th), Hungarian (9th), Latin (7th), Latin-ITB (7th), Latin-PROIEL (7th), Latvian (7th), Dutch (7th), Dutch-lassysmall (6th), Norwegian-Bokmaal (8th), Norwegian-Nynorsk (7th), Polish (7th), Portuguese-BR (9th), Russian (8th), Russian-Syntagrus(9th), Slovak (6th), Slovenian (7th), Swedish-LinES (10th), Turkish (5th), Turkish-PUD (4th) and Ukrainian (9th).

We failed on a subset of the PUD treebanks. As previously explained, the main gap came from the Spanish, Russian, Portuguese and Finnish PUD treebanks. We analyzed those treebanks based on existing UD CoNLL treebanks. We parsed them with the model trained on the largest treebank that shared the language. It turned out that those PUD treebanks that were parsed with suffixed treebanks (e.g. Spanish-Ancora or Russian-SynTagRus) obtained a very low performance, something that did not happen when parsing them with the model trained on the corresponding unsuffixed treebank (e.g. Spanish or Russian). In cases where there was only one UD treebank sharing the language, our approach worked reasonably well, in spite of the simplistic strategy followed (e.g. Turkish-PUD or Czech-PUD).

We did not perform too well either on the set of *small treebanks* (French-ParTUT, Irish, Galician-TreeGal, Kazakh, Slovenian-SST, Uyghur and Ukrainian). This was somewhat expected for two reasons: (1) neural models that are fed with continuous vector representations are usually data-hungry and (2) the submitted model was only trained on our training split; we did not include the *ad-hoc* dev sets for those languages as a part of the final training data.

We believe that the cases where the parser did not work well were due to external causes (e.g. the chosen cross-treebank strategy), as shown in the case of the PUD treebanks. Unofficial results such as the ones in Table 5 show that this can be easily addressed to push BIST-COVINGTON to obtain competitive results in those treebanks too.

5 Hardware requirements and issues

Our models required DyNet (Neubig et al., 2017), which allocates memory when it is launched. We ran them on CPU. To train the models we used two servers with 128GB of RAM memory each. Estimating the required memory to allocate to train each model was a hard task for us. Dynet does not currently have a garbage collector,[9] so many models ran out of memory even before finishing their training, probably due to wrong memory estimations to complete this phase, and our lack of resources to allocate memory for many treebanks at a time. We observed that models such as Arabic with external word embeddings could take up to 64GB during the training phase.

The performance on the dev set of our trained models was close, but not equal, in our training machine and in TIRA. This might be caused by a serialization versioning issue: https://

[9] https://github.com/clab/dynet/issues/418

github.com/clab/dynet/issues/84.

To safely run a large trained model with external embeddings we recommend at least 32GB of RAM memory. We think a safe estimate to run any model without external embeddings would be something between 15 and 20GB.

The current version of BIST-COVINGTON is not very fast. Average speed (tokens/second) over all test treebanks was 18.27. The fastest models were Kazakh (66.36), Uyghur (54.11) and Czech-PUD (45.79) and the slowest ones Czech-CLTT (5.37), Latin-PROIEL (7.69) and Galician-TreeGal (8.19). To complete the testing phase of the shared task, BIST-COVINGTON took around 28 hours. These times correspond to those of the official evaluation on the TIRA virtual machine. Several factors influence these speeds. Firstly, RNN approaches tend to be slower than feedforward approaches (e.g., reported speeds for the original transition-based BIST-parser by Kiperwasser and Goldberg (2016) are an order of magnitude behind those of Chen and Manning (2014), although the latter is also much less accurate). Secondly, parsing UD data for different languages accurately requires using more linguistic information (e.g. feature embeddings), increasing the model size with respect to models evaluated on simpler settings like the English Penn Treebank. Finally, we are aware that Covington's algorithm may become slower when sentences are too long due to its quadratic worst-case complexity, an issue that is likely to happen due to the predicted segmentation (the organizers actually informed that some treebanks contained sentences of about 300 words).

6 Conclusion

This paper presented BIST-COVINGTON, a bidirectional LSTM implementation of the Covington (2001) algorithm for non-projective transition-based dependency parsing. Our model was evaluated on the end-to-end multilingual parsing with universal dependencies shared task proposed at CoNLL 2017. For segmentation and part-of-speech tagging our model relied on the official UDpipe baseline. The official results located us 7th out of 33 teams in the *big treebanks* category, in spite of not using any ensemble method.

As future work, there is room for improvement. Due to lack of resources to train the models and complete the task on time, we could not train all models using external word embeddings, which has been shown to produce a significant overall improvement. Jackniffing (Agić and Schluter, 2017) might be a simple way to improve the LAS scores. Finally, it would be interesting to implement the non-monotonic version of the Covington transition system, together with approximate dynamic oracles (Fernández-González and Gómez-Rodríguez, 2017), shown to improve accuracy over the regular Covington parser.

Acknowledgments

David Vilares is funded by an FPU Grant 13/01180. Carlos Gómez-Rodríguez has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150). Both authors have received funding from the TELEPARES-UDC project from MINECO.

References

Željko Agić and Natalie Schluter. 2017. How (not) to train a dependency parser: The curious case of jackknifing part-of-speech taggers. In *The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pages 166–170. http://dl.acm.org/citation.cfm?id=1596276.1596307.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, pages 740–750. http://www.aclweb.org/anthology/D14-1082.

Michael A Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*. Citeseer, pages 95–102.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, pages 340–345. https://arxiv.org/pdf/cmp-lg/9706003.pdf.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2017. A full non-monotonic transition system for unrestricted non-projective parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (in press)*. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*. pages 959–976. http://www.aclweb.org/anthology/C12-1059.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective parsing. *Volume 2: Short Papers* page 256. http://aclweb.org/anthology/P/P15/P15-2042.pdf.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 1492–1501. http://aclweb.org/anthology/P/P10/P10-1151.pdf.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 917–927. http://aclweb.org/anthology/D/D14/D14-1099.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* https://arxiv.org/pdf/1412.6980.pdf.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327. http://transacl.org/ojs/index.php/tacl/article/view/885.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, pages 673–682. http://aclweb.org/anthology/P/P11/P11-1068.pdf.

André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 34–44. http://aclweb.org/anthology/D/D10/D10-1004.pdf.

Antonio Valerio Miceli-Barone and Giuseppe Attardi. 2015. Non-projective dependency-based pre-reordering with recurrent neural network for machine translation. In *The 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference of the Asian Federation of Natural Language Processing*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. pages 3111–3119. https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* https://arxiv.org/abs/1701.03980.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553. http://dl.acm.org/citation.cfm?id=1479205.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*. Association for Computational Linguistics, pages 351–359. http://aclweb.org/anthology/P/P09/P09-1040.pdf.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017a. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, `http://hdl.handle.net/11234/1-1983`. http://hdl.handle.net/11234/1-1983.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017b. Universal dependencies 2.0 CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-2184.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portoro, Slovenia, pages 1659–1666. http://www.lrec-conf.org/proceedings/lrec2016/pdf/348_Paper.pdf.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*. volume 6, pages 2216–2219. http://www.lrec-conf.org/proceedings/lrec2006/pdf/162_pdf.pdf.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 99–106. http://aclweb.org/anthology/P/P05/P05-1013.pdf.

Martin Potthast, Tim Gollub, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. 2014. Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In Evangelos Kanoulas, Mihai Lupu, Paul Clough, Mark Sanderson, Mark Hall, Allan Hanbury, and Elaine Toms, editors, *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*. Springer, Berlin Heidelberg New York, pages 268–299. https://doi.org/10.1007/978-3-319-11382-1_22.

Mohammad Sadegh Rasooli and Joel Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *arXiv preprint arXiv:1503.06733* https://arxiv.org/pdf/1503.06733.pdf.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. volume 1631, page 1642.

Milan Straka, Jan Hajic, and Jana Straková. 2016. Udpipe: Trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. http://www.lrec-conf.org/proceedings/lrec2016/pdf/873_Paper.pdf.

David Vilares, Carlos Gómez-Rodrıguez, and Miguel A. Alonso. 2016. One model, two languages: training bilingual parsers with harmonized treebanks. In *The 54th Annual Meeting of the Association for Computational Linguistics*. pages 425–431. http://aclweb.org/anthology/P/P16/P16-2069.pdf.

David Vilares, Carlos Gómez-Rodríguez, and Miguel A Alonso. 2017. Universal, unsupervised (rule-based), uncovered sentiment analysis. *Knowledge-Based Systems* 118:45–55.

Tong Xiao, Jingbo Zhu, Chunliang Zhang, and Tongran Liu. 2016. Syntactic skeleton-based translation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*. pages 2856–2862. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11933.

Mo Yu, Matthew R Gormley, and Mark Dredze. 2015. Combining word embeddings and feature embeddings for fine-grained relation extraction. In *HLT-NAACL*. pages 1374–1379.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gökırmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadova, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 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*. Association for Computational Linguistics.