AntNLP at CoNLL 2018 Shared Task: A Graph-based Parser for Universal Dependency Parsing

Tao Ji, Yufang Liu, Yijun Wang, Yuanbin Wu, Man Lan
[1]Department of Computer Science and Technology
[2]East China Normal University
{taoji, yfliu, yjwang}.antnlp@gmail.com
{ybwu, mlan}@cs.ecnu.edu.cn

Abstract

We describe the graph-based dependency parser in our system (AntNLP) submitted to the *CoNLL 2018 UD Shared Task*. We use bidirectional lstm to get the word representation, then a bi-affine pointer networks to compute scores of candidate dependency edges and the MST algorithm to get the final dependency tree.

From the official testing results, our system gets 70.90 LAS F1 score (rank 9/26), 55.92 MLAS (10/26) and 60.91 BLEX (8/26).

1 Introduction

The focus of the *CoNLL 2018 UD Shared Task* is learning syntactic dependency parsers that can work over many typologically different languages, even low-resource languages for which there is little or no training data. The Universal Dependencies (Nivre et al., 2017a,b) treebank collection has 82 treebanks over 57 kinds of languages.

In this paper we describe our system (AntNLP) submitted to the *CoNLL 2018 UD Shared Task*. Our system is based on the deep biaffine neural dependency parser (Dozat and Manning, 2016). The system contains a BiLSTM feature extractor for getting context-aware word representation and two biaffine classifiers to predict the head token of each word and the label between a head and its dependent.

There are three main metrics for this task, LAS (labeled attachment score), MLAS (morphology-aware labeled attachment score) and BLEX (bi-lexical dependency score). From the official testing results, our system gets 70.90 LAS F1 score (rank 9/26), 55.92 MLAS (10/26) and 60.91 BLEX (8/26). In a word, Our system is ranked top 10 according to the three metrics described above.

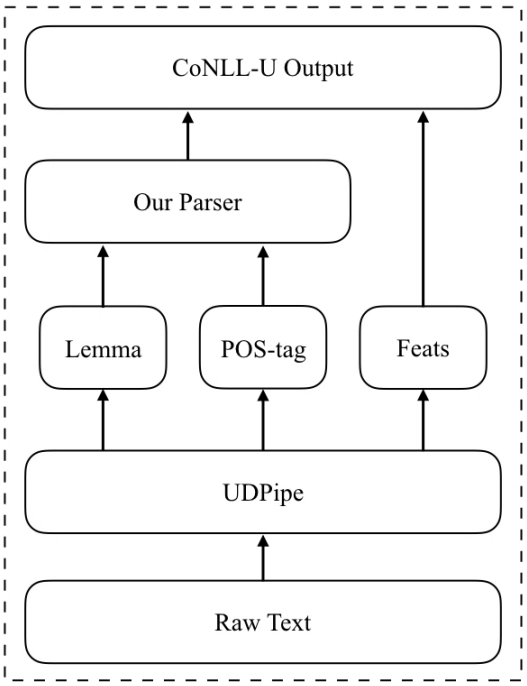

Figure 1: The structure of the entire system.

Additionally, in the categories of small treebanks, our system obtains the sixth place with a MLAS score of 63.73. Besides that, our system ranked tenth in the EPE 2018 campaign with a 55.71 F1 score.

The rest of this paper is organized as follows. Section 2 gives a brief description of our overall system, including the system framework and parser architecture. In Section 3, 4 we describe our monolingual model and multilingual model. In Section 5, we briefly list our experimental results.

2 System Overview

The *CoNLL 2018 UD Shared Task* aims to construct dependency trees based on raw texts, which means that the participants should not only build

Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 248–255
Brussels, Belgium, October 31 – November 1, 2018.
https://doi.org/10.18653/v1/K18-2025

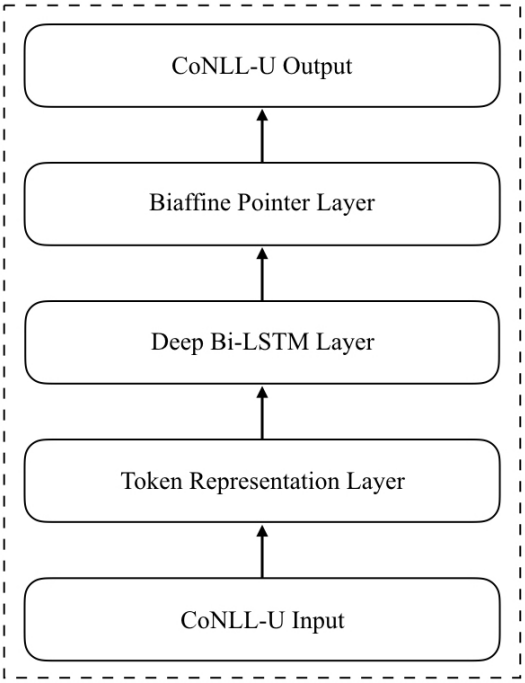

Figure 2: The architecture of our parser system.

the parsing model, but also preprocess systems of the sentence segmentation, tokenization, POS-tagging and morphological analysis. We use preprocessors from the official UDPipe tool in our submission. The structure of the entire system is shown in Figure 1. Our main focus is on building a graph-based parser.

We implement a graph-based bi-affine parser following Dozat and Manning (2016). The parser architecture is shown in Figure 2, which consists of the following components:

- **Token representation**, which produces the context independent representation of each token in the sentence.
- **Deep Bi-LSTM Encoder**, which produces the context-aware representation of each token in the sentence based on context.
- **Bi-affine Pointer Networks**, which assign probabilities to all possible candidate edges.

We describe the three sub-modules in the following sections in detail.

2.1 Token representation

Recent studies on dependency parsing show that densely embedded word representation could help to improve empirical parser performance. For example: Chen and Manning (2014) map words and POS tags to a d-dimensional vector space. Dozat and Manning (2016) use the pre-trained GloVe embeddings as an extra representation of the word. Ma et al. (2018) use Convolutional Neural Networks (CNNs) to encode character-level information of a word.

The token representation module of our parser also uses dense embedding representations. Details on token embeddings are given in the following.

- **Word $\mathbf{e_i^w}$**: The word embedding is randomly initialized from the normal distribution $\mathcal{N}(0,1)$ ($e_i^w \in \mathbb{R}^{100}$).
- **Lemma $\mathbf{e_i^l}$**: The lemma embedding is randomly initialized from the normal distribution $\mathcal{N}(0,1)$ ($e_i^l \in \mathbb{R}^{100}$).
- **Pre-trained Word $\mathbf{e_i^{pw}}$**: The FastText pre-trained word embedding ($e_i^{pw} \in \mathbb{R}^{300}$). We will not update e_i^{pw} during the training process.
- **UPOS $\mathbf{e_i^u}$**: The UPOS-tag embedding is randomly initialized from the normal distribution $\mathcal{N}(0,1)$ ($e_i^u \in \mathbb{R}^{100}$).
- **XPOS $\mathbf{e_i^x}$**: The XPOS-tag embedding is randomly initialized from the normal distribution $\mathcal{N}(0,1)$ ($e_i^x \in \mathbb{R}^{100}$).
- **Char $\mathbf{e_i^c}$**: The character-level embedding is obtained by the character-level CNNs ($e_i^c \in \mathbb{R}^{64}$).

Our parser uses two kinds of token representations, one is a lexicalized representation of the monolingual model, another one is the delexicalized representation of the multilingual model.

The lexicalized representation x_i^l of token w_i is defined as:

$$x_i^l = [\underbrace{e_i^w + e_i^l}_{word}; \underbrace{e_i^u + e_i^x}_{POS}; e_i^{pw}; e_i^c] \quad (1)$$

and the delexicalized representation x_i^d of token w_i is defined as:

$$x_i^d = [e_i^u; e_i^x; e_i^c] \quad (2)$$

In the following sections, we uses x_i to represent x_i^l or x_i^d when the context is clear.

2.2 Deep Bi-LSTM Encoder

Generally, the token embeddings defined above are context independent, which means that the sentence-level information is ignored. In recent years, some work shows that the deep BiLSTM can effectively capture the contextual information of words (Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016; Ma et al., 2018).

In order to encode context features, we use a 3-layer sentence level BiLSTM on top of $x_{1:n}$:

$$\begin{aligned}\vec{h}_t &= \text{LSTM}(\vec{h}_{t-1}, x_i, \vec{\theta}) \\ \overleftarrow{h}_t &= \text{LSTM}(\overleftarrow{h}_{t+1}, x_i, \overleftarrow{\theta}) \\ v_i &= \vec{h}_i \circ \overleftarrow{h}_i\end{aligned}$$

$\vec{\theta}$ are the model parameters of the forward hidden sequence $\vec{h}$. $\overleftarrow{\theta}$ are the model parameters of the backward hidden sequence $\overleftarrow{h}$. The vector v_i is our final vector representation of ith token in s, which takes into account both the entire history $\vec{h}_i$ and the entire future $\overleftarrow{h}_i$ by concatenating $\vec{h}_i$ and $\overleftarrow{h}_i$.

2.3 Biaffine Pointer Networks

How to determine the probability of each dependency edge is an important part of the graph-based parser. The work of Dozat and Manning (2016) shows that the biaffine pointer (attention) networks can calculate the probability of each dependency edge well. Here we used a similar biaffine pointer network structure.

In order to better represent the direction of the dependency edges, we use multi-layer perceptron (MLP) networks to learn each word as the representation of head and dependent words, rather than simply exchanging feature vectors. And we also separate the predictions of dependent edges and their labels. First, for each v_i, we use two MLPs to define two pointers h_i^{arc} and s_i^{arc}, which is the representation of v_i with respect to whether it is seen as a head or a modifier of an candidate edge.

$$\begin{aligned}h_i^{arc} &= \text{MLP}_{head}^{(arc)}(v_i) \\ s_i^{arc} &= \text{MLP}_{dep}^{(arc)}(v_i)\end{aligned}$$

Similarly, we use h_i^{rel} and s^{rel} to describe x_i when determine the relation label of a candidate edge.

$$\begin{aligned}h_i^{rel} &= \text{MLP}_{head}^{(rel)}(v_i) \\ s_i^{rel} &= \text{MLP}_{dep}^{(rel)}(v_i)\end{aligned}$$

We first use the arc-biaffine pointer networks to predict the probability of a dependency edge between any two words. For any two words w_i and w_j in a sentence, the probability $p_{i\to j}^{(arc)}$ that they form a dependency edge $w_i \to w_j$ is as follows:

$$\begin{aligned}a_{i\to j}^{(arc)} &= h_i^{arc} \cdot \text{W}^{(arc)} \cdot s_j^{arc} + u^{(arc)} \cdot s_j^{arc} \\ p_{i\to j}^{(arc)} &= \text{softmax}(a_{i\to *}^{(arc)})[j]\end{aligned}$$

where $\theta^{arc} = \{\text{W}^{(arc)}, u^{(arc)}\}$ are the model parameters, $a_{i\to j}^{(arc)}$ is the computed score of the dependency edge. $a_{i\to *}^{(arc)}$ is a vector, and the k^{th} dimension is the score of the dependency edge $a_{i\to k}^{(arc)}$. $p_{i\to j}^{(arc)}$ is the j^{th} dimension of normalization of the vector $a_{i\to *}^{(arc)}$, meaning the probability of dependency edge $w_i \to w_j$.

We obtain a dependency tree representation T of a complete graph $p_{*\to *}^{(arc)}$ using the maximum spanning tree (MST) algorithm. The probability $p_{i\xrightarrow{r} j}^{(rel)}$ of the relation r of each dependency edge $w_i \to w_j \in T$ is then computed. The definition of $p_{i\xrightarrow{r} j}^{(rel)}$ is as follows:

$$\begin{aligned}a_{i\xrightarrow{*} j}^{(rel)} &= h_i^{rel} \cdot \mathbf{W}^{(rel)} \cdot s_j^{rel} \\ &\quad + \text{V}^{(rel)} \cdot h_i^{rel} + \text{U}^{(rel)} \cdot s_j^{rel} \\ p_{i\xrightarrow{r} j}^{(rel)} &= \text{softmax}(a_{i\xrightarrow{*} j}^{(rel)})[r]\end{aligned}$$

where $\theta^{rel} = \{\mathbf{W}^{(rel)}, \text{V}^{(rel)}, \text{U}^{(rel)}\}$ are the model parameters, $\mathbf{W}^{(rel)}$ is a 3-dimensional tensor. $a_{i\xrightarrow{*} j}^{(rel)}$ is a vector, and the k^{th} dimension is the score of the dependency edge $w_i \xrightarrow{k} w_j$.

2.4 Training Details

We train our model by minimizing the negative log likelihood of the gold standard $(w_i \xrightarrow{r(w_i)} w_{h(w_i)})$ arcs in all training sentences:

$$\begin{aligned}J^{(arc)} &= -\frac{1}{|\tau|}\sum_{S\in\tau}\sum_{i=1}^{N_S} \log p_{i\to h(w_i)}^{(arc)} \\ J^{(rel)} &= -\frac{1}{|\tau|}\sum_{S\in\tau}\sum_{i=1}^{N_S} \log p_{i\xrightarrow{r(w_i)} h(w_i)}^{(rel)} \\ J &= J^{(arc)} + J^{(rel)}\end{aligned}$$

where τ is the training set, $h(w_i)$ and $r(w_i)$ is w_i's gold standard head and relation within sentence S, and N_s is the number of words in S.

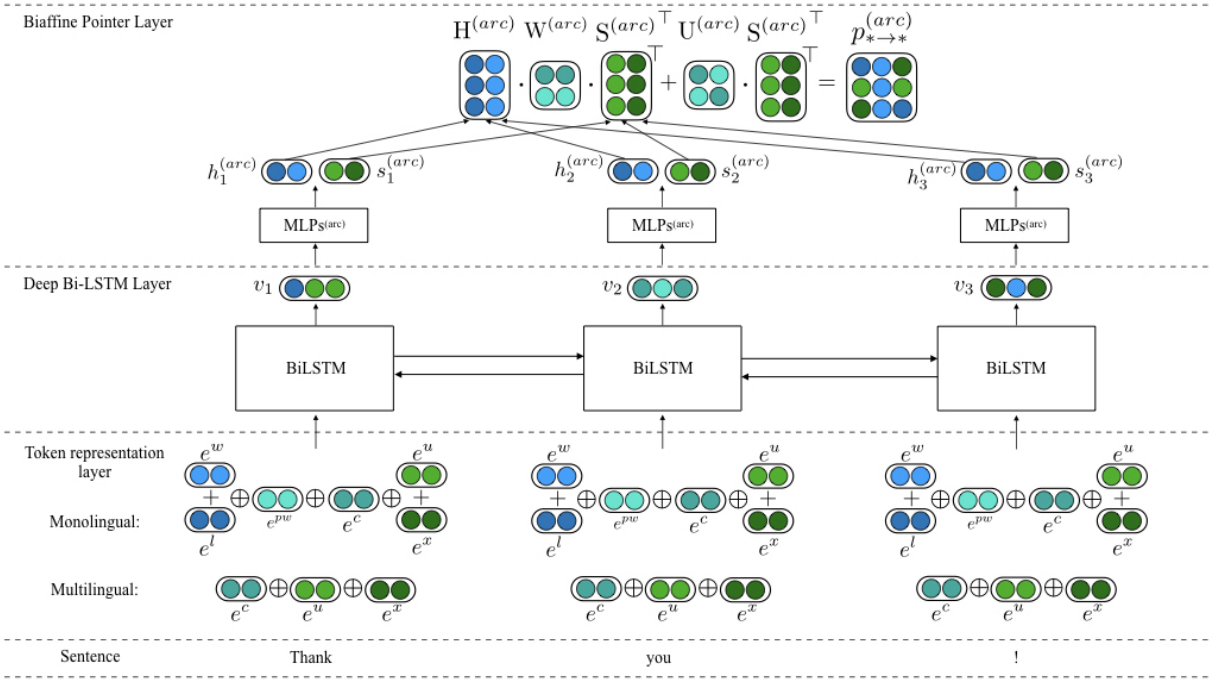

Figure 3: The architecture of our parser system.

3 Monolingual Model

There are 82 treebanks in the *CoNLL 2018 UD Shared Task*, including 61 big treebanks, 5 PUD treebanks (additional parallel test sets), 7 small treebanks and 9 low-resource language treebanks. There are several languages in which there are many treebanks, such as *en_ewt*, *en_gum* and *en_lines* in English. We combine training sets and development sets for multiple treebanks of the same language. And then just train a model for the language and make predictions on its different treebanks.

For each language of UD version 2.2 sets (Nivre et al., 2018; Zeman et al., 2018) with both a training set and a development set, we train a parser using lexicalized token representation and only using its monolingual training set (no cross-lingual features)[1] . The architecture of the monolingual model is shown in Figure 3.

4 Multilingual Model

For 7 languages without a development set, we divide them into two classes based on the size of their training set, which can be fine-tuned (*ga*, *sme*) and can not be fine-tuned (*bxr*, *hsb*, *hy*, *kk*, *kmr*).

For each language of UD version 2.2 sets (Nivre et al., 2018; Zeman et al., 2018) with both a training set and a development set, we train a parser using delexicalized token representation as a cross-language model. The architecture of the multilingual model is shown in Figure 3. The training set of these 5 languages are then used as a development set to validate the performance of each cross-language model (see Table 1). We select the best performance model as a cross-language model for the corresponding language. For both *ga* and *sme*, we manually divide the development set from the training set and fine-tune the cross-language model. Prediction and fine-tuning results are shown in the Table 2.

5 Experimental Results

We trained our system based on a Nvidia GeForce GTX Titan X. We used the official TIRA (Potthast et al., 2014) to evaluate the system.

We used Dynet neural network library to build our system (Neubig et al., 2017). The hyper-parameters of the final system used for all the reported experiments are detailed in Table 5.

5.1 Overall Results

The main official evaluation results are given in Table 4. And the Table 6 shows the per-treebank LAS F1 results. Our system achieved 70.90 F1 (LAS) on the overall 82 tree banks, ranked 9^{th} out

[1] In total, we trained 46 monolingual models.

Language	#Train	Cross language	LAS
Buryat (bxr)	19	Uyghur (ug)	27.45
Upper_Sorbian (hsb)	23	Croatian (hr)	39.35
Armenian (hy)	50	Latvian (lv)	29.35
Kazakh (kk)	9	Turkish (tr)	23.44
Kurmanji (kmr)	19	Persian (fa)	26.03

Table 1: Corpus listed above are languages that don't have development set and the training set size is too small to be fine-tuned. "#Train" means the number of sentences. "Cross Language" means the language with the highest LAS score for corresponding origin language in our delexicalized cross-language model.

Corpus	#Total	#Train	#Dev	Cross language	LAS	Fine-tune
Irish (ga)	566	476	90	Hebrew (he)	36.13	68.49
North_Sami (sme)	2464	1948	516	Swedish (sv)	36.13	63.00

Table 2: Corpus listed above are languages that don't have development set. Because the training set size is much bigger, we decide to divide the training set into two parts, one for training set and the other for development set.

Origin language	language family	Cross Language	Language family
Buryat (bxr)	Mongolic	Uyghur (ug)	Turkic_Southeastern
Upper_Sorbian (hsb)	IE_Slavic	Croatian (hr)	IE_Slavic
Armenian (hy)	IE_Armenian	Latvian (lv)	IE_Baltic
Kazakh (kk)	Turkic_Northwestern	Turkish (tr)	Turkic_Southwestern
Kurmanji (kmr)	IE_Iranian	Persian (fa)	IE_Iranian
Irish (ga)	IE_Celtic	Hebrew (he)	Afro-Asiatic_Semitic
North_Sami (sme)	Uralic_Sami	Swedish (sv)	IE_Germanic

Table 3: language families and genera for origin language and cross language (IE = Indo-European).[2]

Corpus	FLAS	Baseline	Rank	MLAS	Baseline	Rank	BLEX	Baseline	Rank
All treebanks(82)	70.90	65.80	9	55.92	52.42	10	60.91	55.80	8
Big treebanks(61)	79.61	74.14	12	65.43	61.27	11	70.34	64.67	9
PUD treebanks(5)	68.87	66.63	11	53.47	51.75	10	57.71	54.87	8
Small treebanks(7)	63.73	55.01	6	42.24	38.80	7	48.31	41.06	6
Low resource(9)	18.59	17.17	10	3.43	2.82	9	8.61	7.63	8

Table 4: Official experiment results with rank. (number): number of corpora. FLAS means F1 score of LAS.

word/lemma dropout	0.33
upos/xpos tag dropout	0.33
char-CNN dropout	0.33
BiLSTM layers	3
BiLSTM hidden layer dimensions	400
Hidden units in $MLP^{(arc)}$	500
Hidden units in $MLP^{(rel)}$	100
Learning rate	0.002
Optimization algorithm	Adam

Table 5: Hyper-parameter values used in shared task.

of 26 teams. Compared to the baseline obtained with UDPipe1.2 (Straka et al., 2016), our system gained 5.10 LAS improvement on average. Our system shows better results on 7 small treebanks. Performance improvement are more obvious when considering only small treebanks(for example, our system ranked fourth best on *ru_taiga* and *sl_sst*). Besides that, our system ranked tenth in the EPE[3] 2018 campaign with a 55.71 F1 score.

5.2 Discussion on Multilingual Model

As described in section 4, we trained 46 cross-language models and selected the corresponding cross-language model for 7 languages that did not have a development set. Generally, cross-language models are trained in the language of the same family. However, apart from grammatical similarity, the language family division also considers the linguistic history, geographical location and other factors. We want to select a language's cross-language model to consider only grammatical similarity. So we use a cross-language model to predict the results in this language as a basis for selection.

In table 3, the experimental results show that the Cross-language model with the best performance in *hsb*, *hy*, *kk*, and *kmr* languages comes from the same language family, while the Cross-language model with the best performance in *bxr*, *ga*, and *sme* is not from the same language family. Therefore, constructing a cross-language model according to the language of the same family is only applicable to some languages, not all of them. We've only chosen the best performing cross-language model at the moment. In the future, we will try to select the top-k cross-language model.

[2]Information from http://universaldependencies.org.
[3]http://epe.nlpl.eu

6 Conclusions

In this paper, we present a graph-based dependency parsing system for the *CoNLL 2018 UD Shared Task*, which composed of a BiLSTMs feature extractor and a bi-affine pointer networks. The results suggests that a deep BiLSTM extractor and a bi-affine pointer networks is a way to achieve competitive parsing performances. We will continue to improve our system in our future work.

Acknowledgments

References

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR* abs/1611.01734. http://arxiv.org/abs/1611.01734.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. pages 334–343. http://aclweb.org/anthology/P/P15/P15-1033.pdf.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL* 4:313–327. https://transacl.org/ojs/index.php/tacl/article/view/885.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard H. Hovy. 2018. Stack-pointer networks for dependency parsing. *CoRR* abs/1805.01087. http://arxiv.org/abs/1805.01087.

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* .

Corpus	AntNLP	Rank	Best	Baseline	Corpus	AntNLP	Rank	Best	Baseline
af_afr*	82.63	11	85.47	77.88	hy_arm*	25.09	10	37.01	21.79
ar_pad*	70.75	12	77.06	66.41	id_gsd	76.86	16	80.05	74.37
bg_btb	87.24	13	91.22	84.91	it_isd*	89.14	12	92.00	86.26
br_keb	10.06	16	38.64	10.25	it_pos*	72.30	9	79.39	66.81
bxr_bdt	19.53	1	1	12.61	ja_gsd	72.82	17	83.11	72.32
ca_anc*	89.28	13	91.61	85.61	ja_mod*	12.94	22	28.33	22.71
cs_cac	90.47	6	91.61	83.72	kk_ktb	19.26	18	31.93	24.21
cs_fic*	90.14	7	92.02	82.49	kmr_mg	23.20	16	30.41	23.92
cs_pdt	89.41	9	91.68	83.94	ko_gsd	80.15	12	85.14	61.40
cs_pud	84.76	6	86.13	80.08	ko_kai*	85.01	11	86.91	70.25
cu_pro*	68.23	13	75.73	65.46	la_itt*	83.14	12	87.08	75.95
da_ddt	80.56	10	86.28	75.43	la_per*	60.99	5	72.63	47.61
de_gsd	76.88	10	80.36	70.85	la_pro*	66.24	12	73.61	59.66
el_gdt	85.76	12	89.65	82.11	lv_lvt*	75.56	12	83.97	69.43
en_ewt	80.74	12	84.57	77.56	nl_alp*	84.69	11	89.56	77.60
en_gum	79.70	11	85.05	74.20	nl_las*	82.04	8	86.84	74.56
en_lin*	79.25	5	81.97	73.10	no_bok*	89.19	7	91.23	83.47
en_pud	84.60	9	87.89	79.56	no_nyn*	88.26	9	90.99	82.13
es_anc*	88.84	11	90.93	84.43	no_nyn*	66.26	4	70.34	48.95
et_edt	81.37	11	85.35	75.02	pcm_nsc	18.30	6	30.07	12.18
eu_bdt	79.01	11	84.22	70.13	pl_lfg	91.16	13	94.86	87.53
fa_ser*	83.98	11	88.11	79.10	pl_sz	85.03	14	92.23	81.90
fi_ftb	83.72	11	88.53	75.64	pt_bos*	86.71	8	87.81	82.07
fi_pud	85.50	10	90.23	80.15	ro_rrt	84.92	8	86.87	80.27
fi_tdt	83.18	11	88.73	76.45	ru_syn*	90.20	10	92.48	84.59
fo_oft	20.13	21	49.43	25.19	ru_tai*	68.99	4	74.24	55.51
fr_gsd	84.84	10	86.89	81.05	sk_snk	81.14	12	88.85	75.41
fr_seq*	85.32	10	89.89	81.12	sl_ssj	83.26	12	91.47	77.33
fr_spo*	70.96	8	75.78	65.56	sl_sst	56.30	4	61.39	46.95
fro_src*	83.13	12	87.12	79.27	sme_gie*	57.15	13	69.87	56.98
ga_idt	64.38	11	70.88	62.93	sr_set	85.77	10	88.66	82.07
gl_ctg	81.12	8	82.76	76.10	sv_lin*	80.01	10	84.08	74.06
gl_tre*	72.03	8	74.25	66.16	sv_pud	76.54	10	80.35	70.63
got_pro*	62.97	14	69.55	62.16	sv_tal*	83.41	12	88.63	77.91
grc_per*	70.76	9	79.39	57.75	th_pud	0.36	18	13.70	0.70
grc_pro*	73.82	9	79.25	67.57	tr_ims*	59.68	12	66.44	54.04
he_htb	61.43	12	76.09	57.86	ug_udt	61.42	10	67.05	56.26
hi_hdt*	90.44	11	92.41	87.15	uk_iu	79.91	12	88.43	74.91
hr_set	83.48	12	87.36	78.61	ur_udt*	79.85	14	83.39	77.29
hsb_ufa*	31.36	6	46.42	23.64	vi_vtb	42.65	11	55.22	39.63
hu_sze*	73.19	12	82.66	66.76	zh_gsd	62.83	13	76.77	57.91

Table 6: Official experiment results on each treebank. The results in table are F1(LAS). *: some corpus' name too long to display completely, using * to indicate omission.

Joakim Nivre 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 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, Prague, http://hdl.handle.net/11234/1-2184. http://hdl.handle.net/11234/1-2184.

Joakim Nivre et al. 2018. Universal Dependencies 2.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/1-1983xxx. http://hdl.handle.net/11234/1-1983xxx.

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.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portoro, Slovenia.

Dan Zeman et al. 2018. Universal Dependencies 2.2 CoNLL 2018 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/1-2184. http://hdl.handle.net/11234/1-2184.