The parse is darc and full of errors: Universal dependency parsing with transition-based and graph-based algorithms

Kuan Yu **Pavel Sofroniev** **Erik Schill** **Erhard Hinrichs**
Department of Linguistics, University of Tübingen
{kuan.yu,pavel.sofroniev,erik.schill}@student.uni-tuebingen.de
erhard.hinrichs@uni-tuebingen.de

Abstract

We developed two simple systems for dependency parsing: *darc*, a transition-based parser, and *mstnn*, a graph-based parser. We tested our systems in the *CoNLL 2017 UD Shared Task*, with darc being the official system. Darc ranked 12th among 33 systems, just above the baseline. Mstnn had no official ranking, but its main score was above the 27th. In this paper, we describe our two systems, examine their strengths and weaknesses, and discuss the lessons we learned.

1 Introduction

Universal Dependencies (UD) (Nivre et al., 2016) is a cross-linguistically consistent annotation scheme for dependency-based treebanks. UD version 2.0 (UD2) (Nivre et al., 2017b,a) provided the datasets for the *CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* (Zeman et al., 2017). In the shared task participating systems were evaluated through the *TIRA* platform (Potthast et al., 2014). The main evaluation metric was the labeled attachment F_1-score (LAS). 33 systems completed the official evaluation, including the baseline *UDPipe* (Straka et al., 2016).

We submitted a primary system *darc* and a secondary system *mstnn*, with the primary system partaking in the official evaluation. Both are open sourced under the *MIT* license.[1] The two systems differ only in the parsing algorithm. Darc is equipped with a *transition-based non-projective/projective* parser. Mstnn is equipped with a *graph-based non-projective unlabeled* parser and a standalone labeler. Both systems utilize a *neural network classifier* with similar input features.

In this paper, we start with a description of our treatments for different datasets in the shared task, and then the separate descriptions of our two parsers, followed by an analysis of the results.

2 Treatments for datasets

We were tasked with producing parsed outputs for 81 test-sets, either from raw texts or from segmented and tagged inputs produced by the baseline system.

The outputs were required to conform to the *CoNLL-U* format.[2] In this format, each node in a dependency-graph has ten fields named *ID*, *FORM*, *LEMMA*, *UPOSTAG*, *XPOSTAG*, *FEATS*, *HEAD*, *DEPREL*, *DEPS*, and *MISC*, where ID, HEAD, and DEPREL defines an edge. Segmentation establishes the graph/sentence boundaries while filling in ID, FORM, and MISC. Tagging fills in LEMMA, UPOSTAG, XPOSTAG, and FEATS.

63 test-sets have corresponding treebanks in UD2. These treebanks were the only training resources we used.

2.1 Big treebanks

The majority of the treebanks in UD2 (55/63) consist of train-sets and dev-sets. These are the big treebanks.

For segmentation and tagging, we trained UDPipe models on the train-sets and used the dev-sets for parameter tuning.[3] The only hyperparameter tuned by the dev-sets was the number of training iterations. All the other hyperparameters were

[1] https://github.com/CoNLL-UD-2017/darc

[2] http://universaldependencies.org/format.html

[3] Danish, Finnish-FTB, and Slovenian-SST treebanks are missing the `SpaceAfter=No` information in MISC. For training the tokenizers, this information was added.

simply taken from the baseline models (Straka, 2017), because of our limited computing power.

The gold-standard train-sets were re-tagged by our UDPipe models to produce training data for our parsers.

2.2 Small treebanks

The remainder of the treebanks (8/63) consist only of train-sets. These are the small treebanks.

Here we consulted the approach of the baseline system, which split the train-sets into three parts: train, tune, and dev. For our UDPipe models, both the train-sets and tune-sets were used for training, and the dev-sets for tuning. For our parser models, the entirety of treebanks was used for training, and all hyperparameters took the default values.

2.3 Parallel test-sets

The 14 parallel test-sets have no corresponding treebanks, but the corresponding languages exist. For these we used the preprocessed inputs from the baseline system, and picked our parser models according to the languages. If multiple treebanks exist for the same language, we took the model trained on the first one.

2.4 Surprise languages

4 test-sets have no corresponding languages, though small samples of gold-standard data were released as part of the shared task. Again, we used the preprocessed inputs from the baseline system.

For each sample we applied our existing parser models to pick the best treebank for training a delexicalized model. These delexicalized models rely mostly on UPOSTAG, but may utilize FEATS as well. This setting, along with the other hyperparameters, were tuned by the sample data.[4]

3 Primary system: darc

Our primary system employs a transition-based parser.

We adapted our parser from Chen and Manning (2014), who used a neural network classifier in a transition-based parsing algorithm known as the *arc-standard* system (Nivre, 2008).

The neural network classifier requires little feature engineering, and therefore is easily adaptable for different languages, making it ideal for UD parsing. However, the arc-standard system is only applicable to projective parsing, while over half of the treebanks in UD2 have more than 10% non-projective sentences in the train-sets. For this reason, we adopted a non-projective variant by adding a *swap* action to the transition system (Nivre, 2009; Nivre et al., 2009). We chose either the projective or the non-projective algorithm based on how they performed for each treebank. In the end, we used the non-projective one for all but three treebanks.[5]

3.1 The transition algorithm

The algorithm produces a directed acyclic graph from a sequence of nodes $[w_0, w_1, \ldots, w_n]$, which are the syntactic tokens of a sentence, where w_0 is a pseudo root node. The transition system consists of several transition actions defined over *configurations*. Each configuration is a triple consisting of a stack σ, a buffer β, and a set of edges A. From the initial configuration $c_0\colon (\sigma\colon [w_0], \beta\colon [w_1, w_2, \ldots, w_n], A\colon \{\})$, a series of transition actions are taken to produce a terminal configuration $c_m\colon (\sigma\colon [w_0], \beta\colon [\,], A)$.

Possible transition actions are listed below. A different action is defined for each l in the set of dependency labels.

shift

$(\sigma, [w_i|\beta], A) \mapsto ([w_i|\sigma], \beta, A)$

left_arc$_l$

$([w_j, w_i|\sigma], \beta, A) \mapsto ([w_j|\sigma], \beta, A \cup \{(w_j, l, w_i)\})$

right_arc$_l$

$([w_j, w_i|\sigma], \beta, A) \mapsto ([w_i|\sigma], \beta, A \cup \{(w_i, l, w_j)\})$
where $0 \neq i$

swap

$([w_j, w_i|\sigma], \beta, A) \mapsto ([w_j|\sigma], [w_i|\beta], A)$
where $0 < i < j$

3.2 Input features

As input features, we use the set of 18 graph nodes from Chen and Manning (2014):

- The top three words on the stack & buffer
- The first & second leftmost & rightmost children of the top two words on the stack

[4] In the end, we used the Polish treebank for Buryat and Kurmanji, Finnish for North Sami, and Slovenian for Upper Sorbian. The FEATS field was used in the models for Buryat and North Sami.

[5] Persian, Spanish, and Vietnamese.

- The leftmost-of-leftmost & rightmost-of-rightmost children of the top two words on the stack

For each node we take its FORM, LEMMA, UPOSTAG, FEATS, and DEPREL fields. Each field is represented through an embedding into the real vector space. However, some treebanks have no informative LEMMA. For these treebanks we omit the LEMMA embedding, and double the dimension of the FORM embedding.[6]

All embeddings are trainable, except for the FEATS embedding. Each FEATS is represented by a vector of binary values, indicating the presence or absence of any attribute-value pairs in the morphological vocabulary of its affiliated treebank.

In any transition configuration, some nodes may be missing, for which special dummy members are in all embeddings. Special members are also appointed for the root node. For FORM and LEMMA, all hapaxes are replaced with a single arbitrary symbol.

Table 1 lists the number of rows (min, max & avg) and columns (dim) in the embedding matrices.

field	min	max	avg	dim
FORM	70	58562	9070	32
LEMMA	89	29972	6321	32
UPOSTAG	13	19	18	12
DEPREL	21	55	37	16

Table 1: Shapes of embedding matrices

The amount of parameters in the embedding matrices for FORM and LEMMA are substantial. Initializing these parameters with pretrained embeddings has been shown to be beneficial (Chen and Manning, 2014). To produce embeddings which are more suitable for capturing syntactic information, we used the tool developed by Ling et al. (2015).[7]

The embeddings for UPOSTAG and DEPREL are randomly initialized from the $uniform(-0.5, 0.5)$ distribution.

[6] Korean, Portuguese-BR, English-LinES, Indonesian, Swedish-LinES, and Uyghur.

[7] https://github.com/wlin12/wang2vec/ with options {-type 3 -hs 0 -min-count 2 -window 7 -sample 0.1 -negative 7 -iter 20}; Though in fact [-min-count 2] had no effect, as we had all hapaxes replaced by an obscure symbol.

3.3 Feedforward neural network

Our neural network classifier is implemented in *Keras* (Chollet et al., 2015) with the *TensorFlow* backend (Abadi et al., 2015).

The inputs are first transformed by a hidden layer with 256 rectified linear units (*ReLU*), then by a second, similar hidden layer, and finally by a *softmax* layer with as many units as the number of transition actions. The softmax output assigns a probability prediction for each action.

The weights for all layers are initialized in a random uniform distribution following He et al. (2015). The ReLU layers have their biases initialized to ones, in order to alleviate the dying ReLU problem. The network is trained through backpropagation by the *AdaMax* (Kingma and Ba, 2014) algorithm.

In our experiments, we found it helpful to apply *dropout* (Srivastava et al., 2014) to both the trainable embedding layers and the hidden layers. For our network, 25% dropout rate seems to be the optimal. The 50% dropout rate suggested by Srivastava et al. (2014) requires extending the sizes of these layers, which would result in a polynomial amount of increase in the number of parameters. Even though we did find a slight improvement in accuracy with a larger network and a higher dropout rate, we rejected extending the network because of our limited computing power.

For regularization, we apply the *unit-norm* constraint to the trainable embeddings, which ensures that each column of the embedding matrices is a unit vector. We found this helpful for stabilizing the accuracy in later iterations and achieving higher scores. We also experimented with the *max-norm* constraint, which only forces the norms of the column vectors to be no greater than the max-norm; We found that it can be better than the unit-norm constraint, but only for certain optimal max-norm values, which were different for every dataset.

3.4 Training and parsing

These hyperparameters are tuned during training, with their default values marked bold:

- Algorithm: projective, **non-projective**
- Batch-size: 16, **32**, 64
- Iterations: maximally **16**

Our parser is greedy during parsing. From any configuration, only the action with the highest probability prediction is taken to advance into the next configuration. In case the action suggested by the classifier is illegal, the next best action is taken.

The transition algorithm does not prevent multiple nodes from being attached to the pseudo root node. However, this is not allowed in the UD treebanks. When this occurs, we keep the first attachment, and attach the other nodes to that node with the *parataxis* label.

Apart from the regular syntactic nodes, the CoNLL-U format allows for *empty-words* and *multi-words*. We completely ignore the empty-words. We keep track of the multi-words, but ignore them during parsing.

The evaluation is only concerned with the UD labels, and not the language-specific subtypes. For example, *acl:relcl* is considered to be the same as *acl*. We experimented with removing the language specific information before parsing, and we found it to be helpful in some cases, but harmful in others. Either way, the differences are negligible.

3.5 A comparison with Parsito

Our parser is very similar to the *Parsito* parser (Straka et al., 2015) incorporated in UDPipe, which is also a transition-based parser with a feedforward neural network classifier.

The primary difference is in the training. Our parser uses only a *static oracle*, while Parsito supports a *dynamic oracle*, and may additionally utilize a *search-based oracle*.

The static oracle produces transition sequences which must lead to the gold-standard parse trees. A classifier trained only on the gold-standard transition sequences is not robust against its own errors. When an error is made, the parser arrives in a configuration which it has never seen before. To help the classifier make the best decision possible in any configuration, the dynamic oracle (Goldberg and Nivre, 2012) explores erroneous transitions suggested by the classifier itself. Parsito's search-based oracle applies the *SEARN* algorithm (Daumé et al., 2009) to mitigate this problem.

Moreover, in addition to the projective and non-projective transition systems, Parsito supports *link2* (Attardi, 2006), a partially non-projective transition algorithm, which was used for more than one-third of the baseline models.

Despite the limitations of our parser in comparison with UDPipe's Parsito, it achieved comparable results in the shared task.

4 Secondary system: mstnn

In our secondary system, a graph-based non-projective unlabeled parser and a labeler are used.

4.1 Unlabeled parsing

We adapted the *MSTParser* (McDonald et al., 2005) with a neural network classifier. Starting with a fully connected directed graph, the classifier scores the edges between every two nodes, and then the *Chu-Liu-Edmonds' algorithm* (Chu and Liu, 1965; Edmonds, 1967) is applied to find the *maximum spanning arborescence*. The algorithm was implemented using *NetworkX* (Hagberg et al., 2008).

The neural network classifier accepts the following inputs:

- The distance between the two nodes (the arithmetic difference of their ID)
- FORM & LEMMA of the two nodes
- UPOSTAG of the two nodes and their left & right & left-of-left & right-of-right neighbors
- FEATS of the two nodes and their left & right neighbors

All features are constructed the same as in the primary system, except for the added distance feature.[8] The structure of the neural network is also the same, except that the output layer consists of a single *sigmoid* unit. The probability prediction of the sigmoid unit is taken as the score associated with the dependency arc under consideration.

4.2 Labeling

For labeling the edges, we implemented a linear *support-vector classifier* (Cortes and Vapnik, 1995) using *LIBLINEAR* (Fan et al., 2008) through *scikit-learn* (Pedregosa et al., 2011).[9]

Input features to the classifier are the FORM, LEMMA, UPOSTAG, and FEATS fields of the two nodes, plus the UPOSTAG of their left & right neighbors. The FEATS field is represented in the

[8] Also as in the primary system, some features were omitted for certain treebanks.

[9] All hyperparameters used the default values.

	ranking	darc	baseline	ÚFAL	best score	best system
all treebanks	12	68.41	68.35	69.52	76.30	Stanford (Stanford)
big treebanks	15	73.31	73.04	74.38	81.77	Stanford (Stanford)
small treebanks	12	52.46	51.80	53.75	61.49	C2L2 (Ithaca)
parallel test-sets	16	67.96	68.33	69.00	73.73	Stanford (Stanford)
surprise languages	18	34.47	37.07	35.96	47.54	C2L2 (Ithaca)
Old Church Slavonic	7	66.37	62.76	62.76	76.84	IMS (Stuttgart)
Gothic	8	61.92	59.81	62.80	71.36	IMS (Stuttgart)
Hindi	8	87.50	86.77	87.28	91.59	Stanford (Stanford)
German PUD	25	65.09	66.53	66.05	74.86	Stanford (Stanford)
Buryat	25	15.61	31.50	21.58	32.24	IMS (Stuttgart)
Korean	25	58.30	59.09	60.30	82.49	Stanford (Stanford)

Table 2: Official results (LAS)

	UAS				LAS			
	mstnn	darc	baseline	ÚFAL	mstnn	darc	baseline	ÚFAL
all treebanks	71.03	74.22	74.41	**75.39**	61.13	68.41	68.35	**69.52**
Ancient Greek	**66.29**	64.92	62.74	65.37	54.78	**58.20**	56.04	57.39
Irish	**73.61**	72.81	72.08	73.10	57.55	**62.97**	61.52	62.87
Turkish	**61.45**	61.33	60.48	60.78	52.44	**54.70**	53.19	53.78
Uyghur	52.05	**53.79**	53.58	53.49	**34.32**	34.28	34.18	33.21

Table 3: Secondary results

same way as described above, while the other features are constructed through *one-hot* encoding.

5 Results

The official test-run took 1 hour 47 minutes on a single-core *Intel Xeon* CPU, which included segmentation, tagging, and parsing. The secondary system took 3 hours and 14 minutes.

In Table 2, we report our official LAS (labeled attached F_1-score) and our rankings among the 33 systems.[10] We compare our system against the baseline (UDPipe 1.1), ÚFAL (UDPipe 1.2), and the best systems. Included are the macro-averaged scores for all and some subsets of the treebanks, plus three of our best-ranking & worst-ranking per-treebank scores.

Our official system was far from the best, but it was comparable to the two UDPipe systems, despite having a much simpler parser. Parsing aside, it had the highest all-tags F_1-score with 73.92%, thanks to the *MorphoDiTa* (Straková et al., 2014) tool incorporated in UDPipe. However, the baseline was very close with 73.74%.

In general, our primary system (darc) outperformed our secondary system (mstnn), both in LAS and UAS (unlabeled attached F_1-score). However, mstnn occasionally achieved better UAS or LAS, as shown in Table 3.

6 Discussions

In Section 3.5 we made a comparison between our darc parser and UDPipe's Parsito parser. Specifically, Parsito supports better oracles and an additional transition algorithm. We attribute the comparable performance of our much simpler parser to the following factors:

- We used more training data.

  The baseline models were trained on subsets of the train-sets, while using the rest for parameter tuning, leaving the dev-sets untouched. Our models were trained on the entire train-sets, and tuned on the dev-sets.

  We believe this was the reason that we ranked above the baseline but below ÚFAL.

- We used LEMMA when possible.

[10] http://universaldependencies.org/conll17/results.html

In our experiments, we fixed the dimensionality of the lexical space, namely the target real vector space where we embed the lexical representation (FORM and/or LEMMA) of the vocabulary. We found that splitting the dimensions between FORM and LEMMA, as opposed to dedicating exclusively to either one, consistently produced the best results.

Further evidence of this is that for four out of the six treebanks where LEMMA was not used, darc performed worse than the baseline.

Splitting the lexical space between FORM and LEMMA actually decreases the number of parameters in the embedding matrices, comparing with using FORM alone, because LEMMA has fewer types.

- We had a better representation for FEATS.

  Simply treating FEATS as atomic symbols is subject to data sparsity as shown in Table 4.

FEATS	max	avg
type count	2487	430
hapax type count	561	92
entry type count	112	44

Table 4: Statistics for FEATS

  Different types of FEATS are merely different combinations of morphological entries, aka the attribute-value pairs. They are compound symbols with a visible structure, which should be preserved.

  Our representation is explained in Section 3.2. We experimented with normalizing the FEATS vectors into unit vectors by their L^2*-norm*, or into probability distributions by their L^1*-norm* as in Alberti et al. (2015). But simple binary indicators seemed to work the best.

Despite the generally similar performance of the original MSTParser in comparison with transition-based parsers with similar learning algorithms, our own mstnn did not meet the expectation, when compared against darc.

The graph-based approach and the transition-based approach are faced with different challenges, and produces different types of errors (McDonald and Nivre, 2007). The former suffers less from the errors of local decisions, but the latter usually benefits from richer features. In our case, the neural network classifier in mstnn used much less information from neighboring nodes than the classifier in darc.

The separate labeler in mstnn was also suboptimal. From UAS to LAS, the absolute drop was 4% higher for mstnn than it was for darc, which actually means a 15.6% higher increase in error rate. This exemplified a general problem with the pipeline approach: Errors made in each step of the pipeline stack up quickly, which is made even worse by the snowball effect, where errors made in one step bring about more errors in the following steps. Another problem is that in a pipeline, the information necessary for making the correct decisions in one step may not be available until later. We experimented with unlabeled parsing using darc, and despite facing a much simpler task than labeled parsing, it yielded lower UAS.

The pipeline approach is a common weakness of both our systems. We believe that for tasks such as this one, an integrated end-to-end system is more desirable.

Acknowledgments

We thank Dr. Çağrı Çöltekin for his encouragement and expert support, which made this work possible. We also thank Dr. Daniël de Kok for his counseling and the anonymous reviewers for their advice.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. http://tensorflow.org/.

Chris Alberti, David Weiss, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks .

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.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*. pages 740–750.

François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica* 14(10):1396.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20(3):273–297.

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning* 75(3):297–325.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B* 71(4):233–240.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *Journal of machine learning research* 9(Aug):1871–1874.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*. pages 959–976.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, pages 11–15.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. pages 1026–1034.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 523–530.

Ryan T McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*. pages 122–131.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553.

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.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017a. 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.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th international conference on parsing technologies*. Association for Computational Linguistics, pages 73–76.

Joakim Nivre et al. 2017b. 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.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

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.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Milan Straka. 2017. CoNLL 2017 shared task - UDPipe baseline models and supplementary materials. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-1990.

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.

Milan Straka, Jan Hajič, Jana Straková, and Jan Hajič jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *Proceedings of Fourteenth International Workshop on Treebanks and Linguistic Theories (TLT 14)*.

Jana Straková, Milan Straka, and Jan Hajic. 2014. Open-source tools for morphology, lemmatization, pos tagging and named entity recognition. In *ACL (System Demonstrations)*. pages 13–18.

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, Hěctor 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.