The ParisNLP entry at the ConLL UD Shared Task 2017: A Tale of a #ParsingTragedy

Éric Villemonte de la Clergerie, Benoît Sagot, Djamé Seddah

The ParisNLP entry at the ConLL UD Shared Task 2017: A Tale of a #ParsingTragedy

Éric de La Clergerie[1] **Benoît Sagot**[1] **Djamé Seddah**[2,1]
(1) Inria (2) Université Paris Sorbonne
{firstname.lastname}@inria.fr

Abstract

We present the ParisNLP entry at the UD CoNLL 2017 parsing shared task. In addition to the UDpipe models provided, we built our own data-driven tokenization models, sentence segmenter and lexicon-based morphological analyzers. All of these were used with a range of different parsing models (neural or not, feature-rich or not, transition or graph-based, etc.) and the best combination for each language was selected. Unfortunately, a glitch in the shared task's Matrix led our model selector to run generic, weakly lexicalized models, tailored for surprise languages, instead of our dataset-specific models. Because of this #ParsingTragedy, we officially ranked 27th, whereas our real models finally unofficially ranked 6th.

1 Introduction

The Universal Dependency parsing shared task (Zeman et al., 2017) was arguably the hardest shared task the field has seen since the CoNLL 2006 shared task (Buchholz and Marsi, 2006) where 13 languages had to be parsed in gold token, gold morphology mode, while its follow up in 2007 introduced an out-of-domain track for a subset of the 2006 languages (Nivre et al., 2007). The SANCL "parsing the web" shared task (Petrov and McDonald, 2012) introduced the parsing of English non-canonical data in gold token, predicted morphology mode and saw a large decrease of performance compared to what was usually reported in English parsing of the Penn Treebank . As far as we know, the SPMRL shared tasks (Seddah et al., 2013, 2014) were first to introduce a non gold tokenization, predicted morphology, scenario for two morphologically rich languages, Arabic and Hebrew while, for other languages, complex source tokens were left untouched (Korean, German, French...). Here, the Universal Dependency (hereafter "UD") shared task introduced an end-to-end parsing evaluation protocol where none of the usual stratification layers were to be evaluated in *gold* mode: tokenization, sentence segmentation, morphology prediction and of course syntactic structures had to be produced[1] for 46 languages covering 81 datasets. Some of them are low-resource languages, with training sets containing as few as 22 sentences. In addition, an out-of-domain scenario was *de facto* included via a new 14-language parallel test set. Because of the very nature of the UD initiative, some languages are covered by several treebanks (English, French, Russian, Finnish...) built by different teams, who interpreted the annotation guidelines with a certain degree of freedom when it comes to rare, or simply not covered, phenomena.[2] Let us add that our systems had to be deployed on a virtual machine and evaluated in a total blind mode with different metadata between the trial and the test runs.

All those parameters led to a multi-dimension shared task which can loosely be summarized by the following "equation":

$$Lang.Tok.WS.Seg.Morph.DS.OOD.AS.Exp,$$

where $Lang$ stands for Language, Tok for tokenization, WS for word segmentation, Seg for sentence segmentation, $Morph$ for predicted morphology, DS for data scarcity, OOD for out-of-domainness, AS for annotation scheme, Exp for experimental environment.

[1] Although baseline prediction for all layers were made available through Straka et al.'s (2016) pre-trained models or pre-annotated development and test files.

[2] See for example, the discrepancy between fr_{partut} and the other French treebanks regarding the annotation of the not so rare *car* conjunction, 'for/because', and the associated syntactic structures, cf. https://github.com/UniversalDependencies/docs/issues/432.

In this shared task, we earnestly tried to cover all of these dimensions, ranking #3 in UPOS tagging and #5 in sentence segmentation. But we were ultimately strongly impacted by the Exp parameter (cf. Section 6.3), a parameter we could not control, resulting in a disappointing rank of #27 out of 33. Once this variable was corrected, we reached rank #6.[3]

Our system relies on a strong pre-processing pipeline, which includes lexicon-enhanced statistical taggers as well as data-driven tokenizers and sentence segmenters. The parsing step proper makes use for each dataset of one of 4 parsing models: 2 non-neural ones (transition and graph-based) and extensions of these models with character and word-level neural layers.

2 Architecture and Strategies

In preparation for the shared task, we have developed and adapted a number of different models for tokenization[4] and sentence segmentation, tagging—predicting UPOS and the values for a (manually selected, language-independent) subset of the morphological attributes (hereafter "partial MSTAGs")— and parsing. For each dataset for which training data was available, we combined different pre-processing strategies with different parsing models and selected the best performing ones based on parsing F-scores on the development set in the predicted token scenario. Whenever no development set was available, we achieved this selection based on a 10-fold cross-evaluation on the training set.

Our baseline pre-processing strategy consisted in simply using the data annotated using UDPipe (Straka et al., 2016) provided by the shared task organizers. We also developed new tools of our own, namely a tagger as well as a joint tokenizer and sentence segmenter. We chose whether to use the baseline UDPipe annotations or our own annotations for each of the following steps: sentence segmentation, tokenization, and tagging (UPOS and partial MSTAGs). We used UDPipe-based information in all configurations for XPOS, lemma, and word segmentation, based on an *a posteriori* character-level alignment algorithm.

[3] http://universaldependencies.org/conll17/results-unofficial.html

[4] We follow the shared task terminology in differentiating tokenization and word segmentation. A tokenizer only performs token segmentation (i.e. source tokens), and does not predict word segmentation (i.e. wordforms, or tree tokens).

At the parsing level, we developed and tried five different parsers, both neural and non-neural, which are variants of the shift-reduce (hereafter "SR") and maximum spanning-tree algorithms (hereafter "MST"). The next two sections describe in more detail our different pre-processing and parsing architectures, give insights into their performance, and show how we selected our final architecture for each dataset.

3 Pre-processing

3.1 Tagging

Tagging architecture Taking advantage of the opportunity given by the shared task, we developed a new part-of-speech tagging system inspired by our previous work on MElt (Denis and Sagot, 2012), a left-to-right maximum-entropy tagger relying on features based on both the training corpus and, when available, an external lexicon. The two main advantages of using an external lexicon as a source of additional features are the following: (i) it provides the tagger with information about words unknown to the training corpus; (ii) it allows the tagger to have a better insight into the right context of the current word, for which the tagger has not yet predicted anything.

Whereas MElt uses the `megam` package to learn tagging models, our new system, named alVW-Tagger, relies on Vowpal Wabbit.[5] One of Vowpal Wabbit's major advantages is its training speed, which allowed us to train many tagger variants for each language, in order to assess, for each language, the relative performance of different types of external lexicons and different ways to use them as a source of features. In all our experiments, we used VW in its default multiclass mode, i.e. using a squared loss and the one-against-all strategy. Our feature set is a slight extension of the one used by MElt (cf. Denis and Sagot, 2012).

The first improvement over MElt concerns how information extracted from the external lexicons is used: instead of only using the categories provided by the lexicon, we also use morphological features. We experimented different modes. In the baseline mode, the category provided by the lexicon is the concatenation of a UPOS and a sequence of morphological features, hereafter the "full category". In the F mode ("ms mode" in Table 1), only the UPOS is used (morphological features

[5] https://github.com/JohnLangford/vowpal_wabbit/

are ignored). In the M mode, both the full category and the sequence of morphological features are used separately. Finally, in the FM mode, both the UPOS and the sequence of morphological features are used separately.

The second improvement over MElt is that alVWTagger predicts both a part-of-speech (here, a UPOS) and a set of morphological features. As mentioned earlier, we decided to restrict the set of morphological features we predict, in order to reduce data sparsity.[6] For each word, our tagger first predicts a UPOS. Next, it uses this UPOS as a feature to predict the set of morphological features as a whole, using an auxiliary model.[7]

Extraction of morphological lexicons As mentioned above, our tagger is able to use an external lexicon as a source of external information. We therefore created a number of morphological lexicons for as many languages as possible, relying only on data and tools that were allowed by the shared task instructions. We compared the UPOS accuracies on the development sets, or on the training sets in an 10-fold setting when no development data was provided, and retained the best performing lexicon for each dataset (see Table 1). Each lexicon was extracted from one of the following sources or several of them, using an *a posteriori* merging algorithm:

- The monolingual lexicons from the Apertium project (lexicon type code "AP" in Table 1);
- Raw monolingual raw corpora provided by the shared task organizers, after application of a basic rule-based tokenizer and the appropriate Apertium or Giellatekno morphological analyzers (codes "APma" or "GTma");
- The corresponding training dataset (code "T") or another training dataset for the same language (code "T*dataset*");
- The UDPipe-annotated corpora provided by the shared task organizers (code "UDP");
- A previously extracted lexicon for another language, which we automatically "translated" using a dedicated algorithm, which we provided, as a seed, with a bilingual lexicon automatically extracted from OPUS sentence-aligned data (code "TR*source_language*").

All lexical information not directly extracted from UDPipe-annotated data or from training data was converted to the UD morphological categories (UPOS and morphological features).

For a few languages only (for lack of time), we also created expanded versions of our lexicons using word embeddings re-computed on the raw data provided by the organizers, assigning to words unknown to the lexicon the morphological information associated with the closest known word (using a simple euclidian distance on the word embedding space).[8] When the best performing lexicon is one of these extended lexicons, it is indicated in Table 1 by the "-e" suffix.

3.2 Tokenization and sentence segmentation

Using the same architecture as our tagger, yet without resorting to external lexicons, we developed a data-driven tokenizer and sentence segmenter, which runs as follows. First, a simple rule-based pre-tokenizer is applied to the raw text of the training corpus, after removing all sentence boundaries.[9,10] This pre-tokenizer outputs a sequence of "pre-tokens," in which, at each pre-token boundary, we keep trace of whether a whitespace was present in the raw text or not at this position. Next, we use the gold train data to label each pre-token boundary with one of the following labels: not a token boundary (NATB), token boundary (TB), sentence boundary (SB).[11] This model can then be applied on raw text, after the pre-tokenizer has been applied. It labels each pre-token boundary, resulting in the following decisions depending on whether it corresponds to a whitespace in the raw text or not: (i) if it predicts NATB at a non-whitespace boundary, the boundary is removed; (ii) if it predicts NATB at a whitespace boundary, it results in a token-with-space; (iii) if it predicts TB (resp. SB) at a non-whitespace boundary, a token (resp. sentence) boundary is created and "SpaceAfter=No" is added to the preceedings token; (iv) if it predicts

[6] The list of features we retained is the following: Case, Gender, Number, PronType, VerbForm, Mood, and Voice.

[7] We also experimented with per-feature prediction, but it resulted in slightly lower accuracy results on average, as measured on development sets.

[8] We did not used the embeddings provided by the organizers because we experimentally found that the 10-token window used to train these embeddings resulted in less accurate results than when using smaller windows, especially when the raw corpus available was of a limited size.

[9] Apart from paragraph boundaries whenever available.

[10] On languages such as Japanese and Chinese, each non-latin character is a pre-token on its own.

[11] Our tokenizer and sentence segmenter relies on the almost same features as the tagger, except for two special features, which encode whether the current pre-token is a strong (resp. weak) punctuation, based on two manually crafted lists.

Dataset	ours (best setting) lexicon type	ms mode	overall acc.	UDPipe overall acc.	Dataset	ours (best setting) lexicon type	ms mode	overall acc.	UDPipe overall acc.	Dataset	ours (best setting) lexicon type	ms mode	overall acc.	UDPipe overall acc.
ar	AP-e	M	**94.71**	94.57	fr	AP-e		**97.30**	97.08	nl	AP	F	**94.70**	94.07
bg	AP	F	97.61	**97.72**	$fr_{sequoia}$	AP-e	FM	**97.54**	96.60	$nl_{lassysmall}$	AP+Tnl	F	**96.74**	95.65
ca	AP-e	FM	**98.42**	98.15	ga	UDP	M	—	—	$no_{bokmaal}$	AP		**97.66**	97.34
cs	AP	M	**98.83**	98.48	gl	AP	FM	**97.45**	96.77	$no_{nynorsk}$	AP	M	**97.23**	96.74
cs_{cac}	Tcs		**99.24**	98.78	got	T		**94.53**	94.22	pl	AP	M	**97.03**	95.34
cs_{cltt}	AP+Tcs	F	**94.34**	92.06	grc	$Tgrc_{proiel}$-e		**89.56**	81.54	pt	AP	FM	**97.21**	97.00
cu	T	F	**95.15**	94.07	grc_{proiel}	UDP-e	FM	**96.40**	96.01	pt_{br}	AP+Tpt	FM	**97.96**	97.40
da	AP		**96.30**	95.19	he	AP	FM	**96.68**	95.72	ro	AP		**97.34**	96.98
de	AP	M	**92.70**	91.39	hi	AP	F	**96.59**	95.79	ru	AP	M	**96.62**	94.95
el	AP	F	**95.53**	94.17	hr	TRsl	M	**96.94**	96.15	$ru_{syntagrus}$	AP	FM	**98.54**	98.20
en	AP	F	**94.68**	94.43	hu	T		**93.90**	92.31	sk	TRcs	FM	**96.00**	93.14
en_{lines}	AP	FM	**96.08**	94.75	id	AP	M	92.98	**93.36**	sl	AP	FM	**97.82**	96.34
en_{partut}	AP+T	FM	**95.90**	94.39	it	AP	F	**97.55**	97.23	sv	AP	FM	**96.32**	95.17
es	AP		**96.47**	96.24	it_{partut}	Trit	M	**97.89**	95.16	sv_{lines}	AP	F	**96.01**	94.63
es_{ancora}	AP	FM	**98.39**	98.16	ja	*no lexicon*		**96.87**	96.72	tr	APma	FM	**93.65**	92.25
et	GTms	FM	**89.28**	87.52	kk	APms		—	—	ug	UDP		—	—
eu	AP	F	**94.48**	92.80	ko	*no lexicon*		**93.77**	93.68	uk	AP	M	—	—
fa	*no lexicon*		96.04	**96.17**	la_{ittb}	TRit+T		**97.15**	96.86	ur	AP		**93.01**	92.45
fi	GTms	FM	**95.06**	94.52	la_{proiel}	TRit+T-e	FM	**95.62**	95.43	vi	*no lexicon*		88.60	**88.68**
fi_{ftb}	GTms	F	**92.50**	92.34	lv	AP	FM	**93.43**	90.81	zh	AP+UDP		**91.40**	91.21

Table 1: UPOS accuracies for the UDPipe baseline and for our best alVWTagger setting.

TB (resp. SB) at a whitespace boundary, a token (resp. sentence) boundary is created.

We compared our tokenization and sentence segmentation results with the UDPipe baseline on development sets. Whenever the UDPipe tokenization and sentence segmentation scores were both better, we decided to use them in all configurations. Other datasets, for which tokenization and sentence segmentation performance is shown in Table 2, were split into two sets: those on which our tokenization was better but sentence segmentation was worse—for those, we forced the UDPipe sentence segmentation in all settings—, and those for which both our tokenization and sentence segmentation were better.

3.3 Preprocessing model configurations

As mentioned in Section 2, we used parsing-based evaluation to select our pre-processing strategy for each corpus. More precisely, we selected for each dataset one of the following strategies:

1. UDPIPE: the UDPipe baseline is used and provided as such to the parser.
2. TAG: the UDPipe baseline is used, except for the UPOS and MSTAG information, which is provided by our own tagger.
3. TAG+TOK+SEG and TAG+TOK: we apply our own tokenization and POS-tagger to produce UPOS and MSTAG information; sentence segmentation is performed either by us (TAG+TOK+SEG (available for datasets with "yes" in the last column in Table 2) or by the UDPipe baseline (TAG+TOK, available for datasets with "no" in Table 2).

Dataset	ours tok. F-sc.	ours sent. F-sc.	UDPIPE tokenis. F-sc.	UDPIPE sent. F-sc.	Use our sent. seg.?
ar	99.54	**92.75**	**99.99**	77.99	yes
ca	99.95	**99.01**	**99.97**	98.77	yes
cs_{cac}	**100.00**	**99.5**	**100.00**	99.09	yes
cu	99.98	**39.44**	**100.00**	37.09	yes
da	**100.00**	84.01	99.68	**84.36**	no
el	99.57	**92.61**	**99.87**	88.67	yes
et	99.69	**90.82**	**99.79**	84.91	yes
eu	99.93	**99.42**	**99.99**	99.00	yes
fa	99.95	**99.42**	**100.00**	97.14	yes
fi	99.54	**87.76**	**99.69**	86.47	yes
fi_{ftb}	**100.00**	**85.94**	99.94	82.52	yes
gl	**99.73**	**96.8**	99.06	92.44	yes
got	99.99	**28.95**	**100.00**	23.51	yes
hu	99.74	**96.83**	**99.91**	94.55	yes
it	99.73	**96.31**	**99.82**	93.20	yes
ja	**92.63**	99.61	89.53	**99.71**	no
la_{ittb}	**99.90**	**78.71**	99.88	77.38	yes
la_{proiel}	**100.00**	**19.92**	99.99	19.76	yes
lv	**99.21**	91.48	98.91	**96.48**	no
$no_{nynorsk}$	99.82	**94.30**	**99.92**	93.05	yes
pt	**99.89**	**90.23**	99.82	89.27	yes
vi	**87.25**	92.23	83.99	**96.28**	no

Table 2: Tokenization and sentence segmentation accuracies for the UDPipe baseline and our tokenizer (restricted to those datasets for which we experimented the use of our own tokenization).

Whenever we used our own tokenization and not that of the UDPipe baseline, we used a character-level alignment algorithm to map this information to our own tokens.

Table 1 shows the configuration retained for each language for which a training set was provided in advance.[12] For surprise language

[12]Note that our parsing-performance-based selection strategy did not always result in the same outcome as what we would have been chosen based solely on the comparison of our own tools with UDPipe's baseline. For instance, our new tagger gets better results than UDPipe in UPOS tagging on all development corpora but one, yet we used UDPipe-based UPOS for 24 non-PUD corpora.

datasets, we always used the UDPipe configuration.[13] For PUD corpora, we used the same configuration as for the basic dataset for the same language (for instance, we used for the fr_{pud} dataset the same configuration as that chosen for the fr dataset).[14] Table 1 indicates for each dataset which configuration was retained.

4 Parsing Models

We used 4 base parsers, all implemented on top of the DYALOG system (de La Clergerie, 2005), a logic-programming environment (*à la* Prolog) specially tailored for natural language processing, in particular for tabulation-based dynamic programming algorithms.

Non-neural parsing models The first two parsers are feature-based and use no neural components. The most advanced one is DYALOG-SR, a shift-reduce transition-based parser, using dynamic programming techniques to maintain beams (Villemonte De La Clergerie, 2013). It accepts a large set of transition types, besides the usual `shift` and `reduce` transitions of the arc-standard strategy. In particular, to handle non-projectivity, it can use different instances of `swap` transitions, to swap 2 stack elements between the 3 topmost ones. A `noop` transition may also be used at the end of parsing paths to compensate differences in path lengths. Training is done with a structured averaged perceptron, using early aggressive updates, whenever the oracle falls out of the beam, or when a violation occurs, or when a margin becomes too high, etc.[15]

Feature templates are used to combine elementary standard features:

- Word features related to the 3 topmost stack elements $s_{i=0\cdots2}$, 4 first buffer elements $I_{j=1\cdots4}$, leftmost/rightmost children $[lr]s_i$ /grandchildren of the stack elements $[lr]2s_i$, and governors. These features include the lexical form, lemma, UPOS, XPOS, morphosyntactic features, Brown-like clusters (derived from word embeddings), and flags indicating capitalization, numbers, etc.
- Binned distances between some of these elements
- Dependency features related to the leftmost/rightmost dependency labels for s_i (and dependent $[lr]s_i$), label set for the dependents of s_i and $[lr]s_i$, and number of dependents
- Last action (+label) leading to the current parsing state.

The second feature-based parser is DYALOG-MST, a parser developed for the shared task and implementing the Maximum Spanning Tree (MST) algorithm (McDonald et al., 2005). By definition, DYALOG-MST may produce non-projective trees. Being recent and much less flexible than DYALOG-SR, it also relies on a much smaller set of first-order features and templates, related to the source and target words of a dependency edge, plus its label and binned distance. It also exploits features related to the number of occurrences of a given POS between the source and target of an edge (inside features) or not covered by the edge but in the neighborhood of the nodes (outside features). Similar features are also implemented for punctuation.

Neural parsing models Both feature-based parsers were then extended with a neural-based component, implemented in C++ with DyNet (Neubig et al., 2017). The key idea is that the neural component can provide the best parser action or, if aksed, a ranking of all possible actions. This information is then used as extra features to finally take a decision. The 2 neural-based variants of DYALOG-SR and DYALOG-MST, straightforwardly dubbed DYALOG-SRNN and DYALOG-MSTNN, implement a similar architecture, the one for DYALOG-SRNN being a bit more advanced and stable. Moreover, DYALOG-MSTNN was only found to be the best choice for a very limited number of treebanks. In addition to these models, we also investigated a basic version of DYALOG-SRNN that only uses, in a feature-poor setting, its character-level component and its joint action prediction, and which provides the best performance on 3 languages. The following discussion will focus on DYALOG-SRNN.

The architecture is inspired by Google's PARSEYSAURUS (Alberti et al., 2017), with a first left-to-right char LSTM covering the whole

[13] For surprise languages, the UDPipe baseline was trained on data not available to the shared task participants.

[14] Because of a last-minute bug, we used the TAG configuration for tr_{pud} and pt_{pud} although we used the UDPIPE configuration for tr and pt. We also used the TAG setting for fi_{pud} rather than the TAG+TOK+SEG setting used for fi.

[15] By "violation," we mean for instance adding an edge not present in the gold tree, a first step towards dynamic oracles. We explored this path further for the shared task through dynamic programming exploration of the search space, yet did not observe significant improvements yet.

sentence and (artificial) whitespaces introduced to separate tokens.[16] The output vectors of the char LSTM at the token separations are used as (learned) word embeddings that are concatenated (when present) with both the pre-trained ones provided for the task and the UPOS tags predicted by the external tagger. The concatenated vectors serve as input to a word bi-LSTM that is also used to predict UPOS tags as a joint task (training with the gold tags provided as oracle). For a given word w_i, its final vector representation is the concatenation of the output of the bi-LSTM layers at position i with the LSTM-predicted UPOS tag.

The deployment of the LSTMs is done once for a given sentence. Then, for any parsing state, characterized by the stack, buffer, and dependency components mentioned above, a query is made to the neural layers to suggest an action. The query fetches the final vectors associated with the stack, buffer, and dependent *state* words, and completes it with input vectors for 12 (possibly empty) label dependencies and for the last action. The number of considered state words is a hyper-parameter of the system, which can range between 10 and 19, the best and default value being 10, covering the 3 topmost stack elements and 6 dependent children, but only the first buffer lookahead word[17] and no grandchildren. Through a hidden layer and a softmax layer, the neural component returns the best action `paction` (and `plabel`) but also the ranking and weights of all possible actions. The best action is used as a feature to guide the decision of the parser in combination with the other features, the final weight of an action being a linear weighted combination of the weights returned by both perceptron and neural layers.[18]

A dropout rate of 0.25 was used to introduce some noise. The Dynet AdamTrainer was chosen for gradient updates, with its default parameters. Many hyperparameters are however available as options, such as the number of layers of the char and word LSTMs, the size of input, hidden and output dimensions for the LSTMs and feed-forward layers. A partial exploration of these parameters was run on a few languages, but not in a systematic way given the lack of time and the huge number of possibilities. Clearly, even if we did try 380 distinct parsing configurations through around 16K training runs,[19] we are still far away from language-specific parameter tuning, thus leaving room for improvement.

5 Results

Because of the greater flexibility of transition-based parsers, MST-based models were only used for a few languages. However, our results, provided in the Appendix, show the good performance of these models, for instance on Old Church Slavonic (cu), Gothic (got), Ancient Greek (grc), and Kazakh (kk). Already during development, it was surprising to observe, for most languages, a strong preference for either SR-based models or MST-based ones. For instance, for Ancient Greek, the best score in gold token mode for a MST-based model is 62.43 while the best score for a SR-based one is 60.59. On the other hand, for Arabic (ar), we get 74.87 for the best SR model and 71.44 for the best MST model.

Altogether, our real, yet unofficial scores are encouraging (ranking #6 in LAS) while our official UPOS tagging, sentence segmentation and tokenization results ranked respectively #3, #6 and #5. Let us note that our low LAS official results, #27, was the result of a mismatch between the trial and test experimental environments provided by the organizers (cf. Section 6.3). However, we officially ranked #5 on surprise languages, which were not affected by this mismatch.

6 Discussion

While developing our parsers, training and evaluation were mostly performed using the UDPipe pre-processing baseline with predicted UPOS and MSTAGs but gold tokenization and gold sentence segmentation. For several (bad) reasons, only in the very last days did we train on files tagged with our preprocessing chain. Even later, evaluation (but no training) was finally performed on dev files with predicted segmentation and tokenization, done by either UDPipe or by our pre-processing chains (TAG, TAG+TOK+SEG or TAG+TOK). Based on the results, we selected, for each language and treebank, the best preprocessing configuration and the best parsing model.

[16] A better option would be to add whitespace only when present in the original text.

[17] We suppose the information relative to the other lookahead words are encapsulated in the final vector of the first lookahead word.

[18] The best way to combine the weights of the neural and feature components remains a point to further investigate.

[19] We count as a training run the conjunction of a parser configuration, a treebank, and a beam size. Please note that a synthesis may be found at `http://alpage.inria.fr/~clerger/UD17/synthesis.html`

In general, we observed that neural-based models without features often worked worse than pure feature-based parsers (such as `srcat`), but great when combined with features. We believe that, being quite recent, our neural architecture is not yet up-to-date and that we still have to evaluate several possible options. Between using no features (`srnnsimple` and `srnncharsimple` models) and using a rich feature set (`srnnpx` models), where the predicted actions `paction` and `plabel` may be combined with other features, we also tested, for a few languages, a more restricted feature set with no combinations of `paction` and `plabel` with other features (`srnncharjoin` models). These latter combinations are faster to train and reach good scores, as shown in Table 3.

Lang	srcat	srnn charsimple	srnn charjoin	srnn px
sk	76.85	67.85	78.87	80.84
cs_{cac}	84.48	77.07	84.80	84.85
lv	63.86	59.57	66.95	68.74
ko	54.43	57.19	70.91	71.45

Table 3: Neural models & feature impact (Dev)

For the treebanks without dev files, we simply did a standard training, using a random 80/20-split of the train files. Given more time, we would have tried transfer from other treebanks when available (as described below).

To summarize, a large majority of 47 selected models were based on DYALOG-SRNN with a rich feature set, 29 of them relying on predicted data coming from our processing chains (TAG or TAG+TOK+SEG), the other ones relying on the tags and segmentation predicted by UDPipe. 10 models were based on DYALOG-MSTNN, with 5 of them relying on our preprocessing chain. Finally, 5 (resp. 2) were simply based on DYALOG-SR (resp. DYALOG-MST), none of them using our preprocessing.

6.1 OOV Handling

Besides the fact that we did not train on files with predicted segmentation, we are also aware of weaknesses in handling unknown words in test files. Indeed, at some point, we made the choice to filter the large word-embedding files by the vocabulary found in the train and dev files of each dataset. We made the same for the clusters we derived from word embeddings. It means that unknown words have no associated word embeddings or clusters (besides a default one). The impact of this choice is not yet clear but it should be a relatively significant part of the performance gap between our score of the dev set (with predicted segmentation) and on the final test set.[20]

6.2 Generic Models

We also started exploring the idea of transferring information between close languages, such as Slavic languages. Treebank families were created for some groups of related languages by randomly sampling their respective treebanks as described in Table 4. A fully generic treebank was also created by randomly sampling 41k sentences from almost all languages (1k sentences per primary treebank).

Model	Languages	#sent.
ZZNorthGerman	da, no, sv	8k
ZZRoman	fr, ca, es, it, pt, ro, gl	20k
ZZSouthSlavic	bg, cu, hr, sl	16k
ZZWestSlavic	cs, pl, sk	9k
ZZWestGerman	de, du, nl	12k
ZZGeneric	sampling main 46 lang.	41k

Table 4: Generic models partition

The non-neural parsers were trained on these new treebanks, using much less lexicalized information (no clusters, no word embeddings, no lemmas, and keeping only UPOS tags, but keeping forms and lemmas when present). We tested using the resulting models, whose named are prefixed with "ZZ", as base models for further training on some small language treebanks. However, preliminary evaluations did not show any major improvement and, due to lack of time, we put on hold this line of research, while keeping our generic parsers as backup ones.

Some of these generic parsers were used for the 4 surprise languages, with Upper Sorbian using a ZZSSlavic parser[21] (LAS=56.22), North Saami using ZZFinnish (LAS=37.33), and the two other ones using the fully generic parser (Kurmanji LAS=34.8; Buryat LAS=28.55).

6.3 The Tragedy

Ironically, the back-off mechanism we set up for our model selection was also a cause of failure and salvation. Because of the absence of the `name`

[20] An average of 6 points between dev and tests. Dev results available at `goo.gl/lyuC8L`.

[21] We had planned to use a ZZWSlavic parser but made a mistake in the configuration file.

field in the test set metadata, which was nevertheless present in the dev run and crucially also in the trial run metadata, the selection of the best model per language was broken and led to the selection of back-off models, either a family one or in most cases the generic one. The Tira blind test configuration prevented us from discovering this experimental discrepancy before the deadline. Once we adapted our wrapper to the test metadata, the appropriate models were selected, resulting in our real run results. It turned out that our non language-specific, generic models performed surprisingly well, with a macro-average F-score of 60% LAS. Of course, except for Ukrainian, our language-specific models reach much better performance, with a macro-average F-score of 70.3%. But our misadventure is an invitation to further investigation.

However, it is unclear at this stage whether or not mixing languages in a large treebank really has advantages over using several small treebanks. In very preliminary experiments on Greek, Arabic, and French, we extracted the 1000 sentences present in the generic treebank for these languages and trained the best generic configuration (`srcat`, beam 6) on each of these small treebanks. As shown in Table 5, the scores on the development sets do not exhibit any improvement coming from mixing languages in a large pool and are largely below the scores obtained on a larger language-specific treebank.

Lang	generic	small	full	size
Greek	76.25	76.29	78.40	1,662
Arabic	57.94	59.84	64.34	6,075
French	78.28	79.59	84.81	14,553

Table 5: Generic pool vs. small treebank vs. full treebank (with `srcat` models (LAS, Dev)

6.4 Impact of the Lexicon

We also investigated the influence of our tagging strategy with respect to the UDPipe baseline. Figure 1 plots the parsing LAS F-scores with respect to training corpus size. It also show the result of logarithmic regressions performed on datasets for which we used the UDPipe baseline for preprocessing versus those for which we used the TAG configuration. As can be seen, using the UDPipe baseline results in a much stronger impact of training corpus size, whereas using our own tagger leads to more stable results. We interpret this

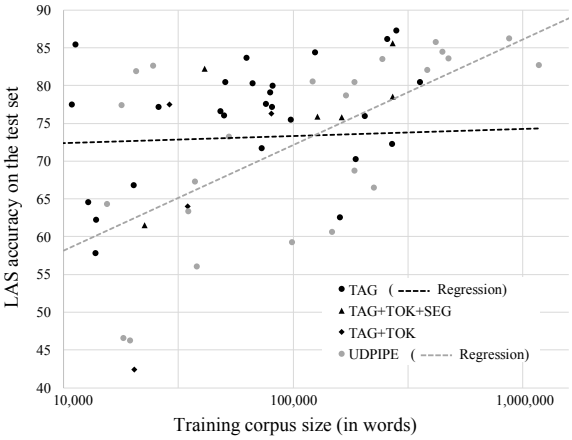

Figure 1: LAS F-score w.r.t. training corpus size

observation as resulting from the influence of external lexicons during tagging, which lowers the negative impact of out-of-training-corpus words on tagging and therefore parsing performance. It illustrates the relevance of using external lexical information, especially for small training corpora.

7 Conclusion

The shared task was a excellent opportunity for us to develop a new generation of NLP components to process a large spectrum of languages, using some of the latest developments in deep learning. However, it was really a challenging task, with an overwhelming number of decisions to take and experiments to run over a short period of time.

We now have many paths for improvement. First, because we have a very flexible but newly developed architecture, we need to stabilize it by carefully selecting the best design choices and parameters. We also plan to explore the potential of a multilingual dataset based on the UD annotation scheme, focusing on cross-language transfer and language-independent models.

Acknowledgments

We thank the organizers and the data providers who made this shared task possible within the core Universal Dependencies framework (Nivre et al., 2016), namely the authors of the UD version 2.0 datasets (Nivre et al., 2017), the baseline UDPipe models (Straka et al., 2016), and of course the team behind the TIRA evaluation platform (Potthast et al., 2014) to whom we owe a lot.

References

Chris Alberti, Daniel Andor, Ivan Bogatyy, Michael Collins, Dan Gillick, Lingpeng Kong, Terry Koo, Ji Ma, Mark Omernick, Slav Petrov, Chayut Thanapirom, Zora Tung, and David Weiss. 2017. SyntaxNet models for the CoNLL 2017 shared task. *CoRR* abs/1703.04929.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of the Tenth Conference on Computational Natural Language Learning*. New York City, USA, pages 149–164.

Éric de La Clergerie. 2005. DyALog: a tabular logic programming based environment for NLP. In *Proceedings of 2nd International Workshop on Constraint Solving and Language Processing (CSLP'05)*. Barcelone, Espagne.

Pascal Denis and Benoît Sagot. 2012. Coupling an annotated corpus and a lexicon for state-of-the-art POS tagging. *Language Resources and Evaluation* 46(4):721–736.

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*. pages 523–530.

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* .

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)*. Portorož, Slovenia, pages 1659–1666.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Prague, Czech Republic, pages 915–932.

Joakim Nivre et al. 2017. 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.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*. Montreal, Canada, volume 59.

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.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*. Dublin, Ireland, pages 103–109.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiorkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clérgerie. 2013. Overview of the spmrl 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proc. of the 4th Workshop on Statistical Parsing of Morphologically Rich Languages: Shared Task*. Seattle, USA.

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)*. Portorož, Slovenia.

Éric Villemonte De La Clergerie. 2013. Exploring beam-based shift-reduce dependency parsing with DyALog: Results from the SPMRL 2013 shared task. In *4th Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL'2013)*. Seattle, USA.

Daniel Zeman, Filip Ginter, Jan Hajič, Joakim Nivre, Martin Popel, Milan Straka, and et al. 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*. Vancouver, Canada, pages 1–20.

Appendix: Overall Results

Dataset		Preproc.	UPOS tagging		Parsing (LAS)				
name	#train wds	mode	acc.	rank	model	real run	real rank	official run	△(real−official)
ar	224k	UDPIPE	88.13	8	SR-nn	66.47	11	56.72	9.75
ar_{pud}	—	UDPIPE	70.27	8	SR-nn	45.38	10	42.73	2.65
bg	124k	TAG	97.29	25	SR-nn	84.41	13	74.15	10.26
bxr	—	UDPIPE	84.12	1	SR-cat (Generic)	28.55	6	28.55	0
ca	418k	UDPIPE	98.04	3	SR-feats	85.78	11	81.94	3.84
cs	1,173k	UDPIPE	98.13	6	SR-cat	82.72	18	71.30	11.42
cs_{cac}	473k	UDPIPE	98.27	8	SR-nn-charjoin	83.56	12	73.62	9.94
cs_{cltt}	16k	TAG	97.77	2	SR-nn	76.72	6	58.65	18.07
cs_{pud}	—	UDPIPE	96.55	2	SR-cat	79.28	18	72.05	7.23
cu	37k	UDPIPE	93.34	11	MST-nn	67.34	4	63.64	3.70
da	80k	TAG+TOK	96.43	2	SR-nn	76.31	8	73.83	2.48
de	270k	TAG	92.04	6	SR-nn	72.27	12	68.45	3.82
de_{pud}	—	TAG	84.58	2	SR-nn	69.79	8	65.16	4.63
el	41k	TAG+TOK+SEG	96.26	3	SR-nn	82.25	5	77.00	5.25
en	205k	TAG	93.52	3	SR-nn	76.00	14	70.20	5.80
en_{lines}	50k	TAG	95.99	2	SR-nn	76.01	7	64.77	11.24
en_{partut}	26k	TAG	94.78	2	SR-nn	77.21	7	69.29	7.92
en_{pud}	—	TAG	94.74	2	SR-nn	80.68	7	75.87	4.81
es	382k	UDPIPE	95.60	5	SR-nn	82.09	13	77.94	4.15
es_{ancora}	445k	UDPIPE	98.15	4	SR-feats	84.47	14	76.61	7.86
es_{pud}	—	UDPIPE	88.15	5	SR-nn	78.15	11	76.57	1.58
et	23k	TAG+TOK+SEG	89.24	5	SR-nn	61.52	8	56.00	5.52
eu	73k	TAG	94.01	3	SR-nn	71.70	8	50.67	21.03
fa	121k	UDPIPE	96.00	7	SR-nn	80.52	9	61.93	18.59
fi	163k	TAG+TOK+SEG	84.59	4	SR-nn	75.82	9	60.19	15.63
fi_{ftb}	128k	TAG+TOK+SEG	93.11	2	MST-nn	75.88	9	40.21	35.67
fi_{pud}	—	TAG	95.98	4	SR-nn	79.68	7	62.41	17.27
fr	356k	TAG	95.51	6	SR-nn	80.44	16	76.79	3.65
fr_{partut}	18k	UDPIPE	94.46	7	SR-nn	77.40	16	75.15	2.25
fr_{pud}	—	TAG	88.54	4	SR-nn	74.82	10	74.70	0.12
$fr_{sequoia}$	51k	TAG	96.78	2	SR-nn	80.48	13	73.55	6.93
ga	14k	TAG	90.00	3	MST-nn	62.23	13	56.35	5.88
gl	79k	TAG	97.42	2	SR-nn	79.06	13	76.81	2.25
$gl_{treegal}$	15k	UDPIPE	90.69	9	SR-nn	64.36	21	65.95	-1.59
got	35k	UDPIPE	93.55	10	MST-nn	63.40	4	58.66	4.74
grc	160k	TAG	88.48	3	MST-nn	62.53	5	47.37	15.16
grc_{proiel}	184k	UDPIPE	95.72	7	MST-altcats	68.72	7	49.41	19.31
he	138k	TAG	81.42	4	SR-nn	57.85	15	44.49	13.36
hi	281k	TAG	96.61	3	SR-nn	87.25	10	45.72	41.53
hi_{pud}	—	TAG	84.62	4	SR-nn	51.46	10	32.39	19.07
hr	169k	UDPIPE	95.67	12	SR-nn	78.67	9	74.81	3.86
hsb	—	UDPIPE	90.30	1	SR-altcats (SSlavic (!))	56.22	9	56.22	0
hu	20k	TAG	92.19	4	SR-nn	66.82	6	49.82	17
id	98k	TAG	93.59	2	SR-nn	75.47	10	64.84	10.63
it	271k	TAG+TOK+SEG	97.38	3	SR-nn	85.59	17	81.20	4.39
it_{pud}	—	TAG+TOK+SEG	93.08	6	SR-nn	84.17	13	81.81	2.36
ja	29k	TAG+TOK	90.00	5	SR-nn	77.52	6	65.15	12.37
ja_{pud}	—	TAG+TOK	88.49	25	SR-nn	76.03	20	62.91	13.12
kk	162k	TAG	67.86	1	MST-nn	26.64	4	24.73	1.91
kmr	0.5k	UDPIPE	90.04	1	SR-cat (Generic)	34.80	12	34.80	0
ko	52k	UDPIPE	93.79	7	SR-nn	73.26	6	40.71	32.55
la	18k	UDPIPE	83.39	11	MST-nn	46.59	13	39.91	6.68
la_{ittb}	270k	TAG+TOK+SEG	97.39	6	SR-nn	78.51	10	52.38	26.13
la_{proiel}	147k	UDPIPE	94.82	8	MST-altcats	60.65	10	42.68	17.97
lv	35k	TAG+TOK	91.10	2	SR-nn	64.03	6	50.52	13.51
nl	186k	TAG	91.57	3	SR-nn	70.28	11	56.11	14.17
$nl_{lassysmall}$	81k	TAG	97.84	2	SR-nn	79.99	10	57.83	22.16
$no_{bokmaal}$	244k	UDPIPE	96.75	6	SR-nn	83.49	14	68.58	14.91
$no_{nynorsk}$	245k	UDPIPE	96.38	7	SR-feats	82.66	10	65.11	17.55
pl	63k	TAG	96.95	2	SR-nn	83.65	6	71.98	11.67
pt	207k	UDPIPE	96.22	6	SR-nn-charjoint	81.87	17	79.21	2.66
pt_{br}	256k	TAG	97.54	2	SR-nn	86.17	13	61.30	24.87
pt_{pud}	—	TAG	88.88	2	SR-nn-charjoint	74.67	10	75.00	-0.33
ro	185k	UDPIPE	96.40	10	SR-nn	80.47	13	76.69	3.78
ru	76k	TAG	96.60	2	SR-nn	77.61	7	66.83	10.78
ru_{pud}	—	TAG	86.66	3	SR-nn	71.55	3	66.17	5.38
$ru_{syntagrus}$	870k	UDPIPE	97.99	4	SR-altcats	86.25	18	54.19	32.06
sk	81k	TAG	95.10	2	SR-nn	77.17	6	67.72	9.45
sl	113k	TAG	97.36	2	SR-nn	85.41	5	80.27	5.14
sl_{sst}	19k	UDPIPE	88.82	9	MST-nn	46.27	17	40.15	6.12
sme	—	UDPIPE	86.81	1	SR-cat (Generic)	37.33	5	37.33	0
sv	67k	TAG	96.74	2	SR-nn	80.30	6	76.77	3.53
sv_{lines}	48k	TAG	95.83	2	SR-nn	76.61	7	63.50	13.11
sv_{pud}	—	TAG	93.43	2	SR-nn	73.65	4	70.83	2.82
tr	38k	UDPIPE	91.22	10	SR-nn	56.03	9	46.38	9.65
tr_{pud}	—	TAG	72.65	2	SR-nn	32.74	20	25.78	6.96
ug	2k	TAG	74.06	9	MST-nn	34.82	10	19.65	15.17
uk	13k	TAG	92.10	2	MST-nn	64.58	6	65.52	-0.94
ur	109k	TAG	92.65	6	SR-nn	77.46	12	39.73	37.73
vi	20k	TAG+TOK	77.64	2	SR-nn	42.40	3	33.00	9.40
zh	99k	UDPIPE	82.69	11	SR-nn	59.28	12	45.83	13.45
Overall (macro-average)			91.79	3		70.35	6	60.02	10.33