The ADAPT Enhanced Dependency Parser at the IWPT 2020 Shared Task

James Barry and **Joachim Wagner** and **Jennifer Foster**
ADAPT Centre
School of Computing, Dublin City University, Ireland
`firstname.lastname@adaptcentre.ie`

Abstract

We describe the ADAPT system for the 2020 IWPT Shared Task on parsing enhanced Universal Dependencies in 17 languages. We implement a pipeline approach using UDPipe and UDPipe-future to provide initial levels of annotation. The enhanced dependency graph is either produced by a graph-based semantic dependency parser or is built from the basic tree using a small set of heuristics. Our results show that, for the majority of languages, a semantic dependency parser can be successfully applied to the task of parsing enhanced dependencies.

Unfortunately, we did not ensure a connected graph as part of our pipeline approach and our competition submission relied on a last-minute fix to pass the validation script which harmed our official evaluation scores significantly. Our submission ranked eighth in the official evaluation with a macro-averaged coarse ELAS F1 of 67.23 and a treebank average of 67.49. We later implemented our own graph-connecting fix which resulted in a score of 79.53 (language average) or 79.76 (treebank average), which would have placed fourth in the competition evaluation.

1 Introduction

The 2020 IWPT Shared Task on enhanced dependency parsing (Bouma et al., 2020) requires participants to predict the enhanced dependencies (DEPS column in the CoNLL-U format) in addition to sentence boundaries, tokenisation, lemmata, POS tags, morphological features and the basic dependency tree. We take a pipeline approach using

1. UDPipe for sentence splitting and tokenisation,

2. ensembles of UDPipe-future basic parsers, that also predict lemmata, POS tags and morphological features, with added support for

multi-treebank models (Stymne et al., 2018), and

3. two types of enhancers: *(a)* copying the basic tree and applying a small set of heuristics (baseline system), and *(b)* a graph-based semantic dependency parser (Dozat and Manning, 2018).

To enable reproduction of our results, we make available our helper scripts and modifications of the semantic parser.[1]

Our approach to the task does not guarantee a connected graph – something that we did not account for. Thus, on submission day, we did not have an appropriate solution ready to fix our outputs but were able to provide a valid submission due to some functionality that was added to the `quick-fix` tool provided by the organisers[2] to alter the enhanced graph. The solution was designed primarily to make the files pass validation but in doing so, harms F1-score. In a post-competition run, we addressed the connected graph issue with an alternative solution which increased our macro-averaged ELAS F1-score from 67.23 to 79.53 and the treebank average from 67.49 to 79.76.

2 System Components

2.1 Segmentation

We use UDPipe[3] (Straka and Straková, 2017) with off-the-shelf UD v2.5 models[4] (Straka and Straková, 2019) for the languages of the shared task to split the raw input text into sentences and tokens. In cases where more than one UDPipe model

---

[1] https://github.com/jbrry/Enhanced-UD-Parsing
[2] https://github.com/UniversalDependencies/tools
[3] http://ufal.mff.cuni.cz/udpipe
[4] http://hdl.handle.net/11234/1-3131

is available for a language, we try all models during development [5] and select for each test language the best overall pipeline according to ELAS on the treebank with the biggest development set for the language.[6]

2.2 Basic Parsing

We choose UDPipe-future (Straka, 2018) for basic parsing and joint prediction of lemmata, POS tags and morphological features so as to not require a separate tagger. We extend UDPipe-future to train multi-treebank models as introduced by (Stymne et al., 2018) with UUParser.[7,8]

Inspired by Straka et al. (2019), we use two types of external word embeddings with UDPipe-future: ELMo contextualised word embeddings (Peters et al., 2018) and FastText character-n-gram-based word embeddings (Bojanowski et al., 2017).[9] For 15 of the 17 test languages, ElmoForManyLangs[10] (Che et al., 2018) provides ELMo models. We train FastText on the raw text provided by the CoNLL'17 shared task for the same 15 languages after shuffling sentences. For the Russian FastText model, we kept getting vectors with large component values even after trying a different machine and a different permutation of sentences, prohibiting effective training of the parser. We then used a model trained on ⅔ of the Russian data for which component values and parser LAS were in the expected range. Furthermore, we train UDPipe-future models using FastText and internal embeddings only.

For Lithuanian and Tamil, we train UDPipe-future without external word embeddings. The parser still uses an internal word embedding covering all words of the training treebank(s) and a word representation obtained with a bidirectional GRU layer over the input characters.

For each target language, we train *(a)* mono-treebank models for each training treebank available with surface strings in UD v2.5, preferring the shared-task version when available, and *(b)* a multi-treebank model for each language using all treebanks for that language for which we also trained mono-treebank models. We train up to seven models with different initialisation for each setting to combine them in ensembles.[11,12] We consider ensembles not just of a single type of model with different initialisation but also combinations of models trained on different treebanks (mono-treebank models) or treebank combinations (multi-treebank models) and in the plain, FastText and ELMo variants.[13] As the number of possible combinations increases exponentially with the number of models, we prune the candidates giving preference to models using all or only one treebank and to models using ELMo. We then test each ensemble on the development data (raw input segmented with UD-Pipe) and pick the best ensembles based on ELAS after applying our heuristic enhancer (Section 2.3) to the basic trees.

To pick the proxy treebank (see description in Footnote 7) for multi-treebank parsing, we use the treebank name in the filename of the raw text during development. However, for final testing, the treebank identifier is unavailable (and if it had been available there would have been cases where

---

[5]Due to a configuration error, we did not try segmentation with UDPipe models trained on `fi_ftb`, `lt_hse` and `sv_lines` in the official submission.

[6]For Czech, we based our decision on results for `cs_cac` instead of `cs_pdt` as we did not have full results available for `cs_pdt`.

[7]Multi-treebank models supply each token with the source treebank ID as additional input with a separate embedding table. Like Stymne et al. (2018), we use a vector size of 12. At test time, a proxy treebank must be chosen when the input sentence does not come from one of the training treebanks or the source is unknown.

[8]https://github.com/jowagner/
UDPipe-Future/tree/multitreebank

[9]The FastText word embedding is restricted to a fixed vocabulary of one million tokens, not taking advantage of FastText's ability to produce new vectors for OOVs. UDPipe-future does not fine-tune these word embeddings. Instead, the parser trains an additional embedding exclusively for training words and a character-based representation. The latter two are added and the result is concatenated with the two externally provided representations. As far as we understand the code, an all-zero vector is used for OOVs, i. e. words not in the selected one-million-word FastText vocabulary.

[10]https://github.com/HIT-SCIR/
ELMoForManyLangs

[11]We trained 68 types of models. We trained seven seeds for 34 of these, five seeds for 30 and three seeds for four. Ensembles sizes three, five and seven are considered, including a combination of $(n+1)/2$ models of one type and $(n-1)/2$ models of another type with $n \in \{3, 5, 7\}$.

[12]We use our implementation https://github.com/
jowagner/ud-combination of the linear combiner of Attardi and Dell'Orletta (2009).

[13]While predicting on development data to facilitate model selection, we temporarily introduced a bug in our system causing it to use the first initialisation seed for all ensemble members only, effectively falling back to a single model when only one model type is used. We fixed this bug before we switched to making test set predictions and tried to account for it in the model selection but, under time pressure, made some hard to explain ad hoc choices, e. g. we used an ensemble of three models for Czech, two mono-treebank models trained on `cs_cac` and one multi-treebank model, even though we also had test set predictions with an ensemble of seven models with the same mixture of model types available. For details, see the reproducibility notes in our code repository.

this treebank is not one of the training treebanks). Given time limits, we decided to simply assign each test set, i. e. each test language, the training treebank with the largest amount of training data as the proxy treebank.[14]

2.3 Heuristic Enhancement

We build a baseline system which concentrates on the two enhanced UD phenomena which are very straightforward to implement using simple heuristic rules, namely, co-reference in relative clauses and modifier relations containing case markers. These rules are applied to the output of the basic parser. We have two versions of the modifier relation rule - one in which the value of the `case` morphological feature is included in the relation label and one without. We also have a rule which adds the lemma of a conjunction to the enhanced label of its head. For each development set, we find the optimal subset of the set of heuristic rules in terms of ELAS among all possible subsets except those combining the two case rules.

This baseline system is clearly suboptimal since it makes no attempt at all to handle those more interesting enhanced UD phenomena which involve the addition or deletion of arcs, i.e. conjunct propagation, ellipsis and control/raising constructions. Nonetheless it is useful as a baseline to check that the main system is performing reasonably and is available as a fall back.

2.4 Semantic Parsing

2.4.1 Modelling Enhanced Dependencies

As our main system to predict the enhanced graph, we follow (Dozat and Manning, 2018) and treat enhanced dependency parsing as a task similar to semantic dependency parsing. In semantic dependency parsing, words may have multiple heads. Thus, Dozat and Manning (2018) apply their deep biaffine graph-based dependency parser (Dozat and Manning, 2017) to the task of semantic dependency parsing but replace the softmax cross-entropy loss with sigmoid cross-entropy loss for edge prediction. The above modification changes the modelling objective such that words are no longer competing with one another to be classified as the appropriate head; rather, the parser chooses whether an edge

exists between each possible pair of words independently. Whether an edge exists between two words is based on a predefined threshold, where a score above this threshold results in an edge being predicted and, subsequently, the edge's label. In our experiments we use an edge prediction threshold of 0.5. If the parser did not predict an edge for a word, we take the edge with the highest probability. As we want to select the label with the highest probability for each chosen edge, standard softmax cross-entropy loss is used for label prediction as in Dozat and Manning (2018).

In order for the semantic dependency parser to be able to model relationships where a word may have multiple heads, we create an adjacency matrix where the ij^{th} entry in the matrix indicates whether an edge exists between tokens i and j with label type k. We also append the dummy *root* token to the adjacency matrix so that an edge can be predicted from the main predicate of the sentence to the dummy *root* token.
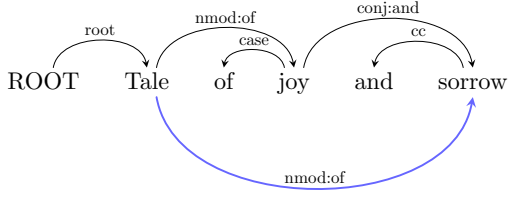
Figure 1a shows the enhanced UD graph for the phrase, *Tale of joy and sorrow*. In the enhanced representation, each conjunct in the conjoined noun phrase is attached to the governor of the modifier phrase, e.g. there is an additional *nmod* relation marked in blue between the noun *Tale* and the second conjunct *sorrow*. Note that the lemma of the *case* and *cc* dependents are appended to the enhanced dependency labels of their heads. The corresponding edge-existence probabilities of the semantic parser trained on `en_ewt` are shown in Figure 1b where the parser correctly predicts an edge from *sorrow* to the first conjunct *joy* as well as the head of the modifier phrase *Tale*.

2.4.2 Feature Representations

In our experiments, each word w_i in a sentence $S = (w_0, w_1, \ldots, w_N)$ is converted to its vector representation $\mathbf{x}_i$. We trained different variants of our semantic parser where $\mathbf{x}_i$ is the concatenation of different combinations of the below features:

- **BERT embedding**: The first word-piece embedding of the wordpiece-tokenised input word from BERT (Devlin et al., 2019) $\mathbf{e}_i^{(BERT)} \in \mathbb{R}^{768}$

- **character embedding**: A character embedding obtained by passing the k characters $ch_1, \ldots, ch_k$ of w_i through a BiLSTM: BiLSTM($ch_{1:k}$), $\mathbf{e}_i^{(ch)} \in \mathbb{R}^{64}$
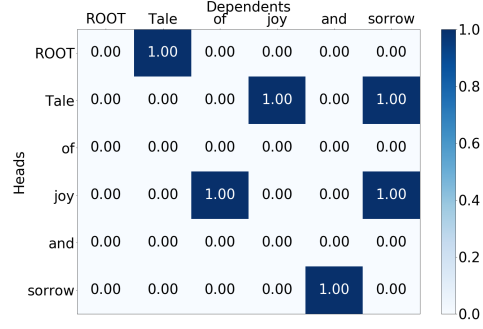
(a) Enhanced UD graph.

(b) Edge-existence probabilities.

Figure 1 The enhanced UD graph and edge-existence probabilities of the semantic parser trained on `en_ewt` for the phrase *Tale of joy and sorrow*.

- **lemma embedding**: The embedding of the word's lemma $\mathbf{e}_i^{(le)} \in \mathbb{R}^{50}$

- **UPOS embedding**: The embedding of the word's universal POS tag $\mathbf{e}_i^{(u)} \in \mathbb{R}^{50}$

- **XPOS embedding**: The embedding of the word's language-specific POS tag $\mathbf{e}_i^{(x)} \in \mathbb{R}^{50}$

- **morphological feature embedding**: The embedding of the word's morphological features $\mathbf{e}_i^{(f)} \in \mathbb{R}^{50}$

- **head-information embedding**: An embedding representing the word's head information from the basic tree $\mathbf{e}_i^{(h)} \in \mathbb{R}^{50}$

- **dependency label embedding**: The embedding of the word's dependency label from the basic tree $\mathbf{e}_i^{(label)} \in \mathbb{R}^{50}$

All model variants use the lexical information of the first BERT word-piece embedding and the character embedding, where ; represents vector concatenation:

$$\mathbf{e}_i^{(l)} = [\mathbf{e}_i^{(BERT)}; \mathbf{e}_i^{(ch)}] \qquad (1)$$

The subsequent variation comes from the other types of features used where we experimented with the below feature settings:

$$\mathbf{x}_i = [\mathbf{e}_i^{(l)}; \mathbf{e}_i^{(u)}] \qquad (2)$$

$$\mathbf{x}_i = [\mathbf{e}_i^{(l)}; \mathbf{e}_i^{(le)}; \mathbf{e}_i^{(u)}; \mathbf{e}_i^{(f)}] \qquad (3)$$

$$\mathbf{x}_i = [\mathbf{e}_i^{(l)}; \mathbf{e}_i^{(le)}; \mathbf{e}_i^{(u)}; \mathbf{e}_i^{(x)}; \mathbf{e}_i^{(f)}] \qquad (4)$$

$$\mathbf{x}_i = [\mathbf{e}_i^{(l)}; \mathbf{e}_i^{(le)}; \mathbf{e}_i^{(u)}; \mathbf{e}_i^{(f)}; \mathbf{e}_i^{(b)}] \qquad (5)$$

$$\mathbf{x}_i = [\mathbf{e}_i^{(l)}; \mathbf{e}_i^{(le)}; \mathbf{e}_i^{(u)}; \mathbf{e}_i^{(x)}; \mathbf{e}_i^{(f)}; \mathbf{e}_i^{(b)}] \qquad (6)$$

For the morphological features, there may be multiple morphological tags $m_1, \ldots, m_t$ for a particular word w_i. Thus, we split the full label into separate features (Hall et al., 2007) and embed each morphological property separately. We then sum the individual embedded representations together and divide by the number of active properties:

$$\mathbf{e}_i^{(f)} = \text{mean}(\mathbf{e}^{(m_{1:t})}) \qquad (7)$$

We follow the same process for the head-information embedding $\mathbf{e}_i^{(h)}$. Rather than encoding the head as an integer value, we obtain a *direction* value and a *distance* value: for each head-dependent pair (i, j), we subtract the indices of i, j giving the distance value. If the value is negative it means the head is to the left or if it is positive, to the right. We then take the absolute distance value and define ranges: *short* (1-4), *medium* (5-9), *far* (10-14) and *long-range* (>15). The qualitative direction (*left* or *right*) and distance labels are embedded in the same way as morphological features, e.g. embedded as separate components, summed together and then divided by the number of features (which in this case is always two):

$$\mathbf{e}_i^{(h)} = \text{mean}(\mathbf{e}^{(h_{1:t})}) \qquad (8)$$

To encode the basic tree, we then concatenate the head representation and the dependency label embedding:

$$\mathbf{e}_i^{(b)} = [\mathbf{e}_i^{(h)}; \mathbf{e}_i^{(label)}] \qquad (9)$$

It is worth mentioning that more sophisticated approaches for modelling head distance and direction exist for basic dependency parsing (Qi et al., 2018) but we leave using this approach for enhanced dependency parsing as future work.

2.4.3 Training Details

Our semantic parser predicts edges in a greedy fashion based on local decisions, i. e. we did not make use of any maximum spanning tree algorithm or enforce any global constraints. One property of enhanced dependency graphs is that the graph may contain cycles, therefore, we did not remove any cycles from the graph but observed that this sometimes causes fragments in the graph which are not reachable from the notional ROOT. For graphs with unreachable nodes, we applied our post-processor to attach these (Section 2.5).

We found that this architecture can be easily applied to enhanced dependency parsing given its similar nature to semantic dependency parsing. One caveat is that in enhanced dependency parsing, the label set can be quite large as modifier lemma and case information can be appended to the dependency label which results in very high memory requirements for certain languages such as Arabic. Additionally, modelling all enhanced labels in this fashion means that the parser is limited in its ability to predict labels for rare modifiers. An examination of the semantic parser output on the `en_ewt` development set shows that, although the parser often predicts the correct label, it can sometimes predict the wrong label containing a frequent modifier which is not in the sentence, e.g. *advcl:if* instead of *advcl:as*.

Our semantic parser is built upon the implementation in AllenNLP (Gardner et al., 2018). Due to time constraints, we trained our semantic parsing models on the gold training data released by the organisers as opposed to creating jack-knifed silver data. Hyperparameters are similar to those in Dozat and Manning (2017) as we found the larger network size of Dozat and Manning (2018) to be too restrictive for certain languages with high memory demands. Full hyperparameters of the semantic parser are given in Table 1. We trained for 75 epochs with early-stopping if the development score did not improve after 10 epochs.

Memory Considerations We trained our semantic parsing models on two GPUs: the first was an NVIDIA RTX 2080 Ti with 12GB of VRAM

Semantic Parser Details	
Parameter	**Value**
Char-BiLSTM layers	2
BiLSTM layers	3
BiLSTM size	400
Char-BiLSTM size	64
Arc MLP size	500
Label MLP size	100
Dropout LSTMs	0.33
Dropout MLP	0.33
Dropout embeddings	0.33
Nonlinear act. (MLP)	ELU
Edge prediction threshold	0.5
BERT word-piece embedding	768
Char embedding	64
Tag embedding (all tags)	50
Optimizer	Adam
Learning rate	0.001
beta1	0.9
beta2	0.9
Num. epochs	75
Patience	10
Batch size	16

Table 1 Chosen hyperparameters for our semantic parser. For the tag embedding, we use the same size embedding for all features (lemma, POS, morphological features, head-information and label embeddings) and concatenate them.

where we had to remove very long sentences ($< .03\%$ of sentences overall) from the treebanks: `cs_cac`, `cs_pdt`, `it_isdt`, `ru_syntagrus` and `sv_talbanken` in order to fit a batch into memory. We were also given access to an NVIDIA V100 GPU with 32GB of VRAM which enabled us to process all treebanks except for `ar_padt` without removing long sentences. For `ar_padt`, after removing the longest 75 sentences, the model still required 29GB of VRAM.

2.4.4 BERT Models

For the BERT models, in early development runs we compared multilingual BERT (mBERT) with a language-specific BERT model if there was one available in HuggingFace's (Wolf et al., 2019) models repository.[15] We used a language-specific BERT model for `ar` (Safaya et al., 2020), `bg+cs` (Arkhipov et al., 2019), `en` (Devlin et al., 2019),

---

[15]https://huggingface.co/models

fi (Virtanen et al., 2019), it[16], nl (de Vries et al., 2019), pl[17], ru (Kuratov and Arkhipov, 2019) and sv[18] and for the rest of the languages we used mBERT (Devlin et al., 2019). We found that the language-specific variant was always better than mBERT except for pl_lfg. For fr_sequoia, we tried using the CamemBERT model (Martin et al., 2020). As this model uses RoBERTA (Liu et al., 2019) as opposed to BERT, we installed AllenNLP from the master repository which uses HuggingFace's `AutoTokenizer` module which supports many BERT-like models. We noticed a trend of lower results when using the master branch for some languages but training was also more stable for certain treebanks where we had previously encountered a `nan` in the loss.[19] Consequently, we include models from the stable release and the bleeding-edge master branch in our development pipeline.

2.5 Connecting the Graph

We had no solution ready to connect fragmented graphs produced by our semantic parser[20] on the system submission day and resorted to using the "connect-to-root" option of the `quick-fix` tool provided by the shared task organisers, who warned that it had not been thoroughly tested.

After the system submission deadline, we investigated the fragmentation issue. The task is to make all nodes reachable from the notional ROOT[21], where reachability is directional. Adding more edges than necessary harms precision and thus F1-score. We found that the `quick-fix` tool with the "connect-to-root" option adds edges to every unreachable node. We also noticed a bug in the implementation where certain reachable nodes were being reported as unreachable.

We then implemented an improved tool to connect fragmented graphs trying to minimise the number of edges added to the graph. We repeatedly

---

[19]We incurred a `nan` loss for cs_cac, cs_pdt, it_isdt and ru_syntagrus using the AllenNLP stable branch 0.9.0 and used the best model from the available epochs.

[20]Between 90.18% (Lithuanian) and 99.51% (Russian) of test sentences in our official submission are not affected, i.e. all nodes are reachable from a root node. This observation excludes Estonian, for which we submitted predictions using our heuristic system.

[21]UD distinguished between the notional ROOT (ID 0) and root nodes. The latter are any nodes that have '0' as a head.

| | ELAS F1 | |
Treebank	sem-frag	heuristic
ar_padt	**70.99**	59.74
bg_btb	**88.09**	86.19
cs_cac	**86.51**	74.41
cs_fictree	**83.23**	77.37
cs_pdt	**79.58**	71.19
en_ewt	**84.71**	82.86
et_edt	62.74	**69.35**
fi_tdt	**83.64**	71.84
fr_sequoia	**88.65**	87.53
it_isdt	**90.13**	88.28
lt_alksnis	**73.63**	57.84
lv_lvtb	**81.82**	71.29
nl_alpino	**89.93**	89.00
nl_lassysmall	79.00	**81.24**
pl_lfg	**94.12**	93.84
pl_pdb	**82.25**	78.27
ru_syntagrus	**88.48**	80.03
sk_snk	**81.30**	75.98
sv_talbanken	**84.54**	81.32
ta_ttb	**55.68**	43.94
uk_iu	**82.41**	76.88

Table 2 Development set ELAS F1 score for the best semantic parser evaluated without connecting fragmented graphs (sem-frag) and for the best combination of heuristic rules (heuristic)

check for each unreachable node how many unreachable nodes can be reached from it. Among the nodes that maximise this number we pick the first node in surface order and make it a child of the notional ROOT, i.e. it becomes an additional root node. This is a rather naive approach which does not try to connect fragments in a sensible manner but, rather, mimics the behaviour of the "connect-to-root" option. Future work could try to show whether our above algorithm adds the minimal number of edges necessary to connect the graph or if a lower optimum exists.

3 Results

Table 2 compares the semantic parser against the heuristic approach on the ELAS F1 metric. The evaluation script was run without connecting fragmented graphs and format validation. For all but two treebanks, the semantic parser performs better than the best heuristic approach. For some lan-

| | ELAS F1 | | |
Treebank	subm	frag fix	re-run
Arabic-PADT	57.19	70.08	70.40
Bulgarian-BTB	77.29	89.58	89.60
Czech-FicTree	70.04	80.77	81.63
Czech-CAC	71.72	86.00	86.38
Czech-PDT	65.94	79.03	79.84
Czech-PUD	64.34	77.37	78.08
Dutch-Alpino	71.44	87.61	87.77
Dutch-L.Small	64.03	77.39	77.24
English-EWT	70.61	83.56	83.56
English-PUD	70.25	86.88	87.03
Estonian-EDT	62.29	68.20	68.37
Estonian-EWT	55.70	61.19	60.67
Finnish-TDT	73.02	84.36	84.33
Finnish-PUD	71.58	84.62	84.62
French-Sequoia	77.44	87.58	88.60
French-FQB	74.30	82.68	83.26
Italian-ISDT	71.98	90.24	90.23
Latvian-LVTB	72.41	81.81	82.40
Lithuanian-AL.	58.36	68.76	68.84
Polish-LFG	61.23	70.89	70.71
Polish-PDB	67.68	80.93	82.43
Polish-PUD	65.64	79.77	80.79
Russian-SynT.	75.27	89.21	89.47
Slovak-SNK	68.43	81.63	81.97
Swedish-Talb.	71.86	86.78	86.72
Swedish-PUD	64.70	79.35	79.37
Tamil-TTB	48.47	57.28	57.10
Ukrainian-IU	66.43	79.81	82.92
Average	67.49	79.76	80.15

Table 3 Test set results: subm = submitted, frag fix = using our own fragment connector and quick-fix.pl without connect-to-root, re-run = a re-run with bug fixes, no new models but new model selection

guages, the difference in performance is large. For et_ewt, which does not have a development set, we suspect that we overfitted our semantic parser on the et_ewt training data by allowing it to train for 75 epochs.

Table 3 shows test set ELAS obtained on the shared task submission site for *(a)* our submission fully relying on the organiser's quick-fix tool to fix issues in the output of our system, *(b)* the same predictions post-processed by our own fragment connector that aims to minimise the number of root edges added, and *(c)* a re-run of our pipeline using the same models for system components as before but with all bugs fixed during development applied to all predictions and new decisions which models to apply to the test sets. While the quick-fix tool enabled us to make a valid submission in time, its approach of adding edges from the root node to all unreachable tokens has a strong negative impact on precision, e. g. 62.26 ELAS precision on the Czech CAC development set vs. 87.37 without post-processing. Our own post-competition fix avoids this and would have brought us to the top half of the competition.

4 Conclusion

In this system submission, we use a graph-based semantic parser to parse enhanced dependencies and compare to a baseline in which we create enhanced graphs from the basic tree using a very limited set of heuristics. Avenues for future work include:

Post-processing Predict the head and label for edges connecting fragments (as opposed to a dummy "0:root" edge) where this information could come from new edges available from lowering the score threshold or from the basic tree.

Label Prediction The semantic parser performs competitively despite treating enhanced dependency labels containing lemmas and case information as atomic units. However, a more sophisticated approach should still be tried.

Multi-treebank Parsing When randomising the proxy treebank for multi-treebank models, use a different randomisation for each ensemble member. Predict the best proxy treebank for each test sentence or paragraph (Wagner et al., 2020).

Elided Tokens Our semantic parser handles elided tokens by appending the elided token to the adjacency matrix and offsetting the head indices. While we used this approach during training on gold data, we did not predict elided tokens and we wish to explore methods for doing so.

Acknowledgments

References

Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. 2019. Tuning multilingual transformers for language-specific named entity recognition. In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy. Association for Computational Linguistics.

Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264, Boulder, Colorado. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2020. Overview of the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, Seattle, US. Association for Computational Linguistics.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülşen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? a study in multilingual parser optimization. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 933–939, Prague, Czech Republic. Association for Computational Linguistics.

Yuri Kuratov and Mikhail Arkhipov. 2019. Adaptation of deep bidirectional multilingual transformers for Russian language. ArXiv 1905.07213.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. ArXiv 1907.11692.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. CamemBERT: a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. To appear. Also available as ArXiv 1911.03894.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium. Association for Computational Linguistics.

Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. Kuisail at semeval-2020 task 12: BERT-CNN for offensive speech identification in social media. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*. To appear.

Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw*

Text to Universal Dependencies, pages 197–207, Brussels, Belgium. Association for Computational Linguistics.

Milan Straka and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Milan Straka and Jana Straková. 2019. Universal dependencies 2.5 models for UDPipe (2019-12-06). LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Milan Straka, Jana Straková, and Jan Hajič. 2019. Evaluating contextualized embeddings on 54 languages in POS tagging, lemmatization and dependency parsing. ArXiv 1908.07448.

Sara Stymne, Miryam de Lhoneux, Aaron Smith, and Joakim Nivre. 2018. Parser training with heterogeneous treebanks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 619–625. Association for Computational Linguistics.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: BERT for Finnish. ArXiv 1912.07076.

Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. 2019. BERTje: A Dutch BERT model. Arxiv 1912.09582.

Joachim Wagner, James Barry, and Jennifer Foster. 2020. Treebank embedding vectors for out-of-domain dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. To appear. Also available as ArXiv 2005.00800.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's transformers: State-of-the-art natural language processing. ArXiv 1910.03771.