Computational Linguistics and Intellectual Technologies:
Proceedings of the International Conference “Dialogue 2020”

Moscow, June 17–20, 2020

THE RUSSIAN LANGUAGE PIPELINE IN THE LIMA MULTILINGUAL ANALYZER

Bocharov V. V. (victor.bocharov@cea.fr),
de Chalendar G. (gael.de-chalendar@cea.fr)
CEA LIST, LASTI, Gif-sur-Yvette, France

In this paper we describe the implementation of Russian language pipeline in LIMA multilingual analyzer and the results obtained in GramEval-2020 shared task. LIMA is a modular pipeline that implements rule-based and machine learning analysis components. Russian language pipeline includes deep neural networks based modules for tokenization, sentence segmentation, part of speech tagging, lemmatization and dependency parsing. Part of speech tags, feature tags and dependency trees conform to Universal Dependencies rules.

Key words: tokenization, part of speech tagging, dependency parsing, lemmatization

DOI: 10.28995/2075-7182-2020-19-93-105

ОБРАБОТКА РУССКОГО ЯЗЫКА В МНОГОЯЗЫЧНОМ АНАЛИЗАТОРЕ LIMA

Бочаров В. В. (victor.bocharov@cea.fr),
де Шаландар Г. (gael.de-chalendar@cea.fr)
CEA LIST, LASTI, Жиф-сюр-Иветт, Франция

В этой статье описана реализация обработки текста на русском языке в анализаторе LIMA и наше участие в соревновании GramEval-2020. Анализатор LIMA—это модульная система обработки текста, включающая статистические и основанные на правилах компоненты. Обработка текста на русском языке реализована при помощи статистических моделей на основе глубоких нейронных сетей и включает токенизацию, морфологический анализ, лемматизацию и построение деревьев зависимостей. Морфологический и синтаксический анализ соответствуют правилам Universal Dependencies.

Ключевые слова: токенизация, морфологический анализ, синтаксический анализ, лемматизация

1. Introduction

The implementation of Russian language pipeline in LIMA is a part of our work on expanding the range of supported languages using machine-learning techniques and Universal Dependencies [16] corpora. LIMA is a modular multilingual toolkit that includes a language agnostic core and a number of analysis modules sharing a common internal representation of text analysis. Our Russian language support is based on Universal Dependencies annotation of the Russian-SynTagRus corpus, recent deep neural networks models and fastText word embeddings. It includes a tokenizer (which does both word and sentence splitting), a morphological analyzer combined with a dependency parser and a lemmatizer. For our participation in GramEval-2020 shared task, we trained another model using annotated corpora supplied by the organizers.

Source code of LIMA including all mentioned components is available on GitHub[1]. Trained models are published in the form of Debian packages[2]. In the following sections, we will first describe related work and then each module in detail. We continue with the evaluation of our results in the shared task before concluding.

2. Related Work

2.1. Universal Dependencies

Universal Dependencies (UD) is an international project and a multilingual annotation framework that provides a universal inventory of linguistic categories and annotations guidelines covering tokenization, part of speech and features tagging and dependency parsing. Within the UD project, a cross-linguistically consistent treebank annotation for many languages is created. A new version of Universal Dependencies treebank collection is released twice a year. Current version UD 2.5 includes 157 treebanks for 90 languages.

There exists a wide range of software[3] (editor, visualising tools, consistency checkers and libraries) that works with Universal Dependencies annotation. UDPipe (see below) is a widely known parsing pipeline that produces output following UD guidelines for many languages.

2.2. NLP pipelines and toolkits

There are many known pipeline-based natural language processing systems. GATE (general architecture for text engineering) [4] is an open-source software toolkit originally developed at the University of Sheffield in 1995. GATE includes many analysis modules (processing resources), graphical environment and an information extraction system called ANNIE (A Nearly-New Information Extraction System).

[1] https://github.com/aymara/lima

[2] https://github.com/aymara/lima-models

[3] https://universaldependencies.org/tools.html

UIMA (Unstructured Information Management Architecture) [6] is an OASIS[4] standard for content analytics developed at IBM, and Apache UIMA is an open-source implementation of this standard. DKPro (The Darmstadt Knowledge Processing Software Repository) is a collection of software components for natural language processing based on the Apache UIMA framework.

Both GATE and UIMA provide pipeline-based frameworks and analysis modules. Within GATE modules are mostly Java-developed. Apache UIMA provides both Java and C++ frameworks and annotators can be written in Java, C++, Perl, Python and TCL.

Apache OpenNLP[5] is a machine learning library that provides analysis components for many NLP tasks: language detection, text segmentation, part of speech tagging, named entity extraction, parsing and coreference resolution. It is also a Java-based toolkit initially released in 2004.

NLTK (Natural Language Toolkit)[6] [1] is a set of Python libraries for solving natural language processing tasks. In addition to analysis modules, NLTK includes also corpora and lexical resources available through the same installer.

spaCy[7] is another open-source Python library offering software components for text analysis. spaCy is partially implemented using Cython[8] and authors claim that their main focus is to provide an industrial tool that is capable to operate at large scale. AllenNLP[9] is a framework for deep learning NLP created on top of spaCy and PyTorch machine learning library.

UDPipe [20] is an open-source tool that implements NLP tasks required to reproduce Universal Dependencies 2.0 annotations: tokenization, sentence segmentation, POS tagging, lemmatization and dependency parsing. UDPipe provides both training and annotation functionality. Training part uses only Universal Dependencies annotation without any supplementary data. UDPipe is written in C++ and bindings for Python, Perl, Java and C# are provided. Several other tools able to analyze Universal Dependencies corpora has participated to CoNLL 2017 and 2018 shared task entitled *Multilingual Parsing from Raw Text to Universal Dependencies* [8], [21]. UDPipe Future is the next version of UDPipe that is under development now.

UDify [12] is a single model for analysis of 75 languages with BERT-based encoder. It uses cased BERT-Base multilingual model pretrained on Wikipedia dumps for 104 languages[10]. Original research paper describes different fine-tuning strategies and their effect on high-resource and low-resource languages.

DeepPavlov is an open source framework for chatbots and virtual assistants development. It includes dependency parsing module with RuBERT-based encoder [13].

---

4 https://www.oasis-open.org/

5 http://opennlp.apache.org/

6 https://www.nltk.org/

7 https://spacy.io/

8 Cython—a compiled language that offers better performance and memory management for Python-like code.

9 https://allennlp.org/

10 https://github.com/google-research/bert/blob/master/multilingual.md

RuBERT is a monolingual BERT trained on the Russian part of Wikipedia and news data. Initial weights of RuBERT were initialized with Google's multilingual BERT.

2.3. LIMA

LIMA [2] is a C++ toolkit and a pipeline-based analysis framework developed by LASTI laboratory of CEA LIST. It was designed and developed with several objectives:

- multilingualism—an ability to work with a broad spectrum of languages;
- diversity of use-cases—LIMA must be useful as a basic component for various text-processing applications such as question-answering systems, automatic summarization, etc.;
- extensibility—an architecture that makes it possible to easily add new functionality or replace the implementation of existing components;
- efficiency—LIMA must be able to process large corpora and work in an industrial context.

LIMA consists of core components defining graph-based language-independent representation of entities like linguistic analysis and processing module and a collection of modules providing several types of functionality including:

- input/output modules: source text readers and analysis writers in various formats,
- dictionary-based and OOV-words annotators,
- POS tags disambiguators,
- named-entities taggers,
- rule-based syntactic analyzers.

Processing modules are executed sequentially and the order of execution is defined by pipeline configuration. All modules have access to a shared graph-based analysis representation. Input-related modules create this representation. Next modules update and enrich analysis and output modules dump generated data into target file format. A shared analysis representation allows modules to be interchangeable to the extent that this is limited by natural dependencies between analysis steps.

Previously, LIMA had rule-based and statistical analysis components implemented for three languages (English, French, and Portuguese) under a free licence and for some other languages commercialy only (German, Spanish, Mandarin Chinese and Arabic) plus some experiments (Russian, Japanese, Czech...). And although the same internal representation of the analysis was built for each of the mentioned languages, the sets of tags for morphological and syntactic categories were defined separately. To support more languages we followed Universal Dependencies 2.0 tagset for parts of speech, features and syntactic dependency labels and developed fully machine learning-based modules.

3. Analysis modules based on deep neural networks

The pipeline approach in the architecture of text analyzers together with rule-based implementations of analysis components offer high flexibility in configuration and explainability of analysis results. At the same time, the development of these rule-based modules is expensive as it involves a lot of human labor. Moreover, most of the rule-based analysis components are difficult to port from one language or domain to another. Machine learning methods are easier in portability and with recent progress in deep neural network architectures, they surpass many rule-based methods in analysis quality. Below we describe our deep neural networks based modules.

3.1. Tokenizer

For token and sentence segmentation we adapt the character labeling approach proposed in Universal segmenter [18]. It is based on bidirectional recurrent neural networks with conditional random fields (BiRNN-CRF) and Viterbi decoder (Figure 2). The tagset consists of token segmentation tags (B—begin of token, I—inside token, E—last character of token, S—single-character token, X—outside of token) and sentence segmentation tags (T—last single character token in the sentence, U—last character of the last token in the sentence).

"	А		я		д	е	р	е	в	о		о	б	н	а	р	у	ж	и	л	!	"		-		"	А		я		с	о	б	а	к	у	!	"
S	S	X	S	X	B	I	I	I	I	E	X	B	I	I	I	I	I	I	I	E	S	T	X	S	X	S	S	X	S	X	B	I	I	I	I	E	S	T

Figure 1: Characters tagging schema

Three concatenated embeddings are given on each RNN step:

- the embedding of the unigram (current character only);
- the embedding of the bigram including previous character and current one;
- the embedding of the trigram that includes previous, current and next characters.

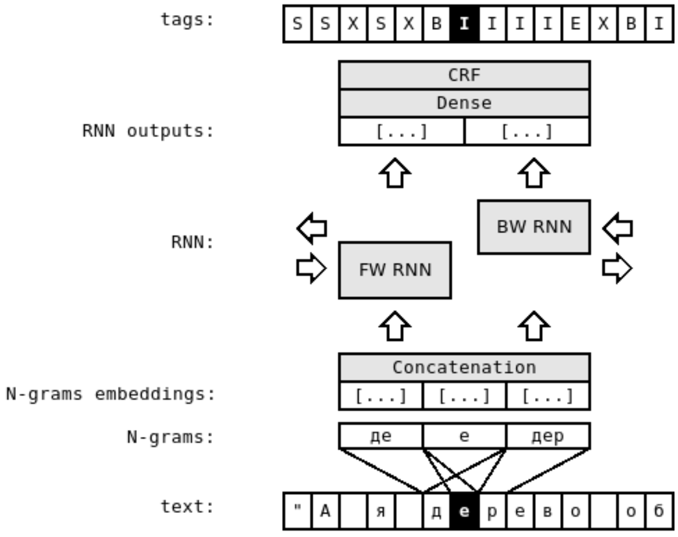

Figure 2: Tokenizer neural network

The dimension of each embedding is calculated at training time as the fourth-degree root of the number of different n-grams of given length found in the training set. This makes the model smaller for most of languages without significant quality reduction.

Table 1: N-gram embeddings dimensions for model trained on Russian-SynTagRus

	Min. frequency	Number of ngrams	Dimension
Unigram	3	153	4
Bigram	4	3,217	8
Trigram	10	12,653	12
		Total	24

3.2. Morphological Tagger and Dependency Parser

The morphological tagger assigns part of speech tags and feature tags for each word in the sentence. For this purpose, we use a similar sequence labeling approach as described above for tokenization. As soon as there are many different types of tags (part of speech tags, number, gender, case, etc), a dedicated classifier is required for each type. We use single BiRNN input for all types of tags with different CRF outputs for each classifier. CRF outputs for taggers are connected to a second BiRNN layer. Remaining layers are used by the dependency parser only.

For dependency parsing, we adapted graph-based parser [11] with deep biaffine attention arcs scoring method [5]. Arc scorer is attached on top of the concatenation of the output of the same BiRNN that is used for morphological tagging and dedicated BiRNN that is used for dependency parsing only. All these tasks (i.e. morphological tagging and dependency parsing) are trained simultaneously.

The BiRNN that is shared by taggers and parser has an input that consists of pre-trained word embeddings for all words and trained word embeddings for frequent words. The sum of word embeddings of these two types is concatenated with the final state of character-level RNN for the corresponding word.

FastText word embeddings with subword information are used as pre-trained word embeddings. In fastText, model word vectors are calculated as an element-wise average of word vector and vectors of all its subwords. In case of out-of-vocabulary words the first element isn't available and subwords vectors only are used [3]. The choice of fastText instead of word2vec [15] or Glove [17] is made for two reasons: fastText provides pretrained models [7] for most languages available in the Universal Dependencies collection, and subword information gives meaningful word vectors even for OOV-words.

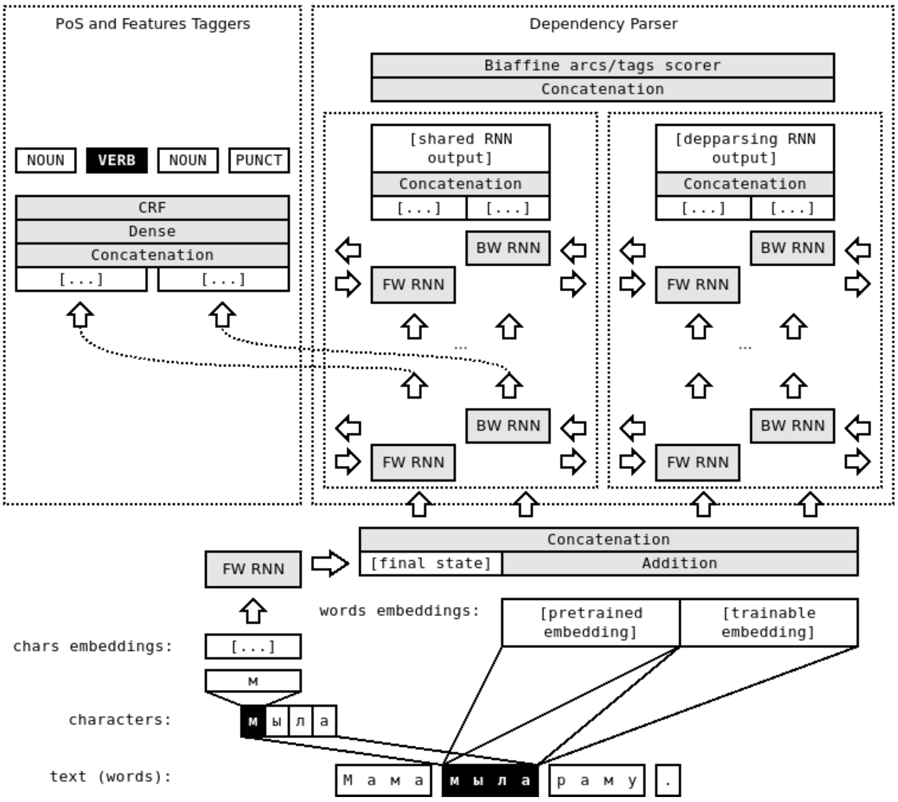

Figure 3: Taggers and dependency parser neural network

3.3. Lemmatizer

The lemmatizer uses the source form of the word and morphologic tags (part of speech and features tags) predicted on the previous step. The lemmatization task is treated as a sequence-to-sequence translation problem at the character level. Our approach is similar to the one adopted in Turku Neural Parser Pipeline [10]: surface word form together with predicted tags is given as input. Instead of encoding tags in form of strings we add them as embeddings of tags to the encoder as initial state. Seq2seq neural network with Bahdanau attention is used (**Figure 4**).

Lemmatization as it is described above is a context-independent task: the neighboring words aren't used to predcit lemma. This makes it possible for the lemmatizer to use the dictionary of pre-lemmatized words (lemmatizer cache) to improve the analysis speed. The lemmatizer cache generated from Russian-SynTagRus training set increases test set lemmatization speed by 2.5 times.

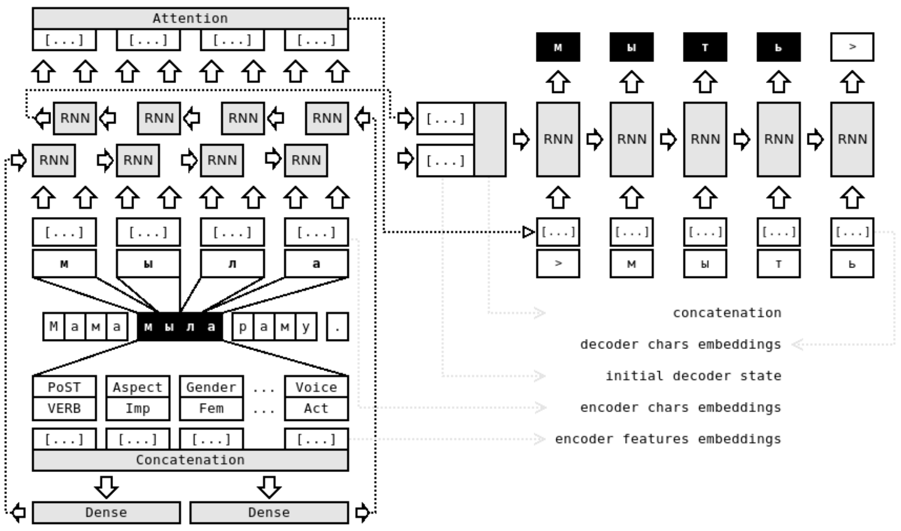

Figure 4: Lemmatizer neural network

4. Embedding Compression

The model size is an important practical issue that can limit the usage of the software on low-memory devices. The largest part of the model described above is a fastText embedding file that takes between 2.5 and 7.3 Gb depending on the language. Tokenizer, morphological tagger, dependency parser, and lemmatizer together take no more than 2% of the total model size. Thus the most important step toward memory footprint reduction is the compression of embeddings.

FastText embeddings file consists of four parts:

- word embedding table;
- n-grams embedding table;
- output table;
- dictionary.

The output table isn't used to calculate word embeddings and can be discarded. With remaining parts following reduction strategies are possible: dimensionality reduction, pruning and quantizing. Within the scope of this work, we have tried word and n-grams embedding tables pruning and quantizing [9].

In the case of pruning, we have removed 50% of the least frequent words and 50% of the least frequent n-grams. By default, both embedding tables and the dictionary have 2,000,000 entries each (i.e. 4,000,000 embeddings in total) and the resulting file contains 1,000,000 most frequent words and 1,000,000 most frequent n-grams.

For quantization, we used product quantization from the fastText library that consists of the approximation of real-valued vectors by the closest vector in a pre-defined set of centroids. This implementation splits each vector into several sub-vectors (subquantizers) and maps each one to some pre-defined point. The number of these pre-defined points and the number of subquantizers are two parameters. The first one is fixed to 256 possible values (8 bits per centroid index) and the second one is variable. With $k = 1$ each real-valued element (32 bit float) of source vector is replaced by 1 byte integer value giving 4:1 compression rate. With $k = 2$ each pair of float32 values is mapped to 1 byte integer giving 8:1 compression rate. This way the embeddings file can be compressed while all entries in words and n-grams tables are preserved.

In Table 2 we compare the effect of mentioned above compression techniques on analysis metrics. The possible combination of pruning and quantization remains for future investigation. The models we distribute include embeddings compressed with a 8:1 ratio. Embeddings with a 4:1 ratio are published separately[11] due to hosting size restrictions.

Table 2: The degradation of the analysis evaluation metrics with embeddings compression

	File size	UPOS	UAS	LAS
original	6.9Gb	98.34	91.87	90.20
original w/o output table	4.6Gb	98.34	91.87	90.20
50% pruning	2.7Gb	98.33 (−0.01)	91.86 (−0.01)	90.19 (−0.01)
4:1 quantization	1.2Gb	98.33 (−0.01)	91.85 (−0.02)	90.17 (−0.03)
8:1 quantization	0.6Gb	98.31 (−0.03)	91.82 (−0.05)	90.10 (−0.10)

5. Evaluation

We compare LIMA performance on Russian-SynTagRus corpus using the official CoNLL 2018 evaluation script and on GramEval-2020 [14] corpus using its official evaluation script. The use of the evaluation script from CoNLL 2018 competition is motivated by the intention to compare our results with previous works.

Both scripts evaluate part of speech tags, morphologic features, lemmatisation and syntax. Tokenization and sentence segmentation were not included into GramEval-2020 and this functionality of the analyser is evaluated only with CoNLL 2018 script on Russian-SynTagRus.

5.1. Evaluation on the Russian-SynTagRus corpus

LIMA has been evaluated in two settings: with the full analysis pipeline and with the gold tokenization and sentence segmentation. The metrics (F1 scores) are provided in Table 3. For comparison, we included the evaluation results of the version 1.2.0 of UDPipe (CoNLL 2018 baseline) and the results of CoNLL 2018 participants which obtained the best result on one of the evaluation metrics[12].

[11] https://zenodo.org/record/3629537

[12] CoNLL 2018 results are given according to official site. UDPipe Future evaluation metrics with gold segmentation are given according to UDify paper.

LIMA significantly outperforms UDPipe v1.2.0 in morphologic and syntactic tasks in all measurement settings but stays behind the best results of CoNLL 2018. Syntactic metrics (UAS and LAS) depends more on the quality of segmentation than part-of-speech, morphologic features and lemmata. BERT-based systems (UDify and DeepPavlov) are significantly better in syntactic metrics than all other systems.

Table 3: Performance metrics (F1) for Russian-SynTagRus corpus

	Tokens	Sentences	UPOS	UFeats	Lemmas	UAS	LAS
LIMA	99.5	98.6	**98.3**	**96.1**	96.1	**91.8**	**90.2**
UDPipe v1.2.0	**99.6**	**98.8**	97.8	93.5	**96.5**	87.6	85.0
CoNLL 2018 shared task results							
HIT-SCIR	99.6	98.0	98.6	93.6	95.5	**93.9**	**92.5**
NLP-Cube	**99.7**	98.8	98.4	96.2	92.3	92.7	90.9
Stanford	99.6	**98.9**	98.3	95.8	97.0	93.1	91.6
Turku NLP	99.6	98.0	98.0	96.6	**98.2**	93.2	91.7
UDPipe Future	99.6	98.6	**98.7**	**97.2**	97.9	93.0	91.5
Results with gold segmentation							
DeepPavlov			97.6	95.7		**95.2**	**93.7**
LIMA			98.8	96.5	94.7	92.7	91.0
UDPipe v1.2.0			98.2	93.9	96.9	88.3	85.7
UDPipe Future			**99.1**	**97.6**	**98.5**	93.8	92.3
UDify+Lang			**99.1**	97.2	96.6	**95.1**	**93.7**
Results with gold segmentation and morphology							
UDPipe v1.2.0						90.3	89.0

5.2. Evaluation on the GramEval-2020 corpus

For GramEval-2020 evaluation (Table 4), gold tokenization and sentence segmentation were given. Official evaluation scripts provide accuracy metrics for part-of-speech, morphologic features, lemmata and LAS. Overall score is an average of the four metrics mentioned above.

On GramEval-2020, metrics aren't directly comparable with metrics mentioned in previous section due to different method of calculation. Nevertheless the situation is similar: BERT-based systems (ADVance and qbic) are still better on GramEval-2020.

Table 4: Performance metrics (accuracy) for the GramEval-2020 corpus

	Overall	PoS	Feats	Lemmas	LAS
Baseline	80.4	91.0	89.6	86.4	54.5
Vocative	85.2	92.8	89.8	88.5	69.6
Lima	87.9	95.1	95.3	88.2	73.0
ADVance	90.8	95.4	95.8	93.1	78.8
qbic	91.6	95.9	96.0	93.4	81.3

5.3. Analysis speed

The analysis speed is an important characteristics of parser and it directly influences its prcatical applicability. There are many factors that influence analysis speed, including: analysis method, model size, underlying computation library, hardware used, compilation options etc. In the table below we provide the analysis speed comparison of several parsers. All the evaluations were performed on laptop with Intel i7-8650U CPU (4 cores, 8 threads) and 32Gb of RAM except UDPipe Future. Figures for UDPipe Future are taken from corresponding article [19]. Here are some important details regarding these measurements:

- spaCy and AllenNLP have no models for Russian language. We used models for English.
- spaCy and UDPipe run in single-thread mode. To make the comparison more meaningful we started the same analysis 8 times in parallel.
- Only dependency parsing[13] without morphologic features have been tested for DeepPavlov.
- Although CPU we used for these experiments is capable to run up to 8 threads simultaneously, not all analyzers used all of them all the time.
- UDPipe v1.2.0 and TensorFlow library used by LIMA were compiled with "-march = native" gcc compilation option. All other software have been installed from official repositories without compilation.

Table 5: Analysis speed comparison on CPU

	Method	Threads	Speed (tok/sec)
spaCy (en_core_web_sm)	transition-based	signle	2,728
spaCy (en_core_web_sm)	transition-based	multi	9,043
UDPipe v1.2.0	transition-based	single	3,000
UDPipe v1.2.0	transition-based	multi	12,000
AllenNLP (English)	RNN + graph-based	multi	250
LIMA (with cached lemmata)	RNN + graph-based	multi	430
LIMA (w/o cached lemmata)	RNN + graph-based	multi	327
UDPipe Future	RNN + graph-based	multi	517
DeepPavlov	BERT + graph-based	multi	166
UDify	BERT + graph-based	multi	108

From these data the difference in analysis speed between transition-based and graph-based parsers is clearly seen. LIMA shows an average speed comparable to other implementations of the same method.

6. Conclusion and future works

Our first implementation of neural network-based modules for LIMA obtains results at the level of the best systems that participated to CoNLL-2018. Current results

[13] The model name is "syntax_ru_syntagrus_bert".

in GramEval-2020 show that transformers-based models are necessary to reach today's state of the art. Anyway, LIMA is readily available for users, easily installable using simple packages.

Our future work include using BERT-like models to reach state of the art performance but also working on analysis speed to make LIMA neural network modules usable in production settings.

References

1. *Bird, S. et al.:* Natural language processing with python. (2009).
2. *Besançon, R. et al.:* LIMA: A multilingual framework for linguistic analysis and linguistic resources development and evaluation. Presented at the May (2010).
3. *Bojanowski, P. et al.:* Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606. (2016).
4. *Cunningham, H. et al.:* GATE: A framework and graphical development environment for robust nlp tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). (2002).
5. *Dozat, T., Manning, C. D.:* Deep biaffine attention for neural dependency parsing. ArXiv. abs/1611.01734, (2016).
6. *Ferrucci, D. et al.:* Unstructured information management architecture (UIMA) version 1.0, https://docs.oasis-open.org/uima/v1.0/uima-v1.0.html, (2009).
7. *Grave, E. et al.:* Learning word vectors for 157 languages. In: Proceedings of the international conference on language resources and evaluation (lrec 2018). (2018).
8. *Hajič, J., Zeman, D. eds:* Proceedings of the conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. Association for Computational Linguistics, Vancouver, Canada (2017).
9. *Joulin, A. et al.:* FastText.zip: Compressing text classification models. arXiv preprint arXiv:1612.03651. (2016).
10. *Kanerva, J. et al.:* Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task. In: Proceedings of the CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. pp. 133–142 Association for Computational Linguistics, Brussels, Belgium (2018).
11. *Kiperwasser, E., Goldberg, Y.:* Simple and accurate dependency parsing using bidirectional lstm feature representations. Transactions of the Association for Computational Linguistics. 4, 313–327 (2016).
12. *Kondratyuk, D., Straka, M.:* 75 languages, 1 model: Parsing universal dependencies universally. In: Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp). pp. 2779–2795 Association for Computational Linguistics, Hong Kong, China (2019).
13. *Kuratov, Y., Arkhipov, M.:* Adaptation of deep bidirectional multilingual transformers for russian language, (2019).

14. *Lyashevskaya, O. et al.:* GramEval 2020 Shared Task: Russian Full Morphology and Dependency Parsing. In: Computational linguistics and intellectual technologies: Papers from the annual conference "dialogue". (2020).
15. *Mikolov, T. et al.:* Efficient estimation of word representations in vector space. CoRR. abs/1301.3781, (2013).
16. *Nivre, J. et al.:* Universal dependencies v1: A multilingual treebank collection. In: Proceedings of the tenth international conference on language resources and evaluation (LREC'16). pp. 1659–1666 European Language Resources Association (ELRA), Portorož, Slovenia (2016).
17. *Pennington, J. et al.:* GloVe: Global vectors for word representation. In: Empirical methods in natural language processing (emnlp). pp. 1532–1543 (2014).
18. *Shao, Y. et al.:* Universal word segmentation: Implementation and interpretation. Transactions of the Association for Computational Linguistics. 6, 421–435 (2018).
19. *Straka, M.:* 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. pp. 197–207 Association for Computational Linguistics, Brussels, Belgium (2018).
20. *Straka, M., Strakova, J.:* Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. Presented at the January (2017).
21. *Zeman, D., Hajič, J. eds:* Proceedings of the CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. Association for Computational Linguistics, Brussels, Belgium (2018).