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

Moscow, June 17–20, 2020

EXPLORING PRETRAINED MODELS FOR JOINT MORPHO-SYNTACTIC PARSING OF RUSSIAN

Anastasyev D. G. (dan-anastasev@yandex-team.ru)
Yandex, Moscow, Russia

In this paper, we build a joint morpho-syntactic parser for Russian. We describe a method to train a joint model which is significantly faster and as accurate as a traditional pipeline of models. We explore various ways to encode the word-level information and how they can affect the parser's performance. To this end, we utilize learned from scratch character-level word embeddings and grammeme embeddings that have shown state-of-the-art results for similar tasks for Russian in the past. We compare them with the pretrained contextualized word embeddings, such as ELMo and BERT, known to lead to the breakthrough in miscellaneous tasks in English. As a result, we prove that their usage can significantly improve parsing quality.

Key words: morphological analysis, dependency parsing, pretrained language models, multitask learning

DOI: 10.28995/2075-7182-2020-19-1-12

АНАЛИЗ ПРЕДОБУЧЕННЫХ МОДЕЛЕЙ ДЛЯ МОРФО-СИНТАКСИЧЕСКОГО ПАРСИНГА РУССКОГО ЯЗЫКА

Анастасьев Д. Г. (dan-anastasev@yandex-team.ru)
Яндекс, Москва, Россия

1. Introduction

Morpho-syntactic parsing is an important part of various natural language processing systems. In our work, we consider the following components of a parser: morphological parser, lemmatizer and syntactic parser.

The whole parsing is typically performed in a pipeline manner when each of the mentioned components works separately and relies on another component's predictions. It may lead to error accumulation and requires multiple trained models to be run sequentially. We designed a joint model that learns to perform the whole morpho-syntactic parsing task in a multi-task mode. This way, all tasks are solved using a single model and there are no dependencies between the components.

In the past few years, pretrained language models have become almost a standard way to improve the model's quality. However, the research of these models was mostly conducted for English. To fill the gap, we compared two of the most popular language models—ELMo [Peters et al., 2018] and BERT [Devlin et al., 2018]—on our task in application to Russian. Despite the reasonable improvement in terms of accuracy, such models are significantly larger and slower than uncontextualized word embedding models. For a fair comparison, we also implemented a model based on character-level word embeddings which have shown the best results for morphological parsing of Russian on the MorphoRuEval-2017 dataset [Sorokin et al., 2017].

We tested our models on the GramEval-2020 shared task. The proposed BERT-based joint model achieved the state-of-the-art results on it.

2. Background

In this section, we give an overview of the parser's components we considered and describe the choices we made designing our parser.

2.1. POS and Morphological Features Tagger

In the Universal Dependencies notation, a morphological parser's prediction consists of a part-of-speech (POS) prediction and a morphological features prediction. Some systems make the prediction separately, some condition the morphological features prediction on the POS prediction [Qi et al., 2019], others predict them simultaneously [Kanerva et al., 2018].

We have chosen the latter approach because it's known to lead to the state-of-the-art results on Russian [Anastasyev et al., 2018]. Thus, the POS tag and morphological features were simply concatenated (e.g., "NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur") and model had to perform multiclass classification with a significant number of classes (about 1200 in our case). It means that the model cannot predict inconsistent grammatical values. A drawback of such an approach is that the model learns to ignore low frequent morphological features at the tail of the distribution. However, we assumed that errors from this tail cannot possibly affect the model's accuracy much.

2.2. Lemmatizer

A straightforward approach is a lemmatization based on a dictionary. Frequent non-homonymous words may be lemmatized using a dictionary only: you just need to look up the word in the dictionary and return the corresponding lemma. Frequent, but homonymous words should be lemmatized with the help of a morphological parser which is required to disambiguate the word. The disambiguated word again may be found in a dictionary. The dictionary might be built from the train data only, yet better results can be achieved with a dictionary based on external resources (such as [Korobov, 2015]). This method fails when an out-of-vocabulary word appears. However, you still can return the word itself as the lemma (this assumption is reasonably good in case of a large enough dictionary).

Another approach utilizes sequence-to-sequence models [Kanerva et al., 2018] to translate the sequence of symbols of the word to the sequence of symbols of its lemma. This way, the problem of the out-of-vocabulary words is solved, but the inference is slower, the model is prone to hallucinations and output of the morphological parser is still required to disambiguate the word.

In our method, we used a method similar to the one in UDPipe [Straka et al., 2016] and in [Sharoff et al., 2011]. It is based on modification rules that have to be applied to a word to obtain its lemma. Only three spelling modification types might be applied: cut N symbols from the prefix of the word, cut N symbols from its suffix and append a specific sequence of symbols to its suffix. Also, three case modifications could be used: a word might be lowercased, uppercased and capitalized. A result modification rule is a combination of such modifications where each modification type appears exactly once. For example: {"cut_prefix": 0, cut_suffix": 1, "append_suffix": "ый", "lower": false, "capitalize": false, "upper": false}.

Therefore, the lemmatizer is a simple classifier that learns to predict a rule to be applied from the rules found in the training set. We considered only rules that appeared more than twice in the train data, which led to less than 1000 classes.

The designed lemmatizer can be trained in the multitask mode with the rest of the parser's components. In this case, the lemmatizer doesn't rely on the morphological parser explicitly anymore, but they learn shared representations. This way, the error propagation issue is solved, and a single model is trained. Also, the model is less likely to hallucinate an invalid lemma than in the sequence-to-sequence approach.

2.3. Dependency Parser

In our work, we considered biaffine dependency parser [Dozat et al., 2017], which applies biaffine classifiers for pairs of words to predict whether a directed arc between these two words should exist and which label this arc should have. We used Edmonds' algorithm for finding minimum spanning trees on directed graphs for decoding.

Typically, this model relies on morphological parser's predictions. We compared such an approach with both unconditional dependency parser (that doesn't receive morphological predictions as an input) and joint model parser (which is trained to predict morphology and syntax relations simultaneously). The latter model uses shared representations between the morphological parser, the lemmatizer and the dependency parser. This approach should help them to learn an additional useful signal from the data and to regularize them.

2.4. Word Embeddings

One of the most important parts of our study is the choice of the word embeddings. We considered the following options.

2.4.1. Character-level Word Embeddings

Character-level word embeddings have led to state-of-the-art models in the past. They were shown to be useful for morphological parsing [Ling et al., 2015] as well as dependency parsing [Qi, 2019]. They are also useful because any word can be encoded and the problem of out-of-vocabulary words doesn't exist. Moreover, the model can learn morphology and semantics from the word's spelling which is likely to be meaningful in Russian.

We used the most common approach to train the character-level embeddings based on BiLSTM. This way, BiLSTM is applied to the sequence of symbol embeddings and word's embedding is constructed as a concatenation of the last states of the forward and backward LSTMs.

2.4.2. Contextualized Word Embeddings

Recently, contextualized word embeddings have led to a breakthrough in many NLP tasks. Contextualized word embeddings are pretrained using a language model task on a large amount of data. The pretraining step helps to learn useful relations from unlabeled data. The contextualization of these embeddings usually leads to better representations of words: it disambiguates the word and encodes some important information about the word's context. We considered the following models: ELMo [Peters et al., 2018] and BERT [Devlin et al., 2018].

The first model uses two LSTMs as forward and backward language models. Those language models encode the context information from the previous and the next words. Yet, they are independent, which leads to a certain lack of context information, because each of them has access only to some part of the sentence. ELMo also utilizes character-level word embeddings to build an uncontextualized word representation. However, unlike the model mentioned in **Section 2.4.1**, their character-level word embeddings are based on the convolutional neural network instead of a BiLSTM.

BERT is based on a transformer model, and it is trained in a masked language model manner. It should help to produce more contextualized word representation because BERT can access both the previous and the next words. To deal with the out-of-vocabulary words, BERT is designed to work on subword-level, which means that a word can be represented using any number of subword embeddings. To obtain a word's embedding, we used mean-pooling of its subword embeddings.

The main drawback of such embeddings is their speed: they tend to be by a few orders of magnitude slower than a simple lookup in a word embeddings table. They are also much larger than character-level word embeddings from the previous section (e.g., in our case, a character-level model was 10 times smaller than an ELMo-based one and almost 30 times smaller than a BERT-based model).

2.4.3. Grammeme Embeddings

It was shown that additional grammeme embeddings can give a significant improvement of the model quality for morphological parsing [Anastasyev et al., 2018]. They encode ambiguous information about the word's grammatical value and give

the model an additional signal about the relations between this word and others. We experimented with grammeme embeddings built using the pymorphy2 dictionary [Korobov, 2015]: we concatenated the word embeddings mentioned in the previous two sections with the grammeme embeddings.

3. Experimental Setup

As already mentioned in **Section 2**, we explored the following embedders in our experiments: character-level BiLSTM, ELMo[1] and BERT[2] with possibly concatenated grammeme embedding.

For the character-level BiLSTM we used a 32-dim character embedding with a dropout rate equal to 0.3 and a single-layer 128-dim BiLSTM (which means that the result word embedding has 256 dimensions).

We considered two types of the encoder: a pass-through encoder which simply passes the word embeddings to the decoder and a single-layer 256-dim BiLSTM encoder. The latter encoder helps to contextualize the word embeddings, while the pass-through one utilizes built-in contextualization of ELMo and BERT models. We applied a dropout with 0.4 rate to the encoders' input word embeddings.

We used a simple logistic regression classifier as the morphological parser's and lemmatizer's decoders (we compared it with a multi-layer feedforward classifier, but it didn't lead to any improvements). We utilized biaffine classifiers for the dependency parser's decoder with the 256-dim tag and 512-dim arc representation. We applied a dropout with 0.1 rate to the encoder's output before the classifiers.

In our proposed multitask approach, all decoders were trained jointly in a single model. We compared it with a single-task mode, where the lemmatizer and dependency parser were either conditioned on the morphology predictions (conditional case) or used only word embedding features (unconditional models). In the first situation, the predictions of the morphological parser were embedded and concatenated to the word embeddings.

We trained the model with slanted triangular learning rates (STLR) and discriminative fine-tuning [Howard et al., 2018]: during the first epoch pretrained embedders were frozen and during the next epochs they were trained with a learning rate by an order of magnitude lower than the learning rate of the trained from scratch parameters. The maximum learning rate of the trained from scratch parameters was 10^{-3}. The warm-up period of the STLR scheduler was 0.1 of the training iterations.

We used the data provided by the organizers of GramEval-2020[3] for training and compared models on the official dev[4] and test[5] sets. We didn't use the 17th-century

---

[1] ruwikiruscorpora_tokens_elmo_1024_2019 from http://rusvectores.org (http://vectors.nlpl.eu/repository/20/195.zip)

[2] RuBERT from DeepPavlov (http://files.deeppavlov.ai/deeppavlov_data/bert/rubert_cased_L-12_H-768_A-12_pt.tar.gz)

[3] https://github.com/dialogue-evaluation/GramEval2020

[4] https://github.com/dialogue-evaluation/GramEval2020/tree/master/dataOpenTest

[5] https://competitions.codalab.org/competitions/22902

subset either for training or for evaluation: it wasn't clean enough and the gains from improvements on the data are not obvious.

We utilized allennlp [Gardner et al., 2018] and transformers [Wolf et al., 2019] libraries for our implementation. We made the code used for those experiments publicly available[6].

Fig. 1 schematically shows the model and summarizes the mentioned options.

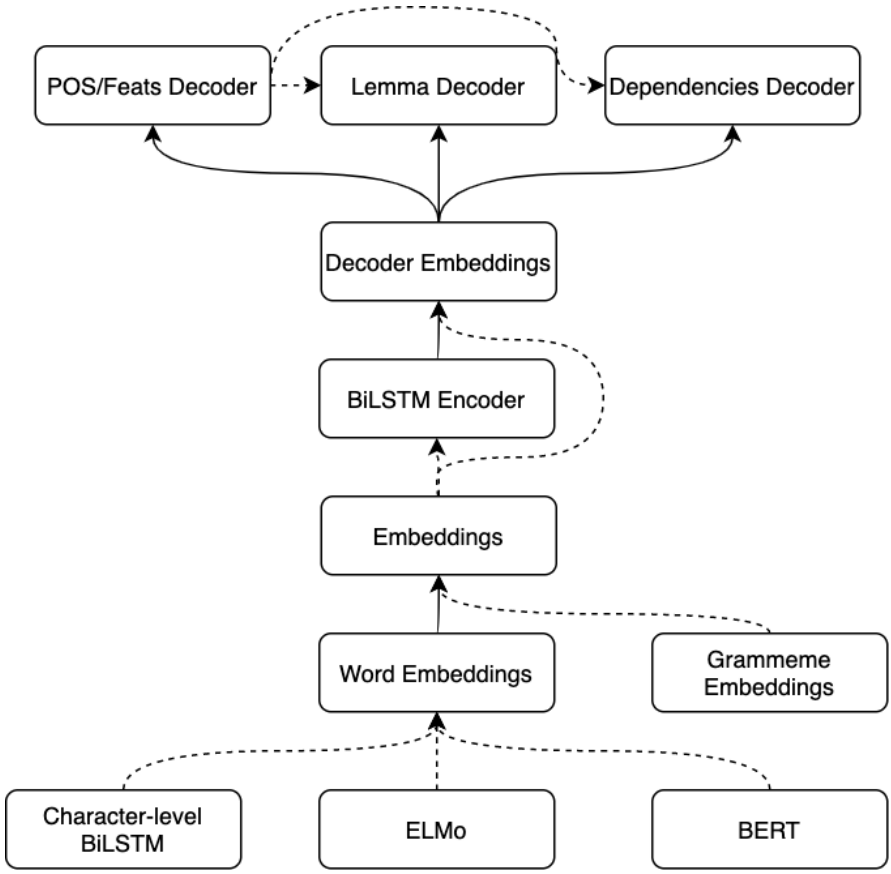

Fig. 1. Schematic representation of the explored options. The dashed lines show optional dependencies between the components

[6] https://github.com/DanAnastasyev/GramEval2020

4. Experiments and Results

4.1. Embedders and Encoders Comparison

Table 1 summarizes the results of the different encoders and embedders described in Section 3. The prefixes in the model names correspond to different embedders: "chars_" denotes the character-level BiLSTM embedder, "elmo_" represents the ELMo-based embedder and "bert_" stands for the BERT-based embedder. Models with an additional grammeme embedding have "_morph_" substring. Models with the "_lstm" suffix use the BiLSTM encoder while models without it use the pass-through encoder. All models were trained in the joint mode.

Table 1. Comparison of different embedders and encoders on dev/test sets

Model	POS	MorphoFeats	Lemma	LAS	Overall
chars	94.7% / 91.7%	92.2% / 90.9%	95.5% / 93.6%	41.1% / 38.1%	80.9% / 78.6%
chars_lstm	97.2% / 94.1%	96.9% / 94.6%	97.3% / 95.0%	87.2% / 77.8%	94.7% / 90.4%
chars_morph_lstm	97.5% / 94.4%	97.7% / 95.2%	98.1% / 95.6%	89.6% / 77.0%	95.7% / 90.6%
elmo	97.4% / 95.4%	96.2% / 95.8%	93.1% / 92.8%	80.3% / 74.1%	91.8% / 89.5%
elmo_lstm	97.9% / 95.9%	97.5% / 95.9%	97.0% / 95.3%	88.9% / 80.3%	95.3% / 91.9%
elmo_morph_lstm	97.8% / 95.7%	97.7% / 96.1%	97.3% / 95.3%	89.5% / 79.6%	95.6% / 91.7%
bert	98.4% / **96.2%**	**98.3%** / **96.4%**	**98.6%** / **96.5%**	93.1% / **84.6%**	**97.1%** / **93.4%**
bert_lstm	**98.6%** / 95.8%	**98.4%** / **96.3%**	**98.5%** / 96.2%	**93.2%** / 83.5%	**97.2%** / 92.9%
bert_morph_lstm	98.4% / 95.9%	**98.4%** / **96.4%**	**98.5%** / **96.4%**	**93.3%** / 84.1%	**97.2%** / 93.2%
bert_random	97.0% / 92.3%	96.7% / 93.3%	97.1% / 93.1%	88.0% / 73.1%	94.7% / 88.0%

Clearly, the BERT-based models significantly outperform others. As expected, the character-level models are worse than the models with contextualized embeddings. However, the difference between the ELMo-based and character-level embeddings model with the grammeme embeddings in terms of quality is relatively insignificant.

The first model (*chars*) shows the possible performance of a completely uncontextualized model. It's quite interesting that the model can solve morphological tasks with reasonably good quality. It demonstrates that the homonymy level in the data is moderate.

The fourth model (*elmo*) is contextualized, but the embeddings are frozen (in accordance with the guidelines from the ELMo authors). The performance of this model shows the amount of morphological and syntactic information the model learned during the pretraining step. It outperforms the uncontextualized *chars* model, especially on the syntax metrics.

BiLSTM encoder is designed to adjust the word representations, to make them more contextualized. In the case of the fine-tunable BERT, it doesn't improve the model's quality. For the character-level embeddings and frozen ELMo the BiLSTM encoder is the only way to adjust the representations to the task. It explains the huge enhancement of all metrics when such encoder is applied.

Grammeme embeddings encode an external knowledge about the given word. Given the performance of the models, we can hypothesize that they can improve the performance of a model trained on the provided data only. In contrast to it, BERT or ELMo seem to be able to learn all the required information from the pretraining step.

However, it's also possible, that larger capacity of those models (the superior number of the learned parameters compared to the *chars* models) leads to their improved quality. To rule out such a hypothesis, we designed an additional experiment. We trained a BERT-based model with randomly initialized weights (*bert_random*). We used a standard random initialization from the transformers library, which is similar to the original BERT implementation. Turns out, this model is on a par with *chars_lstm* model on the dev set and even worse than it on the test set. We consider it as a proof of the importance of the pretraining step.

4.2. Frozen and Tunable Pretrained Embedders Comparison

To compare ELMo and BERT more fairly, both of them should be pretrained and fine-tuned. In Table 2 we summarize the model qualities in those setups. The "frozen_" prefix refers to frozen embedders, while "trainable_" corresponds to the embedders, which were fine-tuned with the rest of the model.

Table 2. Comparison of frozen and fine-tuned models on dev/test sets

Model	POS	MorphoFeats	Lemma	LAS	Overall
frozen_elmo	97.4% / 95.4%	96.2% / 95.8%	93.1% / 92.8%	80.3% / 74.1%	91.8% / 89.5%
frozen_elmo_lstm	97.9% / 95.9%	97.5% / 95.9%	97.0% / 95.3%	88.9% / 80.3%	95.3% / 91.9%
trainable_elmo	98.2% / 95.5%	97.8% / 95.8%	98.2% / 95.9%	91.5% / 79.7%	96.4% / 91.7%
trainable_elmo_lstm	98.3% / 95.7%	97.9% / 95.8%	98.3% / 95.8%	92.2% / 81.2%	96.7% / 92.1%
frozen_bert	96.0% / 94.0%	95.5% / 94.3%	86.6% / 86.6%	81.7% / 76.7%	89.9% / 87.9%
frozen_bert_lstm	97.1% / 95.3%	96.6% / 95.1%	92.3% / 91.0%	86.8% / 82.0%	93.2% / 90.9%
trainable_bert	**98.4%** / **96.2%**	**98.3%** / **96.4%**	**98.6%** / **96.5%**	**93.1%** / **84.6%**	**97.1%** / **93.4%**
trainable_bert_lstm	**98.6%** / 95.8%	**98.4%** / **96.3%**	**98.5%** / 96.2%	**93.2%** / 83.5%	**97.2%** / 92.9%

Frozen ELMo performs surprisingly better than frozen BERT. The main cause, most likely, is the lack of alignment between the BERT's subwords used for pretraining and the complete words required for the parsing tasks. However, BERT's fine-tuning leads to a greater improvement of the final score, while ELMo gains from the fine-tuning step considerably less.

The latter result is somewhat surprising given the common guidelines. Nonetheless, the difference between frozen and fine-tunable ELMo is less significant when an additional encoder is used. It should be noted that the difference in the highest scores achieved by the ELMo- and BERT-based models doesn't necessarily reflect the BERT's architecture superiority. They have to be pretrained on the same corpus to allow a proper comparison.

4.3. Specialized and Joint Models Comparison

To ensure that we didn't lose anything in our joint-training setup, we compared a joint BERT-based model without encoder with similar models but trained in a single-task manner. Table 3 shows the obtained results: *multitask_bert* refers to the baseline model, *conditional_bert* and *unconditional_bert* correspond to the single-task models with and without conditioning on a morphological parser's predictions.

Table 3. Comparison of single and multitask training on dev/test sets

Model	POS	MorphoFeats	Lemma	LAS	Overall
multitask_bert	**98.4% / 96.2%**	**98.3% / 96.4%**	**98.6% / 96.5%**	**93.1% / 84.6%**	**97.1% / 93.4%**
unconditional_bert	**98.5%** / 95.7%	**98.4%** / 96.1%	**98.6% / 96.4%**	92.9% / 83.6%	**97.1%** / 92.9%
conditional_bert	**98.5%** / 95.7%	**98.4%** / 96.1%	**98.7% / 96.5%**	92.7% / 83.2%	**97.1%** / 92.9%
conditional_distorted_bert	**98.5%** / 95.7%	**98.4%** / 96.1%	94.2% / 89.6%	65.8% / 58.7%	89.2% / 85.0%

These models have shown approximately similar accuracies. The multitask model even outperformed single-task ones on the test set. Moreover, it's required to perform about three times more computations on both training and inference stages to use the single-task models.

The lack of difference in the performance of the conditional and the unconditional models shows that the BERT embeddings already contain the required morphological information which is passed explicitly in the conditional model.

However, the conditional model is still prone to error propagation issue. To prove it, we passed random grammatical values to the conditional lemmatizer and the dependency parser of the conditional_bert model (*conditional_distorted_bert* row). It led to a drastic drop in quality, especially in case of dependency parsing performance.

We believe, that these findings can prove the superiority of the proposed joint model.

4.4. Robustness to an Unseen Domain

Mostly, the data we used for training came from the news and fiction domains. However, it is likely to encounter sentences from other domains in a real-life application, which may lead to a considerable drop in the model's performance [Giesbrecht, 2009]. To estimate our model's ability to work on an unseen domain, we evaluated models trained on the data without the poetry and without the social data. We compared them with the baseline model trained on the whole data.

Table 4 summarizes our findings. The last column shows the number of samples removed from the train set.

Table 4. Comparison of models trained without some specific domain

Model	Fiction	News	Wiki	Poetry	Social	DataD
bert	**99.2% / 96.3%**	**97.3% / 96.0%**	**95.9% / 90.6%**	**99.0% / 93.0%**	**94.0% / 91.2%**	0
bert_w/o_poetry	**99.1% / 96.2%**	**97.2% / 95.9%**	**96.0%** / 89.4%	91.1% / 91.9%	**93.8% / 91.3%**	−915
bert_w/o_social	**99.2% / 96.2%**	**97.3% / 95.9%**	**96.0%** / 88.9%	**99.0%** / 92.3%	93.4% / 89.7%	−2773

The quality drops on 1–1.5% on the unseen domains, which seems to be reasonably significant, but not drastic. However, the performance of the model on those genres was poorer compared to the fiction or the news subsets. So, we expect our model to work well enough on other unseen domains, yet not without errors.

4.5. Speed and Size Comparison

As we mentioned in section 2.4.2, the main issue of the large pretrained models such as BERT or ELMo is their lower speed and larger size. We summarize the speed and size of the presented models in Table 5.

Table 5. Comparison of speed in samples per second and size of the models

Model	Speed, it/s, CPU	Speed, it/s, GPU	Size, Mb
chars_morph_lstm	39.3	30.3	25
elmo_lstm	11.8	24.3	243
bert	3.0	8.4	702

The speed evaluation was conducted on an Intel Xeon E5-2660 v4 CPU and a Tesla M40 GPU. We used a batch size equal to 64 for ELMo-based model on GPU, a batch size equal to 16 for BERT-based model on GPU and a single sample batch for all other cases. Surprisingly, the smallest model (chars_morph_lstm) has better performance on CPU rather than on GPU. Most likely, the model is too small to benefit from the faster matrix multiplications that a GPU can provide.

As expected, BERT is significantly larger and considerably slower than others, while character-level is the fastest and the smallest model.

4.6. Comparison on GramEval-2020

We also compared our model with the results of the contestants of GramEval-2020. The results are presented in Table 4. The results on the 17th century subset are ignored.

Table 6. Results of GramEval-2020

Contestant	POS	MorphoFeats	Lemma	LAS	Overall
qbic	**95.8%**	**96.6%**	**96.5%**	**84.2%**	**93.3%**
ADVance	95.3%	96.3%	95.8%	82.2%	92.4%
lima	95.4%	96.5%	93.6%	76.6%	90.5%
vocative	94.0%	91.9%	94.6%	73.5%	88.5%
baseline	92.0%	91.7%	92.2%	57.9%	83.5%

Clearly, our BERT-based models achieve state-of-the-art performance on this dataset.

Interestingly, the uncontextualized *chars* model from Table 1 shows comparable results to the baseline. Most likely, it shows the importance of the training on the data from a similar distribution and with similar markup guidance as in the test set.

Besides, our model with the character-level embeddings (*chars_morph_lstm* from Table 1) might have achieved third place despite its simplicity. Moreover, its lemmatization quality is on par with the second-place result which proves that our proposed lemmatization technique is effective in terms of quality as well as in terms of speed.

As a result, it's possible to choose an optimal model by the quality-size tradeoff varying the embeddings and encoder, and even the smallest model will be reasonably accurate.

5. Conclusions

We proposed a joint morpho-syntactic parser which shows state-of-the-art performance on the GramEval-2020 dataset. We proved that joint training can simplify the parsing pipeline and improve its accuracy. Moreover, our designed lemmatization technique is more straightforward than the dictionary-based or the sequence-to-sequence lemmatization, but it shows promising results.

We explored different pretrained contextualized word embeddings in an application for this task and showed the way to improve the parsing quality using them. We hope that our analysis may offer an insight into the ability of the pretrained language models to increase the model's quality. Based on the results, it should be easier to make a choice on the balance between the accuracy, the size and the speed of the parser.

References

1. *Anastasyev D. et al.* (2018), Improving Part-of-Speech Tagging via Multi-task Learning and Character-level Word Representations, Computational linguistics and intellectual technologies: Proceedings of the International Conference "Dialog 2018", pp. 14–27.
2. *Eugenie Giesbrecht, Stefan Evert* (2009). Is Part-of-Speech Tagging a Solved Task? An Evaluation of POS Taggers for the German Web as Corpus, Proceedings of the 5th Web as Corpus Workshop (WAC5), San Sebastian, Spain.
3. *Jacob Devlin et al.* (2018), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, available at https://arxiv.org/pdf/1810.04805.pdf.
4. *Jenna Kanerva et al.* (2018), Turku Neural Parser Pipeline: An End-to-End System for the CoNLL 2018 Shared Task, Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Brussels, Belgium, pp. 133–142.
5. *Jeremy Howard, Sebastian Ruder* (2018), Universal Language Model Fine-tuning for Text Classification, available at https://arxiv.org/pdf/1801.06146.pdf.
6. *Korobov M.* (2015), Morphological Analyzer and Generator for Russian and Ukrainian Languages, Analysis of Images, Social Networks and Texts, Vol. 542, pp. 320–332.
7. *Ling Wang et al.* (2015), Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, pp. 1520–1530.
8. *Matt Gardner et al.* (2018), A Deep Semantic Natural Language Processing Platform, Proceedings of Workshop for NLP Open Source Software (NLP-OSS), Melbourne, Australia, pp. 1–6.

9. *Matthew Peters et al.* (2018), Deep Contextualized Word Representations, Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, New Orleans, Louisiana, Vol. 1, pp. 2227–2237.
10. *Milan Straka* (2016), UDPipe: Trainable Pipeline Sharoff for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing, Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16), Portorož, Slovenia, pp. 4290–4297.
11. *Peng Qi et al.* (2018), Universal Dependency Parsing from Scratch, Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Brussels, Belgium, pp. 160–170.
12. *Sharoff S., Nivre J.* (2011), The proper place of men and machines in language technology: Processing Russian without any linguistic knowledge. Computational linguistic and intellectual technologies, pp. 591–604.
13. *Sorokin A., et al.* (2017), MorphoRuEval-2017: An Evaluation Track for the Automatic Morphological Analysis Methods for Russian, Proceedings of the International Conference "Dialog 2017". Vol. 1, pp. 297–313.
14. *Thomas Wolf et al.* (2019), HuggingFace's Transformers: State-of-the-art Natural Language Processing, available at https://arxiv.org/pdf/1910.03771.pdf.
15. *Timothy Dozat, Christopher Manning* (2016), Deep Biaffine Attention for Neural Dependency Parsing, available at https://arxiv.org/pdf/1611.01734.pdf.