**BUILDING GAPPING RESOLUTION SYSTEM OVERNIGHT: LESSONS LEARNED**

Gapping is a type of ellipsis where entire predicate is elided while two or more arguments or adjuncts are present. In this paper we describe our submission to Automatic Gapping Resolution for Russian (AGGR) competition. Our system was build in just a few hours and does not rely on any third-party pre-trained models like BERT. Nevertheless, we obtain competitive results.

Keywords: Gapping resolution, Russian language, Neural sequence taggers

**СОЗДАНИЕ СИСТЕМЫ РАЗРЕШЕНИЯ ГЭППИНГА ЗА ОДНУ НОЧЬ: ИЗВЛЕЧЕННЫЕ УРОКИ**

Гэппинг - это тип опущения, в котором исключается весь предикат, в то время как присутствуют два или более аргумента или дополнения. В этой статье мы описываем наши результаты участия в конкурсе «Automatic Gapping Resolution for Russian» (AGGR). Наша система была построена всего за несколько часов и не использует сторонних предварительно обученных моделей, таких как BERT. Тем не менее, мы получаем конкурентоспособные результаты.

Ключевые слова: гэппинг, русский язык, нейронные системы разметки последовательности.

**1. Introduction**

Gapping is a type of ellipsis where entire predicate is elided while two or more arguments or adjuncts are present. Examples of gapping (taken from AGRR organizers website) are ““Ей он рассказывает одно, а нам — совершенно другое”, “Кто любит арбуз, а кто - свиной хрящик”, “Дайте мне две пятерки, а я вам десятку”.The aim of AGRR competition was to conduct systematic study of gappping annotation systems and approaches using large corpus. Previous studies of gapping employed mainly syntactic parsers and rules [Marneffe et al, 2017] and only small annotated corpus. Thus present work can only be meaningfully compared to results, obtained by other AGRR participants, and, at the time of this writings, information about these submissions was limited.

According to AGRR results website, most successful systems employed neural sequence taggers with transfer learning, with contextual embeddings. Contextual embeddings are essentially hidden states or combinations of hidden states of neural networks trained on a large corpus in unsupervised fashion, with objective to predict next token (neural language models), or intermediate token. Such approached are being developed from 2015 [Tarasov et al, 2015] but gained popularity only recently, when computing resources made possible to pre-train large models. As of list of most popular contextual embedding models includes ELMO [Peters et al, 2018], GPT [Radford et al, 2018] and BERT [Delvin et al, 2018], with BERT considered to be state of the art.

Possible disadvantages of such approaches include:

* Models that use large neural networks as feature extractors are slow, which makes them hard to develop, debug and optimize hyperparameters. Same type of disadvantage is present at inference time, although methods to compactify such models exists.
* Underlaying models are too large to study and improve for small research groups, that are forced to use published pre-trained weights. This limits capabilities for true innovation to large corporations and government entities that posses necessary computational resources
* Models are hard to construct. Even with pre-trained BERT models developers need to write lot of code to train and make new predictions. Errors in that code could easily ruin whole system

In many commercial projects the ability to quickly prototype good enough solution for new kind of problem is often more valuable then achieving SOTA results.

Given these considerations, in this paper we describe a method that allowed us to implement competitive solution for gapping annotation within 12 hours, out of which 2 were spent in coding and 10 in model training using single commodity GPU. Our method employs smaller character level contextual embeddings that were pre-trained beforehand for a week, also using single GPU.

**2.Methods and algorithms**

2.1. Character level language model for contextual embeddings

Model described here was pre-trained beforehand for other task and was not part of our gapping resolution system development effort, it is described here for completeness, since its details were not published before and can not be found elsewhere.

Language modeling (LM) is one of the important tasks of natural language processing. The task involves predicting the (*n+1*)th token in a sequence given the *n* preceding tokens, where tokens can be words, subwords or characters. More formally, the goal of a language model is to estimate a distribution *P*(*x*0:T) over sequences of tokens (*x*0 , x1 , . . . , xT ).

The joint distribution over long text spans can then be represented as a product of the predictive distribution over tokens conditioned on the preceding tokens:

Neural language models [Sutskever et al, 2011] usually use recurrent neural networks (RNN) for sequence modeling. Given a sequence of vectors {*x(t)*} where t=1..T , an RNN computes memory and output sequences:

*h(t) = f(Wx(t) + Vh(t−1) + b) (2)*

*y(t) = g(Uh(t) + c) (3)*

where *f* is a nonlinear function, such as the sigmoid or hyperbolic tangent function and *g* is the output function. *W* and *V* are weight matrices between the input and hidden layer, and between the hidden units. *U* is the output weight matrix, *b* and *c* are bias vectors connected to hidden and output units. *h*(0) in equation *(1)* can be set to constant value that is chosen arbitrary or trained by backpropagation.

Recently, it was shown that by learning to predict the next character given previous characters, neural network based language models can learn internal representations that capture syntactic and semantic properties [Radford et al, 2017].

We use Long Short Term Memory (LSTM) [Hochreiter et al, 1997] based neural network. The structure of the LSTM allows it to train on problems with long term dependencies. In LSTM simple activation function *f* from above is replaced with composite LSTM activation function*.* Each LSTM hidden unit is augmented with a state variable *s(t)* The hidden layer activations correspond to the ‘memory cells’ scaled by the activations of the ‘output gates’ *o* and computed in following way:

*h(t) = o(t) \* f(c(t)) (3)*

*c(t) = d(t) \* (c(t-1) + i(t)) \* f(Wx(t) + Vh(t-1) + b) (3)*

where \* denotes element-wise multiplication, *d(t)* is dynamic activation function that scales state by "forget gate" and *i(t)* is activation of input gate.

We train LSTM model with 3 hidden layers, with 3192 LSTM cells in the first layer and 2048 LSTM cells per 2 remaining layers. Model was pre-trained on 2 billion characters of Russian texts, taken from Common Crawl data ([www.commoncrawl.org](http://www.commoncrawl.org/)). Since Common Crawl contains a lot of low quality repetitive text, we filtered out sentences with high perplexity (evaluated with simpler LSTM-based language model trained on Russian Wikipedia data). Model was trained by using truncated backpropagation through time (BPTT) with learning rate controlled by Adam [Kingma et al, 2014] algorithm. One significant difference from similar character level contextual embedding model training that we employed was very large BPTT window size – 500 characters (as compared to typically window size of 100 characters). We halted training by tracking the performance on the validation set, stopping when negligible gains were observed, which happened after 1 week of training on single 1080GTX GPU. Our model reached 1.39 bit per character (BPC) metric value on test set.

2.2. Gapping resolution model

We represented gapping resolution task as sequence tagging problem, assigning each gapping tag (cV, cR1, cR2, R1, R2) to each word. For simplicity we predict gap position to be the same as beginning of R2. These assumptions allowed us to reduce programming and fit into self-imposed development time constraint. Obviously, they had negative impact on final accuracy.

For binary classification task (presence or absence of gapping in the sentence) we simple detected presence or absence of R2 in the sequence tagging output, thus we avoided use of separate model.

We employed differential programming library NeuThink (<https://github.com/meanotekai/neuthink>) which add layer of abstraction on top of PyTorch (pytorch.org) deep learning framework. NeuThink uses expression trees and prototype-based programming to simplify neural model specification and, automatically generates inference code based on model definition. With many routine tasks handled by the library, manual coding was limited to format conversion script, that converted AGRR format to NeuThink sequence tagging data format (39 lines of code) and model specification itself (6 lines of code). The process took 2 hours total (including post-implementation bug fixing).

Our final model uses hidden states of the last layer of character-level language model as input (states are taken at the last character of each word, so our sequence classifier is actually word-level, but can use character-level information for unknown words). The input is fed to two bi-directional LSTM layers with 256 units per layer. Output of the last BLSTM layer is fed into Softmax classifier, that predicts tag for each word. Training was conducted using mini-batch gradient descent with one sentence per minibatch, with initial learning rate set to 0.0001 and cross-entropy loss. Training was stopped when loss on the development set could no longer be improved.

Because of time constraint, trivial bug was introduced into the format conversion code, which was not timely discovered since metrics calculation script that we used to validate results ( provided by organizers) also contained errors. Thus we submitted correct results only after final deadline (code was corrected after corrected metrics calculation script was released by organizers.

Our final solution contains only 45 lines of code and is in this aspect shortest solution from all submitted.

**3. Results and discussion**

Our results are summarized in Tables 1 and 2. For all tables we picked best results from all submissions for each group, regardless of time it was send, since a number of participants also apparently send results with trivial algorithmic errors that were corrected afterward. For true method comparison we should avoid considering results with such errors. Column “Knowledge Tranfer Method” was filled by us by examining published code of other participants and may contain mistakes, since we do not have access to other participants detailed method description.

Overall, while our results are inferior to first two BERT-based models, they are still good (third place for full annotation and fourth place for binary classification in consolidated table). They are superior to most non-contextual embeddings models and also surprisingly superior to two BERT-based models (which is probably due to algorithmic errors in their implementation). Our results are also superior to AWD-LSTM word-based model trained on corpus of comparable size and used in more sophisticated way (ULMfit for classification).

It is interesting to note, that gapping resolution requires model to learn long range dependency between gap position and corresponding verb, and character-level context representation seems to be good enough for doing it. We are presently unaware of of existence of other demonstrations of suitability of charter-level contextual embeddings for the tasks that require understanding of long-range relations.

Table 1. Full annotation task results\*

|  |  |  |
| --- | --- | --- |
| Team | F1 | Knowledge Tranfer Method |
| Fit-predict | 0.892 | BERT-based |
| EXO | 0.836 | BERT-based |
| Meanotek | 0.687 | Custom contextual model (LSTM, character based with very long context window) |
| МГУ-DeepPavlov | 0.652 | FastText |
| Koziev Ilya | 0.646 | word2vec |
| Derise | 0.621 | No pretraining (?) Transformer-based |
| MorphoBabushka | 0.44 | BERT-based |
| nsu-ai | 0.03 | BERT-based |
| Vlad | - | Custom contextual model (LSTM, word based) |

\* For all tables we picked best results from all submissions for each group, regardless of time it was send

Table 2. Binary classification results

|  |  |
| --- | --- |
| Team | F1 |
| Fit-predict | 0.95 |
| EXO | 0.94 |
| МГУ-DeepPavlov | 0.91 |
| Meanotek | 0.87 |
| Derise | 0.85 |
| Vlad | 0.84 |
| Koziev Ilya | 0.83 |
| MorphoBabushka | 0.68 |
| nsu-ai | 0.19 |

4. Conclusion

Using Gapping resolution as an example of novel NLP task, we demonstrated that by using of combination of relatively small pre-trained character-level contextual embeddings and novel neural network model-definition method implemented in NeuThink library, it is possible to obtain completive (3rd and 4th place) and practically useful results in just 2 hours of programmer effort and 10 hours of training time on commodity GPU. Our results can easily be improved by spending some more time in developing model that does not employ our simplifying assumptions (such as presence of R2=presence of gapping) or by doing more extensive hyperparameters search.

We believe, that even in the age when huge pre-trained models seems to dominate in all NLP tasks, there is still significant room for innovation even for groups that do not have access to large computing clusters. Even thought it is hard to build models that perform on par with BERT and GPT-based embeddings, it is still possible to advance methods that allow for fast model prototyping and finding better tradeoff between accuracy and development time.

5. References

1. de Marneffe M. C., Nivre J., Schuster S. Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017) //Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017). – 2017.
2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
3. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780
4. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).
5. Peters, Matthew E., et al. "Deep contextualized word representations." *arXiv preprint arXiv:1802.05365* (2018).
6. Radford, A., Jozefowicz, R., & Sutskever, I. (2017). Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.
7. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.
8. Tarasov D.S. (2015) Deep Recurrent Neural Networks for Multiple Language Aspect-Based Sentiment Analysis // Computational Linguistics and Intellectual Technologies: Proceedings of Annual International Conference “Dialogue”, Issue 14(21), V.2, pp. 65-74