Computational Linguistics and Intellectual Technologies:
Proceedings of the International Conference “Dialogue 2019”

Moscow, May 29—June 1, 2019

FILLING THE GAPS WITH RULES AND NETWORKS[1]

Sorokin A. A. (alexey.sorokin@list.ru)

Moscow Institute of Physics and Technology, Neural Networks and Deep Learning Lab, Dolgoprudny, Russia; Moscow State University, Faculty of Mathematics and Mechanics, Moscow, Russia

In this paper we describe rule-based and neural approaches to gapping resolution task for Russian language. Our study was conducted on the material of AGRR-2019 Shared Task. We demonstrate that neural model definitively outperforms the rule-based one even when only 2000 annotated sentences are available. The rule-based model took the 6th place in AGRR-2019 competition (2nd in terms of precision), while the neural one was better than the second-ranked system[2].

Keywords: Gapping, ellipsis, automatic gapping resolution, neural networks

ПРАВИЛОВЫЕ И НЕЙРОННЫЕ МОДЕЛИ ДЛЯ ГЭППИНГА

Сорокин А. А. (alexey.sorokin@list.ru)

Московский Физико-технический Институт, Лаборатория нейронных систем и глубокого обучения, Долгопрудный, Россия; Московский Государственный Университет, механико-математический факультет, Москва, Россия

Данная работа посвящена автоматическому распознаванию гэппинга. Мы показываем, что нейросетевой подход к этой задаче более эффективен в сравнении с правиловым, в том числе на обучающей выборке небольшого размера. Наша нейросетевая модель показала качество выше второго результата в соревновании AGRR-2019[3], в то время как правиловая модель заняла шестое место, показав при этом вторую точность.

Ключевые слова: гэппинг, автоматическое распознавание гэппинга, эллипсис, семантический парсинг

[1] The research was conducted under support of National Technological Initiative Foundation and Sberbank of Russia. Project identifier 0000000007417F630002.

[2] The neural model was submitted two weeks after the end of AGRR-2019 competition when all the gold answers and the systems of other participants were available.

[3] Данный результат был получен после завершения соревнования.

1. Introduction

In linguistics, gapping is a type of ellipsis that occurs in the non-initial conjuncts of coordinate structures [11], [2]. The elided material usually includes the finite verb as well as some of its dependents. For example, in the sentence *Маша любит чай, а Петя кофе.* (*Mary likes tee and Peter coffee*), the elided segment consists of the main verb *любит* (*likes*). Identifying the presence/absence of gaps and their resolution is important in Natural Language Understanding. For example, consider the sentence

(1) *Президентом Ирака стал Бакр, а вице-президентом — Саддам Хуссейн.*
Bakr became the president of Iraq and Saddam Hussein the vice-president.

To extract the semantic structure of this sentence and transform it, e. g., to Wikidata-like triples object-relation-subject, a system must restore the missing verb *стал* (*became*). The extracted triples can be useful, for example, for question answering or information retrieval.

Gapping has attracted high attention in theoretical linguistics [11], [8], [9], however, there are only a few works that investigate gapping in computational literature [14], [5], [6], [7], [15]. Moreover, none of this works address this problem in modern NLP paradigm, where large amounts either of labeled or unlabeled data are utilized. The main obstacle is the lack of datasets: even large existing treebanks contain not more than several hundreds of gapping examples. The only exception is Automatic Gapping Resolution for Russian Shared Task (AGRR-2019) [16] https://github.com/dialogue-evaluation/AGRR-2019, which provides more than 16,000 sentences in total for training and development. Our system was submitted to participate in this competition and the present work describes the model, as well as the results of its application to the dataset.

The paper consists of the following parts: **Section 2** describes the task and the dataset. **Section 3** describes the pipeline and our initial rule-based model. **Section 4** describes the neural model and **Section 5** is devoted to its training. **Section 6** is devoted to data pre- and postprocessing, **Section 7** measures the quality of our models as well as their individual parts. In **Section 8** we conclude with the directions for future work.

2. Task description

AGRR-2019 organizers postulate gapping resolution task as follows: given a raw sentence

(2) *Президентом Ирака стал Бакр, а вице-президентом — Саддам Хуссейн.*
Bakr became the president of Iraq and Saddam Hussein the vice-president.

detect:

1. Whether the sentence contains a gap (binary presence-absence task in organizers' terms).
2. [item:pos] The position of this gap. The annotation standards located it immediately to the left of the first symbol of right core argument *Саддам Хуссейн*. Naively it seems more natural to treat the hyphen as such position, however, the hyphen is optional. Therefore the organizers' solution is more consistent, though less obvious from the first glance.

3. The position of the predicate, corresponding to the gap (gap resolution task).
4. [item:core] The core arguments of the elided predicate: *вице-президентом* (*vice-president*+Ins) and *Саддам Хуссейн* (*Saddam Hussein*+Nom).
5. [item:core-main] The core arguments of the main predicate, corresponding to the orphaned dependents of the elided one: *Президентом Ирака* (*president*+Ins *of Iraq*+Gen) and *Бакр* (*Bakr*+Nom).

Alltogether these subtasks comprise the full resolution track. The participants were allowed to solve the entire task or simply detect the presence/absence of the gap. Note that a sentence may contain several gaps, corresponding to the same main predicate, such as.

(3) *У двоих были черные волосы, у одного — светлые, а у четвертого — каштановые.*
Two of them had black hair, one blond and the fourth — brown.

The dataset also contains several examples with only one orphaned dependent, such as

(4) *Судья посмотрела на свои часы, затем — на меня.*
The judge looked at her watch and then — at me.

The characteristics of the dataset are given in **Table 1**. Note that the number of gapped sentences and their relative frequency significantly exceeds the corresponding parameters of existing general-purpose corpora, such as Universal Dependencies (e. g. [5], **Table 1**).

Table 1: Statistics of gap sentences in the dataset

	Train	Development	Test
Total	16,407	4,143	2,046
With gaps	5,542	1,382	680
Multiple gaps	369	90	47
Single orphan gaps	174	27	17

3. Rule-based approach

As introduced in [12], the gapping relations in UD 2.x treebanks is treated via promotion (see also [5]). The highest node in “obliqueness hierarchy”[4] is promoted to be the head of the clause containing the gap, while all other core dependents of the elided predicate are attached to it via the orphan relation. It results in the dependency tree below on **Figure 1**.

[4] nsubj > obj > iobj > obl > advmod > csubj > xcomp > ccomp > advcl

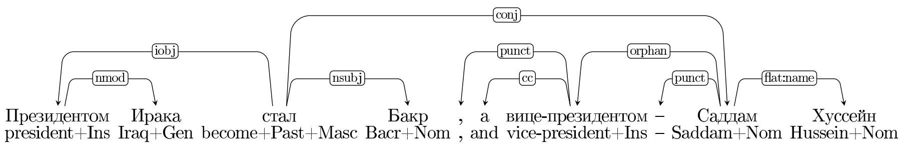
conj
iobj
nmod
nsubj
punct
cc
orphan
punct
flat:name
Президентом Ирака стал Бакр , а вице-президентом – Саддам Хуссейн
president+Ins Iraq+Gen become+Past+Masc Bacr+Nom , and vice-president+Ins – Saddam+Nom Hussein+Nom

Figure 1: A dependency tree for a sentence, containing gap

Given the golden dependency tree, the rule-based pipeline may work as following:

1. Find the orphan dependency and extract the subtrees attached to its left and right edges as elided predicate dependents. If required, remove the punctuation and conjuction between two clauses.
2. Label the first word of the right subtree as gap position.
3. Start from the head of the found orphan relation and follow the dependency edges upward until a verb node is reached. Label this node as the main verb.
4. Find in the subtree of the main verb the two nodes that better match the core dependents found on the first step using their morphology, syntactic roles and semantics (e. g. embeddings).

Provided the syntactic tree is correct, the first three stages are performed algorithmically. The last task was studied in [15] for English, where it was solved by finding the cheapest alignment between the arguments in the full and gapped clause, where the cost of the alignment was based on similarities between the phrases being aligned as well as on the monotonicity of the alignment. The cost of individual alignment links used GloVe similarities between words being aligned and their part-of-speech tags. This method achieved a relatively high quality of remnant attachment with precision and recall of 87% (see **Table 5** in [15]) on golden parses for English language. However, in the real world scenario with automatically generated parse trees using state-of-the-art dependency parsers, the recall falled to 38% and even the precision to 65%.

We met the same problems as in even to a greater extent. For example, in the development set of the dataset our parser found only 338 orphan relations which is less than one quarter of the number of gapped sentences[5]. Even if the model reconstructs an approximately correct parse tree in terms of its topology (Unlabeled Attachment Score), it may fail to label the orphan edge correctly. It is frequently confused with nsubj, obl or nmod, amod relations, for example, in the sentence on **Figure 2** nsubj was predicted instead of orphan.

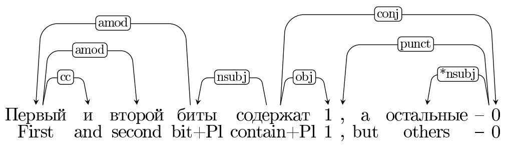
amod
amod
cc
nsubj
obj
conj
punct
*nsubj
Первый и второй биты содержат 1 , а остальные – 0
First and second bit+Pl contain+Pl 1 , but others – 0

Figure 2: Wrong dependency label in automatical parse tree of gapped sequence

[5] We used the UDPipe [18] dependency parser trained on ru-syntagrus corpus together with DeepPavlov morphological tagger, based on [10] and [17].

Some of these parsing errors can be overcome using other clues except the orphan relation, such as the presence of hyphen or "…, *a* …" construction together with the verb-noun conj relation between the main clause and the promoted gap dependent, which is unlikely to occur in other conditions. Therefore we developed a complicated rule-based system, which finds the gap position using the potentially incorrect dependency tree. If the tree was correct, a system would find the gap predicates by picking the shortest edge that covers the gap position detected on the previous stage. However, parsing errors make the rules even more complicated since the parser is prone to establish local dependencies. For example, consider the sentence

(5) *Мне поставили пять, а брату моей подруги детства — четыре* (*They gave (the mark) five to me and to the brother of my childhood friend+Fem — four*)

and its correct and predicted parses on **Figure 3**.

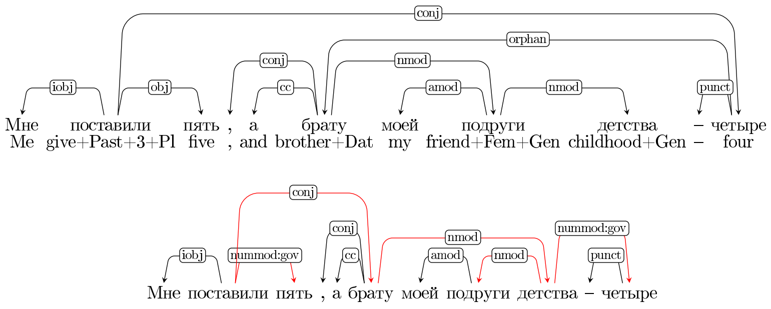
conj
orphan
conj
nmod
iobj
obj
cc
amod
nmod
punct
Мне поставили пять , а брату моей подруги детства – четыре
Me give+Past+3+Pl five , and brother+Dat my friend+Fem+Gen childhood+Gen – four
conj
conj
nmod
nummod:gov
iobj
nummod:gov
cc
amod
nmod
punct
Мне поставили пять , а брату моей подруги детства – четыре

Figure 3: Local attachment error in automatically obtained dependency parse tree (bottom)

Following the obliqueness hierarchy, the gold parse tree the second direct object *четыре* "*four*" is attached to the head of the main clause and the indirect object *брату brother*+Dat — to it via orphan relation. However, the indirect object is much closer to the main verb, therefore the parser selects it as the head of the second clause. Moreover, both numeral objects obtain a wrong dependency label nummod:gov; the structure of the noun phrase *моей подруги детства* "*my*+Gen+Fem *childhood*+Gen *friend*+Gen+Fem" is also incorrect. In this case a system finds the pattern "… , a … " and the conj relation between verb and noun, but fails to restore the second predicate of the gap.

To find the remnants of the gap arguments we utilize the morphological information. Namely, we try to find the descendants of the main verb that have the same part-of-speech and case as the arguments. If only a single word satisfies these constraints, we return this word as remnant. If there are multiple remnants, we rank them using their depth in the tree, their location (to the left or to the right of the main verb) and other features. The hierarchy of features was tuned by hand on the training set.

The final part of the model transforms the constituent heads discovered on the previous stages to the correspondings constituents, which are returned as spans. We use a syntactic parser for this purpose simply considering all dependents of a constituent head as its subtree. We also write some rules to remove the punctuation and function words on constituent boundaries and to deal with format mismatch between UD and the competition (e.g., competition and UD guidelines differently treat numeral constructions such as *десять лет* or fixed prepositional phrases).

4. Neural model

Rule-based models usually suffer from data variability and vagueness. Consequently, we decided to design a neural network that can automatically detect gapping after training on labeled data. We solve the task by stages, using separate networks for gap location, predicate location and remnant matching, although the architecture of all the networks is the same. The networks on latter stages of the pipeline use as inputs (some of) the outputs produced by the previous stages. We describe in details the network for gap prediction and only note the differences for models that perform predicate location and remnant matching.

4.1. Network for gap location

We solve the following task: given the tokenized sentence $w_1, \ldots, w_m$, the verb position i and word position j, predict whether the verb w_i was elided in position j. The network structure can be described as following:

1. Take as input a sequence of pretrained word embeddings $e_1, \ldots, e_m$.
2. Pass this embeddings through two independent bidirectional LSTMs to obtain sequences of context vectors $g_1, \ldots, g_m$ and $h_1, \ldots, h_m$.
3. Calculate similarity scores between g_i and h_j, a natural way to do it is to measure their dot product $s_{ij} = \langle g_i, h_j \rangle$.
4. Pass the similarities through a sigmoid layer $\sigma_{ij} = \sigma(s_{ij})$.
5. Label as gaps for verb w_i all the positions j, such that $\sigma_{ij} > \frac{1}{2}$.

We did not use the softmax activation because there can be several gaps for a single verb in a sentence. After preliminary experiments we replaced the attention-like similarity calculation by an Infersent-like [3] dense layer using the formula:

$$s_{ij} = \langle w, [g_i, h_j, g_i - h_j, g_i \odot h_j] \rangle + b,$$

where w and b are trainable vectors, $\odot$ denotes element-wise product and $[\,\cdot\,]$ concatenation. The intuition behind is that g-embedding of the main verb g_i should match the h-embedding of the gap position h_j.

4.2. Network modifications

For the task of gap argument location we use as inputs the pair of verb and gap positions (i and k respectively). When calculating the similarity scores s^{arg}_{ikj} for the argument in position j we simply use a dense layer $s^{arg}_{ikj} = \langle w^{arg}, [g_i, g_k, h_j] \rangle + b^{arg}$

Since there can be only one left argument for a given gap, we return the position of the highest score σ_{ikj} provided $\sigma_{ikj} > \frac{1}{2}$. We also select only those k that satisfy the inequality $i < j < k$. For the right argument we train an analogous model, using the restriction $j > k$ during decoding phase.

For the problem of remnant matching we pass as inputs the verb index i and corresponding gap predicate index k. Since the predicate of the gapped verb usually resemble its remnant we use again the Infersent-like formula:

$$s_{ikj}^{rem} = \langle w^{rem}, [g_i, g_k, h_j, g_k - h_j, g_k \odot h_j] \rangle + b^{rem}$$

In this case also only the word with the highest score is returned.

Summarizing, given the training instance for a single sentence, which includes the main verb V, its predicates (remnants) R_l and R_r and the gap triples of the form $\langle G_i, P_{i,l}, P_{i,r} \rangle$, the prediction pipeline is expressed in Table 2.

Table 2: Inputs and outputs for different pipeline phases

Pipeline phase	Input	Output
Gap location	V	$G_1, \ldots, G_k$
Gap predicate location	v, G_i	$P_{i,1}, P_{i,r}$
Remnant location	$v, P_{i,l}$	R_l
	$v, P_{i,r}$	R_r

When the heads are found, we recover the complete remnant and predicate spans using the same dependency-based procedure as in the rule-based system.

Theoretically, the information about word morphology and syntax can be useful to detect its gapping status. Our network is enough flexible for this task: one may concatenate the embeddings of word morphological tag and dependency type to the pretrained word embedding and train the corresponding embedding matrix together with the network.

5. Model training

5.1. Loss function

Our loss function consists of two components: L_p, penalizing false positives, and L_r, preventing from false negatives. Both components use standard cross-entropy loss and sum over all verbs in all sentences of the training data. Using notation of the previous section, the loss for a single verb in position i is

$$\begin{aligned} L_p &= -\sum_{j \in \overline{I_g}} \log(1 - s_{ij}), \\ L_r &= -\sum_{j \in I_g} \log s_{ij}, \end{aligned}$$

where I_g is the set of all answers (gaps) for a given input (verb). Actually, $L = L_p + L_r$ equals to standard cross-entropy. Since recall is more important than precision at the early stages of the pipeline, we penalize false negatives higher than false positives by weighting L_r with additional multiple α.

When doing argument prediction and remnant matching we output only the position with the highest matching score s_{ij}. Therefore we must ensure during training, that the score of the correct word in position j_g is maximal among all words. In this case we add auxiliary loss

$$L_{aux} = -\alpha(\log s_{ij_g} - \log \max_j s_{ij}),$$

which is zero only when the correct word has the highest score. It is also weighted by recall weight α, which was selected to be 2 in our experiments.

5.2. Model parameters

We used ELMo embeddings [13] as input for our model, using the implementation from DeepPavlov library [1]. We took the first layer of ElMo network, since it is known to better reflect morphological and syntactic properties that are important for gapping. The size of these embeddings was 1024. Bidirectional LSTMs for sentence processing contained 192 units in each direction.

We collected the inputs in mini-batches, a single batch contains all input-output pairs for 8 sentences (there can be several verbs in a sentence, therefore actual size of the mini-batch is larger). The input data is partitioned in 3/1 proportion to test and development sets. The network was trained for 5 epochs, if F1-score did not improve for 2 epochs, the training was stopped. Training was performed using Adam optimizer with default settings. Our network is implemented using Keras[6].

6. Model application

6.1. Input and output format

Training data consists of raw text sentences together with their annotation. The annotation consists of two parts, the first is the binary label (0/1) indicating whether a gap occurs in the current sentence. For the sentences containing the gap the second part generally contains 6 pairs of numbers, as shown on **Figure 4**. The first pair of numbers refers to the main predicate (typically, a single verb), and the second and the third to its core arguments; the fourth one contains the empty span of the gap, the two remaining ones label the positions of gap predicates. All offsets are given in characters.

```
Некоторые торговцы отлично продают один вид товаров или услуг, а другие - другой.
                   1   27:34   0:18   35:39   74:74   65:71   74:80
```

Figure 4: A typical example of input data sentence

[6] https://github.com/AlexeySorokin/Gapping

For some sentences, the number of input spans differ: if the gapped verb has only one argument remaining, such as *поперек* the one on **Figure 5.1**, then there are only only 4 spans present. On the contrary, if more the sentence contains $r > 1$ gaps, as in **Figure 5.2**, (*домишко* (*казался*) *дворцом*, *дома* (*казались*) *небоскребами*) and the gapped verb has k arguments, then the total number of input spans is $r(k+1)$ (typically k equals 2).

Сосиску разрезаем вдоль на две половинки, затем поперек.
1 8:17 18:23 48:48 48:51

Я смотрел, и каждая улочка казалась мне изящной аллеей, каждый домишко — дворцом, а двухэтажные дома — небоскребами.
1 27:35 13:26 40:54 73:73 103:103 56:70 84:100 73:80 103:115

Figure 5: Other examples of input data sentences

To be evaluated, a system should provide the output of the same format as the input given. If the system does not predict the span, the corresponding column is left empty.

6.2. Data pre- and postprocessing

Since both our systems operate on the level of subtree heads, not the character-based spans, we need to convert the input data to appropriate format before applying the model and transform it back when submitting the output. Input conversion is required on two stages: to prepare training data and to evaluate system output on validation data.

First, character-level spans are converted to word-level spans using the NLTK tokenizer (we also tried the UDPipe one but found it to perform worse). Second, for each span the subtree head is determined using the automatically obtained parse tree. We select as possible heads all words in the span whose parent lies outside this span. If the parse tree is correct, there is only one such head. If multiple heads are returned due to parsing errors, we exclude the instance during training or return None during validation.

During model evaluation we need to convert its predictions back to the competition format. Subtree heads are transformed to word-level spans using the procedure described in **Section 3** and word-level spans are converted to character-level to produce the final answer. We note that this procedure is prone to errors due to incorrect parsing and (potentially) tokenization.

7. Results and discussion

We present the evaluation of the entire system using the official evalution script[7] as well as our own metrics. We pay more attention to our own metrics since they allow to evaluate separate stages of the pipeline.

[7] https://github.com/dialogue-evaluation/AGRR-2019/blob/master/agrr_metrics.py

7.1. Individual models evaluation

Our gapping model essentially consists of model for 3 individual subtasks: gap location, gapped predicates location and remnants matching. Additionally, to predict argument spans, the stage of span prediction is required. We present scores for each of the stages separately, passing the gold input to them (see Table 2 for the list of inputs and outputs for each phase of the pipeline). Since our models return subtree heads, their predictions are judged against the subtree heads extracted from correct spans, when our input preprocessor fails to extract such span, any output is considered as invalid. We evaluate the model on the level of individual inputs, not sentences. We collect all input-output tuples and calculate the number of true positives (present both in gold and predicted answers), false positives and false negatives. For all the tasks except gap location we also count the number of partially correct answers, which is the number of input-output pairs for which more than one half of tuple elements was predicted correctly. These partial matches are added to the number of true positives with weight 0.5 when calculating precision, recall and F1-measure. The metrics are reported using the official test set and the gold answers on it. We compare 3 models: the rule-based one, the full neural model and the ensemble of 3 neural models.

Table 3: Quality of individual pipeline stages on test set of AGRR-2019 competition

Stage	Model	TP	FP	FN	partial	Precision	Recall	F1
Gap location	Rule-based	556	219	180	0	71.74	75.54	73.59
	Neural (single)	663	50	73	0	92.99	90.08	91.51
	Neural (ensemble)	1397	86	103	0	**94.20**	**93.13**	**93.66**
Predicate location	Rule-based	615	2	54	67	94.81	88.11	91.34
	Neural (single)	649	0	13	74	94.88	93.21	94.04
	Neural (ensemble)	1378	0	10	112	**96.24**	**95.60**	**95.92**
Remnant matching	Rule-based	475	0	208	53	94.98	68.14	79.35
	Neural (single)	557	3	106	73	93.76	80.64	86.71
	Neural (ensemble)	576	4	107	53	**95.18**	**81.86**	**88.02**
Span prediction	Rule-based	520	0	116	100	91.94	77.45	84.07

Table 3 demonstrates that rule-based model definitely looses to the neural one. Moreover, the only rule-based component of the neural pipeline, span prediction, occurs to be the weakest part of it. To measure relative impact of different pipeline components, we score the output after each stage of the pipeline.

Table 4 supports the conclusion made from individual stages evaluation: neural model is significantly stronger than the rule-based one. Though their overall performance is comparable in terms of precision, the recall of the neural model is definitely higher. Note that after remnant matching stage the precision of rule-based model goes up because this phase eliminates some arguments of the gapped verb that were incorrectly predicted on the previous stages and do not match any arguments of the main verb in the sentence. It is also the only stage, where individual rule-based model is comparable with the neural one at least in precision terms, however it is achieved at the expense of significant decrease of recall.

Table 4: Quality after each stage of the pipeline on test set of AGRR-2019 competition

Stage	Model	TP	FP	FN	partial	Precision	Recall	F1
Gap location	Rule-based	556	219	180	0	71.74	75.54	73.59
	Neural (single)	663	50	73	0	92.99	90.08	91.51
	Neural (ensemble)	672	45	64	0	**93.72**	**91.30**	**92.50**
Predicate location	Rule-based	537	170	131	68	73.68	77.58	75.58
	Neural (single)	598	31	69	69	90.62	85.94	88.21
	Neural (ensemble)	617	27	62	57	**92.08**	**87.70**	**89.84**
Remnant matching	Rule-based	398	34	266	72	86.11	58.97	70.00
	Neural (single)	527	22	87	122	87.63	79.89	83.58
	Neural (ensemble)	561	19	75	100	**89.85**	**83.02**	**86.30**
Span prediction	Rule-based	322	45	277	137	77.48	53.06	62.98
	Neural (single)	403	38	103	230	77.20	70.38	73.63
	Neural (ensemble)	436	38	94	206	**79.77**	**74.67**	**76.13**

7.2. AGRR-2019 evaluation metrics

In Table 5 we score different stages of the pipeline using the official evaluation metrics of the competition. The organizers evaluated the performance both in term of binary detection of gapping in a sentence, as well as complete gapping resolution, which uses character-wise precision and recall. Since full resolution can be performed only when all the answers are available, intermediate stages are evaluated using only binary metrics.

Table 5: Official evaluation metrics of AGRR-2019 competition after each stage of the pipeline on development (left) and test (right) set

		Binary						Full resolution	
Stage	Model	Precision		Recall		F1			
Gap location	Rule-based	80.6	80.6	82.9	82.5	81.7	81.5	—	—
	Neural (single)	96.1	96.2	93.7	92.5	94.9	94.3	—	—
	Neural (ensemble)	*97.2*	*97.0*	**94.6**	**92.9**	**95.9**	**95.0**	—	—
Predicate location	Rule-based	80.6	80.6	82.9	82.5	81.7	81.5	—	—
	Neural (single)	96.3	96.3	93.3	91.8	94.7	94.0	—	—
	Neural (ensemble)	*97.2*	*97.0*	*94.4*	*92.2*	*95.8*	*94.6*	—	—
Remnants matching	Rule-based	93.1	93.4	63.7	64.6	75.6	76.3	—	—
	Neural (single)	97.4	97.3	91.2	89.1	94.2	93.0	—	—
	Neural (ensemble)	**98.0**	**97.9**	*92.0*	*90.1*	*94.9*	*93.9*	—	—
Span prediction	Rule-based	93.1	93.4	63.7	64.6	75.6	76.3	59.3	60.2
	Neural (single)	97.4	97.3	91.2	89.1	94.2	93.0	87.5	85.3
	Neural (ensemble)	**98.0**	**97.9**	*91.4*	*90.1*	*94.9*	*93.9*	**89.1**	**87.1**

Almost always scores decrease between subsequent stages of the pipeline. The only exception is the rule-based model, where gaps and predicates are located using the same model, therefore there accuracies coincide. On the stage of remnants matching the precision of rule-based model grows up since most incorrectly predicted predicates are not matched with any remnant and are therefore rejected. However, for a significant fraction of correct predicates matches are also not found which deteriorates the overall performance.

The first thing to note is solid performance of our model in terms of precision even without ensembling. Its recall is also rather high: the best scores achieved by AGRR-2019 participants [8] during evaluation were 95.9% for binary gap detection and 89.2% for full resolution, so we are about two percents behind[9].

Our comparison demonstrates the clear superiority of neural models in all phases of the pipeline. However, the training set provided by the organizers of AGRR-2019 was rather large and requires huge amount of manual effort in its collection. It is natural to ask whether neural models can be used in low-resource setting. We selected first 1,600 sentences of the training data for training and 400 sentences of development data for tuning and trained an ensemble of 3 models on this smaller dataset. The results in comparison with our large model is given are given in Table 6.

Table 6: Official evaluation metrics of AGRR-2019 for neural model trained on smaller dataset on development (left) and test (right) set

Stage	Model	Binary						Full resolution	
		Precision		Recall		F1			
Gap location	Rule-based	80.6	80.6	82.9	82.5	81.7	81.5	—	—
	Small neural (ensemble)	96.0	*98.2*	86.2	81.2	90.8	88.9	—	—
	Neural (ensemble)	*97.2*	97.0	**94.6**	**92.9**	**95.9**	**95.0**	—	—
Predicate location	Rule-based	80.6	80.6	82.9	82.5	81.7	81.5	—	—
	Small neural (ensemble)	96.0	**98.2**	85.9	81.2	90.7	88.9	—	—
	Neural (ensemble)	*97.2*	97.0	*94.4*	*92.2*	*95.8*	*94.6*	—	—
Remnants matching	Rule-based	93.1	93.4	63.7	64.6	75.6	76.3	—	—
	Small neural (ensemble)	97.4	**98.6**	76.7	71.5	85.8	82.9	—	—
	Neural (ensemble)	**98.0**	*97.9*	*92.0*	*90.1*	*94.9*	*93.9*	—	—
Span prediction	Rule-based	93.1	93.4	63.7	64.6	75.6	76.3	59.3	60.2
	Small neural (ensemble)	97.4	**98.6**	76.7	71.5	85.8	82.9	73.9	69.4
	Neural (ensemble)	**98.0**	*97.9*	*91.4*	*90.1*	*94.9*	*93.9*	**89.1**	**87.1**

We observe that models trained on smaller sample of data do not loose in precision, however, recall significantly decreases at all stages of the pipeline. Nevertheless, they still outperform the rule-based model. We observed severe overfitting on small datasets, which means that several parameters of the model (e. g., interlayer dropout

[8] https://github.com/dialogue-evaluation/AGRR-2019

[9] We note again that only our rule-based system was submitted during the competition, therefore other participants could potentially improve their scores as well.

or number of recurrent units) must be altered. The recall weight should be also adjusted even further than it is in the basic model. Nevertheless, even datasets of medium size allow to train neural gap detectors that outperform rule-based recognizers.

8. Conclusions

We have designed a high-quality neural model for gap resolution for Russian language, whose quality achieves 93% for binary gap detection and 89% for full gap resolution. The model is based on ELMo embeddings and recurrent neural networks. The are at least three directions for future research: the first is to apply the model to other languages, such as English, Spanish or Czech. Since model architecture is language independent, the main obstacle can be relative lack of data. Though our model shows solid performance for only 2,000 training sentences available, they are much lower than the scores of the model trained on larger datasets. Probably, curated generation of training data (especially negative examples) can make the model more robust. Another problem is the usage of ELMo embeddings which are not available for many languages and whose learning is time- and resource-consuming. The simplest solution is to replace ELMo with newer multilingual BERT [4] which demonstrated high performance in other works of AGRR-2019 competition.

Another problem to investigate is the role of morphological and syntactic information. We found that it does not have stable effect on the full dataset, however, with smaller data its significance can be higher. The main improvement can be achieved in the weakest part of the model: the detection of constituent bounds using a syntactic parser. The simplest way to enhance performance is to retrain the parser with more gapped sentences and thus increase its ability to parse such sentences. Alternatively, one may directly solve the task of constituent bound detection without reducing to syntactic parsing. We leave these questions for future research.

Acknowledgements

The author thanks the organizers of AGRR-2019 competition for giving the opportunity to participate and for their helpful cooperation during the Shared Task. I am also very grateful to the stuff of MIPT Neural Networks laboratory for warm and inspiring atmosphere during the work on the problem. I deeply thank the anonymous reviewers whose comments and suggestions helped to improve the paper.

References

1. *Burtsev M. et al.:* DeepPavlov: Open-Source Library for Dialogue Systems (2018), Proceedings of ACL 2018, System Demonstrations, Melbourne, Australia, pp. 122–127.
2. *Carnie A.:* Syntax: A generative introduction (2006), Blackwell.

3. *Conneau A. et al.:* Supervised learning of universal sentence representations from natural language inference data (2017), arXiv preprint arXiv:1705.02364, available at https://arxiv.org/pdf/1705.02364.pdf.
4. *Devlin J. et al.:* Bert: Pre-training of deep bidirectional transformers for language understanding (2018), arXiv preprint arXiv:1810.04805, available at https://arxiv.org/pdf/1810.04805.pdf.
5. *Droganova K., Zeman D.:* Elliptic Constructions: Spotting Patterns in UD Treebanks (2017), Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017), Gothenburg, Sweden, pp. 48–57.
6. *Droganova K. et al.:* Parse Me if You Can: Artificial Treebanks for Parsing Experiments on Elliptical Constructions (2018), Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018), Miyazaki, Japan, pp. 1845–1852.
7. *Droganova K. et al.:* Mind the Gap: Data Enrichment in Dependency Parsing of Elliptical Constructions (2018), Proceedings of the Second Workshop on Universal Dependencies (UDW 2018), Brussels, Belgium, pp. 47–54.
8. *Jackendoff R. S.:* Gapping and related rules (1971), Linguistic inquiry, Vol. 2., 1., pp. 21–35.
9. *Johnson K.:* Gapping (2014), manuscript, available at http://people.umass.edu/kbj/homepage/Content/gapping.pdf.
10. *Heigold G., Neumann G., van Genabith J.:* An extensive empirical evaluation of character-based morphological tagging for 14 languages (2015), Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers., Vol. 1., pp. 505–513.
11. *Lakoff G., Ross J. R.:* Gapping and the order of constituents (1970), Progress in linguistics: A collection of papers., Vol. 43., p. 249.
12. *Nivre J. et al.:* (2016), Universal Dependencies v1: A Multilingual Treebank Collection, Language Resources and Evaluation (LREC), Portoroz, pp. 1659–1666.
13. *Peters M. E. et al.:* Deep contextualized word representations (2018), arXiv preprint arXiv:1802.05365.
14. *Schuster S., Lamm M., Manning C. D.:* Gapping Constructions in Universal Dependencies v2 (2017), Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017), Gothenburg, Sweden, pp. 123–132.
15. *Schuster S., Nivre J., Manning C. D.:* Sentences with Gapping: Parsing and Reconstructing Elided Predicates (2018), arXiv preprint arXiv:1804.06922, available at https://arxiv.org/pdf/1804.06922.pdf.
16. *Smurov I. et al.:* AGRR 2019: Automatic Gapping Resolution for Russian (2019), International conference on computational linguistics “Dialogue”, to appear.
17. *Sorokin A.:* Improving neural morphological Tagging using Language Models (2018), International conference on computational linguistics “Dialogue”., Vol 1., pp. 707–720.
18. *Straka M.:* UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task (2018), Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies., Brussels, Belgium, pp. 197–207.