Записки научных
семинаров ПОМИ
Том 499, 2021 г.

T. Khakhulin, V. Logacheva, V. Malykh

ROBUST WORD VECTORS: CONTEXT-INFORMED EMBEDDINGS FOR NOISY TEXTS

Abstract. We suggest a new language-independent architecture of robust word vectors (RoVe). It is designed to alleviate the issue of typos and misspellings, common in almost any user-generated content, which hinder automatic text processing. Our model is morphologically motivated, which allows it to deal with unseen word forms in morphologically rich languages. We present the results on a number of natural language processing (NLP) tasks and languages for a variety of related architectures and show that the proposed architecture is robust to typos.

§1. Introduction

Rapid growth in the usage of mobile electronic devices has increased the number of user input text issues such as typos. This happens because typing on a small screen and in transit (while walking, on public transport etc.) is difficult, and people accidentally hit incorrect keys more often than on a standard desktop keyboard. Spell-checking systems widely used in web services can handle this issue, but they can also make mistakes.

Meanwhile, any text processing system is now impossible to imagine without word embeddings, vectors that encode semantic and syntactic properties of individual words [2]. However, to use these word vectors the user input should be clean, i.e., free of misspellings and typos, because a word vector model trained on clean data will not contain misspelled versions of words. There are examples of models trained on noisy data [17], but this approach does not fully solve the problem: typos are unpredictable, and a corpus cannot contain all possible incorrectly spelled versions of a word and, naturally, it is impossible to obtain a sufficiently large corpus to gather sufficient statistics for all possible typos. Instead, we suggest that one should make algorithms for word vector modeling robust to noise.

Key words and phrases: word vectors, distributed representations, natural language processing.

This work was supported by the National Technology Initiative and PAO Sberbank project ID 0000000007417F630002.

We suggest a new architecture **RoVe** (Robust Vectors)[1]. The main feature of this model is its open vocabulary. It encodes words as sequences of characters, which enables the model to produce embeddings for out-of-vocabulary (OOV) words. The idea as such is not new, and many other models either use character-level embeddings [18] or encode the most common n-grams to assemble unknown words from them [4]. However, unlike similar models, *RoVe* is specifically targeted at typos: it is invariant to swapping characters in a word. This property is ensured by the fact that each word is encoded as a bag of characters. At the same time, word prefixes and suffixes are encoded separately, which enables RoVe to produce meaningful embeddings for unseen word forms in morphologically rich languages. Notably, this is done without explicit morphological analysis.

Another feature of RoVe is context dependency; in order to generate an embedding for a word one should also take into account its context. The motivation for such architecture is as follows. Our intuition is that when processing an OOV word our model should produce an embedding similar to that of some similar word from the training data. This behaviour is suitable for typos as well as unseen forms of known words. In the latter case, we would like a word to get an embedding similar to the embedding of its initial form. This process reminds lemmatization (reducing a word to its initial form). Lemmatization is context-dependent since it often needs to resolve homonymy based on word's context. By making the *RoVe* model context-dependent we enable it to perform such implicit lemmatization.

We compare *RoVe* with common word embedding tools: *word2vec* [21] and *fasttext* [4]. We test the models on three tasks: paraphrase detection, identification of textual entailment, and sentiment analysis, and three languages with different linguistic properties: English, Russian, and Turkish.

The paper is organised as follows. In Section 2 we review previous work on the subject. Section 3 contains the description of the model's architecture. In Section 4 we describe the experimental setup, and report evaluation results in Section 5. Section 6 concludes the paper and outlines future work.

§2. Related work

Out-of-vocabulary (OOV) words are a major problem for word embedding models. The commonly used *word2vec* model does not have any

[1]We have made an open-source implementation available here: `https://gitlab.com/madrugado/robust-w2v`.

means of dealing with them. OOV words can be rare terms, unseen forms of known words, or simply typos, and different types of OOV words may require different approaches. The majority of research has concentrated on unknown terms and generation of word forms, and very few works targeted typos.

The work [18] presents an open-vocabulary word embedding model, where word vectors are computed with an RNN over character embeddings. The authors claim that their model implicitly learns morphology, which makes it suitable for morphologically rich languages. However, it is not robust against typos. Another approach is to train a model that approximates original embeddings and encodes unseen words to the same vector space. The work [24] approximates pretrained word embeddings with a character-level model. In [3], pre-trained embeddings are projected to a lower-dimensional space, which allows the authors to train meaningful embeddings for new words from scarce data. However, initial word embeddings needed for these approaches cannot be trained on noisy data.

To tackle noisy training data, the work [22] trains a neural network that filters word embeddings. To do that, the authors take a pretrained word embedding model and learn a matrix transformation in order to denoise it. The transformation makes word embeddings more robust to statistical artifacts in training data. Unfortunately, this does not solve the problem of typos in test data since the model still has a closed vocabulary.

There are examples of embeddings targeted at unseen word forms. In [35], sub-word embeddings are trained with the purpose of combining them into a word embedding via a recurrent (RNN) or convolutional (CNN) neural network. The atomic units here are characters or morphemes. Morphemes give better results in machine translation, in particular for morphologically rich languages. This method yields high-quality embeddings but requires to train a separate model for morphological analysis.

Some other models are targeted at encoding rare words. The *fasttext* model [4] produces embeddings for the most common n-grams of variable length, and an unknown word can be encoded as a combination of its n-grams. This is beneficial for encoding of compound words which are very common in German and occasionally occur in English and other languages. However, such a model is still not well suited for handling typos.

Unseen words are usually represented with subword units (morphemes or characters). This idea has been extensively used in research on word vector models. It not only gives a possibility to encode OOV words but has also been shown to improve the quality of embeddings. The work [38] was the first to show that character-level embeddings trained with a CNN can store the information about semantic and grammatical features of words. They tested these embeddings on multiple downstream tasks. In [29] character-level CNNs are used for intrusion detection, and the work [36] builds a language-independent sentiment analysis model using character-level embeddings, which would be impossible with word-level representations.

Unlike these works, we do not train character embeddings or models for combining them; these are defined deterministically. This spares us the problem of too long character-level sequences which are difficult to encode with RNNs or CNNs. We bring the meaning to these embeddings by making them context-dependent.

It was recently suggested that word context matters not only in general (i.e., word contexts define its meaning), but also in each individual case of word usage. This has resulted in the emergence of word vector models that produce word embeddings with respect to a word's local context. There is evidence that contextualising pre-trained embeddings improves them [13] and raises quality of downstream tasks, e.g., machine translation [20] or question answering [23].

§3. Model architecture

RoVe combines context dependency and open vocabulary, which allows to generate meaningful embeddings for OOV words. These two features are supported by the two parts of the model (see Fig. 1).

3.1. Encoding the context. The *RoVe* model produces context-dependent word representations. It means that it does not generate a fixed vector for a word and needs to produce it from scratch for every occurrence of the word. This structure marginally increases text processing time but yields more accurate, context-informed word representations. The model is conceptually similar to an encoder used to create the representation of a sentence by reading all its words. Such encoders have been used for machine translation [20], question answering [32], and many other tasks.

In order to generate a representation of a word, we need to encode it together with its context. For every context word, we first produce its input

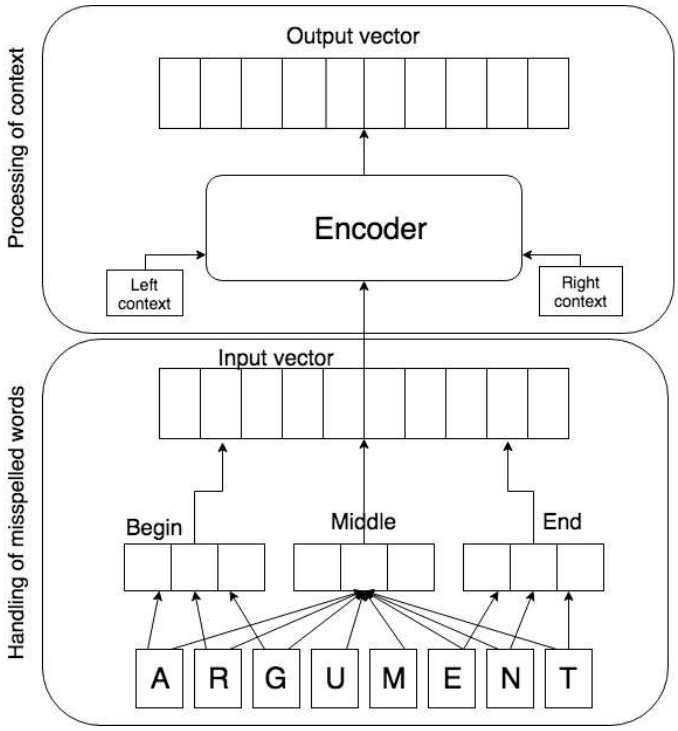

Figure 1. *RoVe* model: generating an embedding for the word *argument*.

embedding (described in Section 3.2). This embedding is then passed to the encoder (top part of Fig. 1) which processes all words from the context. The encoder should be a neural network that can process a string of words and keep the information on their contexts. The most obvious choices are an RNN or a CNN. However, a different type of network could also be used. Having processed the whole context, we obtain an embedding for the target word by passing a hidden state corresponding to the word in question through a fully-connected layer. Therefore, we can generate embeddings for all words in a context simultaneously.

3.2. Handling of misspelled words. Another important part of the model is the transformation of an input word into a fixed-size vector (input

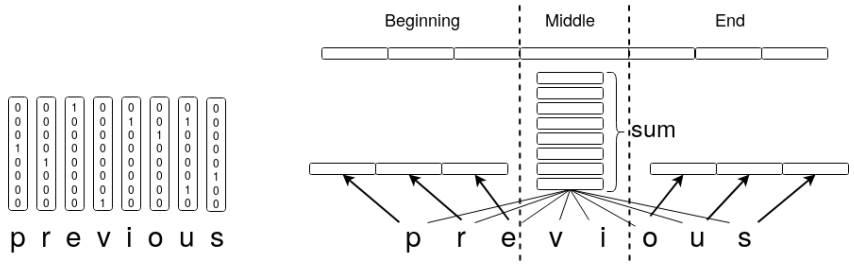

Figure 2. Generation of input embedding for the word *previous*. Left: generation of character-level one-hot vectors, right: generation of BME representation.

embedding). This transformation is shown in the bottom part of Figure 1. This is a deterministic procedure, uniquely defined for a given word; it does not need training. It is executed as follows.

First, we represent every character of a word as a one-hot vector (alphabet-sized vector of zeros with a single 1 in position i where i is the index of the character). Then, we generate three vectors: beginning (**B**), middle (**M**), and end (**E**) vectors. The **M** vector is the sum of one-hot vectors of all characters of a word. The **B** vector is a concatenation of one-hot vectors for n_b first characters in the word. Likewise, the **E** vector is a concatenation of one-hot vectors of n_e last characters in the word. The values of n_b and n_e are hyperparameters that can vary for different languages and datasets. We form the input embedding by concatenating **B**, **M**, and **E** vectors. Therefore, its length is $(n_b + n_e + 1) \times |A|$, where A is the alphabet of a language. This input embedding is further processed by the neural network described above. The generation of the input vector is shown in Fig. 2.

We further refer to this three-part representation as **BME**. It was inspired by the work [28] where the first and the last characters of a word are encoded separately as they carry more meaning than other characters. However, the motivation for our BME representation stems from dividing the words into morphemes. We encode n_b first characters and n_e last characters of a word in a fixed order (as opposed to the rest of the word which is saved as a bag of letters) because we assume that it can be an affix that carries a particular meaning (e.g., the English prefix *un* carries the meaning of reversed action or absence) or grammatical information (e.g.,

the English suffix *ed* indicates the past participle of a verb). Thus, keeping it can make the resulting embedding more informative.

The **M** part of the input embedding discards the order of letters in a word. This feature guarantees the robustness of embeddings against swaps of letters within a word, which is one of the most common typos. Compare *information* and *infromation* that will have identical embeddings in our model, whereas *word2vec* and many other models will not be able to provide any representation for the latter word.

In addition to that, BME representation is not bounded by any vocabulary and is able to provide an embedding for any word, including words with typos. Moreover, if a misspelled word is reasonably close to its original version, its embedding will also be close to that of the original word. This feature is ensured by character-level generation of input embedding: close input representations will yield close vectors. Therefore, even a misspelled word is likely to be interpreted correctly.

The use of our model alleviates the need for spelling correction, because a word does not need to be spelled correctly to be successfully interpreted. Unlike other models that support typos, *RoVe* can handle noise in both training and inference data.

3.3. Training. The *RoVe* model is trained with the *negative sampling* procedure suggested by [33]. We use it as described in [21]. This method serves to train vector representations of words. The fundamental property of word vectors is the small distance between vectors of words with close meanings and/or grammatical features. In order to enforce this similarity, it was suggested that training objective should be twofold: in addition to pushing vectors of similar words close to each other we should increase the distance between vectors of unrelated words. This objective corresponds to a two-piece loss function shown below in equation 1. Here, w is the target word, v_i are positive examples from the context (C), and v_j are negative examples (Neg) randomly sampled from the data. The function $s(\cdot, \cdot)$ is a similarity score for two vectors that should be increasing as the vectors get closer to each other. For our experiments we used cosine similarity because it is computationally simple and does not contain any parameters.

The first part of the loss rewards close vectors of similar words, and the second part penalises close vectors of unrelated words. Words from a window around a particular word are considered to be similar to it since they have a common context. Unrelated words (negative examples) are sampled randomly from data, hence the name of the procedure.

Formally, our model is trained using the following objective:

$$L = \sum_{v_i \in C} e^{s(w,v_i)} + \sum_{v_j \in Neg} e^{-s(w,v_j)} \quad (1)$$

Converting the words into input embeddings is a deterministic procedure, so during training we only update parameters of the neural network that generates context-dependent embeddings and fully-connected layers that precede and follow it.

§4. Experimental setup

We test the performance of word vectors generated with *RoVe* on three tasks:

- paraphrase detection,
- sentiment analysis,
- identification of text entailment.

For all tasks, we train simple baseline models. This is done deliberately to make sure that the performance is largely defined by the quality of vectors that we use. For all the tasks we compare word vectors generated by different modifications of *RoVe* with vectors produced by *word2vec* and *fasttext* models.

We conduct experiments on datasets for three languages: English (analytical language), Russian (synthetic fusional), and Turkish (synthetic agglutinative). Affixes have different structures and purposes in these types of languages, and in our experiments we show that our BME representation is effective for all of them. We did not tune n_b and n_e parameters (lengths of **B** and **E** segments of BME). In all our experiments we set them to 3, following the fact that the average length of affixes in Russian is 2.54 [25]. However, they are not guaranteed to be optimal for English and Turkish.

4.1. Baseline systems. We compare the performance of *RoVe* vectors with vectors generated by two most commonly used models: *word2vec* and *fasttext*. We use the following *word2vec* models:

- **English** – pretrained *Google News* word vectors[2],
- **Russian** – pretrained word vectors *RusVectores* [14],
- **Turkish** – we trained a model on the "42 bin haber" corpus [37].

[2] `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing`

We stem Turkish texts with SnowBall stemmer [8] and lemmatize Russian texts with the *Mystem* tool[3] [31]. This is done in order to reduce the sparsity of the text and interpret rare word forms. In English this problem is not as severe because it has a less developed morphology. In addition to that, *Google News* vectors were trained on much larger corpus which has occurrences of most of forms for common words.

As *fasttext* baselines we use official pretrained *fasttext* models.[4] We also tried an extended version of the *fasttext* baseline for Russian and English: *fasttext* augmented with a spell checker. For downstream tasks we checked the texts with a publicly available spell checker[5] before extracting word vectors. Since spell-checking is a very common way to reduce the effect of typos, we wanted to compare its performance with *RoVe*.

4.2. Infusion of noise. In order to demonstrate the robustness of *RoVe* against typos we artificially introduced noise into our datasets. We model:

- random insertion of a letter,
- random deletion of a letter.

For each input word we randomly insert or delete a letter with a given probability. Both types of noise are introduced at the same time. We test models with different levels of noise from 0% (no noise) to 30%. According to [5], the real level of noise in user-generated texts is 10-15%. We add noise only to the data for downstream tasks, *RoVe* and *word2vec* models are trained on clean data.

4.3. Encoder parameters. The model as described in Section 3 is highly configurable, with many parameters and architectural tweaks subject to change. The main decision to be made when experimenting with the model is the architecture of the encoder. We experiment with RNNs and CNNs, conducting our experiments with the following RNN architectures:

- **Long Short-Term Memory (LSTM)** unit [11] – a unit that mitigates problem of vanishing and exploding gradients that is common when processing of long sequences with RNNs. We use two RNN layers with LSTM cells;

[3] `https://tech.yandex.ru/mystem/`

[4] English: `https://fasttext.cc/docs/en/english-vectors.html`, Russian and Turkish: `https://fasttext.cc/docs/en/crawl-vectors.html`

[5] `https://tech.yandex.ru/speller/`

- **bidirectional LSTM** [30] – two RNNs with LSTM units where one RNN reads a sequence from beginning to end and another one backward;
- **stacked LSTM** [10] – an RNN with multiple layers of LSTM cells; this allows to combine the forward and backward layer outputs and use them as input to the next layer; here, we have experimented with two bidirectional RNN layers with stacked LSTM cells;
- **Simple Recurrent Unit (SRU)** [15] – LSTM-like architecture which is faster due to parallelization;
- **bidirectional SRU** – bidirectional RNN with SRU cells.

We have also tried the following convolutional architectures:

- **CNN-1d** – one-dimensional convolutional neural network as in [12]; this model used 3 convolution layers with kernel sizes 3, 5, and 3 respectively;
- **ConvLSTM** – a combination of CNN and a recurrent network; we first applied the CNN-1d model and then produced vectors as in the biSRU model from a lookup table.

The sizes of hidden layers for RNNs as well as the sizes of fully-connected layers of the model are set to 256 in all experiments.

4.4. RoVe models. We trained our *RoVe* models on the following datasets:

- English – *Reuters* dataset [16],
- Russian – *Russian National Corpus* [1],
- Turkish – *42 bin haber* corpus.

All *RoVe* models are trained on original corpora without adding noise or any other preprocessing. The *RoVe* model for Turkish is trained on the same corpora as the one we used to train the *word2vec* baseline, which makes them directly comparable. For English and Russian we compare *RoVe* models with third-party *word2vec* models trained on larger datasets. We also tried training our *word2vec* models on training data used for *RoVe* training. However, these models were of lower quality than pretrained *word2vec*, so we do not report results for them.

§5. Results

5.1. Paraphrase detection. The task of paraphrase detection is formulated as follows: given a pair of phrases, we need to predict if they have the same meaning. We compute cosine similarity between vectors for phrases.

	English			Russian			Turkish		
noise (%)	0	10	20	0	10	20	0	10	20
BASELINES									
word2vec	0.715	0.573	0.564	0.800	0.546	0.535	0.647	0.586	0.534
fasttext	**0.720**	0.594	0.587	0.813	0.645	0.574	0.632	0.595	0.514
fasttext+spellcheck	**0.720**	0.598	0.585	0.813	0.693	0.453	–	–	–
RoVe									
stackedLSTM	0.672	0.637	0.606	0.723	0.703	**0.674**	0.601	0.584	0.536
SRU	0.707	0.681	0.641	0.823	0.716	0.601	0.647	0.602	0.568
biSRU	0.715	**0.687**	**0.644**	**0.841**	**0.741**	0.641	**0.718**	**0.641**	**0.587**

Table 1. Results of the paraphrase detection task in terms of ROC AUC.

High similarity is interpreted as a paraphrase. Phrase vectors are computed as the average of vectors of words in a phrase. For *word2vec* we discard OOV words since the model cannot generate embedding for them. We measure the performance of models on this task with the ROC AUC metric [9] which defines the proportions of true positive answers in the system's outputs with a varying threshold.

We run experiments on three datasets:

- **English** – *Microsoft Research Paraphrase Corpus* [7] that consists of 5,800 sentence pairs extracted from news sources on the web and manually labelled for presence/absence of semantic equivalence;
- **Russian** – *Russian Paraphrase Corpus* [26] that consists of news headings from different news agencies; it contains about 6,000 pairs of phrases labelled in terms of a ternary scale: "-1" – not paraphrase, "0" – weak paraphrase, and "1" – strong paraphrase; we used only "-1" and "1" classes for consistency with other datasets; there are 4,470 such pairs;
- **Turkish** – *Turkish Paraphrase Corpus* [6] that contains 846 pairs of sentences from news texts manually labelled for semantic equivalence.

The results of this set of experiments are presented in Table 1. Due to limited space we do not report results for all noise levels and list only figures for 0%, 10% and 20% noise. We also omit the results from most *RoVe* variants that never beat the baselines.

As we can see, none of the systems are completely robust to typos: their quality falls as we add noise. However, this decrease is much sharper for baseline models, which means that *RoVe* is less sensitive to typos. Figure 3

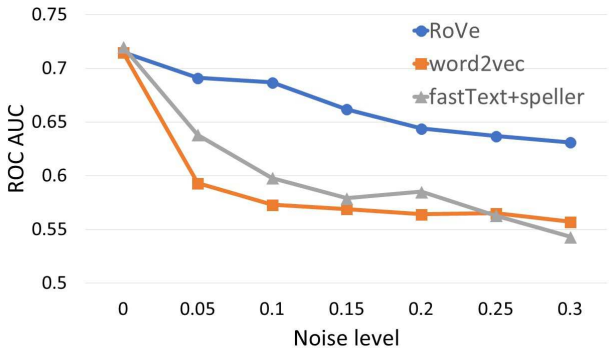

Figure 3. Comparison of *RoVe* with *word2vec* and *fasttext* on texts with a growing amount of noise (paraphrase detection task for English).

	English			**Russian**		
noise (%)	0	10	20	0	10	20
BASELINES						
word2vec	0.649	0.611	0.554	0.649	0.576	0.524
fasttext	**0.662**	0.615	0.524	0.703	0.625	0.524
fasttext + spellcheck	0.645	0.573	0.521	0.703	0.699	0.541
RoVe						
stackedLSTM	0.621	0.593	0.586	0.690	0.632	0.584
SRU	0.627	0.590	0.568	0.712	0.680	0.598
biSRU	0.656	**0.621**	**0.598**	**0.721**	**0.699**	**0.621**

Table 2. Results of the sentiment analysis task in terms of ROC AUC.

shows that while all models show the same result on clean data, *RoVe* outperforms the baselines as the level of noise goes up. Of all *RoVe* variations, bidirectional SRU gives the best result, marginally outperforming SRU.

Interestingly, the use of a spellchecker does not guarantee improvement: the *fasttext+spellcheck* model does not always outperform vanilla *fasttext*, and its score is unstable. This might be explained by the fact that the spellchecker makes mistakes itself; for example, it can occasionally change a correct word into a wrong one.

5.2. Sentiment analysis. The task of sentiment analysis consists in determining the emotion of a text (positive or negative). For this task we use word vectors from different models as features for a naive Bayes classifier (again, a very simple model used to compare the embeddings as directly as possible). The evaluation is performed with the ROC AUC metric. We experiment with two datasets:

- **English** – Stanford Sentiment Treebank [34]; in this corpus the objects are labelled with three classes–positive, negative, and neutral–abd we use only the former two;
- **Russian** – Russian Twitter Sentiment Corpus [19]; it consists of 114,911 positive and 111,923 negative records. Since tweets are noisy, we do not add noise to this dataset and use it as is.

Results for this task (see Table 2) confirm the results reported in the previous section: the biSRU model outperforms others, and the performance of *word2vec* is markedly affected by noise. On the other hand, *RoVe* is more resistant to it.

5.3. Identification of text entailment. This task is devoted to the identification of logical entailment or contradiction between the two sentences. We experiment with the *Stanford Natural Language Inference* corpus [27] labelled with three classes: *contradiction*, *entailment*, and *no relation*. We do not use *no relation* in order to reduce the task to binary classification. The setup is similar to the one for paraphrase detection task: we define the presence of entailment by cosine similarity between phrase vectors, which are represented as averaged vectors of words in a phrase. Pairs of phrases with high similarity score are assigned *entailment* class and the ones with low score are assigned *contradiction* class. The quality metric is ROC AUC.

The results for this task are shown in Table 3. They fully agree with those obtained on the other tasks: *RoVe* with biSRU cells outperforms the baselines and the gap between them gets larger as more noise is added. Note also that here a spellchecker deteriorates the performance of *fasttext*.

5.4. Types of noise. All the results reported above were tested on datasets with two types of noise (insertion and deletion of letters) applied simultaneously. Our model is by definition invariant to letter swaps, so we did not include this type of noise in the experiments. However, a swap does not change an embedding of a word only when this swap happens outside the **B** and **E** segments of a word, otherwise the embedding changes as **B**

	English		
noise (%)	0	10	20
BASELINES			
word2vec	0.624	0.593	0.574
fasttext	0.642	0.563	0.517
fasttext + spellcheck	0.642	0.498	0.481
RoVe			
stackedLSTM	0.617	0.590	0.516
SRU	0.627	0.590	0.568
biSRU	**0.651**	**0.621**	**0.598**

Table 3. Results of the task on identification of textual entailment.

and **E** keep the order of letters. Therefore, we compare the effect of random letter swaps.

We compare four types of noise:

- only insertion of letters,
- only deletion,
- insertion and deletion (original setup),
- only letter swaps.

Similar to the noise infusion procedure for insertion and deletion, we swap two adjacent characters in a word with probabilities from 0% to 30%.

It turned out that the effect of swap is both language- and dataset-dependent. It deteriorates the evaluation metrics stronger for texts with shorter words, because there swaps often occur in the **B** and **E** segments of words. In our experiments on paraphrase and textual entailment tasks all four types of noise produced the same effect on English datasets, where the average length of words is 4 to 4.7 characters. On the other hand, Russian and Turkish datasets (with average word length of 5.7 characters) are more resistant to letter swaps than to other noise types.

However, this holds only for tasks where the result was computed as cosine similarity between vectors, i.e., where vectors fully define the performance. In the sentiment analysis task, where we trained a naive Bayes classifier, all types of noise had the same effect on the final quality for both English and Russian.

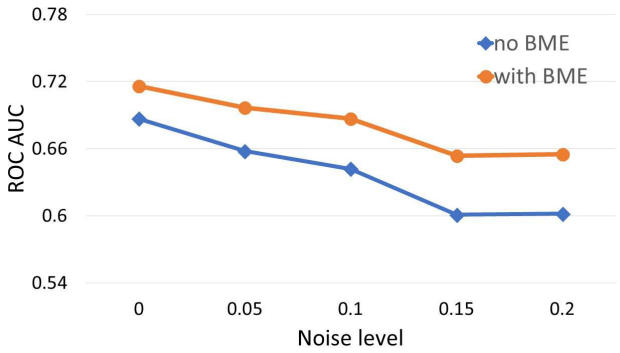

Figure 4. *RoVe* model with and without BME representation (paraphrase detection task for English).

5.5. OOV handling vs context encoding. Our model has two orthogonal features: handling of OOV words and context dependency of embeddings. To see how much each of them contributes to the final quality we tested them separately.

Only context dependency. We discard the BME representation of a word and consider it as a bag of letters (i.e., we encode it only with the **M** segment). Thus, the model still has open vocabulary, but is less expressive. Figure 4 shows the performance of models with and without BME encoding on the paraphrase detection task for English. We see that the BME representation does not make the model more robust to typos: for both settings the scores deteriorate to a similar extent as more noise is added. However, BME increases the quality of vectors for any level of noise. Therefore, prefixes and suffixes contain much information that should not be discarded. Results for other languages and tasks show the same trend.

Only BME encoding. In this setup we discard the context dependency of word vectors. We replace the encoder with a projection layer that converts the BME representation of a word into a 300-dimensional vector.

Figure 5 shows the performance of this model on the paraphrase task for English. The quality is close to random (a random classifier has ROC AUC of 0.5). Moreover, it is not consistent with the amount of noise: unlike our previous results, the quality does not decrease monotonically as noise increases. This is obvious since the encoder is the only trainable part of the model, thus it is the part most responsible for the quality of word

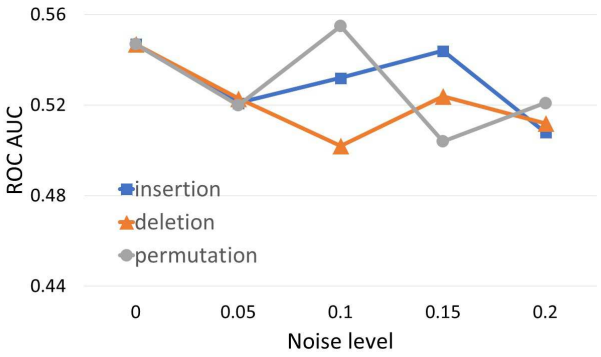

Figure 5. *RoVe* without context information (paraphrase detection task for English).

vectors. In addition, we should mention that we have tested our model on an additional noise type for this task, the permutation. This noise type has not been used in other experiments since robustness to this noise type has already been established in [28].

§6. Conclusions and future work

In this work, we have presented *RoVe*, a novel model for training word embeddings which is robust to typos. Unlike other approaches, this method does not have any explicit vocabulary. Embedding of a word is formed of embeddings of its characters, so *RoVe* can generate an embedding for any string of characters. This alleviates the influence of misspellings, as words with omitted or extra characters have an embedding close to the one of their correct versions.

We tested *RoVe* with different encoders and discovered that the SRU (Simple Recurrent Unit) cell is better suited for it. Bidirectional SRU performed best on all tasks. Our experiments have shown that our model is more robust to typos than *word2vec* and *fasttext* models commonly used to train word embeddings. Their quality falls dramatically as we add even a small amount of noise.

We have an intuition that *RoVe* can produce meaningful embeddings for unseen terms and unseen word forms in morphologically rich languages. However, we have not yet tested this intuition, and in our future work we

will look into possibilities of using *RoVe* for such tasks. This will require the tuning of lengths of prefixes and suffixes. We also plan to test language-dependent and data-driven tuning in future work. Another direction would be to train a *RoVe* model jointly with a downstream task, e.g., machine translation.

References

1. V. M. Andrjushchenko, *Koncepcija i architektura mashinnogo fonda russkogo jazyka*, (1989).
2. S. Arora, Y. Li, Y. Liang, T. Ma, A. Risteski, *Linear algebraic structure of word senses, with applications to polysemy*, (2016).
3. R. Astudillo, S. Amir, W. Ling, M. Silva, I. Trancoso, *Learning word representations from scarce and noisy data with embedding subspaces*, Proc. 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, 2015, pp. 1074–1084.
4. P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, *Enriching word vectors with subword information*, (2016).
5. S. Cucerzan, E. Brill, *Spelling correction as an iterative process that exploits the collective knowledge of web users.*, **4** (2004), 293–300.
6. S. Demir, I. D. El-Kahlout, E. Unal, H. Kaya, *Turkish paraphrase corpus*, Proc. 8th International Conference on Language Resources and Evaluation (LREC'12) (Istanbul, Turkey) (Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, eds.), European Language Resources Association (ELRA), may 2012 (english).
7. B. Dolan, C. Quirk, C. Brockett, *Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources*, (2004).
8. M F. Porter, *Snowball: A language for stemming algorithms*, **1** (2001).
9. T. Fawcett, *An introduction to ROC analysis*, Pattern recognition letters **27** (2006), no. 8, 861–874.
10. A. Graves, A.-R. Mohamed, G. Hinton, *Speech recognition with deep recurrent neural networks*, **38** (2013).
11. S. Hochreiter, J. Schmidhuber, *Long short-term memory*, **9** (1997), 1735–80.
12. N. Kalchbrenner, E. Grefenstette, P. Blunsom, *A convolutional neural network for modelling sentences*, **1** (2014).
13. D. Kiela, C. Wang, K. Cho, *Context-attentive embeddings for improved sentence representations*, CoRR **abs/1804.07983** (2018).
14. A. Kutuzov, I. Andreev, *Texts in, meaning out: neural language models in semantic similarity task for russian*, (2015).
15. T. Lei, Y. Zhang, *Training RNNs as fast as CNNs*, (2017).
16. D. Lewis, F. Li, T. Rose, Y. Yang, *Reuters corpus volume 1 as a text categorization test collection*, (2004).

17. Q. Li, S. Shah, X. Liu, A. Nourbakhsh, *Data sets: Word embeddings learned from tweets and general data*, (2017).
18. W. Ling, T. Luís, L. Marujo, R. Fernández Astudillo, S. Amir, C. Dyer, A. W. Black, I. Trancoso, *Finding function in form: Compositional character models for open vocabulary word representation*, CoRR **abs/1508.02096** (2015).
19. N. Loukachevitch, Y. Rubtsova, *Entity-oriented sentiment analysis of tweets: Results and problems*, Text, Speech, and Dialogue (Chan) (Pavel Král and Václav Matoušek, eds.), Springer International Publishing, 2015, pp. 551–555.
20. B. McCann, J. Bradbury, C. Xiong, R. Socher, *Learned in translation: Contextualized word vectors*, (2017).
21. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, *Distributed representations of words and phrases and their compositionality*, **26** (2013).
22. K.A. Nguyen, S. Schulte im Walde, N. Thang Vu, *Neural-based noise filtering from word embeddings*, CoRR **abs/1610.01874** (2016).
23. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, *Deep contextualized word representations*, CoRR **abs/1802.05365** (2018).
24. Y. Pinter, R. Guthrie, J. Eisenstein, *Mimicking word embeddings using subword RNNs*, CoRR **abs/1707.06961** (2017).
25. A.A. Polikarpov, *Towards the foundations of Menzerath's law*, Contributions to the Science of Text and Language **31** (2007), 215–240.
26. E. Pronoza, E. Yagunova, A. Pronoza, *Construction of a russian paraphrase corpus: Unsupervised paraphrase extraction*, Proc. RuSSIR 2015 (2016), 146–157.
27. S. R. Bowman, G. Angeli, C. Potts, C. Manning, *A large annotated corpus for learning natural language inference*, Proc. EMNLP 2015 (2015), 632–642.
28. K. Sakaguchi, K. Duh, M. Post, B. Van Durme, *Robsut wrod reocginiton via semi-character recurrent neural network*, Proc. AAAI 2017 (2017), 3281–3287.
29. J. Saxe, K. Berlin, *expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys*, CoRR **abs/1702.08568** (2017).
30. M. Schuster, K.K. Paliwal, *Bidirectional recurrent neural networks*, IEEE Transactions on Signal Processing **45** (1997), 2673–2681.
31. I. Segalovich, *A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine*, Machine Learning; Models, Technologies and Applications, 2003, pp. 273–280.
32. M. Seo, A. Kembhavi, A. Farhadi, H. Hajishirzi, *Bidirectional attention flow for machine comprehension*, (2016).
33. N. Smith, J. Eisner, *Contrastive estimation: Training log-linear models on unlabeled data*, Proc. 43rd ACL (2005), 354–362.
34. R. Socher, A. Perelygin, J.Y. Wu, J. Chuang, C.D. Manning, A.Y. Ng, C. Potts, *Recursive deep models for semantic compositionality over a sentiment treebank*, Proc. 2013 EMNLP (2013), 1631–1642.
35. E. Vylomova, T. Cohn, X. He, G. Haffari, *Word representation models for morphologically rich languages in neural machine translation*, CoRR **abs/1606.04217** (2016).

36. J. Wehrmann, W. Becker, H. E. L. Cagnini, R. C. Barros, *A character-based convolutional neural network for language-agnostic twitter sentiment analysis*, IJCNN-2017: International Joint Conference on Neural Networks, 2017, pp. 2384–2391.
37. O. Yildirim, F. Atik, M. F. Amasyali, *42 bin haber veri kumesi*, (2003).
38. X. Zhang, J. J. Zhao, Y. LeCun, *Character-level convolutional networks for text classification*, CoRR **abs/1509.01626** (2015).

Поступило 14 января 2019 г.

Skolkovo Institute of Science and Technology,
Nobelya Ulitsa, 3, 121205, Moscow, Russia
E-mail: taras.khakhulin@phystech.edu

Neural Systems and Deep Learning laboratory,
Moscow Institute of Physics and Technology,
9 Institutskiy per., Dolgoprudny,
Moscow Region, 141701, Russian Federation
E-mail: varvara.logacheva@gmail.com

Steklov Institute
of Mathematics at St. Petersburg,
nab. r. Fontanki, 27,
191023, St. Petersburg;
(ii) Moscow Institute of Physics and Technology,
9 Institutskiy per.,
Dolgoprudny, Moscow Region;
Institute for Systems Analysis,
Federal Research Center "Computer Science and Control"
of Russian Academy of Sciences,
pr. 60-letiya Oktyabrya, 9,
117312, Moscow
E-mail: valentin.malykh@phystech.edu