Robust Word Vectors: Context-Informed Embeddings for Noisy Texts

Valentin Malykh **Varvara Logacheva** **Taras Khakhulin**
Neural Systems and Deep Learning laboratory,
Moscow Institute of Physics and Technology,
Moscow, Russia
valentin.malykh@phystech.edu

Abstract

We suggest a new language-independent architecture of robust word vectors (RoVe). It is designed to alleviate the issue of typos, which are common in almost any user-generated content, and 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 the variety of related architectures and show that proposed architecture is typo-proof.

1 Introduction

The 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 transport (or while walking) is difficult, and people accidentally hit the wrong keys more often than when using a standard 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 encode semantic and syntactic properties of individual words (Arora et al., 2016). However, to use these word vectors the user input should be clean (i.e. free of misspellings), because a word vector model trained on clean data will not have misspelled versions of words. There are examples of models trained on noisy data (Li et al., 2017), but this approach does not fully solve the problem, because typos are unpredictable and a corpus cannot contain all possible incorrectly spelled versions of a word. Instead, we suggest that we should make algorithms for word vector modelling robust to noise.

We suggest a new architecture **RoVe** (Robust Vectors).[1] The main feature of this model is open vocabulary. It encodes words as sequences of symbols. This enables the model to produce embeddings for out-of-vocabulary (OOV) words. The idea as such is not new, many other models use character-level embeddings (Ling et al., 2015) or encode the most common ngrams to assemble unknown words from them (Bojanowski et al., 2016). However, unlike analogous models, RoVe is specifically targeted at typos — it is invariant to swaps of symbols 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 encode its context. The motivation for such architecture is the following. 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 want a word to get an embedding similar to the embedding of its initial form. This process reminds lemmatisation (reduction of a word to its initial form). Lemmatisation is context-dependent since it often needs to resolve homonymy based on word's context. By making RoVe model context-dependent we enable it to do such implicit lemmatisation.

We compare RoVe with common word vector tools: word2vec (Mikolov et al., 2013) and fasttext (Bojanowski et al., 2016). We test the models on three tasks: paraphrase detection, identifi-

[1] An open-source implementation is available, hidden for anonymity.

cation 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 the previous work. Section 3 contains the description of model's architecture. In section 4 we describe the experimental setup, and report the results in section 5. Section 6 concludes and outlines the 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 means of dealing with them. OOVs can be rare terms, unseen forms of known words, or simply typos. These types of OOVs need different approaches. The majority of research concentrated on unknown terms and generation of word forms, and very few targeted typos.

Ling et al. (2015) produce an open-vocabulary word embedding model — 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 encode unseen words to the same vector space. Pinter et al. (2017) approximate pre-trained word embeddings with a character-level model. Astudillo et al. (2015) project pre-trained embeddings to a lower space — this allows them 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 Nguyen et al. (2016) train a neural network that filters word embedding. To do that authors take pre-trained word embedding model and learn matrix transformation which denoise it. It makes them more robust to statistical artefacts in training data. Unfortunately, this does not solve the problem of typos in test data (i.e. the model still has a closed vocabulary).

There are examples of embeddings targeted at unseen word forms. Vylomova et al. (2016) train sub-word embeddings which are combined 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 translation task, in particular for morphologically rich languages. This method yields embeddings of high quality, but it requires training of a model for morphological analysis.

Some other models are targeted at encoding rare words. fasttext model (Bojanowski et al., 2016) produces embeddings for the most common ngrams of variable lengths, and an unknown word can be encoded as a combination of its ngrams. This is beneficial for encoding of compound words which are very common in German and occasionally occur in English. However, such a model is 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 does not only give a possibility to encode OOV words, but has also been shown to improve the quality of embeddings. Zhang et al. (2015) were 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. Saxe and Berlin (2017) use character-level CNNs for intrusion detection, and Wehrmann et al. (2017) build 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 every case of word usage. This resulted in emergence of word vector models which produced word embeddings with respect to words' local context. There are numerous evidences that contextualising pre-trained embeddings improves them (Kiela et al., 2018) and raises quality of downstream tasks, e.g. Machine Translation (McCann et al., 2017) or question answering (Peters et al., 2018).

3 Model architecture

RoVe combines context dependency and open vocabulary, which allows generating meaningful em-

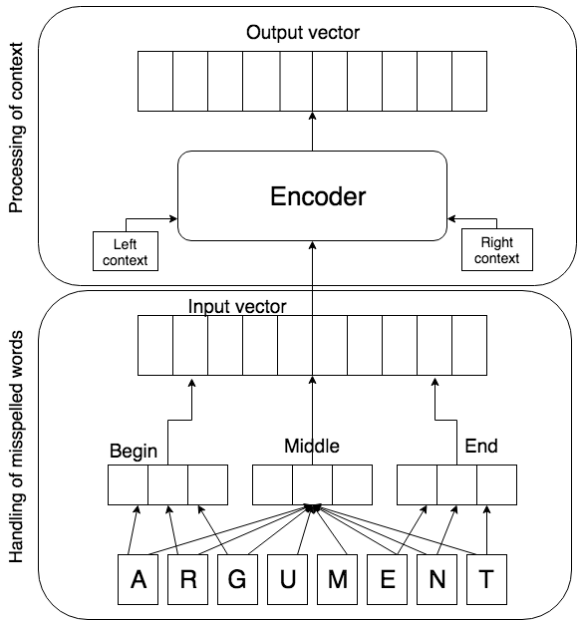

Figure 1: RoVe model: generation of embedding for the word *argument*.

beddings for OOV words. These two features are supported by the two parts of the model (see fig.1).

3.1 Encoding of context

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 the text processing time, but yields more accurate representations context-informed. The model is conceptually similar to encoder used to create representation of a sentence by reading all its words. Such encoders are used in Machine Translation (McCann et al., 2017), question answering (Seo et al., 2016) and many other tasks.

In order to generate a representation of a word, we need to encode it together with its context. For every word of a context we first produce its input embedding (described in section 3.2). This embedding is then passed to the encoder (top part of figure 1) which processes all words of the context. The encoder should be a neural network which 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 can be used. After having processed the whole context we get embedding for the target word by passing a hidden state corresponding to the needed word 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 transformation of an input word into a fixed-size vector (input embedding). The transformation is shown in the bottom part of figure 1. This is a deterministic procedure, it is uniquely defined for a word and 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 the position i where i is the index of the character). Then, we generate three vectors: beginning (**B**), middle (**M**), and end (**E**) vectors. **M** vector is a sum of one-hot vectors of all characters of a word. **B** is a concatenation of one-hot vectors for n_b first characters of a word. Likewise, **E** component is a concatenation one-hot vectors of n_e last characters of a word. n_b and n_e are hyperparameters which can vary for different 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 input vector is shown in fig.2.

We further refer to this three-part representation as **BME**. It was inspired by work by Sakaguchi et al. (2016) where the first and the last symbols of a word are encoded separately as they carry more meaning than the other characters. However, the motivation for our BME representation stems from division of words into morphemes. We encode n_b first symbols and n_e last symbols 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. an English prefix *un* with a meaning of reversed action or absence) or grammatical information (e.g. suffix *ed* which indicates past participle of a verb). Thus, keeping it can make the resulting embedding more informative.

The M part of 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* which will have identical embeddings in our model, whereas word2vec and many other models will not be able to provide any representation for

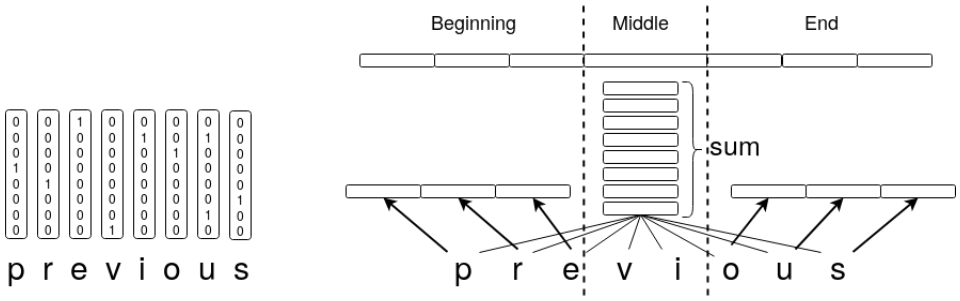

Figure 2: Generation of input embedding for the word *previous*. Left: generation of character-level one-hot vectors, right: generation of BME representation.

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. 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 which support typos, RoVe can handle noise in both training and inference data.

3.3 Training

RoVe model is trained with *negative sampling* procedure suggested by Smith and Eisner (2005). We use it as described by Mikolov et al. (2013). This method serves to train vector representations of words. The fundamental property of word vectors is 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 in equation 1. Here, w is the target word, v_i are positive examples from context (C) and v_j are negative examples (Neg) randomly sampled from data. Function $s(\cdot, \cdot)$ is a similarity score for two vectors which should be an increasing function. For our experiments we use 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 vector 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 are sampled randomly from data, hence the name of the procedure.

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

Our model is trained using this objective. The conversion of words into input embeddings is a deterministic procedure, so during training we update only parameters of a neural network which generates the context-dependent embeddings and fully-connected layers that precede and follow it.

4 Experimental setup

We check 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 the 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 (Polikarpov, 2007). 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** — pre-trained Google News word vectors,[2]
- **Russian** — pre-trained word vectors RusVectores (Kutuzov and Andreev, 2015).
- **Turkish** — we trained a model on "42 bin haber" corpus (Yildirim et al., 2003).

We stem Turkish texts with SnowBall stemmer (F. Porter, 2001) and lemmatise Russian texts Mystem tool[3] (Segalovich, 2003). This is done in order to reduce the sparsity of text and interpret rare word forms. In English this problem is not as severe, because it has less developed morphology.

As fasttext baselines we use official pre-trained fasttext models.[4] We also try an extended version of fasttext baseline for Russian and English — fasttext + spell-checker. For the downstream tasks we checke texts with a publicly available spell-checker[5] prior to extracting word vectors. Since spell-checking is one of the common ways of reducing the effect of typos, we wanted to compare its performance with RoVe.

4.2 Infusion of noise

In order to demonstrate robustness of RoVe against typos we artificially introduce noise to 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 the different levels of noise from 0% (no noise) to 30%. According to Cucerzan and Brill (2004), 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. The main decision to be made when experimenting with the model is the architecture of the encoder. We experiment with RNNs and CNNs. We conduct experiments with the following RNN architectures:

- **Long Short-Term Memory (LSTM)** unit (Hochreiter and Schmidhuber, 1997) — 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.
- **bidirectional LSTM** (Schuster and K. Paliwal, 1997) — two RNNs with LSTM units where one RNN reads a sequence from beginning to end and another one backward.
- **stacked LSTM** (Graves et al., 2013) — an RNN with multiple layers of LSTM cells. It allows to combine the forward and backward layer outputs and use them as an input to the next layer. We experiment with two bidirectional RNN layers with stacked LSTM cells.
- **Simple Recurrent Unit (SRU)** (Lei and Zhang, 2017) — LSMT-like architecture which is faster due to parallelisation.
- **bidirectional SRU** — bidirectional RNN with SRU cells.

We also try the following convolutional architectures:

- **CNN-1d** — unidimensional Convolutional Neural Network as in (Kalchbrenner et al., 2014). This model uses 3 convolution layers with kernel sizes 3, 5 and 3, respectively.
- **ConvLSTM** — a combination of CNN and recurrent approaches. We first apply CNN-1d model and then produce 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 train our RoVe models on the following datasets:

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

[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/

- English — Reuters dataset (Lewis et al., 2004),
- Russian — Russian National Corpus (Andrjushchenko, 1989),
- 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 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 pre-trained 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. High similarity is interpreted as a paraphrase. Phrase vectors are computed as an average of vectors of words in a phrase. For word2vec we discard OOV words as it cannot generate embedding for them. We measure the performance of models on this task with ROC AUC metric (Fawcett, 2006) which defines the proportions of true positive answers in system's outputs with varying threshold.

We run experiments on three datasets:

- **English** — Microsoft Research Paraphrase Corpus (Dolan et al., 2004) 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 (Pronoza et al., 2016) consists of news headings from different news agencies. It contains around 6,000 pairs of phrases labelled in terms of ternary scale: "-1" — not paraphrase, "0" — weak paraphrase, and "1" — strong paraphrase. We use only "-1" & "1" classes for consistency with other datasets. There are 4,470 such pairs.
- **Turkish** — Turkish Paraphrase Corpus (Demir et al., 2012) contains 846 pairs of sentences from news texts manually labelled for semantic equivalence.

The results of this task are outlined 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 of RoVe variants that never beat the baselines. We refer readers to the supplementary material for the full results of these and other experiments.

As we can see, none of the systems are typo-proof — 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 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 variants, bidirectional SRU gives the best result, marginally outperforming SRU.

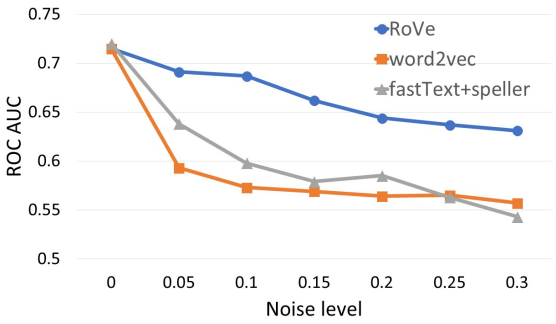

Figure 3: Comparison of RoVe model with word2vec and fasttext on texts with growing amount of noise (paraphrase detection task for English).

Interestingly, the use of spell-checker does not guarantee the improvement: fasttext+spell-checker model does not always outperform vanilla fasttext, and its score is unstable. This might be explained by the fact that spell-checker 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 Naive Bayes classifier. The evaluation is performed with ROC AUC metric. We experiment with two datasets:

- **English** — Stanford Sentiment Treebank (Socher et al., 2013). In this corpus objects are labelled with three classes: positive, negative and neutral, we use only the two former.
- **Russian** — Russian Twitter Sentiment Corpus (Loukachevitch and Rubtsova, 2015). It

	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 + spell-checker	**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.

	Sentiment analysis						Textual entailment		
	English			Russian			English		
noise (%)	0	10	20	0	10	20	0	10	20
BASELINES									
word2vec	0.649	0.611	0.554	0.649	0.576	0.524	0.624	0.593	0.574
fasttext	**0.662**	0.615	0.524	0.703	0.625	0.524	0.642	0.563	0.517
fasttext + spell-checker	0.645	0.573	0.521	0.703	0.699	0.541	0.642	0.498	0.481
RoVe									
stackedLSTM	0.621	0.593	0.586	0.690	0.632	0.584	0.617	0.590	0.516
SRU	0.627	0.590	0.568	0.712	0.680	0.598	0.627	0.590	0.568
biSRU	0.656	**0.621**	**0.598**	**0.721**	**0.699**	**0.621**	**0.651**	**0.621**	**0.598**

Table 2: Results of the sentiment analysis and textual entailment tasks in terms of ROC AUC.

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.

The results for this task (see table 2) confirm the ones 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 Stanford Natural Language Inference corpus (R. Bowman et al., 2015) 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 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 listed in the right part of table 2. They fully agree with those of 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 spell-checker 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 to the experiments. However, a swap does not change an embedding of a word only when this swap happens outside B and E segments of a word, otherwise the embedding changes as B 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.

Analogously to 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 language and dataset dependent. It deteriorates the scores stronger for texts with shorter words, because there swaps often occur in 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 symbols.

On the other hand, Russian and Turkish datasets (average word length is 5.7 symbols) 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 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.

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 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 paraphrase detection task for English. We see that BME representation does not make the model more robust to typos — for both settings scores reduce 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 expose the same trend.

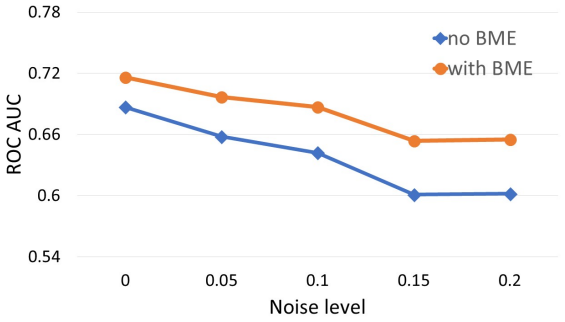

Figure 4: RoVe model with and without BME representation (paraphrase detection task for English).

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

Figure 5 shows the performance of this model on 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 while noise increases. This is obvious since the encoder is the only trainable part of the model, thus, it is the part mostly responsible for the quality of word vectors. In addition we should mention that we have tested our model on additional noise type for this task – the permutation. This noise type hasn't been used for the other experiments, since robustness to this noise type was shown in (Sakaguchi et al., 2016).

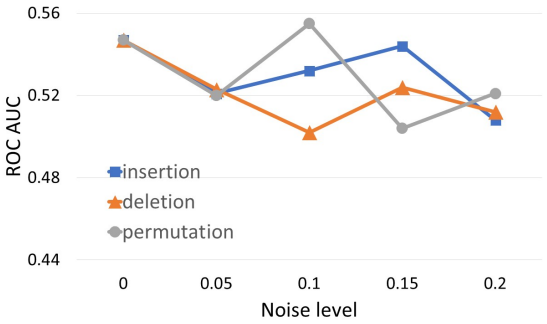

Figure 5: RoVe without context information (paraphrase detection task for English).

6 Conclusions and future work

We 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. This alleviates the influence of misspellings, words with omitted or extra symbols have an embedding close to the one of their correct versions.

We tested RoVe with different encoders and discovered that SRU (Simple Recurrent Unit) cell is better suited for it. Bidirectional SRU performs best on all tasks. Our experiments showed that our model is more robust to typos than word2vec and fasttext models commonly used for training of word embedding. Their quality falls dramatically as we add even 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 did not test this. In our fu-

ture work we will look into possibilities of using RoVe in these tasks. This will require tuning of lengths of prefixes and suffixes. We would like to test language-dependent and data-driven tuning.

Another direction of future work is to train RoVe model jointly with a downstream task, e.g. Machine Translation.

References

Vladislav Mitrofanovich Andrjushchenko. 1989. Koncepcija i architektura mashinnogo fonda russkogo jazyka.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. 2016. Linear algebraic structure of word senses, with applications to polysemy.

Ramón Astudillo, Silvio Amir, Wang Ling, Mario Silva, and Isabel Trancoso. 2015. Learning word representations from scarce and noisy data with embedding subspaces. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1074–1084. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information.

Silviu Cucerzan and Eric Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. 4:293–300.

Seniz Demir, Ilknur Durgar El-Kahlout, Erdem Unal, and Hamza Kaya. 2012. Turkish paraphrase corpus. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources.

M F. Porter. 2001. Snowball: A language for stemming algorithms. 1.

Tom Fawcett. 2006. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. 38.

Sepp Hochreiter and Jrgen Schmidhuber. 1997. Long short-term memory. 9:1735–80.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. 1.

Douwe Kiela, Changhan Wang, and Kyunghyun Cho. 2018. Context-attentive embeddings for improved sentence representations. *CoRR*, abs/1804.07983.

Andrei Kutuzov and Igor Andreev. 2015. Texts in, meaning out: neural language models in semantic similarity task for russian.

Tao Lei and Yu Zhang. 2017. Training rnns as fast as cnns.

David Lewis, Fan Li, Tony Rose, and Yiming Yang. 2004. Reuters corpus volume 1.

Quanzhi Li, Sameena Shah, Xiaomo Liu, and Armineh Nourbakhsh. 2017. Data sets: Word embeddings learned from tweets and general data.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernández Astudillo, Silvio Amir, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *CoRR*, abs/1508.02096.

Natalia Loukachevitch and Yuliya Rubtsova. 2015. Entity-oriented sentiment analysis of tweets: Results and problems. In *Text, Speech, and Dialogue*, pages 551–555, Chan. Springer International Publishing.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors.

Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. 26.

Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. 2016. Neural-based noise filtering from word embeddings. *CoRR*, abs/1610.01874.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking word embeddings using subword rnns. *CoRR*, abs/1707.06961.

Polikarpov. 2007. Towards the foundations of menzerath's law. 31.

Ekaterina Pronoza, Elena Yagunova, and Anton Pronoza. 2016. Construction of a russian paraphrase corpus: Unsupervised paraphrase extraction. 573:146–157.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christoper Manning. 2015. A large annotated corpus for learning natural language inference.

Keisuke Sakaguchi, Kevin Duh, Matt Post, and Benjamin Van Durme. 2016. Robsut wrod reocginiton via semi-character recurrent neural network.

Joshua Saxe and Konstantin Berlin. 2017. expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *CoRR*, abs/1702.08568.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. 45:2673 – 2681.

Ilya Segalovich. 2003. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension.

Noah Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data.

R Socher, A Perelygin, J.Y. Wu, J Chuang, C.D. Manning, A.Y. Ng, and C Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. 1631:1631–1642.

Ekaterina Vylomova, Trevor Cohn, Xuanli He, and Gholamreza Haffari. 2016. Word representation models for morphologically rich languages in neural machine translation. *CoRR*, abs/1606.04217.

Joonatas Wehrmann, Willian Becker, Henry E. L. Cagnini, and Rodrigo C. Barros. 2017. A character-based convolutional neural network for language-agnostic twitter sentiment analysis. In *IJCNN-2017: International Joint Conference on Neural Networks*, pages 2384–2391.

O. Yildirim, F. Atik, and M. F. Amasyali. 2003. 42 bin haber veri kumesi.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626.