Neural text normalization leveraging similarities of strings and sounds

Riku Kawamura[†] **Tatsuya Aoki**[†] **Hidetaka Kamigaito**[†]
Hiroya Takamura[†‡] **Manabu Okumura**[†]

[†]Tokyo Institute of Technology
[‡]National Institute of Advanced Industrial Science and Technology

{riku, aoki, kamigaito}@lr.pi.titech.ac.jp
{takamura, oku}@pi.titech.ac.jp

Abstract

We propose neural models that can normalize text by considering the similarities of word strings and sounds. We experimentally compared a model that considers the similarities of both word strings and sounds, a model that considers only the similarity of word strings or of sounds, and a model without the similarities as a baseline. Results showed that leveraging the word string similarity succeeded in dealing with misspellings and abbreviations, and taking into account the sound similarity succeeded in dealing with phonetic substitutions and emphasized characters. So that the proposed models achieved higher F_1 scores than the baseline.

1 Introduction

Non-standard words such as misspellings or abbreviations are often used in social media such as Twitter. It is difficult to understand the meaning of such words without any extra knowledge, and it is challenging to perform natural language processing over sentences including them. Text normalization plays an important role in dealing with these broken texts by correcting such a sentence into a standard one. The following is an example of text normalization.

before normalization: *r u cuming 2 midcorner dis sunday?*
↓ ↓ ↓ ↓ ↓
after normalization: *are you coming to midcorner this sunday?*

In text normalization, as shown in the example, we need to correct several types, including misspellings (‘*cuming*’ to ‘*coming*’), abbreviations (‘*convo*’ to ‘*conversation*’), phonetic substitutions (‘*dis*’ to ‘*this*’ or ‘*r u*’ to ‘*are you*’), and emphasized characters (‘*yeeeees*’ to ‘*yes*’).

Even though the similarities of character surfaces and phonemes is important for a human to understand non-standard words, current neural network-based text normalization methods do not consider this information. We assume that text normalization more intuitive to humans is possible by explicitly considering such features in a neural network-based method. Based on this assumption, in this work, we propose neural text normalization models that leverage both string and sound similarities. Experimental results show that our proposed models outperformed a baseline and achieved state-of-the-art results in the text normalization track on WNUT-2015.

2 Related Work

Han et al. (2012) proposed a ranking-based text normalization method that incorporates the matching degree of surrounding word n-grams to the target word and the edit distance from existing words. Li and Liu (2012) proposed a text normalization method leveraging phonetic information to translate non-standard words into standard ones. Ansari et al. (2017) proposed an automatic optimization-based nearest neighbor matching approach leveraging string and phonetic similarity. Jin (2015) achieved the best

Proceedings of the 28th International Conference on Computational Linguistics, pages 2126–2131
Barcelona, Spain (Online), December 8-13, 2020

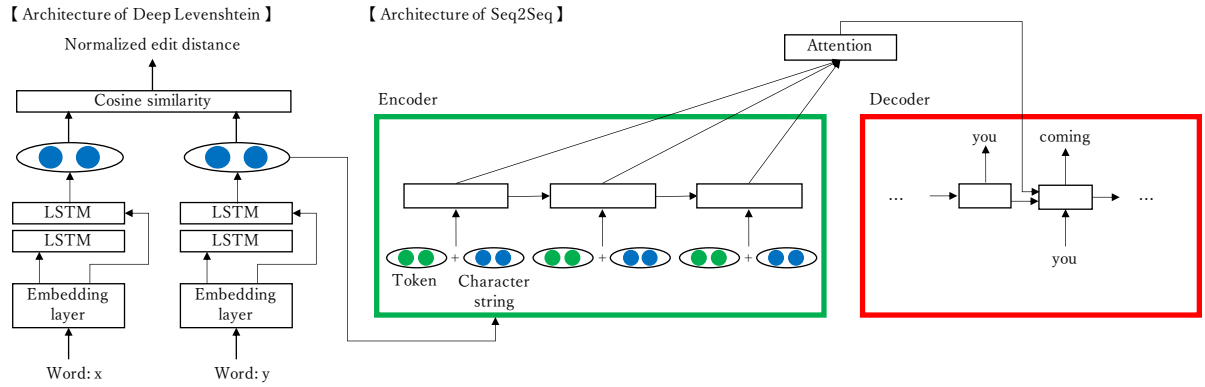

Figure 1: How to introduce the character string feature into Seq2Seq.

performance on the WNUT-2015 task, with a method that generates candidates based on the training data. van der Goot and van Noord (2017) extended this work by leveraging additional resources of Twitter and Wikipedia data. However, their method does not take into consideration contextual information. To solve this problem, Sequence-to-Sequence (Seq2Seq) (Sutskever et al., 2014) has been used for text normalization. Lourentzou et al. (2019) performed highly accurate and fluent text normalization by using the Attention-based Seq2Seq (Bahdanau et al., 2014). Mani et al. (2020) used Seq2Seq in automatic speech recognition error correction, a task similar to text normalization. Taking these trends into account, in this work, we propose a neural text normalization model that leverages both string and sound similarity.

3 Methodology

Figure 1 shows an overview of our proposed method. In this study, we perform text normalization based on the method of Lourentzou et al. (2019), which incorporates token embeddings as an input to the encoder. Our method expands their work by utilizing features related to character strings and sounds.

3.1 Deep Levenshtein

Moon et al. (2018) proposed Deep Levenshtein, a network that captures the feature of character strings based on the Levenshtein edit distance (Levenshtein, 1966) in order to correct any character fluctuations of named entities in a text. In this study, we incorporate this mechanism into the Seq2Seq model to make it more robust to a broken text.

Deep Levenshtein is a neural network that takes two words x and y as an input and then outputs hidden representations for the character strings of the words. Two words x and y are fed into the word embedding layer, and then we obtain word embeddings e_x and e_y, respectively. Each word embedding is an input to the bidirectional LSTM (Hochreiter and Schmidhuber, 1997) to obtain hidden representations c_x and c_y, that capture the feature of the character string. Here, $\overrightarrow{h_x}$ and $\overleftarrow{h_x}$ represent the forward and backward paths of the bidirectional LSTM, respectively. c_x is obtained by concatenating them as $c_x = [\overrightarrow{h_x}; \overleftarrow{h_x}]$, where ; indicates a concatenation operation. Deep Levenshtein learns representations c_x and c_y so that the cosine-based similarity between them approximates the similarity based on the edit distance between the input character strings by minimizing

$$\left|\left| \frac{1}{2}\left(\frac{c_x^{\mathrm{T}} c_y}{||c_x|| \cdot ||c_y||} + 1 \right) - s(x,y) \right|\right|^2, \quad (1)$$

where $s(x,y) = 1 - \frac{d(x,y)}{max(length(x),length(y))}$,[1] that indicates the similarity based on the edit distance between the input words x and y.

Deep Levenshtein can predict the similarity of the character string between two words based on the distance in vector space, so the vectors obtained above capture the character string feature.

[1]We divide the edit distance, $d(x,y)$, by the longer length of the strings x and y to take into account the length of the character strings.

Pattern	Description	Example
1. Do nothing	do not add noise	*python* → *python*
2. Delete characters	delete characters randomly	*python* → *pyhon*
3. Replace characters	replace characters randomly	python → *pyhtno*
4. Extend characters	extend words ending with {u, y, s, r}	*beer* → *beerrrr*
5. Extend short vowels	stretch short vowels {a, i, u, e, o}	*cat* → *caaat*
6. Delete symbol	delete apostrophe	*I'm* → *Im*
7. Misplaced sign	insert apostrophe in different position	*don't* → *do'nt*
8. Typo	replace with another character that is at a near position on a keyboard	*hello* → *jello*
9. Convert to another token	convert to completely different token	*python* → *ruby*

Table 1: Noise generator.

3.2 Deep Metaphone

Raghuvanshi et al. (2019) revealed that relying only on surface text similarities cannot capture phonetic differences between words. Furthermore, Han et al. (2013) showed that sound-related features are effective in text normalization. In this work, we propose Deep Metaphone to capture sound features for text normalization by learning phonetic edit distance.

Deep Metaphone has the same network structure as Deep Levenshtein. The difference between them is the training data they use. Deep Metaphone learns the phonetic edit distance, which is the edit distance of the strings obtained from Double Metaphone (Philips, 2000). The following is an example conversion where we apply the Double Metaphone algorithm over the words '*yeeeees*' and '*yes*'.

$$\underset{Before}{yeeeees} \rightarrow \underset{After}{AS} \qquad \underset{Before}{yes} \rightarrow \underset{After}{AS}$$

By using Double Metaphone, similar to Deep Levenshtein, Deep Metaphone can predict the sound similarity between two words. Therefore, we can use the vector that captures the feature of the sound.

3.3 Incorporating new features to Seq2Seq

In this study, we use a bi-directional LSTM for the encoder and decoder. Lourentzou et al. (2019) incorporated only token embeddings $e_{x_t}^{token}$ as an input to the encoder. We further incorporate the character feature $c_{x_t}^{leven}$, obtained from Deep Levenshtein, and the sound feature $c_{x_t}^{phone}$, obtained from Deep Metaphone.[2] Mansfield et al. (2019) empirically tested addition, concatenation, and multi-layer perceptron to combine new features with token embeddings. They reported that concatenation outperforms the other two methods. Therefore, we choose concatenation. Our new feature vectors $c_{x_t}^{leven}$ and $c_{x_t}^{phone}$ are incorporated into the Seq2Seq encoder as follows:

$$h_t^{enc} = Encoder([e_{x_t}^{token}; c_{x_t}^{leven}; c_{x_t}^{phone}; h_{t-1}^{enc}]). \quad (2)$$

4 Experiments

4.1 Experimental settings

We used WNUT-2015 Shared Task2 (Baldwin et al., 2015),[3] which is a task of normalizing social media texts, for our evaluation. The official dataset consists of 4,917 tweets with 373 non-standard words. The dataset was randomly split by Lourentzou et al. (2019) into 60:40, 2,950 tweets for the training data and 1,967 tweets for the test data. For evaluation metrics, we used precision, recall, and F_1-score.

To train Deep Levenshtein, we applied the noise generator in (Lourentzou et al., 2019) to words in the training data of WNUT-2015 Shared Task2. The noise generator outputs character strings by executing one of the processes in Table 1. We used both those generated words and the original words to train Deep Levenshtein.

[2]Note that Deep Levenshtein and Deep Metaphone take two words as an input during the training steps, but both methods take only a word to extract each feature for Seq2Seq in the inference steps.

[3]https://noisy-text.github.io/2015/

Model	Precision	Recall	F_1
Two-stage Seq2Seq (reported)	90.66	78.14	83.94
Two-stage Seq2Seq (reproduced)	90.24	78.10	83.74
Two-stage Seq2Seq + LS	**91.89**	78.00	**84.38**
Two-stage Seq2Seq + MP	91.32	78.18	84.24
Two-stage Seq2Seq + LS + MP	91.30	77.80	84.13
Random Forest (Jin, 2015)	90.61	**78.65**	84.21
MoNoise (van der Goot and van Noord, 2017)[5]	93.53	80.26	86.39

Table 2: Comparison of our models with the baseline and state-of-the-art model on WNUT-2015.[4] **Bold** indicates the best score. For reference, we also list the result of MoNoise (van der Goot and van Noord, 2017), which incorporates data augmentation with external resources.[5]

Type	Misspelling	Abbreviation	Phonetic substitution	Emphasized character
Input	*wen u cee a car*	*i diss you*	*so funny d temple*	*rt: i go homeee*
Reference	*when you see a car*	*i disrespect you*	*so funny the temple*	*rt: i go home*
Two-stage Seq2Seq	*wen you see a car*	*i diss you*	*so funny d temple*	*rt: i go homeee*
Two-stage Seq2Seq + LS	*when you see a car*	*i disrespect you*	*so funny d temple*	*rt: i go home*
Two-stage Seq2Seq + MP	*wen you see a car*	*i disrespect you*	*so funny the temple*	*rt: i go home*
Two-stage Seq2Seq + LS +MP	*when you see a car*	*i this you*	*so funny d temple*	*rt: i go home*

Table 3: Examples for correcting several types.

For training Deep Metaphone, we first created the training data for Deep Levenshtein as in the above and then applied the Double Metaphone algorithm to words to yield the training data.

Parameters including the size of the token embeddings in the baseline model were set to the same as in (Lourentzou et al., 2019), where it was 100. The sizes of hidden layers, $c_{x_t}^{leven}$ and $c_{x_t}^{phone}$, of LSTM for Deep Levenshtein and Deep Metaphone were tuned from {10, 20, 30, 40, 50} on the validation data, randomly extracted 100 sentences from the training data. When only Deep Levenshtein was used, 20 was selected. When only Deep Metaphone was used, 50 was selected. When both were used, 10 was selected for each of them. The reason why we set the range of smaller values from 10 to 50 compared with the size of token embeddings, 100, is based on the finding in (Sennrich and Haddow, 2016). They reported that the size of the secondary embeddings, i.e., $c_{x_t}^{leven}$ and $c_{x_t}^{phone}$ in our case, should be smaller than that of the primary embeddings, i.e., $e_{x_t}^{token}$ in our case.

4.2 Compared models

In the experiments, we compared a baseline model and our models, which are listed below.

Two-stage Seq2Seq (baseline): A model proposed by Lourentzou et al. (2019). We reimplemented this model and will report its performance in addition to the scores reported in their paper.

Two-stage Seq2Seq + LS (Levenshtein): A model where we add character string features to the input of the encoder in the Two-stage Seq2Seq model.

Two-stage Seq2Seq + MP (Metaphone): A model where we add sound features to the input of the encoder in the Two-stage Seq2Seq model.

Two-stage Seq2Seq + LS + MP: A model where we add both character string and sound features to the input of the encoder in the Two-stage Seq2Seq model.

[4]While the performance degraded when the token embedding size was set to a larger value, 120, it was improved when combining the character string/sound features.

[5]In addition to the WNUT-2015 training data, MoNoise leverages large collections of Twitter and Wikipedia data.

4.3 Results and analysis

We report the performance of the baseline model and our models in Table 2. As shown, our models, i.e., Two-stage Seq2Seq +LS, +MP, +LS +MP, outperformed the baseline model in terms of F_1 score. Table 3 shows example outputs from the Two-stage Seq2Seq model and our models. From the table, we can see that Two-stage Seq2Seq + LS and + LS + MP models corrected the misspelled '*wen*' to '*when*'. Two-stage Seq2Seq + LS and + MP models corrected the abbreviation '*diss*' to '*disrespect*'. Only Two-stage Seq2Seq + MP model corrected the phonetic substitution '*d*' to '*the*'. All of our models corrected the emphasized character '*homeee*' to '*home*'. These results show that Two-stage Seq2Seq + LS model works well for misspellings and abbreviations and that Two-stage Seq2Seq + MP model works well for phonetic substitutions and emphasized characters.

On the contrary, the weakness of Two-stage Seq2Seq + LS model is that it fails to deal with typos where we need to correct from a character to another character that is far on the keyboard layout, such as '*thang*' to '*thing*'. The weakness of Two-stage Seq2Seq + MP model is that it tends to mistakenly correct a word to another word having a similar sound, such as '*nah*' to '*no*'. We also found that, when considering both features, we observed several error cases such as '*favor*' to '*favorite*', where the model wrongly changed a word. We speculate that these are due to the small search range of the sizes of the hidden layers ($c_{x_t}^{leven}$ and $c_{x_t}^{phone}$). Furthermore, all models were not able to handle the correction of an emoji, such as '*b*', meaning thumbs-up. This is because the knowledge in the current models is based only on the training data, and we do not utilize specific knowledge on the connection between linguistic information and other types of information, including visual information, which we leave for our future work.

5 Conclusion and future work

In this paper, we proposed a method that takes into account the similarities of word strings and sounds as features for text normalization. Our evaluation results showed that incorporating such features to Seq2Seq improves the performance of text normalization compared to the baseline method in terms of F_1 score. We also found that the proposed methods managed to effectively consider both surface character and phonetic similarities.

For future work, we will try data augmentation based on the finding by Grundkiewicz et al. (2019). They reported that synthetic data can be generated by substituting words commonly confused with each other to improve performance. On the contrary, we will use the dataset proposed by Sproat and Jaitly (2016), a much larger dataset than the WNUT-2015 Shared Task2, to empirically gain deep insights. We would also like to apply ELMo (Peters et al., 2018), Transformer (Vaswani et al., 2017) and BERT (Devlin et al., 2019) to Seq2Seq for better performance.

References

S. Ansari, Usman Zafar, and A. Karim. 2017. Improving text normalization by optimizing nearest neighbor matching. *ArXiv*, abs/1712.09518.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

Timothy Baldwin, Marie Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–135, Beijing, China, July. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North*.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263, Florence, Italy, August. Association for Computational Linguistics.

Bo Han, Paul Cook, and Timothy Baldwin. 2012. Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 421–432, Jeju Island, Korea, July. Association for Computational Linguistics.

Bo Han, Paul Cook, and Timothy Baldwin. 2013. Lexical normalization for social media text. *ACM Transactions on Intelligent Systems and Technology*, 4:5:1–5:27.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Ning Jin. 2015. NCSU-SAS-ning: Candidate generation and feature engineering for supervised lexical normalization. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 87–92, Beijing, China, July. Association for Computational Linguistics.

VIadimir I Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

Chen Li and Yang Liu. 2012. Normalization of text messages using character- and phone-based machine translation approaches. In *INTERSPEECH-2012*, pages 2330–2333, 09.

Ismini Lourentzou, Kabir Manghnani, and ChengXiang Zhai. 2019. Adapting sequence to sequence models for text normalization in social media. *CoRR*, abs/1904.06100.

Anirudh Mani, Shruti Palaskar, Nimshi Venkat Meripo, Sandeep Konam, and Florian Metze. 2020. Asr error correction and domain adaptation using machine translation. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May.

Courtney Mansfield, Ming Sun, Yuzong Liu, Ankur Gandhe, and Björn Hoffmeister. 2019. Neural text normalization with subword units. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 190–196, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Seungwhan Moon, Leonardo Neves, and Vitor Carvalho. 2018. Multimodal named entity disambiguation for noisy social media posts. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2000–2008, Melbourne, Australia, July. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Lawrence Philips. 2000. The double metaphone search algorithm. *C/C++ Users Journal*, 18:38–43.

Arushi Raghuvanshi, Vijay Ramakrishnan, Varsha Embar, Lucien Carroll, and Karthik Raghunathan. 2019. Entity resolution for noisy ASR transcripts. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 61–66, Hong Kong, China, November. Association for Computational Linguistics.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers*, pages 83–91, Berlin, Germany, August. Association for Computational Linguistics.

Richard Sproat and Navdeep Jaitly. 2016. RNN approaches to text normalization: A challenge. *CoRR*, abs/1611.00068.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks.

Rob van der Goot and Gertjan van Noord. 2017. Monoise: Modeling noise using a modular normalization system. *CoRR*, abs/1710.03476.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ¥L ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.