Phonetic Text Normalization using Recurrent Neural Networks

Jasper Ginn, Arvid Halma, Aaron Swaving & Boaz Manger

January 10, 2019

1 Introduction[1]

Text normalization is the application of linguistic models to text with the aim of reducing tokens to a common form. For example, words may refer to the same underlying concepts but can be spelled in a different way. Examples of normalization are stemming and lemmatization. The former tries to map words to tokens by removing the suffix of a word (e.g. 'economy', 'economist' and 'economic' all map to 'econom'), and the latter approach attempts to bring back each token to its dictionary representation.

Phonetic normalization attempts to map similar-sounding words to a common form [1]. Our interest in this topic is related to the development of our chatbot `ChitChat` [2], which we have implemented in for several international organizations in countries in which English is not the first language. One of our findings is that many of the conversations are characterized by a use of English that strongly resembles its phonetic form. For example, a sentence may contain the word 'aksiom', whereas the author means to write 'axiom'.

Due to the proliferation of social media messaging, there has been an increased interest in phonetic normalization. Text found on social media such as Twitter, which is often referred to as *microtext*, has different challenges than regular, well-written text. It is characterized by a casual writing style, colloquial tone, relaxed spelling, reliance on emoticons, use of out of vocabulary (OOV) words such as acronyms and emphasis on emotional undercurrents for dramatic effect. (See e.g. [3] [4] [5] [6] [7] [8] [9]). While we find that the language used to converse with our chatbot bears some similarities to microtext, the language used is often more formal. This is an important distinction because strict phonetic normalization as we implement it is not suited for the informal style of microtext (see [3]).

In this paper, we present an approach to phonetic normalization that employs a standard encoder-decoder framework common to machine translation tasks. In this setting, the inputs are characters of English words, and the outputs are the pronunciations of these words in various formats. Hence, the goal of this paper is to show that a mapping can be learned from written to phonetic English by using character-level Recurrent Neural Networks (RNNs). One benefit of such an approach is that the map-

[1] All code and data available on GitHub., You can find the repository at https://github.com/JasperHG90/phonorm

ping is learned without specific expertise about such a task. Examples of previous work with respect to text normalization and deep learning are [10], [11] and [12].

This paper is structured as follows. (2) gives a short description of RNNs. (3) outlines the data sets, models and evaluation metrics used in our experiment. (4) interprets the results, and (5) concludes by offering several ways to improve the model.

2 Recurrent neural networks

Recurrent neural networks (RNN) are a type of neural network that model sequences such as series of words or characters. The RNN takes both an input vector, such as a sequence of input characters, as well as outputs that occur earlier in the sequence. Therefore, at timestep t, the results of timestep $t-1$ are fed into the network to learn patterns.

In principle, the length of the sequence is arbitrary, and the input length T_x does not have to match the output length T_y. This is convenient for language modeling because, for example in the case of machine translation, an English sentence rarely has the same length as its counterpart in another language.

Like other models that consist of many layers, RNNs suffer from the ‘vanishing gradient’ problem. This means that they cannot represent long-range dependencies very well, and that, for example, a verb that occurs early in the sentence may not be ‘memorized’ until the end of the sentence, which renders the RNN unable to establish the connection between the verb and a noun or adjective that refers to it. This problem is addressed by making use of RNNs that are able to capture long-range dependencies such as Long Short-Term Memory (LSTM) layers or Gated Recurrent Units (GRU). [13]. In either one of these models, the model can be *unidirectional* or *bidirectional.* In the latter case, the model uses information from both earlier as well as later in the sequence to learn patterns [15].

3 Data and models

This section describes the data used to train the models, and how the models are configured.

3.1 Data

We consider two separate datasets that are subdivided into three categories:

3.1.1 cmudict

This is a dataset containing 133.031 words mapping English words a phonetic representation using ARPAbet phoneme notation [16]. We remove words containing digits, punctuation and special characters. The final dataset contains 123.612 words. The data are subdivided into two data sets:

- *Multiple:* treats phonemes as individual characters. That is, a word like ‘abate’ is translated as ‘AA B EY T’ and would therefore contain the vocabulary entries ‘AA’, ‘B’, ‘EY’ and ‘T’.

- *Single:* in this case, the phonemes are ignored. Hence, the same word ('abate') now contains the vocabulary entries 'A', 'B', 'T', 'E', 'Y'.

This distinction between the datasets allow us to observe whether it is useful to retain the phonemes or not.

3.1.2 Wiktionary

This is a dataset that is retrieved from the wiktionary dump file [17]. We use the 'wikt2pron' python module [18] to extract all English words from the dump from the first of November 2018. The data contains 46.675 words with pronunciations in XSAMPA format after removal of entries containing digits and special characters.

One downside of the wikt2pron module (or from the wiktionary data set) is that it does not distinguish between dialects. That is, the module returns a python dictionary for each word as well as up to 20 pronunciation entries without specifying whether these are British-English, American-English or, for example, Australian-English. Clearly, this is an issue because the pronunciation of words differs across dialects. For the purposes of this analysis, we simply pick the first entry for each word.

3.2 Evaluation metrics

We evaluate the model on three separate metrics.

3.2.1 BLEU score

The BLEU score [19] is a measure of overlap between the predicted word and a reference that outputs a score between $0 - 1$, where a score of 0 means that there is no overlap and 1 means that there is perfect overlap.

Typically, the BLEU score is a combined metric measuring the overlap of one-grams, two-grams, three-grams and four-grams, each of which are weighted equally. Moreover, it takes into account the length of words (since shorter words are easier to predict). In this paper, we use the `sentence_bleu` function from the `nltk` python module [20]. We only present the cumulative 4-gram score (`BLEU-4`).

3.2.2 Accuracy in detecting homophones

One way to assess the accuracy of our model is to evaluate its performance on words that sound the same but that are spelled in different ways. Such words are called *homophones*. We define three metrics to measure the performance of our models in detecting homophones, all of which are proportions. These three metrics are as follows:

1. **all equal**: the proportion of observations for which all homophones map to the same pronunciation.

Let θ_i be the value of the i^{th} development set example. If all the predictions are equal, the value of $\theta_i = 1$, else, it equals 0, then

$$p_{\text{ all equal}} = \frac{1}{n}\sum_{i=1}^{n}\theta_i \qquad (1)$$

2. **equal to reference**: the proportion of observations for which all homophones are equal to the reference.

If we let θ_i be the value of the j^{th} word out of k words in the i^{th} development set example, which equals 1 if the j^{th} word equals the reference and 0 otherwise, then:

$$p_{\text{ equal to reference}} = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{k}\sum_{j=1}^{k}\theta_i\right] \tag{2}$$

3. **largest subgroup**: the average of the proportion of the largest sub-group of unique pronunciations for each example.

If we let u equal the largest subgroup of k possible subgroups, where k is the number of words in the i^{th} development set example. Furthermore, if we let θ_i be the value of the j^{th} word out of k words in the i^{th} development set example, which equals 1 if the j^{th} word equals the value of the largest subgroup u and 0 otherwise, then

$$p_{\text{ largest subgroup}} = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{k}\sum_{j=1}^{k}\theta_i\right] \tag{3}$$

These metrics measure similar concepts. Clearly, a model that scores well on $p_{\text{equal to reference}}$ has a high detection rate for homophones, while at the same time getting the 'actual' pronunciation right most of the time. For the purposes of this analysis, however, it is more important that the pronunciations are all equal to each other ($p_{\text{all equal}}$) than that they equal the actual pronunciation, as this metric still yields a good homophone detection rate while not imposing the strict condition that all predictions must be equal to the reference.

3.2.3 Accuracy in recognizing similar-sounding misspelled words

As a third evaluation of the model, we consider the problem of mapping misspelled - but phonetically similar - words to their phonetic representation. We take a random subset of 100 commonly misspelled words provided by Wikipedia [21] and manually classify these words as either 1 if the misspelled word should be phonetically similar to the correctly spelled word and 0 otherwise. This dataset gives us an idea of the extent to which the models perform on unseen and unusual data. The metrics used to evaluate this data set are accuracy, sensitivity, specificity and the F1 score.

3.3 Cross-validation

The data are split into several train and test sets. We first split the data based on homophones. This process works as follows: we first extract all homophones from the dataset, after which we select a subset as the test set. All English words that map to the same pronunciation are then added to the test set. This ensures that the pronunciation is unseen during training.

To evaluate the general skill (BLEU score) of the model, we then split the train set into a generic train and test split obtained in the previous step.

3.4 Models

This section describes the architecture of our models, all of which are trained using the python library

`Keras` [22].

3.4.1 Hyperparameters and training

All models are trained using stochastic gradient descent and Adam [23] with minibatch size 128 and optimizing crossentropy loss. We apply dropout and recurrent dropout [24] with a drop probability of 10% to both the encoder and decoder. We also use 'teacher forcing', meaning that instead of ingesting the predicted character generated at timestep $t-1$ to predict the next character in the sequence, we use the ground truth character [25]. All models are trained for 10 epochs on a single NVIDIA GTX 1060 with 6GB RAM.

Table 1: Hyperparameters for final models

	batch size	epochs	validation split	hidden dimension	learning rate
cmudict (multiple)	128	10	0.05	512	0.005
cmudict (single)	128	10	0.05	512	0.005
wiktionary	128	10	0.05	512	0.002

3.4.2 Encoder and decoder setup

The English words and their phonetic representation in our dataset are first converted to a one-hot-encoding (OHE). We then use a bidirectional LSTM that takes the OHE English words as input and that passes its internal state to the decoder. The decoder takes as its input the state vectors from the encoder as well as the OHE phonetic words. The decoder then generates the output sequence. (see e.g. [26])

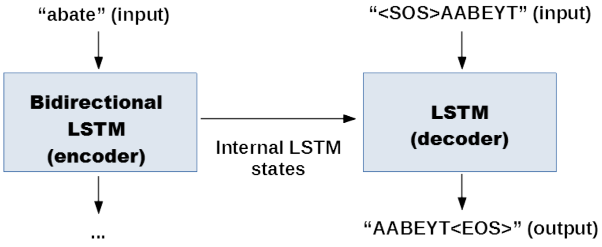

Figure 1: Encoder/decoder architecture

4 Results

This section describes the results of the models. Figure 2 below shows the average BLEU score for words of different lengths.

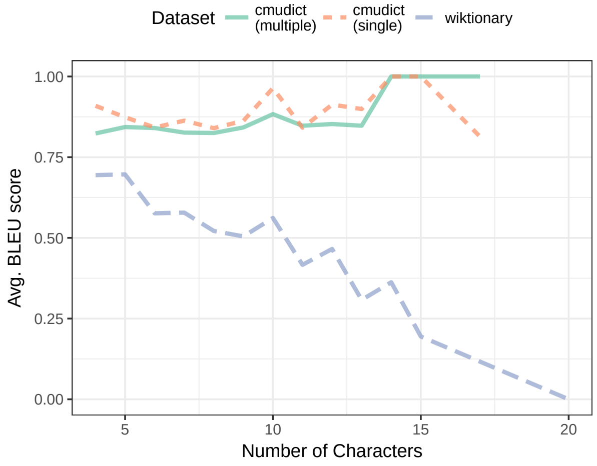

Figure 2: Average BLEU score on development set

The models trained on the `cmudict` data vastly outperform the `wiktionary` data. The `cmudict (single)` model achieves an average BLEU score of 86.78%, which is slightly higher than the `cmudict (multiple)` model at 83.37%.

Table 2 below shows the performance of the models on the homophone test set. The `cmudict (single)` model outperforms the other models on all metrics.

However, it is interesting that the models are much closer in terms of performance on the most important metric ($p_{\text{all equal}}$) than on the other metrics. It should be noted that these metrics do not measure any possible erroneous mappings ('false positives'). Hence, it may be the case that the `wiktionary` model achieves a high score on some of the metrics at the cost of making many more mistakes than the `cmudict` data.

Table 2: Percentage correct on each metric

	All equal	Largest subgroup	Equal to reference
cmudict (single)	77.33%	91.15%	86.94%
cmudict (multiple)	63.56%	86.58%	76.87%
wiktionary	42.11%	80.2%	51.02%

The accuracy across word length is similar across datasets, indicating that they perform well irrespective of the size of a word.

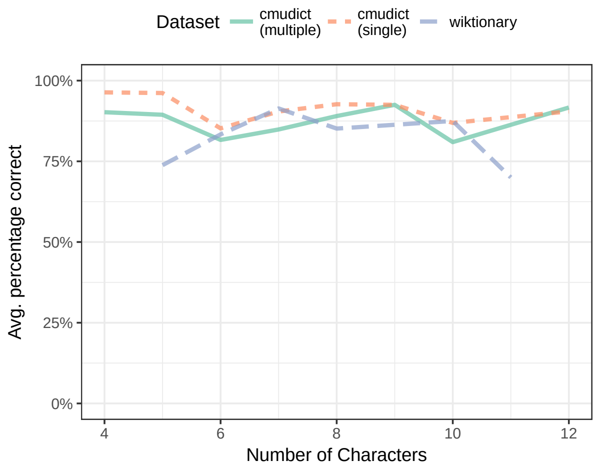

Figure 3: Average percentage correct (largest subgroup) across word length

Finally, we consider the performance of the model on misspelled words. Given that roughly one-third of our manually labelled data is considered to be of the positive class[2], we are most interested in the F1 and sensitivity scores obtained by the model.

Table 3: Evaluation on misspelled data set

	cmudict (single)	cmudict (multiple)	wiktionary
Accuracy	0.94	0.83	0.76
Sensitivity	0.87	0.77	0.69
Specificity	0.92	0.89	0.87
F1	0.90	0.83	0.77

As can be observed from table 2, the `cmudict (single)` model again outperforms the others. It achieves significantly better scores than the `cmudict (multiple)` model.

5 Discussion and conclusion

The RNN approach to phonetic normalization is, as is shown above, succesful in its main objective. That is, the models succeed in creating sensible output, are good at mapping homophones to the same phonetic representation, and perform adequately on misspelled words. On all metrics, the `cmudict (single)` model outperforms the other models. This is a rather curious outcome, as we would expect the model that uses phonemes (`cmudict (multiple)`) to incorporate more information and hence perform better.

There are several improvements to be made to the model. Firstly, there must be some noise in the data because it is not-context aware. This means that the model cannot learn the difference between heteronyms such as a word like 'tear' (to tear a piece of paper) or 'tear' (he teared up). Adding this kind

[2] indicating that the pronunciation of the mispelled word is the same as the word that was intended

of context awareness must be gauged from the sentence somehow, given that the models described in this paper deal exclusively with words at the character level.

Furthermore, while the model performed adequately on the wikipedia data, it faces several bottlenecks. Firstly, the size of the vocabulary is not ideal. This issue could be exacerbated for languages other than English, as we expect the size of vocabularies to be smaller for other languages that are not as widely used on the internet. Secondly, more work should be done to filter different dialects or pronunciations from this data. As mentioned earlier, this project used available software to extract words from the wiktionary. This software does not distinguish between different pronunciations and hence we were not able to filter for, for example, American or British pronunciations.

Finally, we have implemented a rather traditional infrastructure. RNNs are resource-heavy from an architectural point of view. Other models (for example pure Attention models or models that employ convolutional layers) have been shown to perform as good or better while substantially reducing the resources needed to train the model.

References

[1] M. Bisani and H. Ney, “Joint-sequence models for grapheme-to-phoneme conversion,” *Speech Communication*, vol. 50, no. 5, pp. 434–451, May 2008.

[2] A. Halma, “Chitchat: Script information retrieval and conversations.” https://bitbucket.org/arvid/chitchat.

[3] Y. Doval, M. Vilares, and J. Vilares, “On the performance of phonetic algorithms in microtext normalization,” *Expert Systems with Applications*, vol. 113, pp. 213–222, Dec. 2018.

[4] Z. Xue, D. Yin, and B. Davison, “Normalizing Microtext.” 2011.

[5] V. López-Ludeña, R. San-Segundo, J. M. Montero, R. Barra-Chicote, and J. M. Lorenzo, “Architecture for Text Normalization using Statistical Machine Translation techniques,” 2012.

[6] C. Li and Y. Liu, “Normalization of Text Messages Using Character- and Phone-based Machine Translation Approaches,” 2012, vol. 3.

[7] R. Satapathy, C. Guerreiro, I. Chaturvedi, and E. Cambria, “Phonetic-Based Microtext Normalization for Twitter Sentiment Analysis,” in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 407–413.

[8] R. Khoury, “Microtext normalization using probably-phonetically-similar word discovery,” in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2015, pp. 384–391.

[9] R. Khoury, “Phonetic normalization of microtext,” in *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2015, pp. 1600–1601.

[10] W. Min and B. W. Mott, “NCSU_SAS_WOOKHEE: A Deep Contex-

tual Long-Short Term Memory Model for Text Normalization," in *NUT@IJCNLP*, 2015.

[11] G. Chrupała, "Normalizing tweets with edit scripts and recurrent neural embeddings."

[12] N. Jaitly and R. Sproat, "An RNN Model of Text Normalization," 2017.

[13] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[14] K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv:1406.1078 [cs, stat]*, Jun. 2014.

[15] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[16] Carnegie Mellon University, "The CMU Pronouncing Dictionary." http://www.speech.cs.cmu.edu/cgi-bin/cmudict?stress=-s&in=CITATION.

[17] Wikipedia, "Wikimedia downloads," *Index of /enwiktionary/*. https://dumps.wikimedia.org/enwiktionary/20181101/enwiktionary-20181101-pages-meta-current.xml.bz2.

[18] Y. Xiong, "Toolkit converting pronunciation in enwiktionary xml dump to cmudict format: Abuccts/wikt2pron." https://github.com/abuccts/wikt2pron, Oct-2018.

[19] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002, pp. 311–318.

[20] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python.* 2009.

[21] wikipedia, "Wikipedia:Lists of common misspellings/For machines," *Wikipedia.* https://en.wikipedia.org/w/index.php?title=Wikipedia:Lists_of_common_misspellings/For_machines&oldid=838176804, Apr-2018.

[22] F. Chollet, "Keras." https://keras.io, 2015.

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014.

[24] Y. Gal and Z. Ghahramani, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," *arXiv:1512.05287 [stat]*, Dec. 2015.

[25] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor Forcing: A New Algorithm for Training Recurrent Networks," *arXiv:1610.09038 [cs, stat]*, Oct. 2016.

[26] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.