INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

An RNN Model of Text Normalization

Richard Sproat[1], *Navdeep Jaitly*[2]
[1]Google, New York, NY, USA
[2]Google, Mountain View, CA, USA (now at NVIDIA)
[1]`rws@google.com`

Abstract

We present a recurrent neural net (RNN) model of text normalization — defined as the mapping of *written* text to its *spoken* form, and a description of the open-source dataset that we used in our experiments. We show that while the RNN model achieves very high overall accuracies, there remain errors that would be unacceptable in a speech application like TTS. We then show that a simple FST-based filter can help mitigate those errors. Even with that mitigation challenges remain, and we end the paper outlining some possible solutions. In releasing our data we are thereby inviting others to help solve this problem.
Index Terms: text normalization, RNNs, machine learning

1. Introduction

Within the last few years a major shift has taken place in speech and language technology: the field has been taken over by neural deep learning approaches. This change is surely justified by the impressive performance gains to be had by deep learning, something that has been demonstrated in a range of areas from image processing, handwriting recognition, acoustic modeling in automatic speech recognition (ASR), waveform synthesis for text-to-speech (TTS), machine translation, parsing, and *go* playing to name but a few. While improvements with neural models have been significant, it is of course also true that such systems make errors, and depending upon the application and the type of error, the result may range from amusing to unacceptable. In this paper we present an application where the bar is rather high on what counts as an "acceptable" error, namely *text normalization*, in the sense of a system that converts from a written representation of a text into a representation of how that text is to be read aloud — typically a sequence of fully spelled words. The target applications are TTS and ASR (in the latter case mostly for generating language modeling data from raw written text). This problem, while often considered mundane, is very important, and a major source of degradation of perceived quality in TTS systems can be traced to problems with text normalization.

We start by describing why this application area is different other areas like machine translation. Then, after summarizing prior work, we describe our open-source dataset, followed by some experiments we have conducted using recurrent neural nets (RNNs). The RNNs produce very good results when measured in terms of overall accuracy, but they produce errors that would make them risky to use in a real application. We also demonstrate that these errors can be ameliorated with a simple FST-based filter used in tandem with the RNN. We end by suggesting some further ways in which the performance might be improved, and we also invite others to research this problem based on the dataset we have provided.

2. Why text normalization is different

To lay the groundwork for discussion, consider that a native speaker of English would read example 1 as something like 2:

1. A baby giraffe is **6ft** tall and weighs **150lb**.
2. A baby giraffe is **six feet** tall and weighs **one hundred fifty pounds**.

In the original written form there are two *non-standard words* [1], namely the two measure expressions *6ft* and *150lb*. In order to read the text, each of these must be *normalized* into a sequence of ordinary words. In this case both examples are instances of the same *semiotic class* [2], namely measure phrases. But in general texts may include non-standard word sequences from a variety of different semiotic classes, including measures, currency amounts, dates, times, telephone numbers, cardinal or ordinal numbers, fractions, among many others. Each of these involves a specific relation mapping between the written input form and the spoken output form.

If one were to train a deep-learning system for text normalization, one might consider presenting the system with a large number of input-output pairs such as:

a	`<self>`
baby	`<self>`
giraffe	`<self>`
is	`<self>`
6ft	six feet
tall	`<self>`
and	`<self>`
weighs	`<self>`
150lb	one hundred fifty pounds
.	sil

Here we use a special token `<self>` to indicate that the input is to be left alone. In principle this seems like a reasonable approach, but there are a few issues that need to be considered.

First, data of the kind described above needs to be created. Unlike the case of machine translation, where people often translate texts into other languages, and where it is therefore possible to find parallel or near-parallel texts, people rarely feel the need to create normalized versions of text. Thus the training data for a text normalization system needs to be created for every language where we would like to train such a system.

Secondly the set of "interesting" cases in text normalization is usually very sparse: most tokens, as we can see in the example, map to themselves, and while it is certainly important to get that right, one generally does not get any credit for doing so either. What is evaluated in text normalization systems is the interesting cases, the numbers, times, dates, measure expressions, currency amounts, and so forth, that require special treatment.

This leads directly to the third point, namely that with text normalization, the requirements on accuracy are rather stringent: if the text reads *381 kg*, then an English TTS system had better say *three hundred eighty one kilograms*, or maybe *three hundred eighty one kilogram*, but certainly not *four hundred eighty two centimeters*. But these kinds of "silly" errors are errors that neural models trained on these sorts of data will make, as we demonstrate below. In contrast, a well-constructed hand-built system such as [3] might be brittle and fail for many classes of cases, but it would not produce errors of the kind we have just described. So one can have high overall accuracy, but a few bad errors of the kind just described make the system unusable. This is in stark contrast to the situation with neural MT, where such errors also occur [4], but where errors are better tolerated so long as the system has a better BLEU score.

3. Prior work on text normalization

Text normalization has a long history in speech technology, dating back to the earliest work on full TTS synthesis [5]. [6] provided a unifying model for most text normalization problems in terms of weighted finite-state transducers (WFSTs). The first work treating the problem of text normalization as essentially a language modeling problem was [1]. More recent machine learning work specifically addressed to TTS text normalization includes [7, 8, 9]. In the last few years there has been a lot of work that focuses on social media [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21], with some recent neural work in this space [22, 23]. Such work tends to focus on different problems from those of TTS: on the one hand one, in social media one often has to deal with odd spellings of words such as `cu l8r`, `coooooooooooooollll`, or `dat suxxx`, which are less of an issue in most applications of TTS; on the other, for example, expansion of digit sequences into words is critical for TTS, but of no interest to the normalization of social media texts.

4. Dataset

Our data consists of 1.1 billion words of English text, and 290 million words of Russian text, from Wikipedia regions that could be decoded as UTF8, divided into sentences, and run through the Google TTS system's Kestrel text normalization system to produce verbalizations; see [3] for details on how Kestrel works. The problem for present purposes, then, is for the system to learn to reconstruct what an already existing hand-built system does. The format of the annotated data is as in Section 1. Semiotic class instances are verbalized as sequences of fully spelled words, most ordinary words are left alone (represented here as `<self>`), and punctuation symbols are mostly transduced to `sil` (for "silence").

The data were divided into 90 files (roughly 90%) for training, 5 files for online evaluation during training (the "development" set), and 5 for testing. In the test results reported below, we used the first 100K tokens of the final file (99) of the test data, including the end-of-sentence marker, working out to about 92K real tokens for English and 93K real tokens for Russian. A manual analysis of about 1,000 examples from the test data suggests an overall Kestrel error rate of approximately 0.1% for English and 2.1% for Russian. The largest category of errors for Russian involves years being read as cardinal numbers rather than the expected ordinal form. Despite these errors there is no *a priori* reason to believe that the current task is any less valid than working with hand-annotated data would be. Note that although the test data were of course taken from a different portion of the Wikipedia text than the training and development data, nonetheless a huge percentage of the individual tokens of the test data — 98.9% in the case of Russian and 99.5% in the case of English — were found in the training set. The data are available at `https://github.com/rwsproat/text-normalization-data`.

5. Attention-based RNN model

We model text normalization as a sequence-to-sequence problem [24], in particular modeling the task as one where we map a sequence of input characters to a sequence of output words. For the input, the string must be in terms of *characters*, since for a token like *123*, one needs to see the individual digits to know how to read it; similarly it helps to see the individual characters for a possibly OOV word such as *snarky* to classify it as a token to be left alone (`<self>`). On the other hand it is desirable to have the output be in terms of *words*, since if nothing else this shortens the output sequences to be computed.

We use a Tensor Flow [25] model with an attention mechanism [26]. Attention models are crucial to sequence-to-sequence problems with a larger number of steps, since they are able to propagate new information from the encoder states to the decoder at each output step, through the use of the attention mechanism. Our architecture follows closely that of [27]. Specifically, we used a 4-layer bidirectional LSTM reader (but without the pyramidal structure in [27]) that reads input characters, a layer of 256 attentional units, and a 2-layer decoder that produces word sequences. The reader is referred to [27] for more details of the framework.

We learn to predict on a *token-by-token* basis placing each successive token in a (sliding) window of 3 tokens to the left and 3 to the right, marking the to-be-normalized token with a distinctive begin and end tag `<norm>...</norm>`. Thus the token `123` in the context *I live at ... King Ave .* would appear as

```
I live at <norm> 123 </norm> King Ave .
```

on the input side, which would map to

```
one twenty three
```

on the output side. This representation helps to limit the number of input and output nodes over what one would need if one modeled the mapping from a full input sentence to its verbalization.

The training, development and testing data were the same as described in Section 4 above. The English RNN was trained for about five and a half days (460K steps) on 8 GPUs until the perplexity on the held-out data was 1.003; Russian was trained for five days (400K steps), reaching a perplexity of 1.002. See [28] for some further details on the training.

5.1. Results

Results on the full training set are shown in Table 1 for English and Russian.[1] Some problematic errors involving wrong units, and other problems are shown in Table 2.

As noted above, even though the test data are obviously from a completely separate portion of Wikipedia from the training and evaluation data, there is considerable overlap in individual tokens. Could the system simply be memorizing? Evidently not, since for English, 90.6% of the cases *not* found in the training data were correctly produced (compared to 99.8% of

[1]Key non-obvious semiotic classes: ALL = all cases; PLAIN = ordinary word (`<self>`); PUNCT = punctuation (`sil`); TRANS = transliteration; LETTERS = letter sequence; CARDINAL = cardinal number; VERBATIM = verbatim reading of character sequence; ORDINAL = ordinal number; DECIMAL = decimal fraction; ELECTRONIC = electronic address; DIGIT = digit sequence; MONEY = currency amount; FRACTION = non-decimal fraction; TIME = time expression. N is sometimes slightly different for each training condition since in a few cases the model produces no output, and we discount those cases — thus in effect giving the model the benefit of the doubt.

Table 1: *Accuracies for the RNN model. See Footnote 1 for details on the semiotic classes.*

Semiotic Class	English		Russian	
ALL	92416	0.997	93184	0.993
PLAIN	68029	0.998	60747	0.999
PUNCT	17726	1.000	20263	1.000
DATE	2808	0.999	1495	0.976
TRANS	–	–	4103	0.921
LETTERS	1404	0.971	1839	0.991
CARDINAL	1067	0.989	2387	0.940
VERBATIM	894	0.980	1298	1.000
MEASURE	142	0.986	409	0.883
ORDINAL	103	0.971	427	0.956
DECIMAL	89	1.000	60	0.867
ELECTRONIC	21	1.000	2	1.000
DIGIT	37	0.865	16	1.000
MONEY	36	0.972	19	0.842
FRACTION	13	0.923	23	0.826
TIME	8	0.750	8	0.750

the seen cases); for Russian 86.7% of the *unseen* cases were correct (versus 99.4% of the seen cases). Complicated previously unseen cases in Russian, for example include examples like *17.04.1750*, correctly read as *семнадцатое апреля тысяча семьсот пятидесятого года* (‘seventeenth of April of the one thousand seven hundred fiftieth year’).

The results just reported depended on impractically large amounts of training data. To develop a system for a new language one needs a system that could be trained on data sets of a size that one could expect a team of native speakers to hand-label. Assuming one is willing to invest a few weeks’ of work with a small team, it is not out of the question that one could label about 10 million words of Wikipedia-style text.[2] With this point in mind, we retrained the systems on 11.4 million tokens of English from the beginning of the original training set, and 11.9 million tokens of Russian. The system was trained for about 7 days for both languages, until the system had achieved a perplexity on held-out data of 1.002 and for Russian 1.007.

Results are presented in Table 4. The overall performance is not greatly different from the system trained on the larger dataset, and in some places is actually better. The test data overlapped with the training data in 96.9% of the tokens for English and 95.5% for Russian, with the accuracy of the non-overlapped tokens being 95.0% for English and 93.5% for Russian.

The errors made by the system are comparable to errors we have already seen, though in English the errors in this case seem to be more concentrated in the reading of numeric dates. Thus to give just a few examples for English, reading *2008-07-28* as “the eighteenth of september seven thousand two”, or *2009-10-02* as *the ninth of october twenty thousand two*. Some relatively complicated examples not seen in the training data that the English system got right included *221.049 km²* as *two hundred twenty one point o four nine square kilometers*, *24 March 1951* as *the twenty fourth of march nineteen fifty one* and *$42,100* as *forty two thousand one hundred dollars.*

Clearly then, the attention-based models are able to achieve, with reasonable-sized data, performances that are close to what it achieves with larger training data. That said, the system of course still makes “silly” errors, meaning that it will not suffice on its own as the text normalization component of a TTS system.

[2]One of the authors hand-labeled 1,000 tokens of English in about 7 minutes. Data sets of about 10M tokens for four languages are in preparation.

6. Finite-state filters

As we saw in the previous section, an approach that uses attention-based sequence-to-sequence models can produce extremely high accuracies, but is still prone to occasionally producing output that is completely misleading given the input. What if we apply some additional knowledge to filter the output so that it removes silly analyses? One way to do this is to inject knowledge into the decoding by means of a *weak covering grammar*. For example one can construct an FST that maps from expressions of the form `<number> <measure_abbreviation>` to a cardinal or decimal number and the possible verbalizations of the measure abbreviation. Thus *24.2kg* might verbalize as *twenty four point two kilogram* or *twenty four point two kilograms*. The FST thus implements an *overgenerating grammar that includes the correct verbalization, but allows other verbalizations as well.* Such grammars are easy to develop — significantly easier than a full-blown finite-state grammar such as required by Kestrel. They can be used to guide the system in cases where it is error prone, a useful feature in any practical application of a neural system.

We constructed a simple Thrax grammar [29] to cover MEASURE and MONEY expressions, two classes where the RNN is particularly prone to produce silly readings. The grammar incorporates language-specific lexical information (*kg* is *kilogram*/*kilograms*) as well as a number name FST that is learned from a few hundred examples using the algorithm from [30].

During decoding, the FST is composed with the input token being considered by the RNN. If the composition fails — e.g. because this token is not one of the classes that the FST handles — then the decoding will proceed as it normally would via the RNN alone. If the composition succeeds, then the FST is projected to the output, and the resulting output *lattice* is used to restrict the possible outputs from the RNN. Since the input was a specific token — e.g. *2kg* — the output lattice will include only sequences that may be verbalizations of that token. This output lattice is transformed so that all prefixes of the output are also allowed (e.g. *two* is allowed as well as *two kilograms*). This can be done simply by walking the states in the lattice, and making all non-final states final. However, we wish to give a strong *reward* for traversing the whole lattice from the initial to an original final state, and so the exit cost for original final states is set to a very low negative value.

The RNN decoder queries the lattice with a sequence of labels, the object being to find the possible transitions to the next labels and their cost. The label sequence is first transformed into a trivial acceptor, to which is concatenated an FSA that accepts any single output token, and thus has a branching factor of $|V|$, the size of the output vocabulary. This FSA is then composed with the lattice. For strings in the FSA that match against the lattice, the cost will be the exit cost at that state in the lattice; for strings that fail the cost will be `infinity`. Suppose that the input sequence starts with `two hundred`, and that the output lattice allows `two hundred kilogram` or `two hundred kilograms`. Then the FST will return a score of `infinity` for the label `milligram`, for example. However for `kilogram` or `kilograms` it will return a non-infinite cost, and indeed since exiting on one of these corresponds to an original final state of the grammar, it will accrue the reward discussed above. These costs are then summed with the RNN’s own (log probability) scores for the sequence. Since all prefixes of the sequences allowed by the grammar are also allowed, the RNN could produce *two hundred* as the output, but it will get a substantial reward for finishing the sequence (*two hundred kilogram*

Table 2: *Errors from the RNN.*

Input	Correct	Prediction
2 mA	two milliamperes	two million liters
11/10/2008	the tenth of november two thousand eight	the tenth of october two thousand eight
1/2 cc	half a c c	one minute c c
18:00:00Z	eighteen hours zero minutes and zero seconds z	eighteen hundred cubic minutes
750 вольт	семисот пятидесяти вольт	семьсот пятьдесят гектаров
750 volts	seven hundred fifty volts	seven hundred fifty hectares
16 ГБ	шестнадцати гигабайтов	шестнадцати герц
16 GB	sixteen gigabytes	sixteen hertz

Table 3: *Misleading readings corrected by the FST.*

Input	RNN	RNN+FST
£5	five	five pounds
11 billion AED	eleven billion danish	eleven billion dirhams
2 mA	2 megaamperes	2 milliamperes
33 rpm	thirty two revolutions per minute	thirty three revolutions per minute

Table 4: *Accuracies for the RNN on smaller training sets.*

	English		Russian	
ALL	92416	0.996	93184	0.994
PLAIN	68029	0.997	60747	0.999
PUNCT	17726	1.000	20263	1.000
DATE	2808	0.974	1495	0.977
TRANS	–	–	4103	0.942
LETTERS	1404	0.974	1839	0.991
CARDINAL	1067	0.991	2387	0.954
VERBATIM	894	0.977	1298	1.000
MEASURE	142	0.958	409	0.927
ORDINAL	103	0.971	427	0.981
DECIMAL	89	1.000	60	0.917
ELECTRONIC	21	0.952	2	1.000
DIGIT	37	0.703	16	1.000
MONEY	36	0.972	19	0.895
FRACTION	13	0.846	23	0.739
TIME	8	0.625	8	0.750

or *two hundred kilograms*). This is usually sufficient to guide the RNN to better path.[3]

Accuracies in English for the unfiltered and filtered RNN outputs, where the RNN is trained on the smaller training set described in the previous section, are given in Table 5a. The MEASURE and MONEY sets show substantial improvement. None of the other semiotic classes are affected, exactly as desired.

In order to focus in on the differences between the filtered and unfiltered models, we prepared a different subset of the final training file that was rich in MEASURE and MONEY expressions. Specifically, we selected 1,000 sentences, each of which had one expression in that category. We then decoded with the models trained on the smaller training set, with and without the FST filter. Results are presented in Table 5b. Once again, the FST filter improves the overall accuracy for MONEY and MEASURE, leaving the other classes unaffected. Some examples of the improvements in both categories are shown in Table 3. Looking more particularly at measures, where the largest differences are found, we find that the only cases where the FST filter does not help is cases where the grammar fails to match against the input and the RNN alone is used to predict the output. These cases included *1/2 cc*, *30'* (for *thirty feet*), *80'*, *7000 hg* (which uses the unusual unit *hectogram*), and *600 billion kWh* (the measure grammar did not allow for a spelled number like *billion*). In some other cases, the FST does not constrain the RNN enough: *1 g* still comes out as *one grams*, since the FST allows this, but this is more "acceptable" since it is at least not misleading.

[3]We note in passing that this method is more or less the opposite approach to that of [31]. In that work, the FST's scoring is augmented by an RNN, whereas in the present approach, the RNN's decoding is guided by the use of an FST.

Table 5: *(a.) Accuracies for the attention-based sequence-to-sequence models for English on smaller training sets, with and without an FST filter, considering just MEASURE and MONEY. (b.) Similar results on the MEASURE-MONEY-rich test set. Numbers in the left column are the total count of each class.*

a.	RNN		RNN+FST filter
MEASURE	142	0.972	0.993
MONEY	36	0.972	1.000

b.	RNN		RNN+FST filter
MEASURE	979	0.979	0.991
MONEY	312	0.965	0.971

Similar results were obtained for a MEASURE-MONEY-rich set where none of the tokens had previously been seen in the training data, as well as for Russian: the reader is referred to [28] for details. All in all then, the FST-filtration approach seems to be a viable way to improve the quality of the output for semiotic classes where the RNN is prone to make errors.

7. Discussion

We have presented an attention-based RNN for text normalization for speech applications. While the model performs well overall, it has a propensity to make errors that would make it risky to use in an application such as TTS. We have also proposed augmenting the system with an FST filter, and shown that this approach can dramatically improve performance.

In future work we are investigating other techniques for improving performance of the RNN, including augmenting training data with synthetic examples produced by overgenerating grammars reestimated using EM, autoencoding using unannotated data [32], and better tailored loss functions. As we noted before, the data used in our experiments is available open-source. We invite others to make use of the data and perhaps come up with better solutions than what we have reported here.

8. Acknowledgements

We thank Alexander Gutkin, Kyle Gorman, Brian Roark and Suyoun Yoon and four anonymous reviewers for comments on earlier versions of this paper. Alex and Kyle also helped with the data preparation.

9. References

[1] R. Sproat, A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards, "Normalization of non-standard words," *Computer Speech and Language*, vol. 15, no. 3, pp. 287–333, 2001.

[2] P. Taylor, *Text-to-Speech Synthesis*. Cambridge: Cambridge University Press, 2009.

[3] P. Ebden and R. Sproat, "The Kestrel TTS text normalization system," *Natural Language Engineering*, vol. 21, no. 3, pp. 1–21, 2014.

[4] P. Arthur, G. Neubig, and S. Nakamura, "Incorporating discrete translation lexicons into neural machine translation," in *EMNLP*, Austin, TX, 2016.

[5] J. Allen, S. M. Hunnicutt, and D. Klatt, *From Text to Speech: The MITalk System*. Cambridge University Press, 1987.

[6] R. Sproat, "Multilingual text analysis for text-to-speech synthesis," *Natural Language Engineering*, vol. 2, no. 4, pp. 369–380, 1996.

[7] ——, "Lightly supervised learning of text normalization: Russian number names," in *IEEE SLT*, 2010, pp. 436–441.

[8] B. Roark and R. Sproat, "Hippocratic abbreviation expansion," in *ACL*, 2014, pp. 364–369.

[9] R. Sproat and K. Hall, "Applications of maximum entropy rankers to problems in spoken language processing." in *Interspeech*, 2014, pp. 761–764.

[10] Y. Xia, K.-F. Wong, and W. Li, "A phonetic-based approach to Chinese chat text normalization," in *ACL*, Sydney, Australia, 2006, pp. 993–1000.

[11] M. Choudhury, R. Saraf, V. Jain, S. Sarkar, and A. Basu, "Investigation and modeling of the structure of texting language." *International Journal of Document Analysis and Recognition*, vol. 10, pp. 157–174, 2007.

[12] C. Kobus, F. Yvon, and G. Damnati, "Normalizing SMS: are two metaphors better than one?" in *COLING*, Manchester, UK, 2008, pp. 441–448.

[13] R. Beaufort, S. Roekhaut, L.-A. Cougnon, and C. Fairon, "A hybrid rule/model-based finite-state framework for normalizing SMS messages," in *ACL*, Uppsala, Sweden, 2010, pp. 770–779.

[14] M. Kaufmann, "Syntactic normalization of Twitter messages," in *International Conference on NLP*, 2010.

[15] F. Liu, F. Weng, B. Wang, and Y. Liu, "Insertion, deletion, or substitution? Normalizing text messages without pre-categorization nor supervision," in *ACL*, Portland, Oregon, USA, 2011, pp. 71–76.

[16] D. Pennell and Y. Liu, "A character-level machine translation approach for normalization of SMS abbreviations." in *IJCNLP*, 2011.

[17] A. T. Aw and L. H. Lee, "Personalized normalization for a multilingual chat system," in *ACL*, Jeju Island, Korea, 2012, pp. 31–36.

[18] F. Liu, F. Weng, and X. Jiang, "A broad-coverage normalization system for social media language," in *ACL*. Jeju Island, Korea: Association for Computational Linguistics, 2012, pp. 1035–1044.

[19] X. Liu, M. Zhou, X. Zhou, Z. Fu, and F. Wei, "Joint inference of named entity recognition and normalization for tweets," in *ACL*, Jeju Island, Korea, 2012, pp. 526–535.

[20] H. Hassan and A. Menezes, "Social text normalization using contextual graph random walks," in *ACL*, 2013, pp. 1577–1586.

[21] Y. Yang and J. Eisenstein, "A log-linear model for unsupervised text normalization," in *EMNLP*, 2013, pp. 61–72.

[22] G. Chrupała, "Normalizing tweets with edit scripts and recurrent neural embeddings," in *ACL*, Singapore, 2014.

[23] W. Min and B. Mott, "NCSU_SAS_WOOKHEE: A deep contextual long-short term memory model for text normalization," in *WNUT*, 2015.

[24] 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.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR 2015*, 2014.

[27] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *ICASSP*, 2016, pp. 4960–4964.

[28] R. Sproat and N. Jaitly, "Rnn approaches to text normalization: A challenge," Google, https://arxiv.org/abs/1611.00068, Tech. Rep., 2016.

[29] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai, "The OpenGrm open-source finite-state grammar software libraries," in *ACL*, 2012, pp. 61–66.

[30] K. Gorman and R. Sproat, "Minimally supervised models for number normalization," *Transactions of the Association for Computational Linguistics*, 2016.

[31] P. Rastogi, R. Cotterell, and J. Eisner, "Weighting finite-state transductions with neural context," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, 2016, pp. 623–633.

[32] Y. Cheng, W. Xu, Z. He, W. He, H. Wu, M. Sun, and Y. Liu, "Semi-supervised learning for neural machine translation," *CoRR*, vol. abs/1606.04596, 2016.