Semi-supervised URL Segmentation with Recurrent Neural Networks Pre-trained on Knowledge Graph Entities

Hao Zhang and **Jae Ro** and **Richard Sproat**
Google Research
{haozhang,jaero,rws}@google.com

Abstract

Breaking domain names such as openresearch into component words open and research is important for applications like Text-to-Speech synthesis and web search. We link this problem to the classic problem of Chinese word segmentation and show the effectiveness of a tagging model based on Recurrent Neural Networks (RNNs) using characters as input. To compensate for the lack of training data, we propose a pre-training method on concatenated entity names in a large knowledge database. Pre-training improves the model by 33% and brings the sequence accuracy to 85%.

1 Introduction

Word segmentation is a fundamental NLP analysis problem for written languages with no space delimiters between words such as Chinese and Japanese. In the age of digital communications, new URLs (e.g. www.openresearch.org) and hashtags (e.g. #photooftheday), which often include strings of concatenated words (openresearch, photooftheday) are being added every day to a growing set of tokens that an NLP system may need to deal with, and they pose challenges for language and speech applications. For example, a Text-to-Speech (TTS) synthesis system will struggle to pronounce these concatenated tokens, since simply applying a grapheme-to-phoneme system out of the box to something like photooftheday will usually yield poor results. This suggests the need for a model that can split such tokens into the component words. So-called "end-to-end" neural TTS systems (Sotelo et al., 2017; Wang et al., 2017), which learn to map directly from character sequences to speech might seem to hold out the hope of avoiding treating this problem separately. However, the fact that URLs occur relatively rarely in most TTS training data limits the promise of such models on this long-tail problem.

The problem of analyzing URLs does differ in one useful way from more general text normalization problems. For a token such as 123 in a text, one typically needs to know what context it occurs in in order to know how to read it: is it one hundred twenty three or one twenty three; see (Sproat et al., 2001), inter alia. In the case of URLs, these are largely *context-independent* since the output segmentation is usually unaffected by the surrounding words. Hence the problem can be treated as a standalone one that does not require the system to be trained as part of broader text normalization training.

Our training data comes from camel case URLs that naturally define the segment boundaries (e.g. NYTimes.com maps to N Y Times . com) along with manual corrections for non-trivial boundaries. We release our training and evaluation data sets to promote research on this problem. By drawing an analogy with Chinese word segmentation, we cast the URL segmentation problem as a sequence tagging problem. We propose a simple Recurrent Neural Network (RNN) based tagger with an encoder and a decoder.

The model trained on the data set has a decent full sequence accuracy (64%) but fails to generalize to more rare words due to the size of the training data. Inspired by the success of pre-training in many NLP tasks (Peters et al., 2017; Devlin et al., 2019), we propose a pre-training recipe for the segmenter. Based on the observation that URLs are often compound entity names and so are knowledge graph entities

Proceedings of the 28th International Conference on Computational Linguistics, pages 4667–4675
Barcelona, Spain (Online), December 8-13, 2020

(Bollacker et al., 2008), we create a large synthetic training data set by concatenating the knowledge graph entity names. We observe 21% absolute (33% relative) improvement in sequence accuracy after applying pre-training followed by fine-tuning.

2 Related Work

Every text-to-speech system has to be able to read URLs and other electronic addresses, but there is very little in the published literature that discusses this problem specifically as a research topic, and most systems seem not to do much in the way of interesting analysis of the internal components of the address. For example, the Kestrel text normalization system (Ebden and Sproat, 2014) identifies URLs and other electronic addresses using finite-state matchers, and parses the main components into separate tokens based on standard delimiters (`/`, `:`, etc.): thus `www.google.com` would be parsed into `www . google . com`. The individual components are then pronounced separately. Common components such as `.`, `www`, `nytimes` and `com` are handled by table look-up, but other components, such as `jpopasia` in `jpopasia.com` are not otherwise broken down and if they do not match a lexicon entry, may end up being read letter-by-letter.

The problem has applications beyond TTS. In web search, analyzing URLs and hashtags leads to better matching. Wang et al. (2011) termed the problem "URL word breaking". They used a noisy channel model with an n-gram language model trained on word-segmented data and a word-synchronous beam search algorithm for inference. The model is essentially unsupervised. They found that the style of the text used to build the model played a crucial role and document titles yielded the best results. In our pre-training experiments, we also tried web queries and documents. None of them gave the same improvement as the knowledge graph entity names in title case. Srinivasan et al. (2012) improved the model of Wang et al. (2011) by adding a supervised max-margin structured prediction model using individual unsupervised language models as features. Fine-tuning on in-domain data is the counterpart in our system. Both have created training or evaluation data sets of URL domain names for their experiments, but these have not been publicly released. We contribute a data set of URLs crawled from a public repository of Web texts with their internal segments annotated by crowd sourcing.

Chiang et al. (2010) reported experiments on a related artificial problem of splitting of space-free English through Bayesian inference for FSTs following the work of Goldwater et al. (2009).[1]

A closely related problem is compound splitting. Macherey et al. (2011) presented an unsupervised probabilistic model for splitting compound words into parts, with the compound part sub-model being a zero-order model to enable efficient dynamic programming inference. The model is optimized for the task of machine translation. They only reported results for seven (Germanic, Hellenic, and Uralic) languages other than English. More recently, fully supervised letter sequence labelling models have been introduced for German compound splitting (Ma et al., 2016) and Sanskrit word splitting (Hellwig and Nehrdich, 2018). Pre-training can potentially further improve these models.

There is a large body of research on Chinese word segmentation. The best models are supervised ones using structured prediction (Peng et al., 2004), transition-based models (Zhang and Clark, 2007), and most recently RNNs (Ma et al., 2018) and BERT-based models (Huang et al., 2019). The superior results of the BERT-based models demonstrate that pre-training is effective on word segmentation. Unlike BERT, we pre-train the entire model, not just the encoder.

3 RNN Tagging Model

We formulate the segmentation problem as a character sequence tagging model. Given an input character sequence $X = x_1, \ldots, x_I$, the model predicts an output tag sequence $Y = y_1, \ldots, y_I$, with $y_i \in \{\text{B}, \text{I}\}$ being the tag for the character x_i. B indicates the underlying character starts a new segment. I indicates the underlying character continues the current segment. Tag sequences $\{\text{B}, \text{I}\}^+$ correspond one-to-one to segment sequences. For example, B,B,I,B is equivalent to the segment sequence $[x_1], [x_2, x_3], [x_4]$.

[1] It is worth noting that the problem of resegmenting space-free English has a long history: the earliest reference we are aware of is (Olivier, 1968).

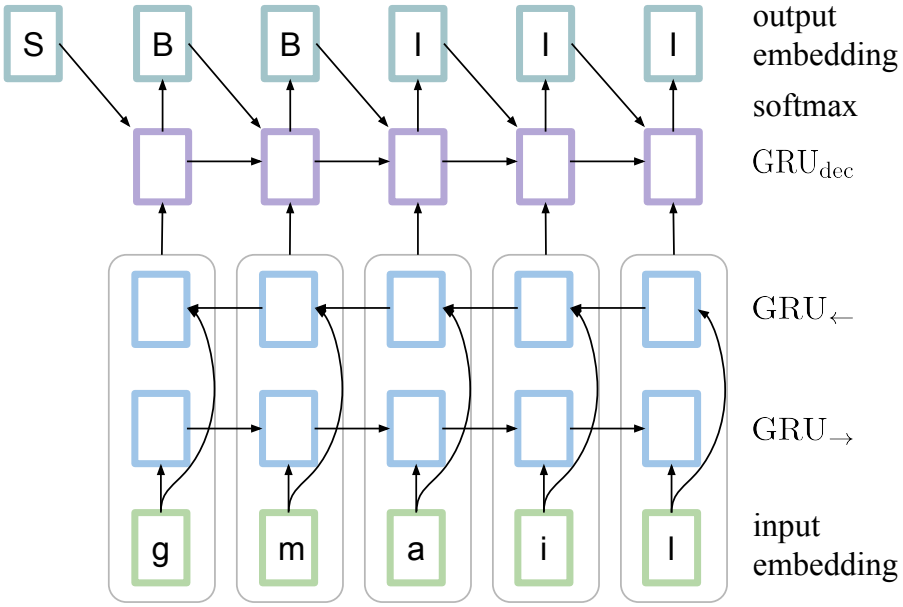

Figure 1: RNN tagging model architecture. GRU$_{\rightarrow}$ and GRU$_{\leftarrow}$ are forward and backward encoder RNNs implemented with Gated Recurrent Units (GRU). GRU$_{dec}$ is the decoder RNN. The concatenation of GRU$_{\rightarrow}$, GRU$_{\leftarrow}$, and the input embedding at the current position, as well as the embedding of the previous tag is fed to GRU$_{dec}$. The output of GRU$_{dec}$ is fed to a softmax layer to generate the output tagging sequence.

We model $P(Y|X)$ with an encoder-decoder RNN. The encoder generates hidden state sequences of length I. At decoding step i, the decoder attends to position i in both the hidden state sequences and the input embedding sequence. Figure 1 illustrates the architecture of the model. Our architecture is a modification of the stacking LSTM architecture of Ma et al. (2018) which can also be understood as using hard attention (Aharoni and Goldberg, 2017) in the decoder. We do not use bigram character features. Instead, we rely on forward and backward RNNs for implicit input feature extraction. We make the decoder auto-regressive by feeding the previously predicted tag to the decoder RNN and applying beam search in inference.

3.1 Pre-training

RNN models demand a large number of training examples. While labelled data is often scarce, un-labelled data with matching domain is often abundant. Pre-training the encoder component of a sequence-to-sequence model for a different task such as language modelling has been extremely successful, as manifested by the BERT model. We take a different approach because we not only can find data that matches the domain of interest but also can construct input-output mappings with high accuracy. Therefore, we pre-train the entire model with the same objective on a synthetic domain-matching data set. The fine-tuning phase follows the pre-training phase by simply switching the training data to the labelled data set.

4 Data

4.1 Camel Case URLs

We use two distinct sources for our camel case URL data set, an internal crawl and Common Crawl[2]. The internal data is comprised of approximately 21k domain names automatically segmented based on case (`DisneylandNews`→`Disneyland News`) and then manually corrected. Manually correcting the

[2]https://commoncrawl.org

data created interesting examples where the segmentation, given casing, is non-trivial (e.g. `Awardsand Honors` vs. `Awards and Honors`). The Common Crawl data was scraped from extracted plain text from web archives and consists of approximately 21k domain names with manual corrections done via crowd sourcing. The Common Crawl URL data set is publicly available on GitHub.[3]

4.2 Knowledge Graph Entity Names

The pre-training data is derived from entity names found in Google's Knowledge Graph[4]. The entity names are naturally space-separated and case-sensitive. Further splitting is also done on entity name segments that are all uppercase consonants since these are virtually always verbalized letter by letter (e.g. `CD Player` to `C D Player`).

4.3 Data Set Statistics

Table 1 lists the statistics of the data sets. The two data sets, namely the internal crawl and the Common Crawl, have approximately equal means of input and output lengths, even though their creation processes are different. On the other hand, the *pre-train* Knowledge Graph data set contains larger proportions of longer sequences. The *pre-train* set is four orders of magnitude larger than either of the two annotated data sets.

	Average Input Length	*Average No. of Segments*	*Total No. of Examples*
internal crawl *train*	12.96	2.47	17036
internal crawl *dev*	12.94	2.48	1893
internal crawl *test*	13.17	2.49	2104
Common Crawl *train*	12.63	2.65	17575
Common Crawl *dev*	12.77	2.66	1953
Common Crawl *test*	12.64	2.67	2170
Knowlege Graph *pre-train*	29.22	4.54	>200m

Table 1: Statistics for camel case URL data sets and Knowledge Graph pre-train data set.

5 Experiments

We choose the hyper-parameters on the internal crawl dev set for the network in Figure 1. Table 2 lists the key parameters. In Section 5.1 and Section 5.2, we develop training strategies using the internal crawl data set. In Section 5.3, we report the final results on both the internal and the Common Crawl test sets based on the hyper-parameters and the training strategy developed on the internal data set.

Input embedding size	256
Output embedding size	64
Number of forward encoder layers	2
Number of backward encoder layers	2
Number of decoder layers	1
Number of decoder GRU units	64
Number of encoder GRU units per layer	256
Beam size	2

Table 2: Network hyper-parameters.

We report results in terms of full sequence accuracy, where $Accuracy = \frac{\#CorrectSegmentations}{\#Sequences}$. For example, `photooftheday` has only one correct segmentation `photo of the day`. There is no

[3]https://github.com/google-research-datasets/common-crawl-domain-names

[4]https://developers.google.com/knowledge-graph

partial credit.

5.1 Lowercase Training

	Lowercase Accuracy
baseline	70.10%
pre-train	82.25%
+fine-tune	*88.80%*

Table 3: Lowercase results. Training and pre-training data sets are lower-cased. Results are on the development set which is also in lowercase.

The camel case training data has implicit word boundary annotations. In order to train a model to predict boundaries when case cues are not present, we need to hide the annotations by normalizing case. For this we simply lowercase both the training and evaluation sets. Table 3 summarizes the results of pure lowercase models. The baseline model uses no external data besides the *train* set.

The improved model is first trained only on the *pre-train* set and then fine-tuned on the *train* set. Pre-training alone is already better than the baseline by a large margin (12%), indicating the importance of learning from a large number of entity names. Fine-tuning yields a further improvement (6%).

5.2 Mixed Case Training

	Lowercase Accuracy	*Camel Case Accuracy*
baseline	69.36%	94.82%
pre-train	81.56%	92.18%
+fine-tune	*89.54%*	*96.67%*

Table 4: Mixed LowerCase/CamelCase results. Training and pre-training sets have equal proportions of lowercase and camel case examples. Results are reported on both lowercase and camel case development sets.

The lowercase model works well on lowercase input. But the accuracy of 88.8% is not very high for camel case input because the simple rule of splitting words based on case switching is 92.55% accurate. One could have a hybrid system in which the neural segmenter is invoked when the input has no case cues and the rule is invoked for camel case input, but we ought in principle to be able to train a model that handles both kinds of input well.

In Table 4, we show our final results of training on mixed lowercase and camel case data. For every camel case example, a lowercase example is added. This is done for both the training set and the pre-training set. Essentially, we assign equal weights to the two types of examples. There is a small degradation in lowercase accuracy for the baseline and the pre-train-only models. But after fine-tuning, not only is the loss recovered, but there is even a slight gain (89.54% versus 88.80%). This can be explained as an effect of transfer-learning if we view the lowercase examples and camel case examples as two different domains. As expected, the camel case accuracy is now close to perfect (96.67%), higher than the accuracy of the simple rule of splitting-by-case (92.55%).

5.3 Final Results

In this section, we report the final results on the *test* portion of the internal crawl and the Common Crawl data sets. The model uses the mixed case training strategy mentioned in Section 5.2. Learning rates are tuned on their *dev* counterparts. The top portion of Table 5 is for internal crawl. The bottom portion is for Common Crawl. The common trends are:

- Pre-training (without fine-tuning) improves over baseline by a large margin when input is lowercase.
- Fine-tuning further improves the results regardless of input casing.

		Lowercase Accuracy	*Camel Case Accuracy*
internal crawl	baseline	69.15%	94.87%
	pre-train	80.89%	91.49%
	+fine-tune	**88.12%**	**96.20%**
Common Crawl	baseline	63.64%	85.48%
	pre-train	75.81%	81.29%
	+fine-tune	**85.21%**	**91.15%**

Table 5: Testing results on the internal crawl and Common Crawl data sets.

5.4 Error Analysis

The pattern of improvement coming from pre-training is clear. Without pre-training, sometimes the model generates word-like segments. The pre-trained model is better at distinguishing real words and fakes ones. Table 6 shows some examples of success in the top region. What remains to be fixed are harder ones shown in the bottom region of Table 6.

	Reference	*Prediction*
Errors fixed *Prediction →Reference*	Mainspring Press	Mains pring Press
	N M animal Control	N Manimal Control
	artists ask art	artist saskart
	planet earth	plane tearth
	D C income	D Cincome
	Pubs history	Pubshistory
	mens weekly	men sweekly
	phoenix tennis	pho enix tennis
Errors uncorrected *Reference →Prediction*	Bulk S e o Tools	Bulk Seo Tools
	Library U s gen net	Library Usgennet
	just a film junkie	justa film junkie
	pet lvr	pet l v r
	ASAP Workouts	A S A P Workouts
	S e o article	Seo article
	iams company breeders	i a m s company breeders
	u s a p a store	u s a pastore

Table 6: Errors fixed and remaining uncorrected by pre-training.

Figure 2 shows how accuracy varies as the number of characters or segments contained in the input increases. The model is highly accurate on input with less than four segments. Due to more strict filtering in the internal crawl data set, there are very few examples with a single segment. There are many more single-segment examples in the Common Crawl data set. Single-segment examples are more challenging probably because they correspond to more rare words or terms. Prediction accuracy is not strongly correlated to input length as shown by the seemingly opposite trends on the two different data sets.

6 Conclusion

URL segmentation has applications in TTS and web search. Our contributions include a curated URL data set and a highly accurate RNN model boosted by pre-training on Knowledge Graph entities.

Acknowledgments

We thank Shankar Kumar, Corinna Cortes, and the reviewers for their comments and suggestions.

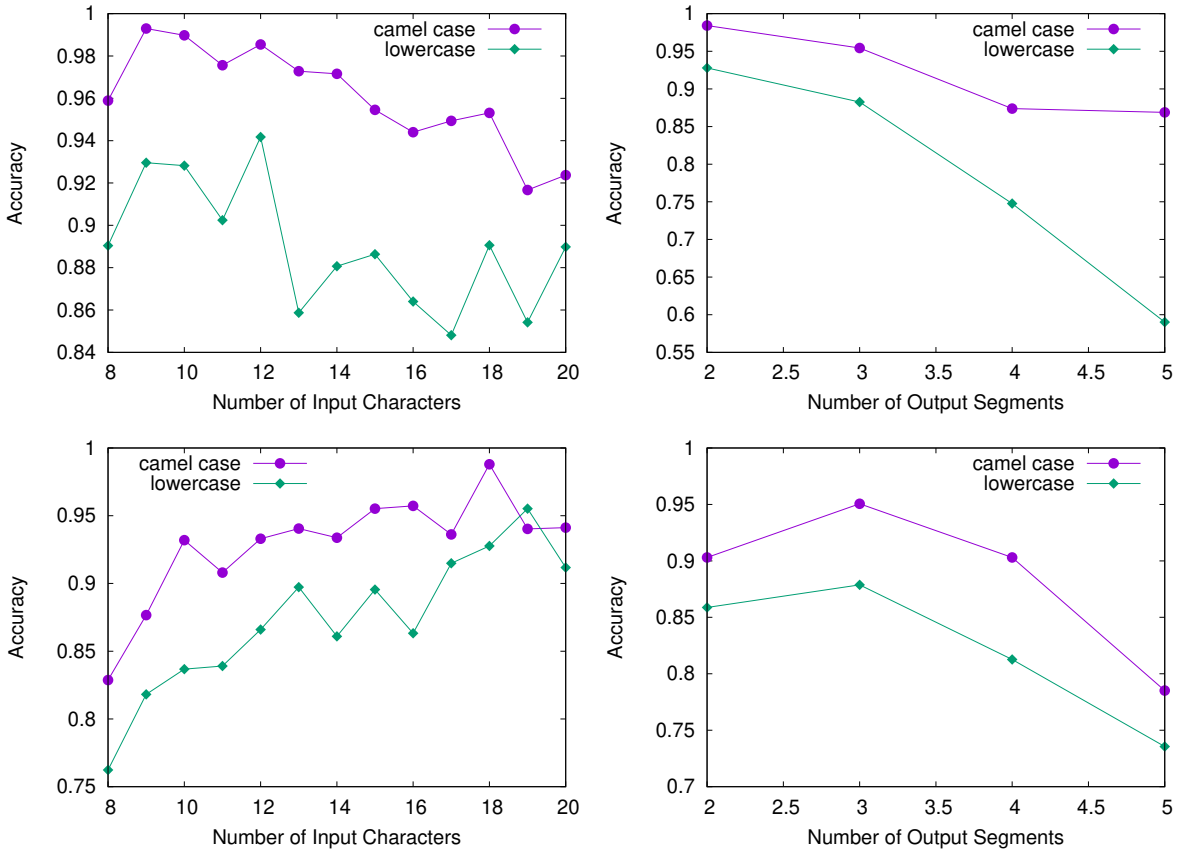

Figure 2: Accuracy as input/output length increases. The leftmost and rightmost lengths are bucketed. Top: internal crawl *dev*. Bottom: Common Crawl *dev*.

References

Roee Aharoni and Yoav Goldberg. 2017. Morphological inflection generation with hard monotonic attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2004–2015, Vancouver, Canada, July. Association for Computational Linguistics.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

David Chiang, Jonathan Graehl, Kevin Knight, Adam Pauls, and Sujith Ravi. 2010. Bayesian inference for finite-state transducers. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 447–455, Los Angeles, California, June. 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. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Peter Ebden and Richard Sproat. 2014. The Kestrel TTS text normalization system. *Natural Language Engineering*, 21:333–353.

Sharon Goldwater, Thomas Griffiths, and Mark Johnson. 2009. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112:21–54, 04.

Oliver Hellwig and Sebastian Nehrdich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763, Brussels, Belgium, October-November. Association for Computational Linguistics.

Weipeng Huang, Xingyi Cheng, Kunlong Chen, Taifeng Wang, and Wei Chu. 2019. Toward fast and accurate neural chinese word segmentation with multi-criteria learning. *CoRR*, abs/1903.04190.

Jianqiang Ma, Verena Henrich, and Erhard Hinrichs. 2016. Letter sequence labeling for compound splitting. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 76–81, Berlin, Germany, August. Association for Computational Linguistics.

Ji Ma, Kuzman Ganchev, and David Weiss. 2018. State-of-the-art Chinese word segmentation with bi-LSTMs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4902–4908, Brussels, Belgium, October-November. Association for Computational Linguistics.

Klaus Macherey, Andrew Dai, David Talbot, Ashok Popat, and Franz Och. 2011. Language-independent compound splitting with morphological operations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1395–1404, Portland, Oregon, USA, June. Association for Computational Linguistics.

D. C. Olivier. 1968. *Stochastic Grammars and Language Acquisition Mechanisms*. Ph.D. thesis, Harvard University, Cambridge, MA.

Fuchun Peng, Fangfang Feng, and Andrew McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 562–568, Geneva, Switzerland, aug 23–aug 27. COLING.

Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1756–1765, Vancouver, Canada, July. Association for Computational Linguistics.

Jose Sotelo, Soroush Mehri, Kundan Kumar, João Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. 2017. Char2wav: End-to-end speech synthesis. In *Proceedings of the 5th International Conference on Learning Representations, Workshop Track*, Toulon, France, April.

Richard Sproat, Alan Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of non-standard words. *Computer Speech and Language*, 15(3):287–333.

Sriram Srinivasan, Sourangshu Bhattacharya, and Rudrasis Chakraborty. 2012. Segmenting web-domains and hashtags using length specific models. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1113–1122, New York, NY, USA. ACM.

Kuansan Wang, Christopher Thrasher, and Bo-June Paul Hsu. 2011. Web scale nlp: A case study on url word breaking. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 357–366, New York, NY, USA. ACM.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. 2017. Tacotron: Towards end-to-end speech synthesis. *Computing Research Repository*, arXiv:1703.10135. version 2.

Yue Zhang and Stephen Clark. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 840–847, Prague, Czech Republic, June. Association for Computational Linguistics.