Building a Word Segmenter for Sanskrit Overnight

Vikas Reddy*[1], Amrith Krishna*[2], Vishnu Dutt Sharma[3], Prateek Gupta[4], Vineeth M R[5], Pawan Goyal[2]

[1]Dept. of Mining Engineering, [2]Dept. of Computer Science and Engineering, [4]Dept. of Mathematics,
[5]Dept. of Electrical Engineering, [1,2,4,5]IIT Kharagpur
[3]American Express India Pvt Ltd
{vikas.challaram, amrith}@iitkgp.ac.in, pawang@cse.iitkgp.ernet.in

Abstract

There is abundance of digitised texts available in Sanskrit. However, the word segmentation task in such texts are challenging due to the issue of *Sandhi*. In Sandhi, words in a sentence often fuse together to form a single chunk of text, where the word delimiter vanishes and sounds at the word boundaries undergo transformations, which is also reflected in the written text. Here, we propose an approach that uses a deep sequence to sequence (seq2seq) model that takes only the sandhied string as the input and predicts the unsandhied string. The state of the art models are linguistically involved and have external dependencies for the lexical and morphological analysis of the input. Our model can be trained "overnight" and be used for production. In spite of the knowledge lean approach, our system preforms better than the current state of the art by gaining a percentage increase of 16.79 % than the current state of the art.

Keywords: Word Segmentation, Sanskrit, Low-Resource Languages, Sequence to sequence, seq2seq, Deep Learning

1. Introduction

Sanskrit had profound influence as the knowledge preserving language for centuries in India. The tradition of learning and teaching Sanskrit, though limited, still exists in India. There have been tremendous advancements in digitisation of ancient manuscripts in Sanskrit in the last decade. Numerous initiatives such as the Digital Corpus of Sanskrit[1], GRETIL[2], The Sanskrit Library[3] and others from the Sanskrit Linguistic and Computational Linguistic community is a fine example of such efforts (Goyal et al., 2012; Krishna et al., 2017).

The digitisation efforts have made the Sanskrit manuscripts easily available in the public domain. However, the accessibility of such digitised manuscripts is still limited. Numerous technical challenges in indexing and retrieval of such resources in a digital repository arise due to the linguistic peculiarities posed by the language. Word Segmentation in Sanskrit is an important yet non-trivial prerequisite for facilitating efficient processing of Sanskrit texts. Sanskrit has been primarily communicated orally. Due to its oral tradition, the phonemes in Sanskrit undergo euphonic assimilation in spoken format. This gets reflected in writing as well and leads to the phenomena of *Sandhi* (Goyal and Huet, 2016). *Sandhi* leads to phonetic transformations at word boundaries of a written chunk, and the sounds at the end of a word join together to form a single chunk of character sequence. This not only makes the word boundaries indistinguishable, but transformations occur to the characters at the word boundaries. The transformations can be deletion, insertion or substitution of one or more sounds at the word ends. There are about 281 sandhi rules, each denoting a unique combination of phonetic transformations, documented in the grammatical tradition of Sanskrit. The proximity between two compatible sounds as per any one of the 281 rules is the sole criteria for sandhi. The *Sandhi* do not make any syntactic or semantic changes to the words involved. *Sandhi* is an optional operation relied solely on the discretion of the writer.

While the *Sandhi* formation is deterministic, the analysis of *Sandhi* is non-deterministic and leads to high level of ambiguity. For example, the chunk '*gardabhaṣcāśvaśca*' (the ass and the horse) has 625 possible phonetically and lexically valid splits (Hellwig, 2015). Now, the correct split relies on the semantic compatibility between the split words.

The word segmentation problem is a well studied problem across various languages where the segmentation is non-trivial. For languages such as Chinese and Japanese, where there is no explicit boundary markers between the words (Xue, 2003), numerous sequence labelling approaches have been proposed. In Sanskrit, it can be seen that the merging of word boundaries is the discretion of the writer. In this work, we propose a purely engineering based pipeline for segmentation of Sanskrit sentences. The word segmentation problem is a structured prediction problem and we propose a deep sequence to sequence (seq2seq) model to solve the task. We use an encoder-decoder framework where the *sandhied* (unsegmented) and the *unsandhied* (segmented) sequences are treated as the input at the encoder and the output at the decoder, respectively. We train the model so as to maximise the conditional probability of predicting the unsandhied sequence given its corresponding sandhied sequence (Cho et al., 2014). We propose a knowledge-lean data-centric approach for the segmentation task. Our approach will help to scale the segmentation process in comparison with the challenges posed by knowledge involved processes in the current systems (Krishna et al., 2017). We only use parallel segmented and unsegmented sentences during training. At run-time, we only require the input sentence. Our model can literally be trained overnight. The best performing model of ours takes less than 12 hours to

*The first two authors contributed equally

[1]http://kjc-sv013.kjc.uni-heidelberg.de/dcs/

[2]http://gretil.sub.uni-goettingen.de/

[3]http://sanskritlibrary.org/

train in a 'Titan X' 12 GB memory, 3584 GPU Cores system. Our title for the paper is inspired from the title for the work by Wang et al. (2015). As with the original paper, we want to emphasise on the ease with which our system can be used for training and at runtime, as it do not require any linguistically involved preprocessing. Such requirements often limit the scalability of a system and tediousness involved in the process limits the usability of a system.

Since Sanskrit is a resource scarce language, we use the *sentencepiece* (Schuster and Nakajima, 2012), an unsupervised text tokeniser to obtain a new vocabulary for a corpus, that maximises the likelihood of the language model so learnt. We propose a pipeline for finding the semantically most valid segmented word-forms for a given sentence. Our model uses multiple layers of LSTM cells with attention. Our model outperforms the current state of the art by 16.79 %.

2. Models for Word Segmentation in Sanskrit

A number of methods have been proposed for word segmentation in Sanskrit. Hellwig (2015) treats the problem as a character level RNN sequence labelling task. The author, in addition to reporting sandhi splits to upto 155 cases, additionally categorises the rules to 5 different types. Since, the results reported by the author are not at word-level, as is the standard with word segmentation systems in general, a direct comparison with the other systems is not meaningful. Mittal (2010) proposed a method based on Finite State Transducers by incorporating rules of *sandhi*. The system generates all possible splits and then provides a ranking of various splits, based on probabilistic ranking inferred from a dataset of 25000 split points. Using the same dataset, Natarajan and Charniak (2011) proposed a sandhi splitter for Sanskrit. The method is an extension of Bayesian word segmentation approach by Goldwater et al. (2006). Krishna et al. (2016) is currently the state of the art in Sanskrit word segmentation. The system treats the problem as an iterative query expansion problem. Using a shallow parser for Sanskrit (Goyal et al., 2012), an input sentence is first converted to a graph of possible candidates and desirable nodes are iteratively selected using Path Constrained Random Walks (Lao and Cohen, 2010).

To further catalyse the research in word segmentation for Sanskrit, Krishna et al. (2017) has released a dataset for the word segmentation task. The work releases a dataset of 119,000 sentences in Sanskrit along with the lexical and morphological analysis from a shallow parser. The work emphasises the need for not just predicting the inflected word form but also the prediction of the associated morphological information of the word. The additional information will be beneficial in further processing of Sanskrit texts, such as Dependency parsing or summarisation (Krishna et al., 2017).So far, no system successfully predicts the morphological information of the words in addition to the final word form. Though Krishna et al. (2016) has designed their system with this requirement in mind and outlined the possible extension of their system for the purpose, the system currently only predicts the final word-form.

3. Method

We use an encoder-decoder framework for tackling our segmentation problem, and propose a deep seq2seq model using LSTMs for our prediction task. Our model follows the architecture from Wu et al. (2016), originally proposed for neural machine translation. We consider the pair of *sandhied* and *unsandhied* sentences as source and target sentences, respectively. Following the insights from Sutskever et al. (2014), we reverse the sequence order at the input and we find that the reversal of the string leads to improvement in the results. We also use a deep architecture with 3 layers each at the encoder and decoder, as it is shown that deeper models perform better than shallow LSTM Models. We also experiment with models with and without attention and find that the model with attention leads to considerable improvement in performance of the system (Wu et al., 2016). Given the training set $\mathcal{S}$, our training objective is to maximise the log probability of the segmented sequences T where the unsegmented sequences S are given. The training objective is to maximise (Sutskever et al., 2014)

$$\frac{1}{|S|} \sum_{(T,S) \in \mathcal{S}} log\ p(T|S)$$

For a new sentence, we need to output a sequence T' with maximum likelihood for the given input (Sutskever et al., 2014).

$$T' = \underset{T}{\operatorname{argmax}}\ p(T|S)$$

LSTMs are used both at the encoder and decoder. We use softmax layer at the decoder and perform greedy decoding to obtain the final prediction. The outputs are then passed to the loss function which calculates the log-perplexity over the data samples in the batch. We then update the parameters via backpropagation and use Adam optimiser (Kingma and Ba, 2015) for our model.

Vocabulary Enhancement for the model - Sanskrit, being a resource poor language, the major challenge is to obtain enough data for the supervised task. While there are plenty of sandhied texts available for Sanskrit, it is hard to find parallel or unsandhied texts alone, as it is deterministic to get sandhied text from unsandhied texts.

In our case we use 105,000 parallel strings from the Digital Corpus of Sanskrit as released in Krishna et al. (2017). To handle the data sparsity, we adopt a purely engineering based approach for our model. Rather than relying on the real word boundaries, we use the '*sentencepiece*' model, an unsupervised text tokeniser (Schuster and Nakajima, 2012) to obtain a new vocabulary for the corpus. The method was originally proposed for segmentation problem in Japanese and Korean speech recognition systems. In the method, a greedy approach is used to identify new word units from a corpus that maximises the likelihood of the language model so learnt (Schuster and Nakajima, 2012).

Figure 1 shows the instance of words learnt from the *sentencepiece* model corresponding to the original input from the corpus. In the *sentencepiece* model, the 'space' in the original input is also treated as a character and is replaced

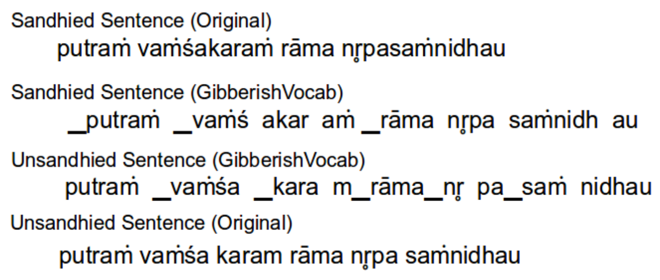

Sandhied Sentence (Original)
putraṁ vaṁśakaraṁ rāma nr̥pasaṁnidhau
Sandhied Sentence (GibberishVocab)
_putraṁ _vaṁś akar aṁ _rāma nr̥pa saṁnidh au
Unsandhied Sentence (GibberishVocab)
putraṁ _vaṁśa _kara m_rāma_nr̥ pa_saṁ nidhau
Unsandhied Sentence (Original)
putraṁ vaṁśa karam rāma nr̥pa saṁnidhau

Figure 1: Sandhied and unsandhied sentence expressed in original writing and with the new learnt vocabulary 'GibberishVocab'.

with the special symbol '_'. So 'aṃ_rāma' is a word in our vocabulary, which originally is part of two words.
Our model is fed only the 'words' from the new vocabulary, henceforth to be referred to as '*GibberishVocab*'. Note that the decoder also outputs words from *GibberishVocab*. The output from decoder is then converted to the original vocabulary for evaluating the outputs. This is trivially done by reclaiming the original 'space' as the delimiter for the old vocabulary.

4. Experiments

4.1. Dataset

We used a dataset of 107,000 sentences from the Sanskrit Word Segmentation Dataset (Krishna et al., 2017). The dataset is a subset of the Digital Corpus of Sanskrit. From the dataset we only use the input sentences and the ground truth inflected word-forms. We ignore all the other morphological and lemma information available in the dataset.

4.2. Baselines

We compare the performance of our system with two other baseline systems.
supervisedPCRW - This is the current state of the art for word segmentation in Sanskrit. The method treats the problem as an iterative query expansion task. This is a linguistically involved approach, as at first a lexicon driven shallow parser is used to obtain all the phonetically valid segments for a given sentence. The sentence is then converted into a graph with the segments as the nodes. The edges are formed between every pair of nodes which can co-exist in a sentence and are not competing for the same input position. The edge weights are formed by weighted sum of random walks across typed paths. The authors use typed paths to obtain extended contexts about the word pairs from the candidate pool. The typed paths are designed with human supervision which is also linguistically involved.
GraphCRF - We use a structured prediction approach using graph based Conditional Random Fields, where we first obtain the possible candidate segments using the shallow parser and then convert the segments into a graph. For every node segment, we learn a word vector using fastText (Bojanowski et al., 2016).
segSeq2Seq - This is the proposed model as described in Section 3. but without attention.
attnSegSeq2Seq - This is the proposed model as described in Section 3. with attention.

Model	Precision	Recall	F-Score
GraphCRF	65.20	66.50	65.84
SupervisedPCRW	76.30	79.47	77.85
segSeq2Seq	73.44	73.04	73.24
attnsegSeq2Seq	90.77	90.3	90.53

Table 1: Micro-averaged Precision, Recall and F-Score for the competing systems on the test dataset of 4200 strings.

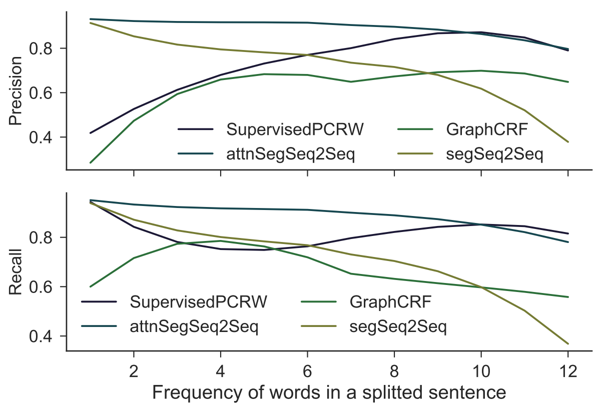
We report all our results on a test data of 4,200 sentences which was not used in any part of the training. From the dataset we ignore about 7,500 sentences which are neither part of training nor the test set. We used 214,000 strings both from input and output strings in the training data to obtain the *GibberishVocab* using *sentencepiece* model.
We use string-wise micro averaged precision, recall and F-Score to evaluate our model as is the standard with evaluating word segmentation models. We find that the default vocabulary size of 8,000 for the *GibberishVocab* works best. Of the 8,000 'words', the encoder vocabulary size is 7,944 and the decoder vocabulary size is 7,464. This shows the high overlap in the vocabulary in *GibberishVocab* at both input and output sides, in spite of the difference in phonetic transformations due to sandhi. Originally the training data contained 60,308 segmented words at the output side. By reducing the vocabulary size at decoder side to 7,464, we make the probability distribution (softmax) at the decoder layer denser. Even if we followed a linguistic approach there were 16,473 unique lemmas in the training dataset.

4.3. Training Procedure and Hyperparameters

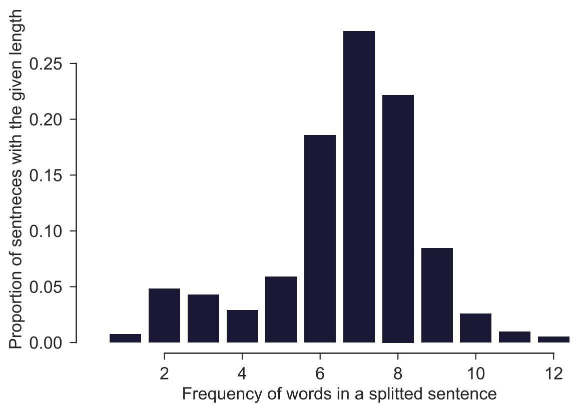
Our models have 3 layers at both the encoder and decoder. The models contain an embedding layer which is a trainable matrix with individual word vector having a size of 128. Our LSTM layers consist of 128 cells at both the encoder and decoder layers. We train the sentences in a batch size of 128 and keep the sequence length of each sequence to be 35. The initial learning rate was set at 0.001 and we trained our system for 80 epochs after which the network parameters converged. We used Adam optimiser with parameter values β_1,β_2 as 0.9 and 0.999, respectively. We use dropout in the hidden layers with different settings from 0.1 to 0.4 in step sizes of 0.1. We find that a dropout of 0.2 is the best performing configuration. Dropout helps to avoid over-fitting of data (Srivastava et al., 2014). Both the 'segSeq2Seq' and 'attnSegSeq2Seq' models follow the same architecture and have the same hyperparameter settings and vary only on the attention mechanism.

4.4. Results

Table 1 shows the performance of the competing systems. We can find that the system 'attnSegSeq2Seq' outperforms the current state of the art with a percent increase of 16.29 % in F-Score. The model 'segSeq2Seq' falls short of the current state of the art with a percent decrease of 6.29 % in F-Score. It needs to be noted that the systems 'attnSegSeq2Seq' and 'segSeq2Seq' are exactly the same architectures other than the addition of attention in the former. But there is a percentage increase of 23.61 % for



(a) Precision and Recall for the competing systems grouped based on the count of words in each of the test sentence.



(b) Distribution of strings in test dataset grouped based on the count of words in each sentence.

Figure 2: Results on the test dataset. The sentences are grouped based on the count of words in the segmented sentences

both the systems. One probable reason for this is due to the free word order nature of sentences in Sanskrit. Since there are multiple permutations of words in a sentence which are valid syntactically and convey the same semantic meaning, the entire input context is required to understand the meaning of a sentence for any distributional semantic model.

Figure 2 shows the results of the competing systems on strings of different lengths in terms of words in the sentence. This should not be confused with sequence length. Here, we mean the 'word' as per the original vocabulary and is common for all the competing systems. For all the strings with up to 10 words, our system 'attnSegSeq2Seq' consistently outperforms all the systems in terms of both precision and recall. The current state of the art performs slightly better than our system, for sentences with more than 10 words. It needs to be noted that the average length of a string in the Digital Corpus of Sanskrit is 6.7 (Krishna et al., 2016). The proportion of sentences with more than 10 words in our dataset is less than 1 %. The test dataset has slightly more than 4 % sentences with 10 or more words. The 'segSeq2Seq' model performs better than the state of the art for both Precision and Recall for strings with less than or equal to 6 words. Figure 2a shows the proportion of sentences in the test data based on the frequency of words in it. Figure 2b shows the proportion of strings in the test dataset based on the number of words in the strings. Our systems attnSegSeq2Seq takes overall 11 hours 40 minutes and for 80 epochs in a 'Titan X' 12GB GPU memory, 3584 GPU Cores, 62GB RAM and Intel Xeon CPU E5-2620 2.40GHz system. For segSeq2Seq it takes 7 hours for the same setting.

5. Discussion

The purpose of our proposed model is purely to identify the word splits and correctness of the inflected word forms from a sandhied string. The word-level indexing in retrieval systems is often affected by phonetic transformations in words due to *sandhi*. For example, the term '*parameśvaraḥ*' is split as '*parama*' (ultimate) and '*īśvaraḥ*' (god). Now, a search for instances of the word '*īśvaraḥ*' might lead to missing search results without proper indexing. String matching approaches often result in low precision results. Using a lexicon driven system might alleviate the said issues, but can lead to possible splits which are not semantically compatible. For *parameśvaraḥ*', it can be split as '*parama*' (ultimate), '*śva*' (dog) and '*raḥ*' (to facilitate). Though this is not semantically meaningful it is lexically valid. Such tools are put to use by some of the existing systems (Krishna et al., 2016; Mittal, 2010) to obtain additional morphological or syntactic information about the sentences. This limits the scalability of those systems, as they cannot handle out of vocabulary words. Scalability of such systems is further restricted as the sentences often need to undergo linguistically involved preprocessing steps that lead to human in the loop processing. The systems by Krishna et al. (2016) and Krishna et al. (2017) assume that the parser by Goyal et al. (2012), identifies all the possible candidate chunks.

Our proposed model is built with precisely one purpose in mind, which is to predict the final word-forms in a given sequence. Krishna et al. (2017) states that it is desirable to predict the morphological information of a word from along with the final word-form as the information will be helpful in further processing of Sanskrit. The segmentation task is seen as a means and not an end itself. Here, we overlook this aspect and see the segmentation task as an end in itself. So we achieve scalability at the cost of missing out on providing valuable linguistic information. Models that use linguistic resources are at an advantage here. Those systems such as Krishna et al. (2016) can be used to identify the morphological tags of the system as they currently store the morphological information of predicted candidates, but do not use them for evaluation as of now. Currently, no system exists that performs the prediction of wordform and morphological information jointly for Sanskrit. In our case, since we learn a new vocabulary altogether, the real word boundaries are opaque to the system. The decoder predicts from its own vocabulary. But predicting morphological information requires the knowledge of exact word boundaries. This should be seen as a multitask learning set up. One possible solution is to learn '*GibberishVocab*' on the set of words rather than sentences. But this leads to increased vocabulary at decoder which is not beneficial, given the scarcity of the data we have. Given the importance of morphological segmentation in morphologically rich languages such as Hebrew and Arabic (Seeker and Çetinoğlu, 2015), the same applies to the morphologically rich Sanskrit as well (Krishna et al., 2017). But, we leave this work for future.

6. Conclusion

In this work we presented a model for word segmentation in Sanskrit using a purely engineering based appraoch. Our model with attention outperforms the current state of the art (Krishna et al., 2016). Since, we tackle the problem with a non-linguistic approach, we hope to extend the work to other Indic languages as well where sandhi is prevalent such as Hindi, Marathi, Malayalam, Telugu etc. Since we find that the inclusion of attention is highly beneficial in improving the performance of the system, we intend to experiment with recent advances in the encoder-decoder architectures, such as Vaswani et al. (2017) and Gehring et al. (2017), where different novel approaches in using attention are experimented with. Our experiments in line with the measures reported in Krishna et al. (2016) show that our system performs robustly across strings of varying word size.

Code and Dataset

All our working code can be downloaded at `https://github.com/cvikasreddy/skt`. The dataset for training can be downloaded at `https://zenodo.org/record/803508#.WTuKbSa9UUs`

7. Bibliographical References

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October. Association for Computational Linguistics.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252.

Goldwater, S., Griffiths, T. L., and Johnson, M. (2006). Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 673–680. Association for Computational Linguistics.

Goyal, P. and Huet, G. (2016). Design and analysis of a lean interface for sanskrit corpus annotation. *Journal of Language Modelling*, 4(2):145–182.

Goyal, P., Huet, G. P., Kulkarni, A. P., Scharf, P. M., and Bunker, R. (2012). A distributed platform for sanskrit processing. In *COLING*, pages 1011–1028.

Hellwig, O. (2015). Using recurrent neural networks for joint compound splitting and sandhi resolution in sanskrit. In *4th Biennial Workshop on Less-Resourced Languages*.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *In 3rd International Conference on Learning Representations (ICLR) 2015*.

Krishna, A., Santra, B., Satuluri, P., Bandaru, S. P., Faldu, B., Singh, Y., and Goyal, P. (2016). Word segmentation in sanskrit using path constrained random walks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 494–504, Osaka, Japan, December. The COLING 2016 Organizing Committee.

Krishna, A., Satuluri, P. K., and Goyal, P. (2017). A dataset for sanskrit word segmentation. In *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 105–114, Vancouver, Canada, August. Association for Computational Linguistics.

Lao, N. and Cohen, W. W. (2010). Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 81(1):53–67.

Mittal, V. (2010). Automatic sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 85–90. Association for Computational Linguistics.

Natarajan, A. and Charniak, E. (2011). S3-statistical saṃdhi splitting. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, pages 301–308. Association for Computational Linguistics.

Schuster, M. and Nakajima, K. (2012). Japanese and korean voice search. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 5149–5152. IEEE.

Seeker, W. and Çetinoğlu, Ö. (2015). A graph-based lattice dependency parser for joint morphological segmentation and syntactic analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Wang, Y., Berant, J., and Liang, P. (2015). Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on*

Natural Language Processing (Volume 1: Long Papers), pages 1332–1342, Beijing, China, July. Association for Computational Linguistics.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xue, N. (2003). Chinese word segmentation as character tagging. *Computational Linguistics and Chinese Language Processing*, 8(1):29–48.