Neural compound-word (Sandhi) generation and splitting in Sanskrit language

Sushant Dave
IIT Delhi
sushant.dave@gmail.com

Dr. Prathosh A. P.
IIT Delhi
prathoshap@ee.iitd.ac.in

Arun Kumar Singh
IIT Delhi
mailtoarunkus@gmail.com

Prof. Brejesh Lall
IIT Delhi
prathoshap@ee.iitd.ac.in

Abstract

This paper describes neural network based approaches to the process of the formation and splitting of word-compounding, respectively known as the Sandhi and Vichchhed, in Sanskrit language. Sandhi is an important idea essential to morphological analysis of Sanskrit texts. Sandhi leads to word transformations at word boundaries. The rules of Sandhi formation are well defined but complex, sometimes optional and in some cases, require knowledge about the nature of the words being compounded. Sandhi split or Vichchhed is an even more difficult task given its non uniqueness and context dependence. In this work, we propose the route of formulating the problem as a sequence to sequence prediction task, using modern deep learning techniques. Being the first fully data driven technique, we demonstrate that our model has an accuracy better than the existing methods on multiple standard datasets, despite not using any additional lexical or morphological resources. The code is being made available at https://github.com/IITD-DataScience/Sandhi_Prakarana

1 Introduction

Sanskrit is one of the oldest of the Indo-Aryan languages. The oldest known Sanskrit texts are estimated to be dated around 1500 BCE. It is the one of the oldest surviving languages in the world. A large corpus of religious, philosophical, socio-political and scientific texts of multi cultural Indian Subcontinent are in Sanskrit. Sanskrit, in its multiple variants and dialects, was the Lingua Franca of ancient India (Coward, 1990). Therefore, Sanskrit texts are an important resource of knowledge about ancient India and its people. Earliest known Sanskrit documents are available in the form called *Vedic Sanskrit*. *Rigveda*, the oldest of the four Vedas, that are the principal religious texts of ancient India, is written in *Vedic Sanskrit*. In sometime around 5th century BCE, a Sanskrit scholar named *pARini* (Cardona, 1997) wrote a treatise on Sanskrit grammar named *azwADyAyI*, in which *pARini* formalized rules on linguistics, syntax and grammar for Sanskrit. *azwDyAyI*[1] is the oldest surviving text and the most comprehensive source of grammar on Sanskrit today. *azwADyAyI* literally means eight chapters and these eight chapters contain around 4000 sutras or rules in total. These rules completely define the Sanskrit language as it is known today. *azwADyAyI* is remarkable in its conciseness and contains highly systematic approach to grammar. Because of its well defined syntax and extensively well codified rules, many researchers have made attempts to codify the *pARini's* sutras as computer programs to analyze Sanskrit texts.

1.1 Introduction of Sandhi and Sandhi Split in Sanskrit

Sandhi refers to a phonetic transformation at word boundaries, where two words are combined to form a new word. Sandhi literally means 'placing together' (*samdhi-sam* together + *daDAti* to place) is the principle of sounds coming together naturally according to certain rules codified by the grammarian *pARini* in his *azwADyAyI*.

```
vidyA + AlayaH = vidyAlayaH (Vowel Sandhi)
```

[1] https://www.britannica.com/topic/Ashtadhyayi

```
vAk + hari = vAgGari (Consonant Sandhi)
punaH + api = punarapi (Visarga Sandhi)
```

Sandhi Split on the other hand, resolves Sanskrit compounds and "phonetically merged" (sandhified) words into its constituent morphemes. Sandhi Split comes with additional challenge of not only splitting of compound word correctly, but also predicting where to split. Since Sanskrit compound word can be split in multiple ways based on multiple split locations possible, split words may be syntactically correct but semantically may not be meaningful.

```
tadupAsanIyam = tat + upAsanIyam
tadupAsanIyam = tat + up + AsanIyam
```

1.2 Existing Work on Sandhi

The current resources available for doing Sandhi in open domain are not very accurate. Three most popular publicly available set of Sandhi tools are mentioned in table 1.

Tool Name	Description
JNU Sandhi Tools [2]	This application has been developed under the supervision of Dr. Girish Nath Jha from JawaharLal Nehru University. It facilitates Sandhi as well as Sandhi Split.
UoH Sandhi Tools [3]	These tools were developed at the Department of Sanskrit Studies, University of Hyderabad under the supervision of Prof. Amba Kulkarni
INRIA Tools [4]	Called as Sandhi Engine and developed under the guidance of Prof. Gerard Huet at INRIA.

Table 1: Sandhi Tools Summary

An analysis and description of these tools is present in the paper on Sandhikosh (Bhardwaj et al., 2018). The same paper introduced a dataset for Sandhi and Sandhi Split verification and compared the performance of the tools in table 1 on that dataset. Neural networks have been used for Sandhi Split by many researchers, for example (Aralikatte et al., 2018), (Hellwig and Nehrdich, 2018) and (Hellwig, 2015). The task of doing Sandhi has been mainly addressed as a rule based algorithm e.g. (Raja et al., 2014). There is no research on Sandhi using neural networks in public domain so far. This paper describes experiments with Sandhi operation using neural networks and compares results of suggested approach with the results achieved using existing Sandhi tools (Bhardwaj et al., 2018).

1.3 Existing Work on Sandhi Split

Many researchers like (Huet, 2005) and (Kulkarni and Shukl, 2009) have tried to codify *pARini's* rules for achieving Sandhi Split along with a lexical resource. (Natarajan and Charniak, 2011) proposed a statistical method based on Dirichlet process. Finite state methods have also been used (Mittal, 2010). A graph query method has been proposed by (Krishna et al., 2016).

Lately, Deep Learning based approaches are increasingly being tried for Sandhi Split. (Hellwig, 2015) used a one-layer bidirectional LSTM to two parallel character based representations of a string. (Reddy et al., 2018) and (Hellwig and Nehrdich, 2018) proposed deep learning models for Sandhi Split at sentence level. (Aralikatte et al., 2018) uses a double decoder model for compound word split. The method proposed in this paper describes an RNN based, two stage deep learning method for Sandhi Split of isolated compound words without using any lexical resource or sentence information.

[2] http://sanskrit.jnu.ac.in/sandhi/gen.jsp

[3] http://tdil-dc.in/san/sandhi/index_dit.html

[4] https://sanskrit.inria.fr/DICO/sandhi.fr.html

In addition to above, there exist multiple Sandhi Splitters in the open domain. The prominent ones being JNU Sandhi Splitter [5] , UoH Sandhio Splitter [6] and INRIA Sanskrit reader companion [7]

The paper (Aralikatte et al., 2018) compares the performance of above 3 tools with their results. This was an attempt to create benchmark in the area of Sanskrit Computational Linguistics.

2 Motivation

Analysis of sandhi is critical to analysis of Sanskrit text. Researchers have pointed out how a good Sandhi Split tool is necessary for a good coverage morphological analyzer (Akshar Bharati, 2006). Good Sandhi and Sandhi Split tools facilitate the work in below areas.

- Text to speech synthesis system
- Neural Machine Translation from non-Sanskrit to Sanskrit language and vice versa
- Sanskrit Language morphological analysis

Most Sandhi rules combine phoneme at the end of a word with phoneme at the beginning of another word to make one or two new phonemes. It is important to note that Sandhi rules are meant to facilitate pronunciation. The transformation affects the characters at word boundaries and the remaining characters are generally unaffected. The rules of Sandhi for all possible cases are laid out in *azwADyAyI* by *pARini* as 281 rules. The rules of Sandhi are complex and in some cases require knowledge of the words being combined, as some rules treat different categories of words differently. This means that performing Sandhi requires some lexical resources to indicate how the rules are to be implemented for given words. Work done in Sandhikosh (Bhardwaj et al., 2018) shows that currently available rule based implementations are not very accurate. This paper tries to address the problem of low accuracy of existing implementations using a machine learning approach.

3 Method

3.1 The Proposed Sandhi Method

Sandhi task is similar to language translation problem where a sequence of characters or words produces another sequence. RNNs are widely used to solve such problems. Sequence to sequence model introduced by (Sutskever et al., 2014) is especially suited to such problems, therefore same was used in this work. The training and test data were in ITRANS Devanagari format [8]. This data was converted to SLP1 [9] as SLP1 was found more suited for proposed approach. The code was implemented in python 3.5 with Keras API running on TensorFlow backend.

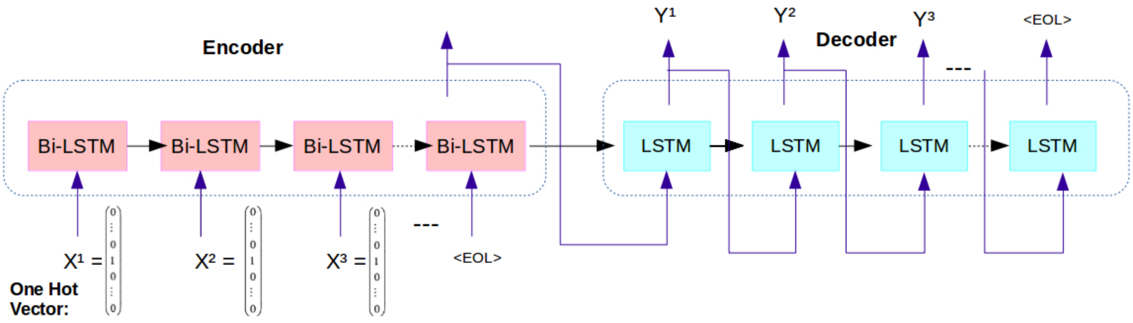

Figure 1: Model Architecture for Sandhi

[5] http://sanskrit.jnu.ac.in/sandhi/viccheda.jsp
[6] http://sanskrit.uohyd.ac.in/scl/
[7] https://sanskrit.inria.fr/DICO/sandhi.fr.html
[8] https://www.aczoom.com/itrans/
[9] https://en.wikipedia.org/wiki/SLP1

Proposed model expects 2 words as input and outputs a new word as per the Sandhi rules of *azwADyAyI* as given in below example.

```
sAmAnyaDvaMsAn + aNgIkArAt => sAmAnyaDvaMsAnaNgIkArAt
```

Results achieved from this approach are rather poor. Analysis showed that this is mainly due to the excessive length of words in some cases. Proposed approach tries to address this problem by taking last n characters of the first input word and first m characters of the second input word to do Sandhi between the 2 smaller new words. Resulting compound word is appended with the characters which are omitted from first and second input word at the beginning and end of the compound word, respectively. This approach works well for long words but suffers from the problem of losing information after truncation of words. Some Sandhi rules are specific to word category and truncated words lose the category information. Proposed method does not use any external lexical resource and learns to incorporate the word category related rules if possible and therefore words should be truncated without losing too much information.

It was found that best results were achieved with $n = 5$ and $m = 2$ for Seq2seq model used in the paper. Below example explains the truncation approach as mentioned above:

```
sAmAnyaD   vaMsAn + aN  gIkArAt
         => sAmAnyaD  + "vaMsAn + aN" + gIkArAt
         => sAmAnyaD  + vaMsAnaN + gIkArAt
         => sAmAnyaDvaMsAnaNgIkArAt
```

Input sequence is set as the two input words concatenated with a '+' character between the 2 words. Output sequence is the sandhified (compound) single word. Characters '&' and '$' are used as start and end markers respectively in the output sequence. A single dictionary is created for all the characters in input and output and a unique one hot vector is assigned to each token in the dictionary. The input and output character sequences are then replaced by their one hot encoded vector sequences. The best results are achieved with LSTM (Hochreiter and Schmidhuber, 1997) as basic RNN cell for decoder and bidirectional LSTM as basic RNN cell for encoder. Both the encoder and decoder use the hidden unit size as 16. The training vectors were divided in batches of 64 vectors and total of 100 epochs were run to get the final results.

3.2 Sandhi Split Method

Conceptually, the Sandhi model architecture explained in Section 3.1 can be used for Sandhi Split as well, where the input and output are swapped with compound word as input and the two initial words concatenated with '+' character as output. However the accuracy achieved with this approach is very poor due to the similar reason observed while doing Sandhi with full word length i.e. Excessive length of words which makes training difficult. But the solution used for Sandhi that employed the method of truncation of words to train the model, is not feasible in Sandhi Split due to the multiple possibilities of split point in the compound word.

Other researchers have tried to solve this problem in two stages i.e. predicting the split point and then splitting the sandhified/compound word. (Aralikatte et al., 2018) achieved significantly good results by suggesting a double-decoder model which operates in two stages as mentioned above. An empirical analysis of the sandhi dataset showed that the following 2 observations holds for almost all the sandhified words:

- No more than *last 2 characters of first word* and *first 2 characters of second word* participate in sandhi process i.e undergo a change post sandhi.
- The *last 2 characters of first word* and *first 2 characters of second word* combine to produce *no more than 5 and no less than 2 characters.*

Exceptions to above rules were found to be mostly errors and thus helped clean the dataset. Above 2 rules leads to the conclusion that the portion of compound word which needs to be analyzed for change post Sandhi should be no more than 5 characters in length. This sequence of 5 characters hereafter referred to as sandhi-window becomes the target of Sandhi Split. Hence, characters before sandhi-window should belong to first word and characters after sandhi-window should belong to the second word. Applying this reasoning, the method described in this paper breaks the problem of Sandhi Split in 2 stages. In Stage 1, sandhi-window is predicted using a RNN model. In Stage 2, sandhi-window predicted in stage 1 is split into 2 different words using a seq2seq model similar to the one used in sandhi described in section 3.1.

Sandhi Split - Stage 1: The task of Stage 1 is to predict the sandhi-window. The input sequence is the compound word and the target output is an integer array with same number of elements as the characters in compound word. All elements in this array are 0 except the elements corresponding to the sandhi-window characters which are set to 1. For example in Fig 2, the sandhi-window is considered from 13th character to 17th character, both inclusive.

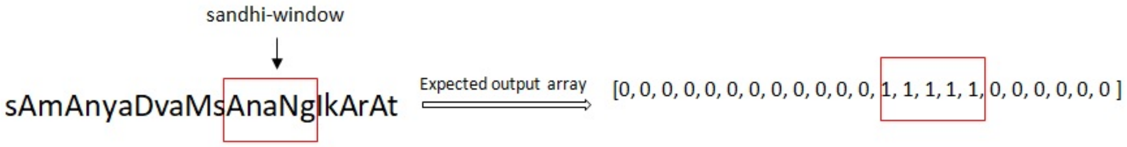

Figure 2: sandhi-window as prediction target in compound word

For an input compound word, the RNN model trained thus is expected to produce an array that has the size equal to input compound word length. All elements in this array must be zero except the elements at sandhi-window location, which must be equal to 1. For this decoder, a RNN was used with bidirectional LSTM as basic RNN cell. The output vector at each time step is connected to a dense layer with output array of unit length. This single output value is the output array element corresponding to input character. One-hot encoded vectors of input character sequences were used. Hidden unit size of Bi-LSTM cell was chosen as 64. The training vectors were divide into batch size of 64. The training was done for 40 epochs. Model was trained with RMSProp optimizer and mean squared error loss.

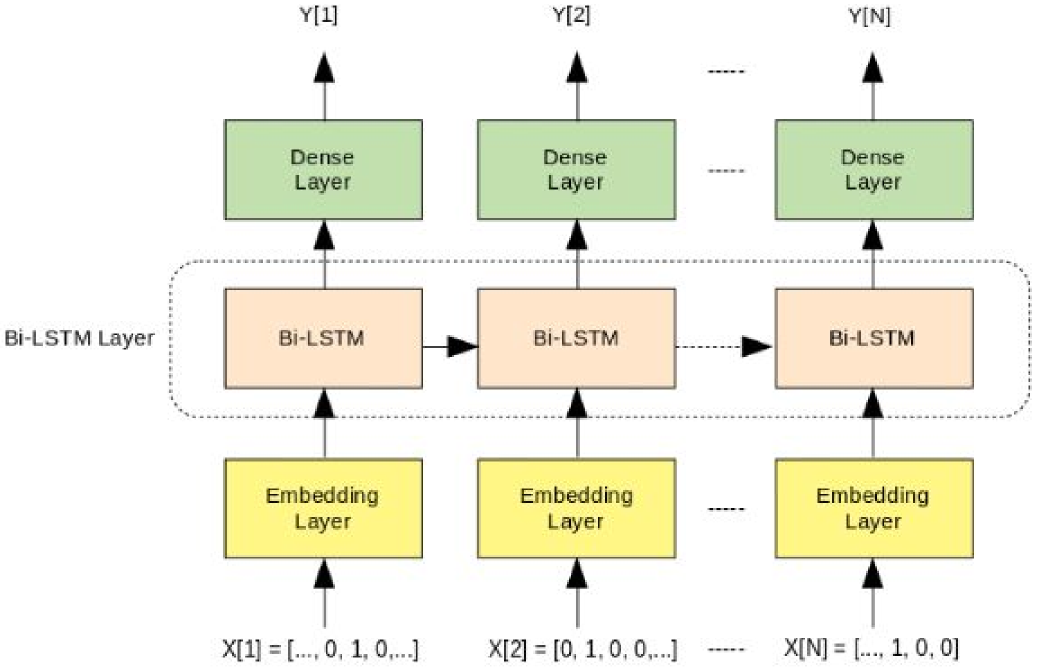

Figure 3: Model Architecture for Sandhi Split - Stage 1

Sandhi Split - Stage 2: The task of Stage 2 is to split the sandhi-window. The model for Stage 2 uses the same architecture as the one used for Sandhi (see section 3.1). Output sequence is set as truncated words T_{W_1} and T_{W_2} (see section 4.2 for detail) concatenated with a '+' character between the 2 words. Input sequence was the sandhi-window of the compound word. Characters '&' and '$' were used as start and end markers respectively in the output sequence. A single dictionary was created for all the characters in input and output and a unique one hot vector was assigned to each token in the dictionary.The input and output character sequences were then replaced by their one hot encoded vector sequences.The best results were achieved with LSTM (Hochreiter and Schmidhuber, 1997) as basic RNN cell for decoder and Bi-LSTM as basic RNN cell for encoder. Both the encoder and decoder used the hidden unit size as 128. Batch size for training was set as 64 vectors and total 30 epochs were run to get the final results. Model was trained with RMSProp optimizer and categorical cross-entropy error loss. The detailed steps for Sandhi Split method are as follows:

- Stage 1 - Predict the sandhi-window in a compound word that is to be split, using a RNN model.
- Stage 2 - The sandhi-window is then split in 2 words using a seq2seq model. Lets call the first split of sandhi-window as P_{S_1} and second split of sandhi-window as P_{S_2}.
- Post-Processing step - The characters N_{W_1} before the sandhi-window are the preceding part of first predicted split P_{W_1} of compound word. The characters N_{W_2} after the sandhi-window are the succeeding part of second predicted split P_{W_2} of compound word. P_{W_1} is obtained by concatenating N_{W_1} and P_{S_1} as preceding and succeeding words respectively. P_{W_2} is obtained by concatenating P_{S_2} and N_{W_2} as preceding and succeeding words respectively.

4 Data and Evaluation Results

4.1 Sandhi

The data for training a neural network model was taken from UoH corpus created at the University of Hyderabad [10]. This dataset has more than 100,000 Sandhi examples. There are some errors in this dataset, some of which were removed using manually created check rules. This dataset has examples from all the 3 types of Sandhi. For current implementation, only those examples were chosen in which 2 words combine to give 1 compound word. The cases where the words can't be combined or more than 2 words combine to produce 1 or more words were discarded. Analysis showed that most Sandhi examples in our dataset followed the relationship given below.

$$-2 <= N_c - (N_{w_1} + N_{w_2}) <= 1$$

Where N_c, N_{w_1} and N_{w_2} are the number of SLP1 characters in compound word, first input word and second input word respectively. Of all the cases which violated this rule, most were obvious errors in dataset and the remaining cases were too few in number and discarded as outliers. It is to be noted that the above equation is consistent with the second of the two empirical rules introduced in section 3.2. Using this rule, total examples left in our dataset were 81029. 20% examples out of these i.e. 16206 were separated as test set. Out of the remaining 65124 examples, 80% examples (51858) were used for training and 20% examples (12965) were used for model validation. Evaluation is based on exact matching of whole compound word. Even if the model does Sandhi over the word boundaries correctly but makes an error in a character before or after, it is considered a failure.

Results from method described above were compared with results from other publicly available tools as provided in Sandhikosha (Bhardwaj et al., 2018). Sandhikosha divides its data in 4 main sources: Ashtadhyayi, Bhagavad Gita, UoH corpus and other literature. In case of UoH corpus, test-set was selected which comes from UoH corpus and which is not used in training the seq2seq model described in this paper. For the the other 3 sets, only those test cases were chosen which have 2 input words and produce 1 output word just like it was done for training set. Since test cases used in this paper and

[10] `http://sanskrit.uohyd.ac.in/Corpus/`

Sandhikosha test cases are not exactly same, the results below are indicative in nature, but they do point to a clear trend.

The comparison is shown in the table 2. Every cell in the table indicates successful test cases, overall test cases and success percentage.

Corpus	JNU	UoH	INRIA	Proposed Method
Literature	53/150 (14.8%)	130/150 (86.7%)	128/150 (85.3%)	109/115 (94.78%)
Bhagavad-Gita	338/1430 (23.64%)	1045/1430 (73.1%)	1184/1430 (82.1%)	575/753 (76.36%)
UoH	3506/9368 (37.4%)	7480/9368 (79.8%)	7655/9368 (81.7%)	14734/16206 (90.92%)
Ashtadhyayi	455/2700 (16.9%)	1752/2700 (64.9%)	1762/2700 (65.2%)	1070/1574 (68.0%)

Table 2: Benchmark Results for Sandhi

As can be seen in the table 2, proposed method outperforms the existing methods of doing Sandhi in every case except the INRIA Sandhi tool in case of *Bhagavada Gita* word-set and it does so without using any additional lexical resource.

4.2 Sandhi Split

The UoH corpus was used for Sandhi Split task. The same dataset was used in sandhi task also (refer section 4.1). Similar to approach taken for Sandhi data preparation, this dataset was converted from Devanagari format to SLP1 format. Only those examples were selected for benchmark where two words combine to produce one word. Examples which violated the two rules mentioned in section 3.2 were discarded. Of the total 77842 remaining examples, 20% examples (15569) were used for testing and 80% examples (62273) were used for model training. The steps for dataset preparation are as follows:

1. Take an example from dataset consisting of a compound word C_W, first word W_1 and second word W_2 where W_1 and W_2 are the words which combine to produce C_W
2. Mark the sandhi-window S_W in C_W
3. Let n_1 and n_2 be the number of characters in C_W before and after S_W respectively
4. Remove the first n_1 characters from W_1. Call the resulting word T_{W_1}
5. Remove the last n_2 characters from W_2. Call the resulting word T_{W_2}
6. The compound word C_W and the location of sandhi-window pair is the data example for Stage 1
7. The sandhi-window S_W, T_{W_1} and T_{W_2} tuple makes data example for Stage 2
8. Repeat the above step for all the example selected from UoH Corpus for Sandhi-split

The results thus obtained for the 15569 test examples were used for benchmark. Criteria of correct prediction was considered based on exact word match between actual split words and predicted split words. We compared our test results with seq2seq paper (Aralikatte et al., 2018) in table 3 on dataset taken from same source. The number of examples in full dataset is also sufficiently close (77842 for us vs. 71747 for (Aralikatte et al., 2018)) All the rows in the table below are taken from ((Aralikatte et al., 2018), Table 1) except the last row which contains the result of approach described in this paper. To evaluate the results for JNU, UoH and INRIA tool, Sandhi Split ground truth was matched with top 10 results returned by these tools and if match found, it was considered a success.

Model	Location Prediction Accuracy	Split Prediction Accuracy
JNU	-	8.1%
UoH	-	47.2%
INRIA	-	59.9%
DD-RNN	95.0%	79.5%
Proposed Method	92.3%	86.8%

Table 3: Benchmark Results for Sandhi Split

It is evident form results that proposed method improves upon the existing state of the Art methods by a decent margin. In addition, proposed models are much simpler and do not require attention mechanism thereby reducing model complexity as well as training and inference time.

5 Conclusion

In this research work, we propose novel algorithms for Sandhi word formation and Sandhi Split that can be trained without use of any external resources such as language models, morphological or phonetic analyzers, and still manage to match or outperform existing approaches. Due to the simplicity of the models, these are computationally inexpensive to train and execute. In future we intend to extend current work to internal Sandhi and internal Sandhi-split using machine learning methods. Proposed models can be further refined by using additional training data as well as investigating techniques to reduce the errors in current training data.

References

V Sheeba Akshar Bharati, Amba Kulkarni. 2006. Building a wide coverage sanskrit morphological analyzer : A practical approach. In *TheFirst National Symposium on Modelling and Shal-low Parsing of Indian Languages, IIT-Bombay*.

Rahul Aralikatte, Neelamadhav Gantayat, Naveen Panwar, Anush Sankaran, and Senthil Mani. 2018. Sanskrit sandhi splitting using seq2(seq)2. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4909–4914, Brussels, Belgium, October-November. Association for Computational Linguistics.

Shubham Bhardwaj, Neelamadhav Gantayat, Nikhil Chaturvedi, Rahul Garg, and Sumeet Agarwal. 2018. SandhiKosh: A benchmark corpus for evaluating Sanskrit sandhi tools. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

George Cardona. 1997. *Panini: A Survey of Research*. Motilal Banarsidass.

Harold G. Coward. 1990. Note: Sanskrit was both a literary and spoken language in ancient india. *The Philosophy of the Grammarians, in Encyclopedia of Indian Philosophies*, 5:3–12, 36–47, 111–112.

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.

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Gérard Huet. 2005. A functional toolkit for morphological and phonological processing, application to a sanskrit tagger. *J. Funct. Program.*, 15:573–614, 07.

Amrith Krishna, Bishal Santra, Pavankumar Satuluri, Sasi Prasanth Bandaru, Bhumi Faldu, Yajuvendra Singh, and Pawan Goyal. 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.

Amba Kulkarni and Devanand Shukl. 2009. Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70:169–177, 01.

Vipul Mittal. 2010. Automatic Sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 85–90, Uppsala, Sweden, July. Association for Computational Linguistics.

Abhiram Natarajan and Eugene Charniak. 2011. s^3 - statistical sandhi splitting. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 301–308, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

S. V. Kasmir Raja, V. Rajitha, and Meenakshi Lakshmanan. 2014. A binary schema and computational algorithms to process vowel-based euphonic conjunctions for word searches. *ArXiv*, abs/1409.4354.

Vikas Reddy, Amrith Krishna, Vishnu Sharma, Prateek Gupta, Vineeth M R, and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.