Using Recurrent Neural Networks for joint compound splitting and Sandhi resolution in Sanskrit

Oliver Hellwig

University of Düsseldorf, Düsseldorf, Germany

Abstract

The paper describes a novel approach that performs joint splitting of compounds and of Sandhis in Sanskrit texts. Sanskrit is a strongly compounding, morphologically and phonetically complex Indo-Aryan language. The interacting levels of its linguistic processes complicate the computer based analysis of its corpus. The paper proposes an algorithm that is able to resolve Sanskrit compounds and "phonetically merged" (sandhied) words using only gold transcripts of correct string splits, but no further lexical or morphological resources. In this way, it may also prove to be useful for other Indo-Aryan languages, for which no or only limited digital resources are available.

1. Introduction

Sanskrit, an Old Indo-Aryan language, whose first texts date back to around 1.500 BCE, has produced one of the most voluminous text corpora in the world. There are two principal layers of Sanskrit. The earlier Vedic layer, which has been created between 1.500 and the middle of the first millenium BCE, may have preserved a spoken form of Sanskrit, at least in its oldest strata (Witzel, 1995). Classical Sanskrit, which is the topic of this paper, is a literary language that is largely regulated by the famous grammar of Pāṇini. Its literary production extends from the end of the Vedic period until the present day, its current use being confined to literary and scientific circles.

Although Sanskrit texts are the primary sources for understanding the cultural and political history of premodern India, there exist comparatively few tools and resources that provide easy access to its text corpus and its underlying structures. The paper aims at contributing a method that facilitates the automatic analysis of digital Sanskrit corpora, especially for researchers from the Humanities, and that can be easily applied to the large, though widely unexplored corpora of Middle and early New Indo-Aryan languages (Prākrits, Old Hindi and Marathi, the language of Nepalese royal edicts, etc.), which share some basic linguistic peculiarities with Sanskrit.

Sanskrit shows a number of linguistic phenomena that complicate its automatic analysis, including a very voluminous vocabulary, which was assembled and expanded over a period of more than 3.000 years, a rich morphology, a weakly regulated word order, and an extremely liberal orthography that permits considerable variations in spelling and word separation. Most demanding from a computational perspective, however, are the euphonical rules called Sandhi ("connection"), which were first formalized by Pāṇini, and the very productive compound formation.

1.1. Sandhi

Most Sandhi rules combine two adjacent phonemes into one or two other phonemes to facilitate the pronounciation of a string. They are applied while deriving an inflected form from its root (inner-word Sandhi), and while combining multiple inflected words into the final sentence (inter-word Sandhi). As the presented algorithm aims at splitting sentences and compounds into un-sandhied word forms, it deals only with the second type of Sandhi rules.

As an example for inter-word Sandhi, consider the string *gardabhaścāśvaśca* ("the ass and the horse"), which contains three Sandhis. The phonetic sequences *śc* are created by combining word final *ḥ* with word initial *c*. The long *ā* is the combination of a word final short *a* and a word initial short *a*. Sandhi can be resolved as follows:

gardabhaścāśvaśca
gardabha(ḥ-c)(a-a)śva(ḥ-c)a
gardabhaḥ ca aśvaḥ ca
"the ass and the horse and"

It is important to keep in mind that the application of Sandhi rules is strictly deterministic, i.e. there is only one acceptable output for a given combination of phonemes, and that these rules must be applied.[1] Analysis of Sandhis, however, does not need to be deterministic. The long *ā* in the sample string may have been derived from one of the four combinations (*a*-*a*), (*a*-*ā*), (*ā*-*a*), or (*ā*-*ā*) (Sandhi and compound split in all cases), it may be the terminating vowel of a feminine noun on *ā* (no Sandhi, but a compound split), or it may belong to the stem of a lexeme (neither Sandhi nor compounding). Which of these six solutions should be chosen, depends on the lexical and semantic context. The short string *hāhākārāḥ* ("the exclamations 'hāhā'"), for example, produces $5^4 = 625$ possible splits using only this set of Sandhi rules, and a substantial number of these splits can be resolved into morphologically and lexically valid substrings.

It should be apparent that learning and applying Sandhi rules cannot proceed mechanically, but requires lexical and semantic context information. So, the rule (*t*-*j*) → (*j*-*j*) is used correctly for *tajjalam* (*ta(t-j)alam*, "this water"), because the resulting split is lexically and semantically meaningful. The string *kajjalam*, however, should not be split by this rule, because the resulting solution *ka(t-j)alam* (*"how many water?") makes sense from a purely lexical, but not from a semantic perspective. Instead, the analyzer should leave this string unchanged (lexeme *kajjala*, "lampblack", nom./acc./voc. sg. neutre).

[1] Epic texts, for instance, frequently do not adhere strictly and consistently to these rules (Oberlies, 2003).

observed	u	t	t	a	m	ā	dh	a	m	a	m	a	dh	y	ā	n	ā	ṃ
target	u	t	t	a	m	**a-a**	dh	a	m	a	m	**a-**	dh	y	ā	n	ā	**m**

Table 1: Desired output for the string *uttamādhamamadhyānāṃ*. Bold letters mark differences between observed and target sequences. Output units containing a hyphen (-) indicate that the string should be split at this position; output of the form x-y indicates Sandhi rules to be learned.

1.2. Compound formation

Sanskrit grammar distinguishes three main types of compounds. *dvandva*s ("pairs") are n-ary compounds listing a set of coordinate members (*aśvāvyuṣṭrāḥ* = *aśva-avi-uṣṭrāḥ*, "horses, sheep, and camels"). *tatpuruṣa*s ("his man") indicate a relation between the governing first and the subordinate second member. *bahuvrīhi*s ("(one who has) much rice") describe the possessed argument in a possessive relation. This compound type may refer to another noun that denotes the possessor. While *dvandva*s and *tatpuruṣa*s grammatically remain nouns, *bahuvrīhi*s inflect like the external possessing argument and can, therefore, be interpreted as adjectives. All Sandhi rules that are operational when combining two independent strings are also applied during compound formation. Because compound formation is recursive, any compound can be composed with another word or compound into a new, more complex compound; as, for instance, in $((aśvāvyuṣṭra)_{dvandva}\text{-}darśanam)_{tatpuruṣa}$ ("visual perception of horses, sheep, and camels").

Apart from the fact that the number of possible Sandhi and compound splits most often increases exponentially with the string length, any decompounding algorithm will also have to deal with lexicalized compounds recorded in the dictionary. The string *mahāratnāni*, for instance, should be split as *mahā-ratnāni* ("big jewels") in most contexts. Gemmological texts, however, know *mahāratna* as a technical term for a class of precious jewels, so the string should not be split in this domain.

1.3. Contribution

The overall aim of the learning algorithm is to (1) split compounds at the correct positions, (2) resolve Sandhi, if it has occurred, and (3) produce the Sandhi rules according to the definitions in Section 3. that were operative when forming the current string. As an example, Table 1 shows the observed input sequence and the desired output for the string *uttamādhamamadhyānāṃ* ("of the highest, middle, and lowest", gen. pl. masc./fem./neuter).[2] The correct decomposed and unsandhied form of this string is *uttama-adhama-madhyānām*, and the classifier should learn to produce this result.

The rest of the paper is organized as follows. Section 2. gives a short overview of current NLP related methods for processing Classical Sanskrit. Section 3. describes which data are used and how they are encoded for the learning task. Section 4. describes the neural network used for compound and Sandhi splitting. Results for different learning scenarios are reported in Section 5. Section 6. summarizes the paper.

[2]Aspirates such as *dh* and diphthongs (*ai*, *au*) are single phonemes in Sanskrit and are, thus, interpreted as one phonetic unit.

2. Related research

A formal description of Sanskrit was first undertaken by the grammarian Pāṇini, who probably lived around 350 BCE in Northwestern India (Cardona, 1976). His grammar Aṣṭādhyāyī ("eight [*aṣṭan*] chapters [*adhyāya*]") describes the late Vedic level of Sanskrit by applying concepts such as thematic roles, rewrite rules, abstract derivation levels, and phonemes (Kiparsky, 2009). Pāṇini's seminal work was continued and refined during the following millenia in works such as the Mahābhāṣya (150 BCE) or the Siddhāntakaumudī (16. c. CE) (Scharfe, 1977).

Many modern approaches to the NLP of Sanskrit try to make use of the Pāṇinian system. Mishra reformulates the rules of the Aṣṭādhyāyī using ideas from set theory to build a generator for valid Sanskrit forms (Mishra, 2009). Huet (Huet, 2005) and Kulkarni (Kulkarni and Shukla, 2009) combine formal methods from the Aṣṭādhyāyī with a statistical scorer for the analysis of Sanskrit. Mittal reports 92.8% split accuracy in Sandhi resolution by combining an FSA trained on a parallel corpus of sandhied and unsandhied texts with lexical frequencies and a morphological analyzer (Mittal, 2010). Hellwig applies a Sandhi rule base, a morphological analyzer, and a language model estimated from a corpus to generate lexical and morphological analyses of unrestricted Sanskrit text (Hellwig, 2009; Hellwig, 2015).

This paper interprets Sandhi and compound resolution as a sequence labeling task. Sequence labeling is an important topic in the machine learning community, and algorithms proposed for this task include, among others, Hidden Markov Models (HMM), Conditional Random Fields (CRF) (Lafferty et al., 2001), and Recurrent Neural Networks (RNN, e.g., simple Elman networks (Elman, 1990)). While the context of HMMs and CRFs is restricted to few (mostly not more than 3) positions to the left of the input symbol, RNNs are in theory able to capture much larger ranges. This property makes them natural candidates for the present task, because Section 1. has shown that the correct resolution of compounds and Sandhis depends on the lexical and semantic neighbourhood of a phoneme, which extends further than a few characters in most cases. Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) neural cells have been shown to be numerically more stable than "normal" neural network cells used in RNNs, and they have demonstrated their ability to understand context-sensitive languages (Gers and Schmidhuber, 2001). Recent research in Computational Linguistics strongly relies on recurrent and deep neural architectures to learn, for instance, the compositional meaning of German compound phrases (Dima and Hinrichs, 2015) or to derive word and morpheme embeddings in the same training process (Qiu et al., 2014). Complex neural architectures are also applied to learn features located on the sentence level from character sequences (Santos and Zadrozny, 2014).

3. Data

The training data are extracted from the corpus of SanskritTagger (Hellwig, 2009). Because Sandhi information is not stored permanently in the database of this corpus, each sentence in the corpus is re-analyzed, and the Sandhi information is extracted from the analysis that matches the gold analysis of the respective sentence stored in the database. Next, each string is split into its phonemes p (observed sequence), and each phoneme is associated with the desired type of transformation rule (target sequence). The full data set consists of 2.591.000 phoneme sequences.

For this paper, the Pāṇinian prescriptions for inter-word Sandhi are reduced to five rule types R_{1-5}. This distinction is based on two criteria: (1) Does the phoneme change from the observed to the target sequence? (2) Is a word or compound split inserted into the target sequence? In addition, the paper distinguishes between two classes of Sandhi rules, which do not coincide with the Pāṇinian classification of Sandhi rules (Wackernagel, 1978, I, 301ff.). If the result of applying a Sandhi rule is a single vowel, this Sandhi type is called vocalic Sandhi in this paper. All other Sandhi types are called non-vocalic Sandhis.

1. $p \rightarrow p$ (R_1): Leave the observed phoneme unchanged; example: *varāhaḥ* ("boar"): $\bar{a} \rightarrow \bar{a}$, result: *varāhaḥ*.

2. $p \rightarrow$ *a-b* (R_2): Undo a vocalic Sandhi, and add a compound split (hyphen, -) between its two phonemes a and b; example: *c**ai**va* ("and indeed"; refer to Footnote 2): *ai* $\rightarrow$ *a-e*, resulting split string: *ca-eva*.

3. $p \rightarrow p$- (R_3): Leave p unchanged, and add a compound split; example: *mahāgiriḥ* ("high mountain"): $\bar{a} \rightarrow \bar{a}$-, result: *mahā-giriḥ*.

4. $p \rightarrow a$- (R_4): Apply a non-vocalic Sandhi, and add a compound split; example: *aśvaśca* ("and the horse"): *ḥ* $\rightarrow$ *ś*-, result: *aśvaḥ-ca*. – Non-vocalic Sandhi rules depend on the directly following observed phoneme. Therefore, *ḥ* is transformed into *ś* only if the next phoneme is an voiceless palatal. This kind of contextual information is not encoded in the training data for two reasons. First, the classifier should learn to infer the rules from the context - that's why a bidirectional recurrent neural network is used as the learning algorithm. Second, lexemes such as *āścarya* ("wonder") contain the sequence *śc*, but should not be split at this point. Omitting the explicit formulation of context rules is therefore intended to keep the algorithm as "unprejudiced" as possible.

5. $p \rightarrow a$ (R_5): Apply a Sandhi; example: *ratnaṃ* ("the jewel"): *ṃ* $\rightarrow$ *m*, result: *ratnam*. – Because the string-final Sandhi depends on the first phoneme $p_{1\,next}$ of the following string, $p_{1\,next}$ is added to the observed sequence. The target of $p_{1\,next}$ is set to the dummy class BOW in this case. All phonemes with the gold annotation BOW are ignored in the final evaluation, while any silver BOW annotations that are not found at the end of a sequence are counted as false negatives. If the start of the next string is, for instance, the consonant c, training data for this instance will look like this:

observed	r	a	t	n	a	ṃ	$\underbrace{c}_{p_{1\,next}}$	
target		r	a	t	n	a	m	BOW

Some non-vocalic Sandhis (e.g., (*n-c*) $\rightarrow$ (*ṃś-c*)) replace a single phoneme (*n*) by multiple phonemes (*ṃ* and *ś*). To keep the format of the data consistent, "superfluous" phonemes of the observed sequence are marked as deletable (x) in this case:

observed	t	ā	ṃ	ś	c	a	g
target	t	ā	x	n	c	a	BOW

Internally, this deletion is interpreted as an instance of R_5.

Table 2 describes the composition of the full data set in terms of Sandhi types, grouped by the count of phonemes in the observed sequences. The table shows that Sandhi and compounding are frequent phenomena, as about 1 of 10 phonemes is subjected to one of the transformation rules. As could be expected, the frequency of the transformations increases with the length of strings (Table 3).

R_1	R_2	R_3	R_4	R_5
90.96	1.49	2.62	0.88	4.05

Table 2: Percentual proportions of Sandhi rules R_1-R_5 in the data

Length	Proportion	$\|R\|$/string
≤ 5	33.89	0.0317
≤ 10	45.89	0.2174
≤ 15	13.7	0.9373
≤ 20	4.75	1.9707
≤ 40	1.7	3.3038
> 40	0.06	6.9375

Table 3: Composition of the full data set: String length classes, their proportion in the dataset in percent, and the average number of transformation rules R_i per string

4. Algorithm

A Recurrent Neural Network is used for the labeling task. The network consists of an input layer, a hidden forward and backward layer (Schuster and Paliwal, 1997), and an output layer. At time step t, the input layer receives the phoneme observed at position t in a string in 1-of-n encoding. The size of the input layer is, therefore, identical with the number of distinct input phonemes. The output layer contains as many units as there are target classes in the current learning problem. Because sample size is varied systematically to study its influence on the labeling accuracy (refer to Table 7), and smaller samples may not contain all types of rules, the size of the output layer varies between 120 classes for 10.000 samples and 155 classes for the full data set. The forward and backward hidden layers capture the left and right context of t, respectively. LSTM cells are used in the hidden layer, because they have been shown to be less susceptible to the "vanishing gradient" problem than regular neural network cells (Hochreiter et al., 2001). The structure of the LSTM cells follows the formulation

Rule type	P	R	F
R_1	99.59	99.44	99.51
R_2	92.96	93.23	93.09
R_3	89.53	91.67	90.59
R_4	89.35	95.55	92.35
R_5	98.28	98.63	98.46

Table 4: Precision (P), recall (R), and F score (F) per rule class (refer to Section 3.); full data set

Len. class	0	1	2	3	≥ 4
≤ 5	33.56	0.4	0.02	0	0
≤ 10	43.43	2.41	0.14	0.01	0
≤ 15	11.72	1.81	0.2	0.02	0
≤ 20	3.44	0.85	0.19	0.04	0.01
≤ 40	1.05	0.43	0.14	0.05	0.02
> 40	0.02	0.01	0.01	0.01	0.01

Table 5: Proportion of errors per string length class; full data set

in (Graves, 2012). The output layer receives the individual outputs from both hidden layers and performs a softmax regression for the desired target values. The network is trained with stochastic gradient descent, and implemented in C++.

5. Evaluation

All evaluations, except for that of the full data set, are performed with 10 crossvalidations (CV), and the averaged values from these CVs are reported. The evaluation of the full data set is only performed once with 90% of training and 10% of test data due to time limitations. The labeler is trained for 100 epochs with a momentum of 0.9 and a learning rate of 0.0005 for each fold of the CVs. The labeling of one string (forward pass of the network) takes approximately 1 millisecond on a normal desktop computer. This means that the trained labeler can process around 1.000 strings per second.

Table 4 evaluates the results for all phonemes in the full data test set with regard to the types of rules involved (refer to Section 3.). The labeler achieves high F scores for R_1 ($p \rightarrow p$) and R_5 ($p \rightarrow a$), and is, therefore, in general quite "conservative", pleading mainly for keeping the original phoneme or replacing it with another one without introducing a compound split. Because R_5 is mostly found in string final position, the high F score for R_5 just means that the labeler has learned the Sandhi rules that depend on the initial phoneme of the following string. While this task is largely deterministic – penultimate *ṃ* is, for example, always mapped to *m* –, the F scores of the complicated classes R_{2-4} are markedly lower. Table 5 describes the stringwise accuracy of the labeler for the full data set, grouped by string length classes (refer to Table 3). Most mislabelings occur for strings with 6-15 phonemes in which one phoneme is not labeled correctly.

For a more detailed analysis, Table 6 gives precision, recall, and F scores for all occurrences of the true target classes of phoneme *ā*, which is notoriously difficult to analyze (remember the example of *hāhākārāḥ* in Section 1.1.)

Target	P	R	F	Proportion
ā	98.09	97.82	97.95	0.82
ā-	84.6	87.72	86.13	0.02
a-a	89.08	92.67	90.84	0.07
a-ā	88.26	86.48	87.36	0.03
ā-a	83.59	75	79.06	0.01
ā-ā	72.45	58.97	65.02	< 0.01
āḥ	73.13	77.66	75.33	0.03

Table 6: Detail results for the observed phoneme *ā*; full data set. The column Target records the desired output rule.

and which can raise R_2 and R_3 rules. Most importantly, the table shows that the F scores of the non-deterministic mappings, which depend on the lexical and semantic context, are correlated with the amount of training data available for each mapping. A closer look at some of the results supports this impression. The string *ātreyādayaḥ* ("(the man called) Ātreya etc.") is labeled correctly as *ātrey(a-ā)dayaḥ*, most probably because *ādayaḥ* is a very common final member of compounds. Another example is *śarkarāmaricopetaṃ* ("mixed with sugar and pepper", labeled as *śarkar(ā-)maric(a-u)peta(m)*). The string is part of the medicinal subcorpus that contains several texts enumerating similar lists of ingredients. In contrast, the string *prakīrṇakeyūrabhujāgramaṇḍalaḥ* ("whose round forearm is scattered with bracelets", labeled: *prakīrṇ(a-)keyūr(a-)bhuj(ā-a)gr(a-)maṇḍalaḥ*) contains one error at *bhuj(ā-a)gra*, which should be labeled as *bhuj(a-a)gra* instead. The word is part of poetic description of a god in the Matsyapurāṇa, and the whole passage is strongly indebted to the demanding style of Sanskrit poetry. It should be noted that the proposed solution offers a lexically, though not semantically valid interpretation of the string, because *bhujā* ("by the enjoying one") is quite regularly found as a compound termination. In addition, the proposed analysis is definitely preferable over possible unlexical solutions such as *bhuj(ā-)gra*.

To estimate how the amount of training data influences the accuracy of the labeler, training and testing are repeated for randomly drawn samples of 10.000, 20.000, 50.000, and 500.000 strings. Table 7 shows that the stringwise accuracy increases with the amount of training data. The accuracy for the full data set comes close to the results reported in (Hellwig, 2015), where a language model and morphological resources are used for analysis. Nevertheless, the labeler shows acceptable accuracy rates even for samples as small as 10.000 strings. As many hand labeled data sets in the Humanities may have similar sizes, the labeler seems to be applicable even in such research scenarios.

6. Conclusion

The paper has presented a novel algorithm for Sandhi resolution and compound splitting that can be trained with shallow annotations and without the use of external resources such as language models, morphological or phonetic analyzers. This is an important prerequisite for its application in South-Asian Studies, because such resources

Sample size	Accuracy
10.000	77.92
20.000	80.98
100.000	87.68
500.000	91.03
full data set	93.24

Table 7: Overall string-wise labeling accuracy w.r.t. sample size

are missing or are still under construction for many ancient languages of India.

On the computational side, it should be explored if a deep architecture improves the accuracy of the labeler. The paper of dos Santos et al. (Santos and Zadrozny, 2014), who insert a convolutional layer right after the input, appears to be a promising starting point for this track of research. A primary area of application is the initial linguistic analysis of under-resourced Middle Indo-Aryan languages such as Old Marathi, or of premodern Nepali. Given the undemanding phonetic structure of Nepalese, for example, the algorithm may be a good choice for suffix splitting in this language. As the algorithm is fast when compared with a full-fledged linguistic processor, it should also be useful for analyzing larger corpora of Sanskrit, which may become available through OCRing printed texts and manuscripts. Here, the algorithm could either be used as a pure word splitter, or be integrated into the existing Sanskrit tagger as a preprocessing step that determines the most probable split of a sentence and feeds this proposal into the full linguistic analysis pipeline.

7. References

Cardona, George, 1976. *Pāṇini. A Survey of Research*. The Hague - Paris: Mouton.

Dima, Corina and Erhard Hinrichs, 2015. Automatic noun compound interpretation using deep neural networks and word embeddings. In *Proceedings of the 11th International Conference on Computational Semantics*.

Elman, Jeffrey L., 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

Gers, Felix A. and Jürgen Schmidhuber, 2001. LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 12:1333–1340.

Graves, Alex, 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*. Heidelberg: Springer Verlag.

Hellwig, Oliver, 2009. SanskritTagger, a stochastic lexical and POS tagger for Sanskrit. In Gérard Huet, Amba Kulkarni, and Peter Scharf (eds.), *Sanskrit Computational Linguistics. First and Second International Symposia*, Lecture Notes in Artificial Intelligence, 5402. Berlin: Springer Verlag.

Hellwig, Oliver, 2015. Morphological disambiguation of Classical Sanskrit. In Cerstin Mahlow and Michael Piotrowski (eds.), *Systems and Frameworks for Computational Morphology*. Cham: Springer.

Hochreiter, Sepp, Y. Bengio, P. Frasconi, and J. Schmidhuber, 2001. *Gradient flow in recurrent nets: The difficulty of learning long-term dependencies*. New York: IEEE Press, pages 237–243.

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

Huet, Gérard, 2005. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *Journal of Functional Programming*, 15(04):573–614.

Kiparsky, Paul, 2009. On the architecture of Pāṇini's grammar. In Gérard Huet, Amba Kulkarni, and Peter Scharf (eds.), *Sanskrit Computational Linguistics*, volume 5402 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer, pages 33–94.

Kulkarni, Amba and Devanand Shukla, 2009. Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70(1-4):169–177.

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira, 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*.

Mishra, Anand, 2009. Simulating the Pāṇinian system of Sanskrit grammar. In *Sanskrit Computational Linguistics*. Springer, pages 127–138.

Mittal, Vipul, 2010. Automatic Sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*. Stroudsburg, PA, USA: Association for Computational Linguistics.

Oberlies, Thomas, 2003. *A Grammar of Epic Sanskrit*. De Gruyter.

Qiu, Siyu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu, 2014. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014*.

Santos, Cicero D. and Bianca Zadrozny, 2014. Learning character-level representations for Part-of-Speech tagging. In Tony Jebara and Eric P. Xing (eds.), *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. JMLR Workshop and Conference Proceedings.

Scharfe, Hartmut, 1977. *Grammatical Literature*. A History of Indian Literature, Volume 5, Fasc. 2. Wiesbaden: Otto Harrassowitz.

Schuster, M. and K.K. Paliwal, 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Wackernagel, Jakob, 1978. *Altindische Grammatik*. Göttingen: Vandenhoek und Ruprecht. Reprint from 1896.

Witzel, Michael, 1995. Early Indian history: Linguistic and textual parametres. In George Erdosy (ed.), *The Indo-Aryans of Ancient South Asia. Language, Material Culture and Ethnicity*, volume 1. Berlin, New York: Walter de Gruyter, pages 85–125.