Sanskrit Word Segmentation Using Character-level Recurrent and Convolutional Neural Networks

Oliver Hellwig
SFB 991, University of Düsseldorf
IVS, University of Zurich
hellwig7@gmx.de

Sebastian Nehrdich
Center for Buddhist Studies
University of Hamburg
nehrdbsd@googlemail.com

Abstract

The paper introduces end-to-end neural network models that tokenize Sanskrit by jointly splitting compounds and resolving phonetic merges (Sandhi). Tokenization of Sanskrit depends on local phonetic and distant semantic features that are incorporated using convolutional and recurrent elements. Contrary to most previous systems, our models do not require feature engineering or extern linguistic resources, but operate solely on parallel versions of raw and segmented text. The models discussed in this paper clearly improve over previous approaches to Sanskrit word segmentation. As they are language agnostic, we will demonstrate that they also outperform the state of the art for the related task of German compound splitting.

1 Introduction

Sanskrit is an Indo-Aryan language that served as lingua franca for the religious, scientific and literary communities of ancient India. Text production in Sanskrit started in the 2. millenium BCE and has continued until today.[1] A 19th century cataloguing project recorded more than 40,000 Sanskrit texts known at that time (Aufrecht, 1891–1903), which covers only a small part of the extant Sanskrit literature. Apart from the oldest Vedic texts, Sanskrit has little diachronic variation on the morphological level, because it was regularized by the grammarian Pāṇini in the 3rd c. BCE.

NLP of Sanskrit is challenging due to compounding (see Ex. 1) and the phonetic processes called Sandhi ('connection'; see Ex. 2–5). Compounding is widely used in other languages, and NLP has developed methods for analyzing compounds (Macherey et al., 2011; Ma et al., 2016). In Sanskrit, however, syntactic co- and subordination tend to be diachronically replaced by compounding (Lowe 2015; see also Sec. 3), so that many sentences in later literature consist only of a few long compounds that are loosely connected by a semantically light verb or an (optional) copula, as shown in this example:

(1) *āśrayabhūtakhādikathanena*
foundation-become-air-etc.-mentioning
"(Something is described) by mentioning air etc. that have become [its] foundations."

The term Sandhi denotes a set of phonetic processes by which the contact phonemes of neighboring word tokens are changed and merged, and which create unseparated strings spanning multiple tokens (Whitney, 1879). Sandhi occurs between adjacent vowels (vocalic Sandhi; Ex. 2), between consonants and vowels (Ex. 4) and between adjacent consonants (Ex. 5):

(2) *rājā+uvāca* 'the king said' $\xrightarrow{\text{ā+u=o}}$ *rāj$\underline{o}$vāca*

(3) **rāja+uvāca* 'O king, he said' $\xrightarrow{\text{a+u=o}}$ *rāj$\underline{o}$vāca*

(4) *prāc+eva* 'before indeed' $\xrightarrow{\text{c+e=ge}}$ *prā$\underline{ge}$va*

(5) *tad+hi* 'because this ...' $\xrightarrow{\text{d+h=ddh}}$ *ta$\underline{ddh}$i*

In addition, Sandhi occurs between independent inflected words (Ex. 2–5) as well as between members of compounds.[2] Because different combinations of unsandhied phonemes can result in the same surface phoneme, Sandhi resolution is non-deterministic and depends on the semantic context of the sentence (see Ex. 3 for a morphologically and lexically valid, but semantically dispreferred reading of the string *rājovāca*).

[1] Text production was oral until the first centuries BCE (Falk, 1993). The texts were transmitted by memorization in this period, making them less (!) prone to transmission errors than in written form.

[2] The compound in Ex. 1 is split as *āśraya-bhūta-kha-ādi-kathanena*, and *kha* +*ādi* = *khādi* is a Sandhi phenomenon.

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2754–2763
Brussels, Belgium, October 31 - November 4, 2018. ©2018 Association for Computational Linguistics

Scriptorial and editorial conventions further complicate the analysis of compounds and Sandhi. While most Indian manuscripts don't insert spaces between strings, modern editors use spaces a gusto. Moreover, the (correct) application of Sandhi is not followed by all authors and editors to the same extent, so that the unsandhied tokens *tat hi asti* ('as this is ...') can occur as *taddhyasti*, *taddhy asti*, *tad dhy asti*, *tad dhyasti* or even *tat hi asti* (unchanged).

Our models aim at transforming a given sentence into a sequence of unsandhied tokens. We refer to this task as *Sanskrit word splitting* (SWS), and subsume Sandhi and compounding phenomena under the common term *splits*. We address SWS by using a combination of convolutional and recurrent elements. The recurrent elements integrate sentence level information that leads to qualified decisions about the semantic meaningfulness of possible compound and Sandhi splits (see Ex. 2 and 3), while the convolutional elements are meant to replace n-gram extraction, which is frequently used in word segmentation architectures. As our models operate on the character level, SWS can be formulated in a sequence labeling framework.

Consequently, this paper has three main contributions:

1. We introduce novel character-based models for SWS that beat state of the art models by large margins.

2. We compare against sequence-to-sequence models and demonstrate that our models work on par with them, but need significantly less time for training and inference.

3. We publish a new dataset for Sanskrit word splitting that consists of more than 560,000 sentences with manually validated splits. The dataset and the code are released at `https://github.com/OliverHellwig/sanskrit/papers/2018emnlp`.

In the rest of this paper, we use the following terminology. A *token* is an unsandhied word that is not itself a compound. A *string* is a sequence of characters that is delimited by a space or a *daṇḍa*. Each string contains at least one token, at least one compound (that itself consists of at least two tokens) or a Sandhied mixture of both. A *sentence* is a piece of Sanskrit text that is terminated by the punctuation mark called *daṇḍa* "stick" (|) and consists of at least one string. Any sentence can consist of multiple independent clauses, which are not demarcated by punctuation in Sanskrit, or consist of a part of a larger clause only.

The paper proceeds as follows: Section 2 gives an overview of related work in NLP. Section 3 introduces our SWS dataset. Section 4 describes the sequence labeling models developed for this paper and three baseline systems, whose evalution is presented in Sec. 5. Section 6 summarizes the paper.

2 Related Research

Most NLP systems for SWS combine Pāṇinis phonetic and morphological rules with a lexical resource, either by using formal (Huet, 2005; Goyal et al., 2009; Kulkarni and Shukla, 2009) or statistical methods, including Dirichlet processes (Natarajan and Charniak, 2011), finite state methods (Mittal, 2010), graph queries (Krishna et al., 2016) and hybrid systems (Hellwig, 2015a).

A number of recent papers approaches SWS with deep learning models. Hellwig (2015b) splits isolated strings by applying a one-layer bidirectional LSTM to two parallel character based representations of a string. The restriction to isolated strings is problematic, because SWS relies on the grammatical and semantic context of the full sentence in many cases. Restricting a model to isolated strings ignores these linguistic clues.

Reddy et al. (2018) formulate SWS as a translation task on the sentence level. They transform surface and unsandhied sentences using the *sentencepiece* model and "translate" the surface into the unsandhied sentence using a seq2seq model with attention. Gantayat et al. (2018) use an encoder-decoder architecture with a global attention mechanism and apply their model to isolated strings from a small dataset (Bhardwaj et al., 2018). So far, no direct comparison of deep learning models for SWS has been done, because the authors used different, partly unpublished datasets and reported performance on different linguistic levels (sentence, string) and with different evaluation methods. We will therefore try to make a fair and comprehensive comparison with the state of the art in Sec. 5.

SWS is closely related to word segmentation for other Asian languages such as Thai (Haruechaiyasak et al., 2008), Chinese or Japanese

(Kanji), with most research being done for Chinese and Japanese. Contrary to Sanskrit, Chinese and Japanese don't exhibit Sandhi phenomena and their logographic scripts condense information, making it possible to use "word-level" CRFs on the output, for example. Chen et al. (2015) interpret Chinese word segmentation (CWS) as a sequence labeling task and evaluate a range of (stacked) bidirectional recurrent architectures that are combined with a final sentence level likelihood layer (Collobert et al., 2011) maximizing the transition score of the BMES encoded target sequence. Their best model uses a single layer bidirectional LSTM with bigrams of pre-trained character embeddings as inputs. Cai and Zhao (2016) deal with CWS by first forming word hypotheses from characters using a gated unit and then processing the word hypotheses with an LSTM-based language model. They minimize the combined word and sentence level scores using a structured margin loss and achieve better performance than Chen et al. (2015) on standard CWS datasets. Kitagawa and Komachi (2017) adapt the model proposed by Chen et al. (2015) for Japanese word splitting, but use characters, character n-grams and lexicon-based word boundary features as inputs. The authors report state of the art performance, but observe a clear drop in the F score of their model, when texts contain a high proportion of Hiragana characters and thus come closer to syllabic or alphabetic scripts.

3 Data

Several datasets for SWS have been published in the last years. While the dataset of Bhardwaj et al. (2018) may be too small and unvaried for training deep learning models, Krishna et al. (2017) re-analyze 560,000 sentences from the Digital Corpus of Sanskrit (DCS)[3] using the Sanskrit Heritage Reader (Goyal and Huet, 2016). Re-analysis is necessary, because the DCS stores the morpho-lexical analysis of strings, but does not record split points and Sandhi rules applied. Due to different linguistic choices (Pāṇinian vs. corpus-oriented) and to different ideas about the (non-)compositional meanings of compounds their final dataset contains only 115,000 sentences (see the discussion in Krishna et al. 2017 and the analysis in Sec. 3.1). As the size of the dataset is crucial for most deep learning methods, we decided to release a new dataset along with this paper. Each sentence contained in the DCS is re-analyzed using the SanskritTagger software (Hellwig, 2009). Our dataset contains the surface forms of sentences in the DCS and the split points and Sandhi rules that the tagger proposes for their morpho-lexical gold analyses stored in the DCS. We didn't differentiate between compound and inter-word splits, as this distinction introduces morphological categories into the dataset. Table 1 shows an example of the annotation format.

Surface	r	ā	j	o	v	ā	c	a
Unsandhied	r	ā	j	ā-u	v	ā	c	a

Table 1: Data extracted from the string *rājovāca*, which is split into the two tokens *rājā* ("king") and *uvāca* ("(he, she) said"; see Ex. 2).

Table 2 shows the statistics of our dataset, split by text genres (first column). The dataset contains 2,978,509 strings and 4,171,682 tokens in 561,596 sentences. Most sentences come from the Epic and scientific (medicine, alchemy, astronomy) domain. While Epic texts are mostly written in easy, plain Sanskrit, the scientific works use many uncommon terms (likely to reoccur in the lexicographic domain) and long compounds. Sentence length is higher in the prose subcorpora (Buddhist, Vedic prose, ritualistic texts).

The fourth column shows that split phenomena are frequent in Sanskrit, occurring for more than 8% of all characters. Columns 5 and 6 report the proportions of complicated splits in relation to all splits. While 15% of all splits are resolved into a vocalic Sandhi, compound breaks are the dominant split type, which is also responsible for the majority of errors and ambiguities (see Sections 3.1 and 5.2). The last column also reflects the diachronic development from earlier texts with limited compounding (Vedic, ritualistic and Dharma texts) towards classical Sanskrit, which shows a strong preference for compounding. We use a fixed split of 90% of the sentences for training, a development set of 5% for parameter optimization and 5% for testing.

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

Genre	#sen	$\lvert\bar{S}\rvert$	$\frac{s\bar{p}l}{S}$	$\frac{v\bar{o}c}{spl}$	$\frac{\bar{cp}}{spl}$
Epos	322811	42	0.081	0.144	0.355
Science	105597	50	0.096	0.148	0.478
Literature	36989	50	0.085	0.173	0.382
Religious	24055	48	0.091	0.174	0.413
Dharma	18506	42	0.08	0.162	0.276
Buddhist	13739	78	0.083	0.143	0.442
Lexicogragphy	13015	44	0.077	0.146	0.376
Vedic prose	11425	58	0.081	0.181	0.097
Philosopy	7277	50	0.088	0.185	0.379
Vedic poetry	4355	46	0.071	0.106	0.079
Ritual	3222	71	0.069	0.183	0.215
Grammar	605	32	0.054	0.194	0.344
Overall	561596	46	0.084	0.150	0.375

Table 2: Statistics of the full dataset; $|\bar{S}|$: average sentence length in characters; columns 4-6 give the average proportion of splits/string $\frac{s\bar{p}l}{S}$, of vocalic Sandhis/split $\frac{v\bar{o}c}{spl}$and of compound splits/split $\frac{\bar{cp}}{spl}$

3.1 Quality of the training data

The dataset released by Krishna et al. (2017) and the one released with this paper both build on the DCS as gold standard. As this corpus was curated by a single user and the project never released a proper annotation guideline, one may suspect that it contains a certain level of inconsistencies and errors that influence the quality of the models and impose an upper limit for the model accuracy. In order to estimate the size of these effects, the authors of this paper independently corrected the analyses of 50 sentences randomly drawn from the training set (250 words, 2,354 characters including spaces). The corrections made by the authors differed at 23 character positions, corresponding to 20 strings in 15 sentences. 16 of these differences concerned compound splits, where the authors disagreed about the (non-)compositional meaning of compounds. A good example for such a disagreement is the string *rājayoga*, which was split as *rāja-yoga* “king-Yoga” = “Yoga of a king” by one author (compositional reading), but left unchanged as the name of a school of Yoga by the other one (non-compositional reading). After adjudicating these disagreements, there remain 5 of 250 strings with annotation errors in the training data, which corresponds to an error level of 2% of all strings and 0.2% of all characters for this sample.

We further explored the effect of compositionality by independently splitting 56 sentences of the Buddhist treatise Triṃśikāvijñaptibhāṣya, which is not part of the DCS. As the text uses highly technical terminology, the degree of disagreement can be expected to be higher than for plain narrative texts. We adjudicated our Sandhi annotations, but kept conflicts in compound splitting unresolved. 94.5% of all strings (394 of 417) and 69.7% of all sentences obtained the same compound analysis by both authors. Again, the majority of differences (11 of 23) showed up when a compound can have a non-compositional meaning that is closely connected with its compositional reading. Evaluation will show that these cases are responsible for a large parts of the model errors.

4 Models

4.1 Input Features

The character based models are trained with embeddings of the indidual surface characters, which are initialized with uniform random values from $[-1, +1]$ and updated during training. Following Kitagawa and Komachi (2017), the input can be enriched with multinomial *split probabilities* that are built from the training data. When the training data contain a split rule for surface character t_i at position i, we extract left ($g^L_{i,n}$) and right ($g^R_{i,n}$) character n-grams with lengths $n \in [2, 7]$[4] that end/start at position i, so that $g^L_{i,n} = \{t_{i-n+1}, t_{i-n+2}, \ldots t_i\}$ and $g^R_{i,n} = \{t_i, t_{i+1}, \ldots t_{i+n-1}\}$. Counts $\#(.)$ for individual n-grams are accumulated over the whole training set. At training and test time, a vector $\mathbf{v}_p \in \mathbb{R}^{2\cdot(7-2+1)=12}$ is assigned to each character position. Its element corresponding to the left n-gram of length 2, for example, is calculated as

$$\mathbf{v}_p(L, 2) = \frac{\#(g^L_{i,2})}{\max \#(g^L_{*,2})} \tag{6}$$

We evaluate the influence of split probabilities in the ablation study (Sec. 5.2).

4.2 Extern Models for Comparison

We compare our models against the following baselines:

Bidirectional RNN We re-implement the model described in Hellwig (2015b), but apply it to full sentences instead of isolated strings. Character embeddings are fed into a bidirectional recurrent

[4] Longer n-grams did not produce improvements on the dev set.

layer with LSTM units. The output of the recurrent layer is additionally regularized by using dropout (Srivastava et al., 2014), and classification is performed using softmax with cross-entropy loss. We decode the output of the softmax in a greedy fashion without considering interactions between adjacent output classes.

seq2seq We retrain the model described in Reddy et al. (2018) with our data after pre-processing them with the unsupervised text tokenizer *sentencepiece* (Schuster and Nakajima, 2012).[5]

Transformer As an alternative to recurrency based seq2seq, we apply the model described in Vaswani et al. (2017) to the input pre-processed with *sentencepiece*. This model relies entirely on an attention mechanism to draw global dependencies between input and output. To our best knowledge, this is the first time that this model has been used for SWS. We use the publicy available implementation tensor2tensor.[6].

4.3 Models Combining RNN and CNN

Convolutional Element Combinations of recurrent and convolutional (LeCun et al., 1998) elements are effective for tasks where complex local features are extracted by the convolutional element and then considered in larger contexts by the recurrent element (and vice versa; see Bjerva et al. 2016 or Ma and Hovy 2016). We use convolutional features c_i as proposed by Kim (2014). Let w denote the width of the input matrix $\mathbf{X}$ of the convolution (= number of time steps), h its height, n the width of the convolutional filter $f^n \in \mathbb{R}^{n \times h}$, $\sigma(.)$ a non-linearity (Rectified Linear Units (Nair and Hinton, 2010) in this paper) and b a bias. A convolutional feature at character position i and for filter j is defined as:

$$c_{ij}^n = \sigma(f_j^n \cdot \mathbf{X}_{[i:i+n-1,*]} + b) \qquad (7)$$

The feature map $\mathbf{c}_i^n$ for m different filters is formed by concatenating the convolutional features ($\mathbf{c}_i^n = [c_{i1}^n, c_{i2}^n, \ldots c_{im}^n]$) and the output $\mathbf{c}$ of the convolutional element is formed by concatenating the feature maps ($\mathbf{c}_i = c_i^1 \oplus c_i^3 \oplus \ldots$). We use use odd filter widths only to avoid problems with patch alignment. We tested convolution with small quadratic filters as used in image convolution as well as other methods for combining the learned filter such as averaging, addition or max-pooling of the stacked filters, but did not observe improved performance on the dev set.

Model 1: Convolution → Recurrency (crNN) As an alternative to n-gram extraction (Chen et al., 2015; Kitagawa and Komachi, 2017), a convolutional element is applied to the character embeddings (see Fig. 1a). Its outputs (Eq. 7) are fed into a bidirectional recurrent layer (Schuster and Paliwal, 1997). As in the baseline RNN (Sec. 4.2), dropout is inserted after the recurrent layer, and classification is performed using softmax with cross-entropy loss and greedy decoding.

Model 2: Recurrency → Convolution (rcNN) The order of convolutional and recurrent elements is switched (see Fig. 1b), so that the convolutional operation replaces additive n-gram formation before the classification layer. The remaining architecture is identical to that of crNN

Model 3: rcNN with Shortcuts (rcNN$_{\text{short}}$) This model extends rcNN by adding shortcut connections (Bishop, 2000) that concatenate the character embeddings and the RNN outputs with the concatenated feature maps $\mathbf{c}$ (see Fig. 1c). When $\mathbf{e}_i$ denotes the embedding of character i and $\mathbf{r}_i$ the output of the recurrent layer at position i, the input to the classification layer is defined as $\mathbf{e}_i \oplus \mathbf{r}_i \oplus \mathbf{c}_i$. Shortcuts are evaluated because we hypothesized that the access to unconvolved information about the input sequence and the output of the recurrent layer would facilitate the exact prediction of split locations. For a better control of information flow, we also experimented with residual (He et al., 2016) and highway (Srivastava et al., 2015) instead of shortcut layers, but could not observe improvements on the dev set, most probably because our models are not deep enough for these layer types to show effects.

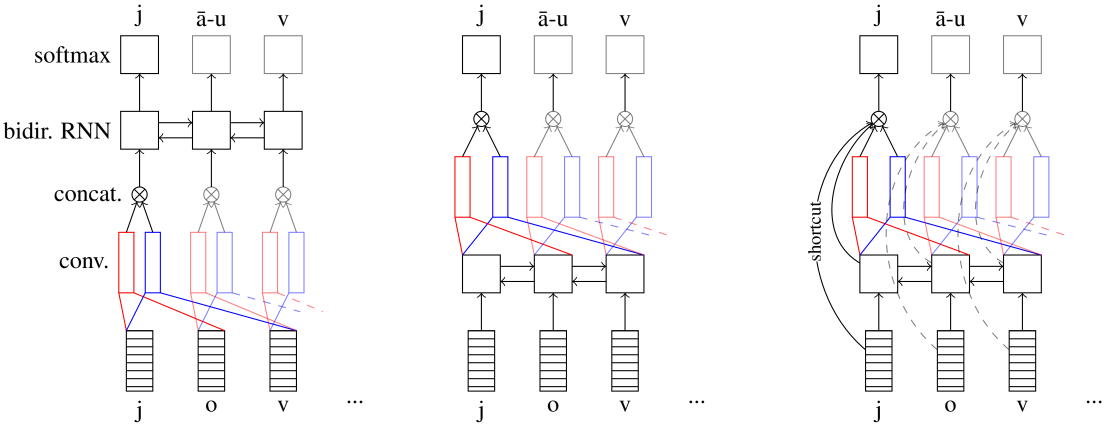
5 Evaluation

5.1 Evaluation Settings

We use the following settings found on the dev set for the character based models: embedding size: 128; 200 hidden recurrent units; 100 convolutional feature maps with filter widths of 3,5 and 7. We use regularized (Zaremba et al., 2014) instead of

[5] Code for the model: https://github.com/cvikasreddy/skt; for the tokenizer: https://github.com/google/sentencepiece

[6] https://github.com/tensorflow/tensor2tensor



(a) crNN

(b) rcNN

(c) rcNN$_{\text{short}}$

Figure 1: Character based models, unfolded for the sequence labeling task $j{+}o{+}v \rightarrow j{+}a\text{-}u{+}v$.

vanilla LSTM units. All models are trained with the Adam optimizer (Kingma and Ba, 2015), an initial learning rate of 0.005 and batch size of 100. Gradients with a magnitude higher than 5.0 are cut. The models used for model selection (Sec. 5.2) are trained for 5, the other character-based models for 10 iterations. We train the Transformer in its default configuration as described in Vaswani et al. (2017) with a vocabulary size 5k[7] and report performance on the test set based on evaluations on the dev set. The model of Reddy et al. (2018) is trained for 80 epochs with our training data and the same parameters as described in the original paper. All calculations are run on a Maxwell Titan X GPU. We compare the models using sentence accuracy ($\frac{\text{\#sens. with errors}}{\text{\#all sens.}}$) and string based P(recision), R(ecall) and F score (Ma et al., 2016), where P and R are equivalent to the measures used in the CWS bakeoffs (Sproat and Emerson, 2003).

5.2 Model Selection

The upper half of Tab. 3 compares the evaluation metrics for the three character based models introduced in this paper trained with and without split probabilities (Sec. 4.1). We test differences in string accuracy using the McNemar test.[8] In general, all models that use recurrency *before* convolution (rcNN*) have string accuracy rates that are significantly higher at the 0.001 level than for models that use convolution before recurrency (crNN).

Table 3 shows that the differences in the performance of crNN and rcNN* are almost as large as between the RNN baseline and the best model from this paper (lower half of Tab. 3), although crNN and rcNN* differ only by the switched order of recurrent and convolutional elements. We found this result surprising, because applying convolution to the character embeddings appeared like a good parametrized alternative to n-gram extraction, which is often the first step in architectures for Chinese and Japanese word segmentation.

To further investigate this phenomenon, we evaluated 60 randomly chosen strings from the test set in which either crNN$^{\text{split}}$ or (XOR) rcNN$^{\text{split}}_{\text{short}}$ made an error. 45 of the errors relate to compound splitting, partly combined with vocalic Sandhi, either by missing a split (rcNN$^{\text{split}}_{\text{short}}$: 11, crNN$^{\text{split}}$: 15) or by oversegmenting compounds (rcNN$^{\text{split}}_{\text{short}}$: 13, crNN$^{\text{split}}$: 6). Most notably, rcNN$^{\text{split}}_{\text{short}}$ tends to insert more splits than crNN$^{\text{split}}$. This behavior can be observed for missing splits and especially for oversegmentations. A more detailed inspection shows that 11 of 13 oversegmentations actually induce a compositional reading of a compound. *saralāṅga* “name of a pine resin”, for example, is oversegmented into *sarala-aṅga* “pine-limb”, which is the etymological derivation of this compound. In contrast, crNN creates oversegmentations such as *śṛṅgavanti-aḥ*, where *śṛṅgavanti* “having horns” (nom. pl. neutre) is a valid form, while *aḥ* is not an independent word form in Sanskrit. Interestingly, rcNN$^{\text{split}}_{\text{short}}$ mis-segments the same string into *śṛṅga-vantyaḥ* in another sentence of the test set. Though differing from the

[7] Larger vocabulary sizes did not improve on the dev set, but performance gains by further decreasing the vocabulary size appear to be possible.

[8] Testing the sentence accuracy produced highly correlated test statistics. Results are therefore not discussed.

		Sen.	String		
Model	sp?	A	P	R	A
Model selection					
crNN		75.7	90.8	90.8	94.2
crNN	✓	75.4	90.6	90.9	94.2
rcNN		81.9	93.2	93.6	95.8
rcNN	✓	*82.2*	*93.3*	*93.8*	*95.9*
rcNN$_{\text{short}}$		81.7	93.1	93.5	95.8
rcNN$_{\text{short}}$	✓	**82.6**	**93.6**	**93.8**	**96**
Comparison with other models					
Hellwig (2015b)		77.7	91.8	91.8	94.8
Reddy et al. (2018)		72.3	90.2	88.4	93.3
Transformer 5K		84.9	**94.9**	94.5	96.5
rcNN$^{\text{split}}_{\text{short}}$	✓	**85.2**	94.6	**94.8**	**96.7**

Table 3: Upper half: Results for model selection (Sec. 5.2); lower half: Comparison with baseline models (Sec. 4.2 and 5.3)

	Hellwig (2015b)			rcNN$^{\text{split}}_{\text{short}}$ (this paper)		
Rule	P	R	F	P	R	F
ā	98.09	97.82	97.95	99.24	99.1	99.17
ā-	84.6	87.72	86.13	89.81	93.27	91.5
a-a	89.08	92.67	90.84	94.36	96.34	95.34
a-ā	88.26	86.48	87.36	91.5	95.6	93.51
ā-a	83.59	75	79.06	92.29	84.76	88.36
ā-ā	72.45	58.97	65.02	90.76	72.24	80.45
āḥ	73.13	77.66	75.33	91.48	89.44	90.45

Table 4: P, R and F for rules that produce the surface phoneme *ā*. Data in the left half are from the original publication. As all metrics are consistently better for this paper, we refrain from highlighting the best results in the right half of the table.

gold analysis, this segmentation gives the correct derivational analysis of the adjective (noun *śṛṅga* "horn" + inflected form of the adjectivizing possessive affix *-vat*). The results of rcNN$^{\text{split}}_{\text{short}}$ thus reflect the inherent inconsistencies of the dataset on the level of compound splitting (see Sec. 3.1), and their erroneous splits are frequently semantically meaningful while glossing over minute semantic distinctions. Errors of crNN, in contrast, tend to be real mis-segmentations, indicating that its ability to reflect the semantic level is underdeveloped.

Split probabilities (Sec. 4.1) have a small, but positive effect on string accuracy of the rcNN* models. When the same model with and without split probabilities is compared using the McNemar test, split probabilities significantly increase string accuracy at the 0.1 level for rcNN and at the 0.001 level for rcNN$_{\text{short}}$, while they don't result in significantly better performance for crNN.

5.3 Comparison with Baseline Models

The lower half of Tab. 3 compares the best model introduced in this paper (rcNN$^{\text{split}}_{\text{short}}$) with baselines proposed for SWS in previous research. rcNN$^{\text{split}}_{\text{short}}$ outperforms the character based RNN described in Hellwig (2015b) by a wide margin. While Tab. 3 shows differences of almost 8% in sentence and 3% in string accuracy, Tab. 4 presents the improvements for the single surface character *ā*, which can correspond to a compound split (*ā-*) or to various vocalic Sandhis (*a-a* etc.). For this complicated character, rcNN$^{\text{split}}_{\text{short}}$ achieves consistent improvements of up to 15% on all metrics. We found it especially relevant to observe that rcNN$^{\text{split}}_{\text{short}}$ made large progress for rare rule types such as *ā-ā* or *ā-a*, indicating its increased ability for semantic generalization.

The seq2seq model (Reddy et al., 2018) performs on a similar level of accuracy as the one proposed in Hellwig (2015b). Similar to crNN (Sec. 5.2) it tends to miss splits and to insert faulty ones (e.g., *dānādānaratiḥ*, should: *dāna-ādānaratiḥ* "pleasure in giving and taking", is: *dānāt-ānaratiḥ* "from giving . . . UNK").

Gantayat et al. (2018) evaluate their model using location and split[9] prediction accuracy. The authors report 95.0 location and 79.5 split accuracy, but don't specify how they calculated these values. For this reason and because they evaluate on isolated strings only, we cannot compare directly against their work, but would like to report the following measures for rcNN$^{\text{split}}_{\text{short}}$:

- P, R and F for location prediction[10]: 97.64/98.19/97.91
- Micro-averaged P, R and F for individual rule types such as vocalic Sandhi or compound

[9] This seems to mean prediction of the correct Sandhi rule; see Gantayat et al. (2018, 4.2).

[10] $P = \frac{TP}{TP+FP}$, $R = \frac{TP}{TP+FN}$, TP: number of characters for which gold and model both record a split (though not necessarily of the same type), FP: number of characters at which the model over-segmentates, FN: number of character where the model fails to detect a valid split.

Model	S/A	P	R	A
Triṃśikāvijñaptibhāṣya				
$\text{rcNN}^{\text{split}}_{\text{short}}$	73.9	78.7	68.6	87.1
Transformer (5K)	72.6	77.5	66.9	86.8
Nyāyamañjarī				
$\text{rcNN}^{\text{split}}_{\text{short}}$	60.2	66.4	63.3	84.8
Transformer (5K)	62.1	68.6	63.5	85.2

Table 5: Results for evaluation on the Triṃśikāvijñaptibhāṣya and the Nyāyamañjarī; S/A: sentence accuracy

Model	P	R	A
Ma et al. (2016)	0.955	0.941	0.943
$\text{rcNN}_{\text{short}}$, no sp	0.958	0.958	0.955

Table 6: Results for splitting German compounds; evaluation metrics according to Koehn and Knight (2003)

split: 95.12/95.12/95.12

The Transformer performs almost on par with $\text{rcNN}^{\text{split}}_{\text{short}}$, and the differences in string accuracy are not statistically significant, although $\text{rcNN}^{\text{split}}_{\text{short}}$ takes less time for training (2 h vs. 55 h) and inference (less than 1 min vs. 30 min when analyzing the test set). To better understand if the systems make orthogonal errors and could therefore be used in a mixture of experts, we performed a domain-specific evaluation with 73 sentences from the Buddhist treatise *Triṃśikāvijñaptibhāṣya* and 104 sentences from the philosophical text *Nyāyamañjarī*. We preserved the non-standard orthography of both texts in order to simulate the application of the models to real-world data. This includes the presence of typos, unsolved textual problems and erratic (non)-application of Sandhi.

Both models show a significant drop in overall performance when applied to these data (see Tab. 5). This is not surprising, because the input conventions of these files do not match the conventions of the training-data. Most errors arise again from disagreement about the (non-)compositional reading of technical compounds such as *sarvajña-tva* "all-knowing-ness" (see Sec. 3.1). It has to be noted that both models agree well in their correct decisions and in the type of errors they produce on these data. This indicates that the discrepancy in the orthographical conventions is indeed responsible for a large part of the drop in performance. Given the fact that both texts exhibit a lot of special vocabulary that is not present or used in a very different way in the training set, both models perform surprisingly well. Typical errors common to both models are for example *svalpam* instead of *su-alpam* "very small". Both models have difficulties to seperate Sandhi in passages that do not adhere to the common practice for typesetting of Indian texts in Latin transliteration. *ayaṃpariṇāmaḥ*, for example, was not separated into the usual form *ayaṃ pariṇāmaḥ*. There are certain cases of disagreement between both models that are noteworthy. While Transformer has changed the misspelled word *abhupagamyate* to the correct form *abhyupagamyate* in one case (overlooked by $\text{rcNN}^{\text{split}}_{\text{short}}$), $\text{rcNN}^{\text{split}}_{\text{short}}$ correctly identified the verbal form *upacaryante iti*, where Transformer inserted the semantically dispreferred, but grammatically possible present participle *upacaryantaḥ iti*. Overall, none of the models shows a generally better or worse performance in these cases of disagreement.

5.4 Application to German Compounds

In order to test if the character based models generalize well to other languages with limited training resources, we applied $\text{rcNN}_{\text{short}}$ with split probabilities and the same settings as for SWS to the task of splitting German compounds. The current state of the art is set with a CRF operating on n-grams of characters (Ma et al., 2016). Table 6 shows that our model achieves an improvement of about 1% for recall and accuracy when trained with the training set of Ma et al. (2016) only. We sampled 20 examples for the three error classes "wrong split", "wrong faulty split" and "wrong non-split" (Ma et al., 2016, 78). While our model failed to detect splits for all 20 examples of the type "wrong non-split", the type "wrong split" contained 10 cases, where the split(s) proposed by the model make(s) good sense for us, but are not recorded in the test set (e.g. "Viermaster" 'four-master', "Viermaster" in test; already remarked by Ma et al. 2016). We observed a similar level of inconsistencies for the "wrong faulty split" type (8 instances), where, for example, our model analyzed "Bundes-tags-vize-präsident" 'vice president of the Federal Parliament', while the test set had "Bundes-tags-vizepräsident".

6 Conclusion

While the models discussed in this paper have produced clear performance gains when compared with previous research on SWS, we expect that future research will improve over our results, but it will be difficult to approach error-free performance. The reservation is due to the errors in the training data and especially the question of (non-)compositional readings of compounds, which seems to produce related levels of confusion for human annotators and ML models. While following this track of research, we would like to expand its scope to joint learning of splits, lexical and morphological annotations. Here, we expect that especially lexical and morphological analysis will benefit from a joint model. We hypothesize that CTC (Graves, 2012) trained as a co-task or segmental NNs (Lu et al., 2016) with a modified objective (including split probabilities) may be suitable for this task.

References

Theodor Aufrecht. 1891–1903. *Catalogus Catalogorum. An Alphabetical Register of Sanskrit Works and Authors*. Otto Harrassowitz, Leipzig.

Shubham Bhardwaj, Neelamadhav Gantayat, Rahul Garg, and Sumeet Agarwal. 2018. SandhiKosh: A benchmark corpus for evaluating Sanskrit Sandhi tools. In *Proceedings of the LREC*.

Christopher M. Bishop. 2000. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford.

Johannes Bjerva, Barbara Plank, and Johan Bos. 2016. Semantic tagging with deep residual networks. *arXiv preprint arXiv:1609.07053*.

Deng Cai and Hai Zhao. 2016. Neural word segmentation learning for Chinese. In *Proceedings of the 54th Annual Meeting of the ACL*, volume 1, pages 409–420.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015. Long Short-Term Memory neural networks for Chinese word segmentation. In *Proceedings of the 2015 Conference on EMNLP*, pages 1197–1206.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Harry Falk. 1993. *Schrift im alten Indien: Ein Forschungsbericht mit Anmerkungen*. Gunter Narr Verlag, Tübingen.

N. Gantayat, R. Aralikatte, N. Panwar, A. Sankaran, and S. Mani. 2018. Sanskrit sandhi splitting using $seq2(seq)^2$. *ArXiv e-prints*.

Pawan Goyal, Vipul Arora, and Laxmidhar Behera. 2009. Analysis of Sanskrit text: Parsing and semantic relations. In *Sanskrit Computational Linguistics*, pages 200–218. Springer.

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

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

Choochart Haruechaiyasak, Sarawoot Kongyoung, and Matthew Dailey. 2008. A comparative study on Thai word segmentation approaches. In *Proceedings of ECTI-CON*, volume 1, pages 125–128.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 770–778.

Oliver Hellwig. 2009. SanskritTagger, a stochastic lexical and POS tagger for Sanskrit. In *Sanskrit Computational Linguistics. First and Second International Symposia*, Lecture Notes in Artificial Intelligence, 5402, pages 266–277, Berlin. Springer Verlag.

Oliver Hellwig. 2015a. Morphological disambiguation of Classical Sanskrit. In *Systems and Frameworks for Computational Morphology*, pages 41–59, Cham. Springer.

Oliver Hellwig. 2015b. Using Recurrent Neural Networks for joint compound splitting and Sandhi resolution in Sanskrit. In *Proceedings of the 7th LTC*, pages 289–293.

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

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on EMNLP*, pages 1746–1751.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the ICLR*.

Yoshiaki Kitagawa and Mamoru Komachi. 2017. Long Short-Term Memory for Japanese word segmentation. *ArXiv e-prints*.

Philipp Koehn and Kevin Knight. 2003. Empirical methods for compound splitting. In *Proceedings of the Tenth Conference of EACL, Volume 1*, pages 187–193.

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 the COLING*, pages 494–504.

Amrith Krishna, Pavan Kumar Satuluri, and Pawan Goyal. 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.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

John J. Lowe. 2015. The syntax of Sanskrit compounds. *Language*, 91(3):71–115.

Liang Lu, Lingpeng Kong, Chris Dyer, Noah A Smith, and Steve Renals. 2016. Segmental recurrent neural networks for end-to-end speech recognition. *arXiv preprint arXiv:1603.00223*.

Jianqiang Ma, Verena Henrich, and Erhard Hinrichs. 2016. Letter sequence labeling for compound splitting. In *Proceedings of the 14th SIGMORPHON Workshop*, pages 76–81.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the ACL*, volume 1, pages 1064–1074.

Klaus Macherey, Andrew M Dai, David Talbot, Ashok C. Popat, and Franz Och. 2011. Language-independent compound splitting with morphological operations. In *Proceedings of the 49th Annual Meeting of the ACL*, pages 1395–1404.

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

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve Restricted Boltzmann Machines. In *Proceedings of the 27th ICML*, pages 807–814.

Abhiram Natarajan and Eugene Charniak. 2011. Sˆ3-statistical Sandhi splitting. In *IJCNLP*, pages 301–308.

Vikas Reddy, Amrith Krishna, Vishnu Dutt Sharma, Prateek Gupta, M R Vineeth, and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In *Proceedings of the LREC*.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean voice search. In *Proceedings of the ICASSP*.

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

Richard Sproat and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *Proceedings of the second SIGHAN Workshop on Chinese Language Processing*, pages 133–143.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. In *Proceedings of the Deep Learning Workshop at ICML*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

William Dwight Whitney. 1879. *A Sanskrit Grammar*. Breitkopf and Härtel, Leipzig.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.