NLP Techniques for Sanskrit

Sushant Dave

NLP Techniques for Sanskrit

Thesis submitted to
Indian Institute of Technology Delhi
for the award of the degree

of

Master of Science(Research)

by

Sushant Dave

under the guidance of

Dr. Brejesh Lall
(Department of Electrical Engineering)

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI

January 2020

Department of Electrical Engineering
Indian Institute of Technology Delhi
Hauz Khas, New Delhi-110016, INDIA

Certificate

This is to certify that this thesis titled **NLP Techniques for Sanskrit**, being submitted by **Sushant Dave** for the award of the degree of Master of Science (Research) in Electrical Engineering, is a record of bona-fide research work carried out by him under our supervision and guidance at the Department of Electrical Engineering, Indian Institute of Technology Delhi. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

Prof. Brejesh Lall
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Prof. Prathosh A.P.
Department of Electrical Engineering
Indian Institute of Technology, Delhi

- To my family -

Acknowledgment

I am grateful to my supervisor Dr. Brejesh Lall and co-supervisor Dr. Prathosh A.P. for giving me the opportunity to work with them. Their support and guidance have been valuable in carrying out the work described in this thesis.
I am specially grateful to Dr. Prathosh A.P. for motivating me to work on the area of NLP applications for Sanskrit. I feel fortunate to have had the opportunity to work with him.
I thank all the Professors at IIT Delhi who have always been very helpful to me and provided their valuable support and assistance whenever I needed it.
I'd like to thank my fellow student Arun Kumar Singh for having contributed to my research in multiple ways.
I also thank IIT Delhi for providing me with a wonderful environment for research and a chance to meet a lot of hardworking and highly motivated people. My learning here is an unforgettable experience.

Sushant Dave

Abstract

Applying NLP techniques to Sanskrit is an area where despite the advancements in recent years, a lot of work need to be done. Sanskrit had its unique grammar and syntax and pose unique challenges to NLP researchers. In this thesis, we have experimented with standard NLP techniques on Sanskrit and documented the results along with conclusions. We also present our analysis of the results. This thesis also documents the importance, current state and challenges in morphological analysis of Sanskrit Language. Based on the study, 2 specific Sanskrit Language analysis problems were identified that are still unaddressed or only partially addressed - *Sandhi* and *Pratyaya* word analysis, as described below.

This thesis 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.

Along with *Sandhi*, this thesis presents first benchmark corpus for Sanskrit *Pratyaya* (suffix) analysis along with neural network based approaches to process the formation and splitting of word-compounding using suffixes. *Pratyaya* are an important dimension of morphological analysis of Sanskrit texts. There have been Sanskrit Computational Linguistics tools for processing and analyzing Sanskrit

texts. So far, there has not been any work to standardize & validate these tools specifically for *Pratyaya* Analysis. In this work, we prepared a Sanskrit suffix benchmark called *PratyayaKosh* to evaluate the accuracy of tools. We also present our own neural approach results for *Pratyaya* analysis while evaluating the same on most prominent Sanskrit tools.

The work on *Sandhi* and *Pratyaya* has led to 2 research papers, with the paper on *Sandhi* being accepted for publication at a reputed conference.

Contents

List of Abbreviations

NLP	Natural Language Processing
ML	Machine Learning
NN	Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CRF	Conditional Random Fields
MLE	Maximum Likelihood Estimation
LSTM	Long Short Term Memory
TF	TensorFlow

List of Figures

List of Tables

CHAPTER 1

Introduction

Sanskrit is one of the oldest surviving languages today. The oldest known Sanskrit texts are estimated to be dated around 1500 BCE. 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 some regions of ancient India [1]. 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 5^{th} century BCE, a Sanskrit scholar named *pARini*(पाणिनि) [2] wrote a treatise on Sanskrit grammar named *azwADyAyI*(अष्टाध्यायी)[1], in which *pARini*(पाणिनि) formalized rules on linguistics, syntax and grammar for Sanskrit. *azwDyAyI*(पाणिनि)[2] is the oldest surviving text and the most comprehensive source of grammar on Sanskrit today. *azwADyAyI*(अष्टाध्यायी) literally means eight chapters and these eight chapters contain around 4000 *sutra* or rules in total. These rules completely define the Sanskrit language as it is known today. *azwADyAyI*(अष्टाध्यायी) 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 *sutra* by *pARini*(पाणिनि) as computer programs to analyze

[1] All Sanskrit words in this thesis are represented in SLP1 format

[2] `https://www.britannica.com/topic/Ashtadhyayi`

Sanskrit texts. With the recent advancements in high speed computing, Natural Language Processing and Machine Learning techniques like Deep Neural Networks and Recurrent Neural Networks, it should be very interesting to apply Machine Learning based NLP techniques to Sanskrit.

1.1 Motivation

As mentioned earlier, there is an immense volume of ancient Indian texts in Sanskrit which cover a vast range of topics. The analysis of these texts provide us with valuable knowledge of the times during which these texts were written and the people around then. Many Sanskrit scholars over the past many years have worked tirelessly to translate and decode these Sanskrit texts around us. However, this is an arduous tasks considering the vastness and complexity of Sanskrit texts around and the fact that there are very few computational tools available for Sanskrit analysis. Our work in this thesis tries to take a step in this direction by applying modern NLP techniques to Sanskrit and tries to identify and solve a few problems in this area.

1.2 Background

As a highly inflected, free order language, Sanskrit presents unique challenges in parsing and analyzing text. Many scholars have tried to codify the 4000 *sutra* by *pARini*(पाणिनि) but there are significant challenges in doing so. The rules laid out in *azwADyAyI*(अष्टाध्यायी) are sometimes complex and due to the recursive nature of these rules, sometimes ambiguities arise. While some rules like *Vibhakti* rules are well understood and can be easily codified, some other rules like *Pratyaya* formation rules are vast, sometimes ambiguous and difficult to formulate as a computer program. The challenges in analysis of Sanskrit have been analyzed in detail in this paper by Subhash Kak [3].

Considering the problems in analysis of Sanskrit using rule based programming, an alternative could be to use learning based techniques that rely on data to understand the rules of Sanskrit grammar. It will also be an interesting exercise to see if the Sanskrit rules that are difficult to codify, can be learnt by Machine Learning models by training them on large Sanskrit texts.

1.3 Research Objectives

This thesis objectives can be listed as follows:

- Applying basic learning based NLP techniques to Sanskrit texts and analyze the results.
- Study of current literature on NLP for Sanskrit and identifying the challenges and shortcomings.
- Solving a few of the problems identified in order to contribute to the field of Computational linguistics for Sanskrit.
- Share all the data and code used in this project to other researchers for further research in this area.

1.4 Contributions

The main contributions of this thesis can be summarized as follows:

- **Analysis of standard NLP techniques for Sanskrit:** This thesis analyses the application of standard NLP techniques like word-vector creation, N-gram language modelling and data driven translation on Sanskrit. The report establishes that these techniques have limited effectiveness and touches upon the reason for this.
- **State-of-the-art model for *Sandhi* and *Sandhi-Vicceda*:** This thesis introduces an RNN based model for *Sandhi* and *Sandhi-Vicceda* that does not use any external lexical resource and outperforms existing models despite being simpler and quicker to train.
- ***Pratyaya* dataset and RNN model for their formation and split:** This thesis introduces a dataset for *Pratyaya* words along with an RNN based model for their formation and split that outperforms existing tools available for the same.

1.5 Thesis Organization

The rest of the thesis is structured as follows.

- Chapter 2 focuses on creating word vectors for Sanskrit using large Sanskrit corpus and the analysis of the properties of resultant word vectors.
- Chapter 3 describes the experimentation to create Language models for Sanskrit and Machine Translation from Sanskrit to English. The chapter briefly discusses the results and need for Morphological Analyzer for Sanskrit. The chapter also discusses the current state of research on Sanskrit morphological analyzers.
- Chapter 4 introduces *Sandhi* in Sanskrit, challenges in analyzing *Sandhi* and a novel method of formulating and solving the problem of *Sandhi* as a sequence to sequence prediction task, using modern deep learning techniques.
- Chapter 5 introduces *Pratyaya-Kosh*, a benchmark corpus to help researchers new to Sanskrit in building AI based Morphological Analyzer for Sanskrit derivative nouns along with a novel RNN based approach for learning derivative noun formation and split.
- Chapter 6 Describes the conclusions from this thesis and path ahead for future research and improvements in this area.

CHAPTER 2

Word Vectors for Sanskrit

2.1 Introduction

Word Vector representation, also called word embeddings, has had a very significant impact in the area of NLP. It allows us to easily analyze and visualize relationship between words, sentences and documents. Word vectors have allowed massive improvements in the field of Named entity recognition, language translation, semantic text analysis and numerous other problems in NLP. Word vectors, also referred to as word embeddings have become integral to Neural Networks, SVM and other machine learning techniques for NLP since these methods require some numerical representation for words.

Word vectors are simply vectors of numbers that represent words in any given language. There is a different word vector for each word. The relationship between these vectors can capture the relationship between the corresponding words in a manner that is not possible with traditional token based representations. In traditional methods of word representations, words are represented as unique tokens and a one-hot vector is assigned to each token. The problem with this approach is that it does not provide us any insight into the nature of relationships between words. For example, if we take the words 'prison' and 'jail', using one-hot encoded

vectors for their representations does not provide us any indication that these 2 words may be synonyms. The typical measures of distance between 2 vectors like Euclidean distance or Cosine similarity is exactly the same between any 2 vectors under the one hot encoding scheme. The one-hot encoding can suffice for simple tasks like spam mail classifier, but for complex tasks where the semantic and syntactic relationship between words across word collections need to be considered, word vector representations become necessary.

Word vectors have been discussed in the research area of distributional semantics as early as 1950s. The underlying idea is to quantify and categorize semantic similarities between words based on their distribution in large corpus of text. More simply put, the contextual information of a word/phrase alone is sufficient to represent that word/phrase. The "distributional hypothesis" by Firth famously states that "A word is characterized by the company it keeps"[4]. The earliest works on word vector representation represented the word distributional data in sparse vector space of very high dimension. These dimensions were then reduced by techniques like "Singular Value Decomposition". This approach to generate contextual features led to development of topic modelling techniques like Latent Semantic Analysis[5]. Modern methods use architectures like Convolutional Neural Networks for generating the contextual features. The difference between topic modelling and word distribution models is the type of contextual information they use. Topic modelling use documents as contexts, while Neural language models and distributional semantic models instead use words as contexts. These different contextual representations capture different types of semantic similarity. The document based models capture semantic relatedness (e.g. "boat" - "water") while the word-based models capture semantic similarity (e.g. "boat" – "ship")[6].

A very efficient word vector representation was introduced by Mikolov at Google in 2013[7],[8]. It is popularly referred to as Word2Vec and has become very popular among researchers on NLP. It can use either of two models "continuous bag of words" (CBOW) or continuous skip-gram to produce word vector representations. The CBOW model takes the context of each word as input and tries to predict the word from that context, while the skip-gram model takes a word and tries to predict the context, or surrounding words from that word. The Word2Vec model tries to model either of the CBOW or skip-gram for all the

words on a large corpus of text. In the modelling, the words are given a one hot representation and the word vector learning is a side effect of the aforementioned process. The original Word2Vec model was introduced for English, but it has been successfully applied to many different languages. It has been successfully applied in NLP applications for many languages because of the very interesting and useful features (described in sections ahead). It therefore makes sense to try and apply the same word vector representation to Sanskrit words.

2.2 Related Work

There is very little work in public domain on word vector representation for Sanskrit. The only research paper that is available is a paper by Ishank et. al.[9]. The paper uses Mikolov's method to create word embeddings for Sanskrit words and does analysis of word groupings and additive compositional property of Word2Vec embeddings.

2.3 Proposed Work

In this experiment, I used the Mikolov's method to generate word embeddings for Sanskrit. The Sanskrit text was taken from various sources on web containing a mixture of old and modern Sanskrit texts. There are overall 532 Sanskrit documents used, the major ones being the following:

1. Digital Corpus of Sanskrit [10]
2. Bhagavatapuranam
3. Garudapuranam
4. Charakasamhita
5. Bhashaparicheda
6. Vyutpattivada
7. Geethasankarbhshyam

8. Saabdataringini
9. Arthashastra
10. Mahabharata

Other than the above mentioned documents, a lot of other texts were used that were collected from multiple sources on the internet, major ones being corpus provided by University of Hyderabad [11] and Sanskrit Wikipedia [12]. Among these texts, there are over 500,000 unique tokens or words. The text in the corpus needed a bit of pre-processing and cleaning up. There were many non-Sanskrit words in some documents while many Sanskrit texts contained multiple special characters. Some Sanskrit texts used the "purna-viram" character used in Devanagari script, at sentence end and some texts used the "full-stop" from English at sentence end. All the clean up and dividing the text in sentences was done using a Python script that converted all the text in lists of sentences, each of which is a list of words.

Gensim [13] provides Word2Vec implementation which is easy to use and fairly popular among researchers. Gensim library was used with Python3 in my experiment to generate word embeddings for the Sanskrit corpus. Gensim library provides the function Word2Vec(), which takes in the following arguments:

1. Word vector size (number of real number used to represent each word).
2. Minimum number of times a word occurs in corpus for it to be considered.
3. The window size used for a bag-of-words used in word prediction.

Once the word embeddings are generated we need to visualize them. Tensorflow provides a great tool to visualize word embeddings [14]. It also allows to find words closest to a particular word based on either cosine similarity or Euclidean distance in word vector space. It just expects the word and their word vectors to be tsv (tab separated values) format. A python script was used to convert the output of Gensim word-vector generator to tsv format and the results were analyzed in the TensorFlow tool.

2.3 Proposed Work

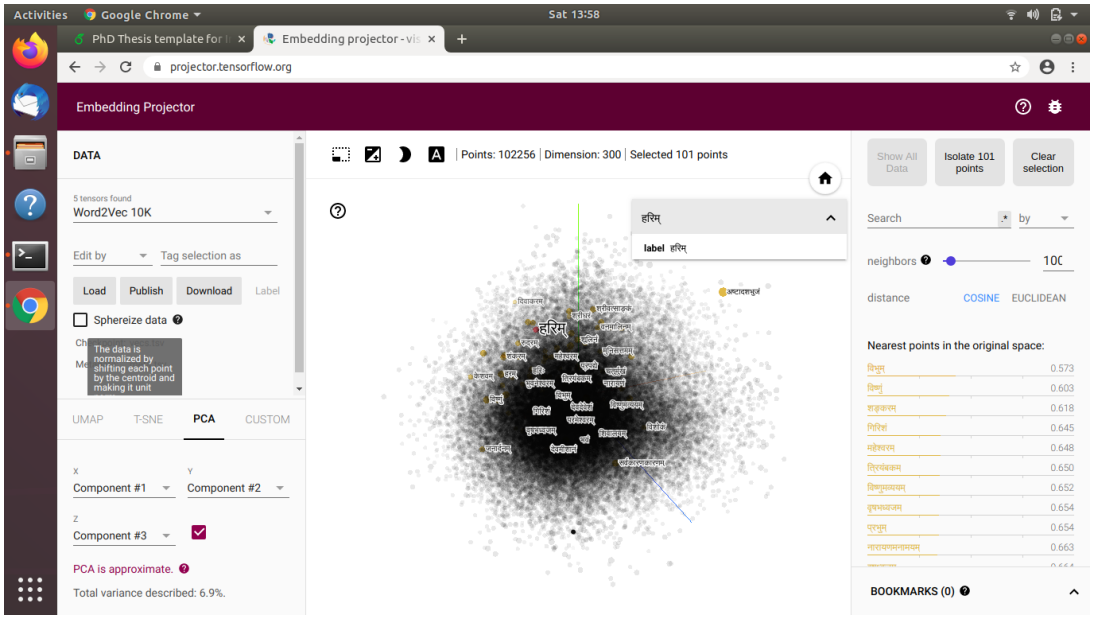

Figure 2.1: TensorFlow projection

To start with, the size of word embeddings 100, 200 and 300 were selected and the results of each compared. With 300 as embedding size, best results were achieved. Beyond this, there didn't seem to be any significant improvement in results, hence 300 was chosen as embedding size. The minimum occurrence count for every word was chosen as 5 and window size was chosen as 5. The restriction on minimum number of occurrences of each word reduced the token count to 102,265 (from over 1,000,000 unique tokens). My analysis is limited to simple inflected words. No compound words were considered in analysis. Most complex compound words are rare and unlikely to occurs more than 5 times in corpus.

The image 2.1 shows the word vector projection in Tensorflow visualizer. The blue colored cluster of tokens in the centre of image is the word vector projected in 3-D space. The user can click on any word and check the nearest words based on either the cosine similarity or Euclidean distance. The tables 2.1 and 2.2 show the neighboring words for some nouns and verbs, respectively.

Table 2.1: Nouns: Neighboring words

No.	Word(Meaning)	Neighboring words(meaning)
1	देव(God)	सार्थवाह(group of pilgrims), वीर(brave), सूत(charioteer), त्वम्(You), माधव(Krishna)
2	रामः(Rama)	लक्ष्मणः(Lakshamana), रामो(Rama), राघवः(Rama), रावणः(Ravana), सः(He)
3	भावना(feeling)	चिन्ता(worry), जिज्ञासा(curiosity), नूतना(fresh), निर्दिष्टा(specified), कल्पना(fiction, creation)
4	रहस्यं(secret)	रहस्यम्(secret), गुह्यं(hidden, mystical), मतं(belief, opinion), कृत्यं(deed, task), कार्यं(act, deed)
5	कार्यं(act, deed)	कर्तव्यं(charge, duty), कार्यम्, कर्म(act, deed), कृतं(completed, carried out), प्राप्तं(achieved, received)
6	साधु(saint, correct)	असाधु(unholy, wrong), साध्विति(virtue), वासाधु(derivative of original word), साधुत्वात्(derivative of original word), साध्व्(derivative of original word)
7	ध्रुवम्(stable, fixed)	नात्र(sage, praise), क्षणात्(instantly), संशयः(doubt, risk), जाम्बूनदसमप्रभम्(having splendor of Jambu river), तत्तारं(unknown)
8	नरः(man)	मानवः(man), नरो(man), मर्त्यः(mortal), यः(who), जितक्रोधो(one who conquered anger)
9	पीत(yellow)	शुक्ल(white), नीलः(blue), हरित(green), एवमग्रेपि(unknown), ज्ञानोत्तरं(after receiving knowledge)
10	प्रकाशं(brightness)	अप्रकाशं(dark, hidden), वैचित्र्यं(despair, strangeness), वारयेयुः(keep away), उपकारं(benevolence, patronization), वादित्रं(musical instrument)
11	शुक्रः(Venus, semen)	पीयूषः(thick fluid, nectar), शनैश्चरो(slow moving, Saturn), शनिः(Saturn), सोमपो(related to moon), सहस्रधारः(flowing in thousand streams)
12	मृत्यु(death)	अर्थनाशं(loss of money), ससर्पे(having serpents), रम्यताम्(pleasantness), क्लीबस्य(weak, emasculated), अमात्यराक्षसस्य(Last ruler of Nanda dynasty who died early)
13	ग्रीष्मे(summer)	वसन्ते(spring), शरदि(winter), शीते(cold), वर्षासु(rainy), पञ्चाग्निमध्यस्थो(between meditation among five fires)
14	निद्रा(sleep)	क्लमः(fatigue), मूर्छा(stupor, faint), पिपासा(thirst), तन्द्रा(fatigue), मूर्च्छा(stupor, faint)
15	वैद्यः(physician)	ब्राह्मणः(priest, divine), भिषक्(beggar), करोत्य्(does), साधुर्(sage), एलिसरः(unknown)

Table 2.2: Verbs: Neighboring words

No.	Word(Meaning of root)	Neighboring words(meaning of root)
1	भवति(happen, become)	स्यात्(perhaps), भवेत्(should be), वा(or, is it?), एव(only), न(no)
2	गच्छति(goes)	याति(goes), प्रयाति(make tracks, depart), विन्दति(find), गच्छतीति(go), प्राप्नोति(reach)
3	दास्यति(give)	दत्तवान्(give), करिष्यति(do), विधास्यति(provide), दास्यते(give), प्रदास्यति(give)
4	पिबेत्(drink)	पिबेद्(drink), पिबेत्सदा(drink), पाययेत्(give to drink), पिबेन्नरः(drink), भक्षयेत्सदा(eat)
5	करोति(do)	कुरुते(do), कुर्वन्ति(do), प्राप्नोति(reach), कुर्यात्(do), क्रियते(do)
6	कुप्यति(get angry)	कफो(phlegm), हृष्यति(be glad), रूक्षैः(rough, coarse), क्लेदयन्त्यापो(moisten), भृत्ये(care of)
7	तिष्ठति(sit)	तिष्ठेत्(sit), स्थास्यति(remain), संस्थितः(still), भ्रमति(move), वर्तते(to be, to exist)
8	नश्यति(lose, perish)	जायते(take birth), विनश्यति(lose, vanish), प्रणश्यति(disappear, vanish), नश्यते(lose, perish), शाम्यति(discontinue)
9	प्रयतेत(make tracks, depart)	तितिक्षुर्(patient, enduring), भिषक्(beggar), कर्तुश्च(unknown), कोविदो(skilled, clever), आर्द्रकस्य(moist, wet)
10	क्रियते(do)	करोति(do), शक्यते(make executable), युज्यते(be fit for, be united), गृह्यते(be bound,controlled), सिध्यति(be perfect, hit a mark)

The word vector representation displays a property of having directions of semantic and syntactic meaning that can be exposed through simple operations on word vectors. For example, we can have the following semantic relationship expressed mathematically:

```
vector(Paris) - vector(France) + vector(India) = vector(Delhi)
```

We can express similar relations based on singular-plural or male-female semantic relation:

```
vector(Kings) - vector(King) + vector(man) = vector(men)
vector(King) - vector(Queen) + vector(woman) = vector(man)
```

In this experiment, I tested the compositional property over linga (gender) and vacana (count) for Sanskrit words. The compositional property is expected to change the female word to male word in table 2.3 and singular word to plural word in table 2.4.

Table 2.3: Compositional addition: Gender based

No.	Word1	Word2	Word3	Word1 - Word2 + word3
1	स्त्री(woman)	सा(she)	सः(he)	कश्चन(some male)
2	कन्या(girl)	सा(she)	सः(he)	कश्चन(some male)
3	उमा(Parvati)	सा(she)	सः(he)	एषः(this male)
4	किशोरी(girl)	सा(she)	सः(he)	बेन्झियोनः(A young male character)
5	जननी(mother)	सा(she)	सः(he)	पितृव्यः(father's brother)
6	पत्नी(wife)	सा(she)	सः(he)	कश्चन(some male)
7	पुत्री(daughter)	सा(she)	सः(he)	स्पन्दनदासः(A young male character)
8	बालिका(girl)	सा(she)	सः(he)	श्रावणः(July-August)
9	लक्ष्मी(fortune, Goddess)	सा(she)	सः(he)	एषः(thismale)
10	राक्षसी(female demon)	सा(she)	सः(he)	गौरवः(dignity)

Table 2.4: Compositional addition: Count based

No.	Word1	Word2	Word3	Word1 - Word2 + word3
1	भवति(he becomes/happens)	युवकः(lad)	युवकाः(lads)	भवन्ति(they become/happen)
2	राजा(king)	युवकः(lad)	युवकाः(lads)	शिक्षकाः(teachers)
3	सा(she)	युवकः(lad)	युवकाः(lads)	ये(these)
4	वदसि(says, talks)	युवकः(lad)	युवकाः(lads)	क्लीबा(weak, emasculatedplural)
5	पुरुषः(man)	युवकः(lad)	युवकाः(lads)	सामग्र्या(stocks, reserves)
6	रामः(Rama, pleasant)	युवकः(lad)	युवकाः(lads)	शिक्षकाः(teachers)
7	किम्(who)	युवकः(lad)	युवकाः(lads)	वयम्(we)
8	आसीत्(he was)	युवकः(lad)	युवकाः(lads)	आसन्(they were)
9	पण्डितः(wise, scholar)	युवकः(lad)	युवकाः(lads)	मर्मैकदेशे(unknown)
10	अहम्(I)	युवकः(lad)	युवकाः(lads)	वयम्(we)

2.4 Results

As mentioned above, the tables 2.1 and 2.2 show the neighboring words for some nouns and verbs, respectively and the tables 2.3 and 2.4 show the compositional property for gender based and count based relationship respectively.

Regarding the neighboring words for nouns in table 2.1, we can make the following observations for each entry in the table (in order):

1. The neighboring words are close to the original word(God) but can hardly be called synonyms or can replace the original word in any context.

2. All the neighboring words except one, are from the same epic and the last word is a common pronoun.

3. Three of the words are types of feelings, so there is some contextual relation there, but its not too obvious.

4. Only first two words are close in meaning to the first word. Rest are unrelated.
5. All the words are related to the original word in meaning
6. All the words are related to the original word in meaning. In fact the words contain the same root and one word is the opposite of original word.
7. None of the neighboring words are similar to the original word.
8. All the words except the last one are the synonyms of the original word.
9. The first three words are colors like the original word. The last two words don't seem to have any similarity with original word.
10. Except for the first word, which is an antonym, the neighboring words don't have any obvious connection to the original word.
11. The first neighboring word has similarity with one meaning of original word (fluid) and the next three words are similar in different context (celestial body). The last word is unrelated.
12. Seeing the similarity with original word requires a bit of a stretch. Money loss and having serpents can lead to death. Being emasculated can be compared to death as per scriptural norms. The last word here is the name of an evil king who died prematurely with his whole family.
13. Original word is a season name and first four neighbors are seasons. Last neighbor is unrelated.
14. Other than one neighbor, all words are related to original word i.e. sleep.
15. None of the words are related to the original word in any meaningful sense.

Regarding the neighboring words for nouns in table 2.2, we can make the following observations for each entry in the table (in order):

1. Other than one word which shares root with original word, none of the words have any relation to original word

2. All the neighbors are verbs and either share the root with original word or have root with similar meaning as root of original word.

3. All the neighbors are verbs and either share the root with original word or have root with similar meaning as root of original word.

4. All the neighbors are verbs and either share the root with original word or have root with similar meaning as root of original word.

5. All the neighbors are verbs and either share the root with original word or have root with similar meaning as root of original word.

6. There seems to be no similarity between original word and neighboring words.

7. All the neighbors are verbs and either share the root with original word or have root with similar meaning as root of original word.

8. Three out of five words share the root with original word. Of the other two, one word is unrelated and one is opposite in meaning.

9. There seems to be no similarity between original word and neighboring words.

10. Other than the closest neighbor which shares root with original word, no other word has any contextual or semantic relation with original word.

Regarding the word clustering, there are a few observations:

1. The Euclidean distance based similarity and cosine similarity produce almost the same results.

2. While the results for nouns are not as good as seen in languages like English, there does seem to be some semantic relationship in some cases.

3. For the verbs, there appears to be better semantic closeness among the words on group.

4. The nearest neighbors of nouns are nouns and verbs are verbs.

Regarding the gender based compositional property for words in table 2.3, we can make the following observations:

1. The output is not what is expected in any of the cases.
2. In 4 and 5, there seems to be a closeness to the expected output.
3. In each case the output word is male although not the exact one.

Regarding the gender based compositional property for words in table 2.4, we can make the following observations:

1. For 1, 8 and 10, the output is exactly as expected.
2. For 3 an 7, there is a bit of similarity between expected and actual output.
3. For remaining test cases, the output is not as per expectation.
4. In every case, the output word is plural even if it is not exactly as expected.

It is to be noted that there is some scope for ambiguity in analysing some of the above results. In absence of any objective scoring methodology, some subjective bias can cause the results to be interpreted differently. Keeping this in mind, 100 words each belonging to both categories i.e. nouns and verbs were selected for analysis of word neighbors. These words along with their four neighboring words and their meanings in English were shared with 5 participants who are not familiar with Sanskrit and they were asked if the words belong to similar context in their opinion (i.e. have similar meaning or related semantics). The users had to provide a simple yes or no for each word. The results are shown in table 2.5.

Table 2.5: Neighboring words: User results

Word Type/User response	Noun	Verb
Yes	53	81
No	47	19

In case of verbs, very often the neighbors of the chosen word were observed to be different form of the same verb root as the original word. For nouns, the score

is pretty low if compared to similar words in English where such groupings work extremely well. It was also observed that for complex or compound words, the groupings don't work well. Better results were observed for simple words. The results for nearest words by cosine similarity were almost identical to the nearest words selected using Euclidean distance.

2.5 Summary

From the results above, it seems clear that while word vectors for Sanskrit work to some extent, they do not work quite as well as the word vectors in English. After analyzing some Sanskrit text and comparing their English translations, it seems that the following reasons might explain the relatively poor experience of Sanskrit word vectors compared to English word vectors:

1. Sanskrit is a free order language i.e. it allows the user to write words of a sentence in any order without losing the meaning of sentence which is not possible in English. For example consider the English sentence:

   dog bites man

   If we swap the order of first and last words, the meaning changes entirely. However, for the same sentence in Sanskrit:

   कुक्कुरः पुरुषम् दंशति

   The above 3 words can be written in any order (total 3! = 6 possible sentences) and the meaning wouldn't change. This is because the words for dog and man in Sanskrit are inflected to convey their exact contextual meaning, irrespective of their position in sentence. This means that Sanskrit sentences lacks the structure that is present in English, that may be a reason for low performance of Sanskrit word vectors.

2. As explained in above point, all words in Sanskrit except a set of words called Avyaya (indeclinables) are inflected. Inflection is seen in languages like Hindi and English as well, but Sanskrit is very highly inflected. A noun

root in Sanskrit has 3 linga (gender), 3 vacana (number) and 8 vibhakti (cases), so a noun root can have 72 different forms. A verb root in Sanskrit can have 10 different Gana (semantic classes), 10 different lakara (tense), 3 purusha (person) and 3 vacana (number) which means there are 900 different inflected forms for a verb root. In addition, some of these roots can be combined with different suffixes or prefixes to create to create new roots with their own forms and this means that there can be even more words with similar contextual meaning. This can be a problem because every word will be initially assigned a different token (or vector) before the training starts in Word2Vec model. This will make the learning of word vectors difficult. This problem of inflected languages with Word2Vec has been noted by other researchers as well [15].

3. Sanskrit allows a user to be very terse with words used and this is a direct outcome of the fact that it is highly inflectional. For example the following English sentence:

   I will be going

   can be represented by the following single word in Sanskrit:

   गमिष्यामि

   The information about tense, person and number is embedded in a single word. This means that the strict need for pronouns and prepositions can be eliminated in many cases and their usage is left to the user's discretion. This leads to different sentence formations for the same content and makes the learning of word distributions based on context difficult.

4. Sanskrit allows for Sandhi (word compounding) thereby allowing the users to create new words on the fly. This is a huge problem, specially in the ancient texts which make extensive use of Sandhi. Since new words can be created as per the user's choice, it is possible to create rarely used new words from commonly used words. This makes it difficult for the model to learn the context since the model sees these words as entirely new words.

2.5 Summary

Therefore, the conclusion from my work on word vectors representation for Sanskrit is the following:

1. The Sanskrit word vectors attained by Mikolov's method may not be adequate for many NLP applications on Sanskrit.

2. The first step towards word vector creation for Sanskrit must be to develop a morphological analyzer that can resolve the Sandhi words into its constituent words and also find the root and inflections of each word, thus allowing the model to know when to treat the words as same and not different thus reducing the number of tokens.

The rest of this thesis is dedicated to contributing towards morphological analysis of Sanskrit words by solving the problem of *Sandhi* spilt and understanding the root and inflection of a set of Sanskrit words known as *Kridanta* and *Taddhita* words.

CHAPTER 3

Problems in applying standard NLP techniques to Sanskrit and literature review on morphological analysis

3.1 Introduction

This chapter discusses two standard NLP techniques namely language modelling and translation using RNN on Sanskrit language and describes my experiments on these two techniques and their results. At the end of this chapter, I discuss the morphological analysis of Sanskrit words, its current state as per the research papers available on this topic and the main challenges in this area. This is made necessary because of the results achieved by aforementioned experiments.

3.1.1 Language modeling

Statistical Language Modeling, also commonly referred to as Language Modeling, is the development of probabilistic models that are able to predict the next word

in the sequence given the words that precede it. A statistical language model is a probability distribution over sequences of words. Given such a sequence, say of length m, it assigns a probability

$$P(W) = P(w_1, w_2, \ldots, w_m)$$

to the whole sequence. Such a model can then, be used to compute the probability of an upcoming word as per the following formula.

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) = \frac{P(w_1, w_2, \ldots, w_n)}{P(w_1, w_2, \ldots, w_{n-1})}$$

Thus, a model that assigns a probability to a sentence in a language can also be used to predict a word in a sequence given preceding words. Language models are useful in:

1. Speech recognition system by discriminating between similar sounding words.
2. In statistical machine translation, a language model characterizes the target language.
3. For selecting alternatives in summarization, generation.
4. Text classification (style, reading level, language, topic etc.)

There are primarily 2 types of language models:

1. Statistical Language Models: These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM) and certain linguistic rules to learn the probability distribution of words.
2. Neural Language Models: These models are based on modern neural networks and widely used due to their superior performance compared to statistical modelling techniques. They use different kinds of Neural Networks to model language.

3.1.2 N-Gram language model

Using the chain rule of probability, the probability of a sequence of m words can be calculated as follows:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) * P(w_2|w_1) * P(w_3|w_2, w_1) * \ldots * P(w_n|w_{n-1}, w_{n-2}, \ldots, w_1)$$

In practice, the above model is difficult to calculate for the following reasons:

1. There are too many combinations to consider.
2. There will not be enough examples to estimate the correct probabilities.

To get past the above 2 problems, Markov assumption is used. A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state i.e. given the present, the future does not depend on the past [16]. Markov assumption describes a model where the Markov property is assumed to hold. Markov assumption can be generalized to a fixed number of previous words that impact the probability of a following word. In an N-gram model, we assume that the probability of a new word depends only on the previous n-1 words. Under this assumption, the following equation holds:

$$P(w_m|w_1, w_2, \ldots, w_{m-1}) = P(w_m|w_{m-1}, w_{m-2}, \ldots, w_{m-n+1})$$

For n = 1, the model is called Unigram and the equation simplifies to:

$$P(w_m|w_1, w_2, \ldots, w_{m-1}) = P(w_m)$$

Similarly, for n = 2 and n = 3, the model is called Bigram and Trigram respectively and the equations for each simplify to the following:

$$P(w_m|w_1, w_2, \ldots, w_{m-1}) = P(w_m|w_{m-1})$$

$$P(w_m|w_1, w_2, \ldots, w_{m-1}) = P(w_m|w_{m-1}, w_{m-2})$$

The estimation for the N-gram model from a large corpus can be done by the following equation:

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) \approx \frac{Count(w_1, w_2, \ldots, w_n)}{Count(w_1, w_2, \ldots, w_{n-1})}$$

The above count based estimation suffers from the following problems:

1. If a particular sequence used in estimation of another sequence does not exist it will cause a problem. For example, if the numerator term in RHS expression of above equation is 0, the probability will be estimated to 0. It would mean ruling out the occurrence of LHS expression altogether which may not be correct.

2. If the denominator in above equation's RHS expression is 0, the LHS expression can't be evaluated.

The problem of not being able to find a particular sequence in the language corpus is referred to as *sparsity* problem. There are multiple ways to deal with *sparsity* problems. The most commonly used way is smoothing, which involves assigning a small probability to all possible sequences. Another way is to use a shorter N-gram, referred to as *backoff*.

In recent times Neural Networks have been used to model N-gram probabilities. With Neural Networks, there is no *sparsity* problem and there is no need to store all observed N-grams. RNNs are specially useful in this case since they can model dependencies of multiple length sequences.

3.1.3 Evaluation of Language Models

A very commonly used measure of evaluation for language models is *Perplexity*. *Perplexity* is defined as the inverse probability of the test set, normalized by the number of words:

$$PP(W) = \sqrt[M]{\frac{1}{P(w_1, w_2, \ldots, w_M)}}$$

Using the chain rule for N-gram model, the above equation can be written as:

$$PP(W) = \sqrt[M]{\frac{1}{\prod_{i=1}^{M} P(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-(N-1)})}}$$

For Bigram, the expression for *Perplexity* simplifies to:

$$PP(W) = \sqrt[M]{\frac{1}{\prod_{i=1}^{M} P(w_i|w_{i-1})}}$$

Perplexity is an inverse indicator, i.e. the lower its value, the better the language model.

Perplexity for the test can also be estimated from *Cross Entropy* between the estimated probability distribution on training set and the probability distribution of the test set.

$$PP(W) = K^{Entropy(W)}$$

Where K is the base over which entropy for word sequence W was calculated. In information theory, K is generally 2 and represents the average number of bits requires to encode sequence W.

3.1.4 Neural Machine Translation

Neural Networks, especially the RNNs have made a huge impact in the area of automatic language translation. Neural translation has provided better results over traditional statistical methods for language translation, both in terms of performance and memory requirements. A very popular RNN model for neural translation is the *Seq2Seq* model introduced by Sutskever et. al. [17]. A *Seq2Seq* model is an end-to-end model made up of two recurrent neural networks:

1. An encoder, which takes the input sequence as input and encodes it into a fixed-size context vector.

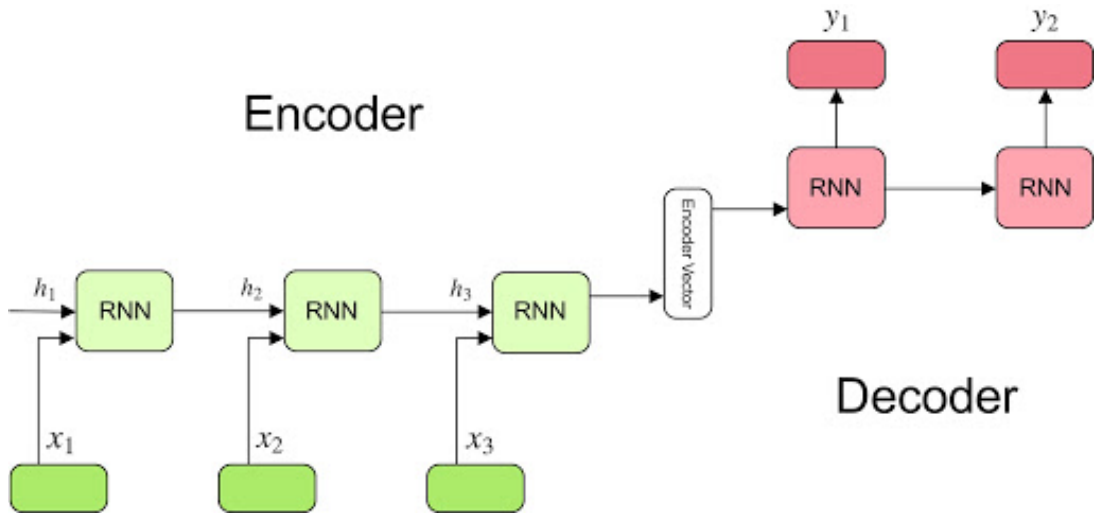

Figure 3.1: Seq2Seq Architecture

2. A decoder, which uses the above context vector as input to generate an output sequence.

Because of this architecture, *Seq2Seq* model is also referred to as encoder-decoder model. The architecture for is shown in the image 3.1, where the input sequence of X produce output sequence of Y. It is to be noted that the decoder operation starts once the encoder operation is complete. *Seq2Seq* has been used with considerable success for language translation for multiple languages.

In this chapter, I describe my experiments with Sanskrit language modelling and translation.

3.2 Related Work

At the time of writing this thesis, there was no work in public domain for either language modelling of Sanskrit or for translation of Sanskrit text to any other language.

3.3 Proposed Work

3.3.1 Language modelling for Sanskrit

The N-gram modelling for Sanskrit requires a large corpus. The corpus that was used in word vector experiments was taken for N-gram modelling as well. Pre-processing was done on the data to remove sentences with words that occurred less than 5 times in the corpus. The dataset consisted of 1,87,850 sentences containing 12,14,462 words out of which there were over 1,00,000 unique words. This dataset was divided into 2 sets: training set containing 1,50,000 sentences and a test set containing 37,850 sentences. After extracting N-gram sequences for N = (1, 2, 3, 4), there were over 8,00,000 sequences available for every value of N.

The count based model was used for N-gram models for N = 1, 2, 3 and 4. To resolve the *sparcity* problem, Laplace smoothing was applied. Under Laplace smoothing, every possible N-gram sequence is assigned a minimum count of 1 before counting. This ensures that the probability of a sequence is always non-zero. The implementation was done in python using NLTK library [18].

The RNN based language model takes one hot encoded Sanskrit words and passes them to an embedding layer that converts them to a vector of 8 float values. The embedding layer is a fully connected layer and its parameters are learnt with other parameters of the model. The new word representations are then passed to an LSTM cell of output size 8. This output is then passed to another fully connected layer with output the size of the vocabulary and softmax activation is applied to the last layer. This is expected to be the one hot representation of output word. In this experiment I tried RNN models for Bigram and Trigram. The code was implemented in Python using Keras library running Tensorflow as backend. Categorical Cross entropy loss and Adam optimizer were used in training the model. This model was trained on the same data that was used for count based technique.

For the translator, the dataset consisted of two documents *Ramayana* and Bible. These documents were taken because these are the only 2 sufficiently large documents available in public domain, for which there is a line by line Sanskrit to English translation available. The Sanskrit and English versions of *Ramayana*

is available here [19]. The Sanskrit and English versions of Bible are available here [20] and here [21], respectively. The Sanskrit verses were taken as input and English sentences were the expected output of the model. Overall, there are around 40,000 sentence pairs to train the translator.

A Seq2Seq model was used for implementing the translator. A dictionary was created for the input words and each word was converted to a one hot vector. The length of every input and output sentence was calculated as input and output size. A special character was designated for padding and every sentence was padded to make the length of all sentences equal. The input one hot vectors were input to an embedding layer, the output of whom was passed to the Encoder RNN of the Seq2Seq model. Softmax loss and Adam optimizer were used to train the translator model.

3.4 Results

For the count based N-gram modelling of the Sanskrit text perplexity scores were calculated on the test set. The results are shown in table 3.1:

Table 3.1: Count based N-gram: Perplexity scores

N-Gram	Perplexity score
1-gram	25189
2-gram	9594
3-gram	4723
4-gram	2700

To put above results in perspective, similar count based N-gram models for English have achieved *perplexity* scores of 962, 170 and 109 for Unigram, Bigram and Trigram respectively [22]. The scores for Sanskrit in comparison, indicate that the model does a poor job of modelling the probability distribution of Sanskrit words in the given Corpus.

For the RNN based N-gram modelling, training was tried for Bigram and Trigram models. Of the total training samples (>7,00,000), 5% were set as validation data and the model was trained on the remaining data. During the training, the

Figure 3.2: RNN Trigram training: Validation Loss(green) vs. Training Loss(red)

training loss decreased continuously, while the validation loss stayed more or less at similar levels thus indicating that the model was not learning anything. The training and validation loss for the RNN based Trigram models are shown in the image 3.2. It is clear that the model does not learn anything useful and the over-fitting starts from the very beginning, leading to a model that is not fit to be used in Sanskrit language modelling task.

A very similar trend was noticed in the translation task, where the training loss kept on decreasing but the validation loss stops decreasing after a point. If the model is trained till training loss becomes very small (< 0.01), the validation loss stays very high (> 5). Such a model will accurately translate any sentence in the training set (except for some extra words at sentence end in some cases), but for any sentence outside the training set, the model outputs a sequence of prepositions like 'and' or 'or', thus indicating that the best approach according to the model, to decrease loss is to output a list of most commonly occurring words.

It is noteworthy that RNN for N-gram modelling and Seq2Seq for translation are widely used for multiple languages and produce good results over similar sized datasets.

3.5 Summary

3.5.1 Need for morphological analysis in Sanskrit

It is obvious from the results of the experiments in this chapter (language modelling and translation) and the previous chapter (word vector creation), that the standard NLP techniques do not work very well on Sanskrit. The lack of a proper structure, very high inflection of words and vast usage of *Sandhi* ensures that a new paradigm is required to be able to do the kind of tasks in Sanskrit, that standard NLP methods can achieve for other languages.

A lot more simplification in Sanskrit can be achieved if there is a morphological analyzer available for Sanskrit that has good accuracy and coverage. As mentioned earlier in Section 2.5, there are thousands of forms of a single verb or a noun. This makes it very difficult for any NLP model to understand the relationship between words. A good morphological analyzer can allow NLP model to understand which words have similar meanings and under what context. In addition, the relationship between words in Sanskrit is indicated by the inflectional form of the words and not by prepositions and conjunctions. This further accentuates the need of a good Morphological analyzer for Sanskrit.

3.5.2 Current status and challenges present in Morphological analysis for Sanskrit

A fair bit of work has been done in the area of Morphological analysis of Sanskrit. Akshar Bharati et. al. have pointed out the major challenges in morphological analysis of Sanskrit [23]. As per their work, the main tasks in this area are:

1. *Sandhi* Analysis: *Sandhi* words remain a significant challenge in morphological analysis. These words are discussed in detail in upcoming chapters.

2. *Samasa* Analysis: *Samasa* words are created by joining of 2 words. These words, unlike the *Sandhi* words, do not undergo phonetic transformation but the meaning of the new emergent word can be very different from original words. Like *Sandhi* words, they are also very common and are difficult to analyze in absence of a dictionary or any such lexical resource.

3. *Subanta* analysis: *Subanta* words refer to nouns, pronouns, adjectives and indeclinables. The nouns, pronouns and adjectives are inflected as per the *Vibhakti* rules. The rules of inflection are well understood and can be generated from word root using rules based logic. Most of *Subanta* words can be analyzed without much difficulty

4. *Tinanta* analysis: *Tinanta* words refer to verbs. There are around 2000 verb roots and all the inflectional forms and the rules for their formation are well known. Just like the *Subanta* words, *Tinanta* words can also be analyzed without much difficulty.

5. *Kridanta* analysis: *Kridanta* words are created when verb roots are combined with suffixes or prefixes called *Krit*. *Kridanta* can be nouns or indeclinables. There are around 135 *Kridanta* possible for each root. Unlike *Subanta* or *Tinanta*, the rules of *Kridanta* are too many, complex and not codified. *Kridanta* words are discussed in detail in coming chapters in this thesis.

6. *Taddhita* analysis: *Taddhita* words are nouns that are created from other nouns, pronouns and adjectives by adding *Taddhita* suffixes. The aforementioned paper does not address the problem of analyzing these words. This thesis discusses about *Taddhita* words in upcoming chapters.

The above points describe the problem of Sanskrit morphology in a broad sense. There are finer issue to be addressed even in case of *Subanta* and *Tinanta* words analysis. Some of these issues have been discussed in [24]. Many researchers have tried to increase the coverage and accuracy of Sanskrit Morphological analysis. Hellwig suggested usage Deep RNN and CRF for improving the accuracy of Morphological analysis of Sanskrit [25]. Jha et. al. have suggested a Database search based method for morphological analysis [26], but their method assumes

Sandhi free text and works only for *Subanta*, *Tinanta* and indeclinables. Sanskrit text, especially ancient texts make heavy usage of *Sandhi* and so the method proposed by Jha et. al. is not very useful in such cases.

The analysis of *Sandhi*, *Samasa*, *Kridanta* and *Taddhita* words remain a significant challenge. Most methods proposed to analyze these words make use of some lexical resource or dictionary. In many cases, the method is just referring to a look up table. While such method may some have limited use in some cases, they do not shed any light on the rules that may have led to their creation and are unlikely to be helpful in many applications like Sanskrit text generation. These methods are also limited in their coverage and do not have the scope of being extended to different words of similar category. While many researchers have tried to solve the problem of Sandhi analysis, their accuracy remains low, reducing their effectiveness in Sanskrit text analysis. A good Sandhi, *Kridanta* and *Taddhita* analysis engine will provide a big boost to Sanskrit morphological analysis and subsequently many other NLP applications for Sanskrit.

CHAPTER 4

Sandhi and Sandhi-split for Sanskrit words

4.1 Introduction

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)
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
```

4.2 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 [27]. 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 [28] 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.

4.3 Related Work

4.3.1 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 viz. JNU[1], UoH[2] & INRIA[3] tools are mentioned in table 4.1.

Table 4.1: Sandhi Tools Summary

Tool Name	**Description**
JNU Sandhi Tools	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	These tools were developed at the Department of Sanskrit Studies, University of Hyderabad under the supervision of Prof. Amba Kulkarni
INRIA Tools	Called as Sandhi Engine and developed under the guidance of Prof. Gerard Huet at INRIA.

An analysis and description of these tools is present in the paper on Sandhikosh [28]. The same paper introduced a dataset for Sandhi and Sandhi Split verification and compared the performance of the tools in table 4.1 on that dataset. Neural networks have been used for Sandhi Split by many researchers, for example [29], [30] and [31]. The task of doing Sandhi has been mainly addressed as a rule based algorithm e.g. [32]. 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 [28].

[1] `http://sanskrit.jnu.ac.in/sandhi/gen.jsp`
[2] `http://tdil-dc.in/san/sandhi/index_dit.html`
[3] `https://sanskrit.inria.fr/DICO/sandhi.fr.html`

4.3.2 Existing Work on Sandhi Split

Many researchers like [33] and [34] have tried to codify *pARini's* rules for achieving Sandhi Split along with a lexical resource. [35] proposed a statistical method based on Dirichlet process. Finite state methods have also been used [36]. A graph query method has been proposed by [37].

Lately, Deep Learning based approaches are increasingly being tried for Sandhi Split. [31] used a one-layer bidirectional LSTM to two parallel character based representations of a string. [38] and [30] proposed deep learning models for Sandhi Split at sentence level. [29] 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.

In addition to above, there exist multiple Sandhi Splitters in the open domain. The prominent ones being JNU Sandhi Splitter [4] , UoH Sandhi Splitter [5] and INRIA Sanskrit reader companion. The paper [29] compares the performance of above 3 tools with their results. This was an attempt to create benchmark in the area of Sanskrit Computational Linguistics.

4.4 Proposed Work

4.5 Method

4.5.1 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 [17] is especially suited to such problems, therefore same was used in this work. The training and test data were in ITRANS Devanagari format [6]. This data was converted to SLP1 [7] as

[4] `http://sanskrit.jnu.ac.in/sandhi/viccheda.jsp`
[5] `http://sanskrit.uohyd.ac.in/scl/`
[6] `https://www.aczoom.com/itrans/`
[7] `https://en.wikipedia.org/wiki/SLP1`

4.5 Method

SLP1 was found more suited for proposed approach. The code was implemented in python 3.5 with Keras API running on TensorFlow backend.

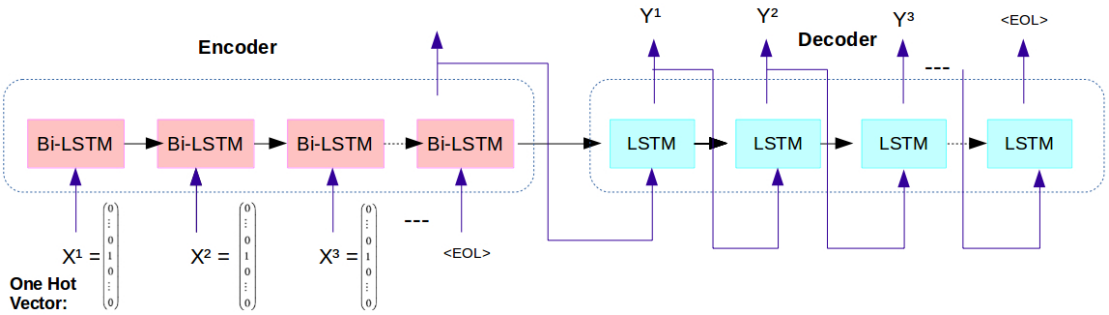

Figure 4.1: Model Architecture for Sandhi

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 [39] 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.

4.5.2 Sandhi Split Method

Conceptually, the Sandhi model architecture explained in Section 4.5.1 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. [29] 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 4.5.1.

4.5.2.1 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 4.2, the sandhi-window is considered from 13^{th} character to 17^{th} character, both inclusive.

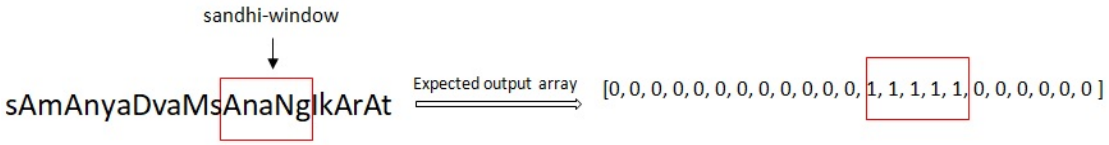

Figure 4.2: sandhi-window as prediction target in compound word

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.

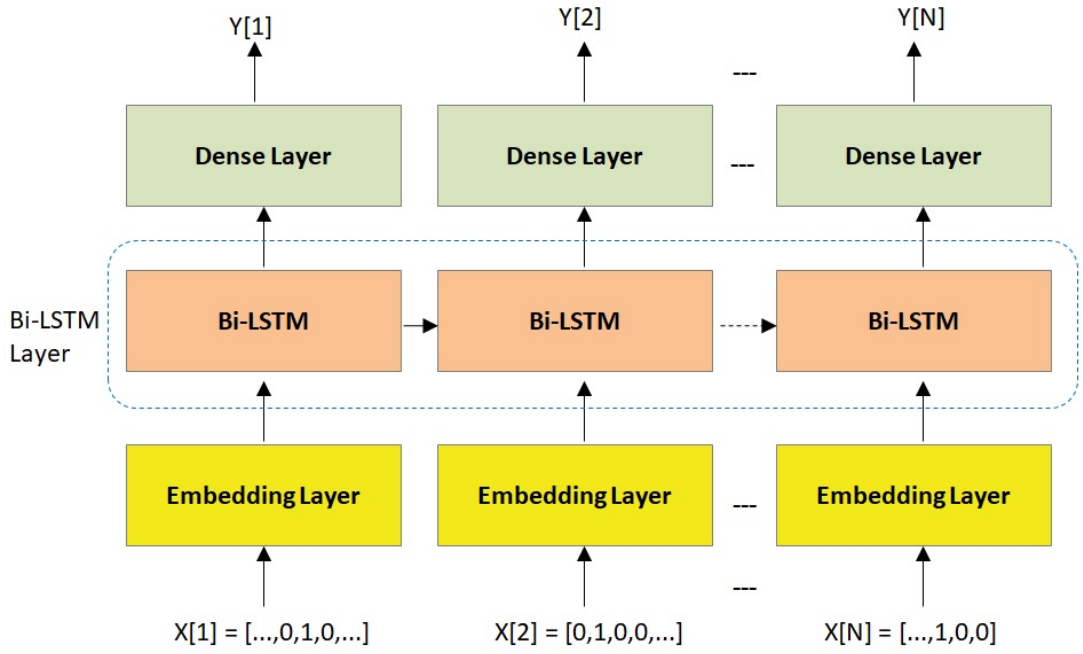

Figure 4.3: Model Architecture for Sandhi Split - Stage 1

4.5.2.2 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 4.5.1). Output sequence is set as truncated words T_{W_1} and T_{W_2} (see section 4.7.2 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 [39] 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.

4.6 Results

4.7 Data and Evaluation Results

4.7.1 Sandhi

The data for training a neural network model was taken from UoH corpus created at the University of Hyderabad [8]. 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

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

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 4.5.2. 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 [28]. 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 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 4.2. Every cell in the table indicates successful test cases, overall test cases and success percentage.

Table 4.2: Benchmark Results for Sandhi

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%)

As can be seen in the table 4.2, 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.

4.7.2 Sandhi Split

Table 4.3: Benchmark Results for Sandhi Split

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%

The UoH corpus was used for Sandhi Split task. The same dataset was used in sandhi task also (refer section 4.7.1). 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 4.5.2 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 [29] in table 4.3, on dataset taken from same source. The number of examples in full dataset is also sufficiently close (77842 for us vs. 71747 for [29]) All the rows in the table are taken from ([29], 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.

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.

4.8 Summary

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.

CHAPTER 5

Analysis of Taddhita and Kridanta pada

5.1 Introduction

As mentioned earlier in this thesis, the analysis of word created from *Krit* and *Taddhita* affixes are a challenge in Sanskrit morphological analysis. The work in this chapter tries to address this problem.
Note: Work described in this chapter is a collaborative effort.[1]

5.1.1 Introduction of Pratyaya in Sanskrit

Different ways of inflectional word formation as mentioned by [40] are as below:

- [Root + Suffix] Root: desideratives, intensives.
- [Word + Suffix] Root: denominal verbs.
- [Root + Suffix] Stem: primary (*krit*) suffixes.
- [Word + Suffix] Stem: secondary (*taddhit*) suffixes.
- [Word + Word] Stem: compounding.

[1]The work described in this chapter was carried out in collaboration with Arun Kumar Singh(2015EEY7542)

- [Root + Suffix] Word: verb inflection.
- [Stem + Suffix] Word: noun inflection.

Here in this introduction, we will explain more about primary and secondary suffixes. Sanskrit is a rich inflected language and depends on nominal and verbal inflections for communication of meaning [41]. A fully inflected unit is called pada. The *subanta padas* are the inflected nouns and the *tinanta padas* are the inflected verbs.

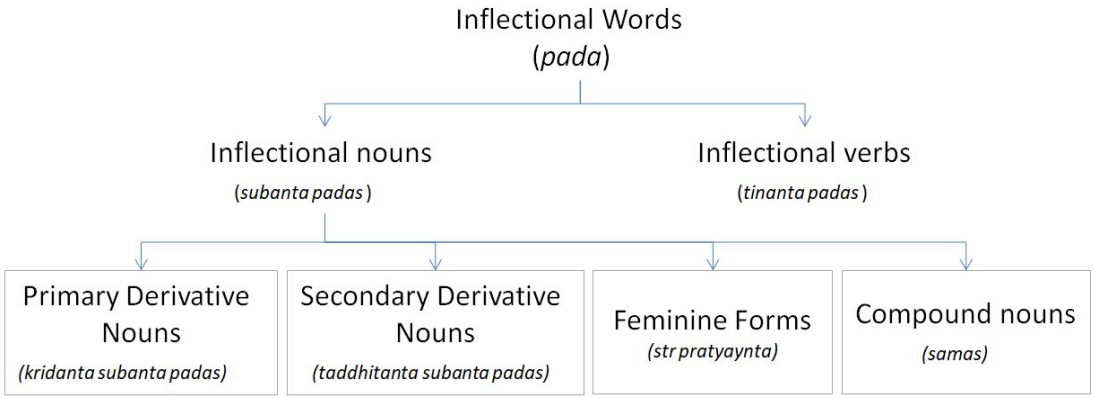

Figure 5.1: Inflectional Word Hierarchy

5.1.1.1 Kridanta subanta (Primary Derivative Nouns)

These are formed when the primary affixes called *krit* are added to verbs to derive substantives, adjectives or indeclinable. *DAtum nAma karoti iti krit. Kridanta* play a vital role in understanding Sanskrit language. Many morphological analyzers are lacking the complete analysis of *Kridanta*. Examples of *krit* pratyaya are as below:

paW (root verb)+ *tavya* (*krit* suffix) = *paWitavyam*
paW (root verb)+ *tumun* (*krit* suffix) = *paWitum*

krit suffixes are mainly of seven types viz. *tavyat, tavya, anIyara,Ryat, yat, kyap, kelimer.*

5.1.1.2 Taddhitanta subanta (Secondary Derivative Nouns)

The secondary derivative affixes called *taddhit* derive secondary nouns from primary nouns. Some examples of *taddhit* pratyaya are as below:

Indra (noun) + *aR* (*taddhit* suffix) = *Endra*
Dana (noun) + *vatup* (*taddhit* suffix) = *Danavat*

taddhit suffixes are mainly of fourteen types.

5.2 Motivation

Researchers have earlier attempted to develop morphological analyzer for Sanskrit such as CDAC [2], Sanskrit academy [3] and [42]). But having constraint of limited coverage or not being available openly, it is difficult to reuse it for further applications in NLP area. Also it is not possible for a person from a non-Sanskrit background to develop applications and systems in Sanskrit morphology, even though Sanskrit is well codified language. Among various facets of Morphological Analysis, suffix analyzer is necessary for a good coverage morphological analyzer. Our proposed Sanskrit noun derivatives analyzer will 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

Pratyayas are not easily available for building Sanskrit based applications either in printed or e-forms. Though some sources are available such as *Panini Pratyayartha Kosha Taddhita Prakaranam* and *Panini Kridanta Pratyaya Artha Kosha* by Dr. Gyanprakash Shastri [4] , *Laghu Siddhant Kaumudi* [5] and *Kridant Karak Prakaranam* by Bhimsen Shashtri but one has to have good understanding of Sanskrit to analyze it and use it to develop some analysis tool or system. We

[2]http://www.cdac.in/html/ihg/ihgidx.asp
[3]http://www.sanskritacademy.org/
[4]https://archive.org/
[5]https://chaukhambapustak.com/

have prepared Pratyaya-Kosh in the form of root, suffix and derivative noun due to affix. This data can be easily used to develop analyzers and can be trained to learn rules using various algorithms automatically.

5.3 Related Work

[43] presented a model in the form of a tool for recognition and analysis of nominal morphology (Sanskrit *subanta padas*) in Sanskrit text for Machine Translation. The tool recognizes nominal morphology with the help of avyaya and verb database and does analysis of Sanskrit *subanta padas*. [41] provided an approach to deal with *Kridanta*. Morphological dictionaries for upapadas, upasargas, roots and suffixes were created and rule based avyaya analyzer was developed which does Morphological Analysis based on identification & filtering of upapadas, upsargas based on dictionary.

The method adopted by [44] was a paradigm based approach where a student is taught the word forms of a common word e.g. deva in Sanskrit and that it is the default paradigm for 'a' ending masculine words. Further the list of exceptional words and the forms where they differ are taught separately. Following this method, a simple algorithm was developed which is described in [45]. This algorithm has been used to develop morph analyzer for different Indian languages (IIIT-H). Separate modules have been developed to handle *subanta, tinanta* and *Kridanta* words. A list of lexicon is extracted from Monier William's dictionary. *Kridanta* analyzer works based on rule based approach to retrieve pratyaya and upasarga form lexicon dictionary. It doesn't provide any module to analyze other derivational suffixes such as *taddhit* but claims for a provision of plug in. [46] proposed an approach for analysis of derivational nouns in Sanskrit. This approach attempted to build a semi supervised model that can identify usage of derived words in a corpus and map them to their corresponding source words. Special interest was given to the usage of secondary derivative affixes in Sanskrit ie. *taddhit*. When it comes to automating *taddhit*, the Sanskrit Heritage System [47] is an existing system that can recognize *taddhit* and perform the analysis, but it does not generate the *taddhit* and only the lexicalized *taddhit* are recognized during the analysis.

[48] attempted to automate the process of deriving *taddhit* in complete adherence to *Ashtadhyayi*. The proposed system adopted a completely object oriented approach in modelling *Ashtadhyayi*. The rules of *Ashtadhyayi* were modeled as classes and were the environment that contains the entities for derivation. In their work the rule group was achieved through formation of inheritance network. The current resources available for finding out derivational nouns in open domain are not very accurate. Two most popular publicly available set of Pratyaya Analysis tools viz. JNU Sanskrit *Kridanta* Analyzer [6] and UoH Morphological Generator [7] are mentioned in table 5.1.

Tool Name	**Description**
Sanskrit *Kridanta* Analyzer (JNU)	The "Sanskrit *Kridanta* Analyzer" was partially completed as part of M.Phil. research submitted to Special Center for Sanskrit Studies, JNU in 2008 by Surjit Kumar Singh (M.Phil 2006-2008) under the supervision of Dr. Girish Nath Jha . It facilitates the split of *Kridanta* into root verb and *krit* pratyaya
Morphological Generator (UoH)	Morphological Generator shows the inflectional, and (some) derivational forms of a given noun or a verb. This tool has been developed by University of Hyderabad (UoH)

Table 5.1: Open Available Tools for Derivative Nouns (Pada) Analysis

5.4 Proposed Work

5.5 Method

5.5.1 Kridanta Pada Formation Method

Kridanta Pada formation task is similar to language translation problem where a sequence of characters or words produces another sequence and lengths of the

[6] http://sanskrit.jnu.ac.in/kridanta/ktag.jsp
[7] http://sanskrit.uohyd.ac.in/scl/

inputs and outputs are not fixed. RNNs are widely used to solve such problems. Sequence to sequence model introduced by [17] is especially suited to such problems, therefore it 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.

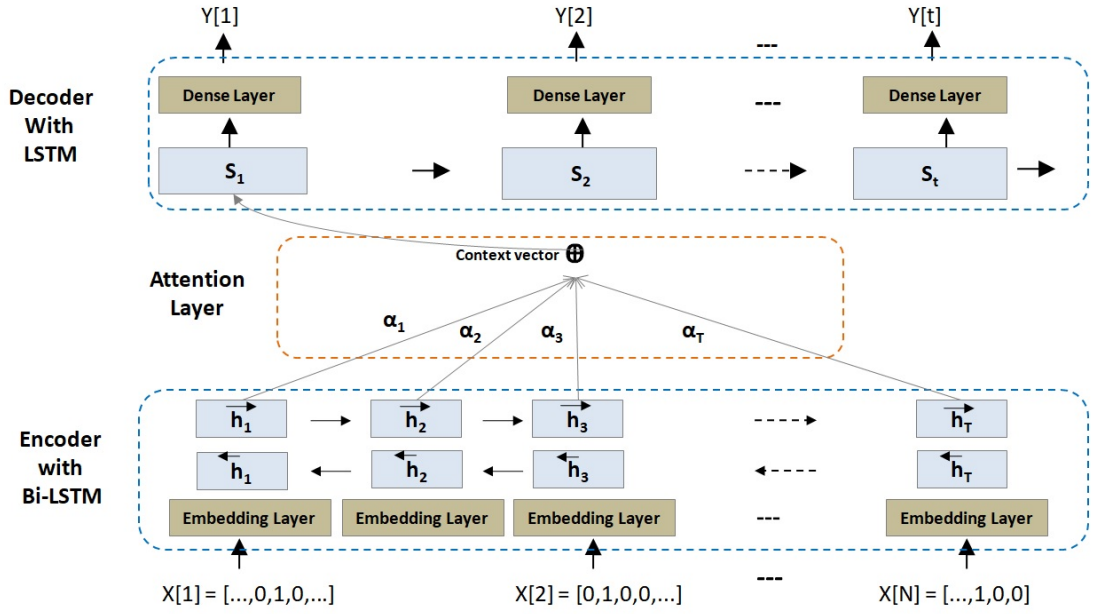

Figure 5.2: Architecture for Learning Derivative Noun (Pada) Formation

Proposed model expects 2 words (root + *krit* pratyaya) concatenated using a "+" symbol as input and outputs the primary derivative noun (*Kridanta*) as shown in the example below:

tul + *lyuw* = *tolanam*

Results achieved from this approach seem to be good. Proposed approach tries to address this problem by giving all the characters of the first input word and all the characters of the second input word concatenated with a '+' symbol as the input to the encoder LSTM of the sequence to sequence model. Due to unavailability of enough data we based our analysis on the 10 pratyayas out of 12. Input

[8] https://www.aczoom.com/itrans/
[9] https://en.wikipedia.org/wiki/SLP1

sequence is set as the two input words concatenated with a '+' character between the 2 words. Output sequence is the *Kridanta* (primary derivative noun) single word. Characters '&' and '$' were used as start and end markers respectively in the output sequence. The maximum lengths of input and output sequences were 17 and 18 respectively. So we used '*' for padding shorter input sequences. We used the same dictionary for input and output characters. The dictionary contained just 53 characters and characters usually don't have too much correlation as can be observed for words. This is the reason that we used one hot encoded representation of the characters instead of training any word embedding. The best results were achieved with an LSTM [39] as basic RNN cell for decoder and bidirectional LSTM as basic RNN cell for encoder. Adding attention to the encoder part improved the accuracy slightly so we opted for using attention in our model. Both the encoder and decoder use the hidden unit size (latent dimension) as 128. The training vectors were divided in batches of 32 vectors and total of 70 epochs were run to get the final results.

5.5.2 Taddhitanta Pada Formation Method

Taddhitanta are formed from *taddhit* pratyayas much like *Kridanta* are formed from *krit* pratyayas. Below example shows this formation:

$$sUrasena + Rya = sOrasenyaH$$

This is similar to *Kridanta* formation and thus we used the same sequence to sequence model. However the data that we had in case of *taddhit* pratyaya was limited and imbalanced for training the neural network. The rules for *Taddhitanta* formation moreover are considered to be more complicated than those for *Kridanta* formation. We believe these are the reasons we did not get very high but a satisfactory accuracy on *Taddhitanta* unlike *Kridanta* which performed much better. The accuracy mentioned is calculated as:

(Number of correct formations) / (Total number of test samples)

The character wise accuracy, however is quite high (98%), which means that the model incorrectly predicts just one or two characters rather than whole words in incorrect formations.

5.5.3 Derivative Noun (Pada) Split Method

Conceptually, the model architecture explained in Section 5.5.1 for *Kridanta* formation and *Taddhitanta* formation respectively, can also be used for splitting the *Kridanta* and *Taddhitanta* back into root and pratyaya, where the input and output are swapped with compound word as input and the two initial words concatenated with '+' character as output. The accuracy achieved with this approach was better than that obtained for pada formation. Our model seems to perform fairly well even for longer sequences of length greater than 15.

5.6 Results

5.7 Data and Evaluation Results

5.7.1 Pratyaya-Kosh

Data for Pratyaya-Kosh was extracted from 4 different sources viz. Panini Pratyayarth kosh, Sanskrit Hindi Kosh dictionary, KridantaRupaMala & Sankrit Abhyas Portal developed based on rules of *Ashtadhyayi*. Majority of data was extracted from Sanskrit Abhyas Portal. Table 5.2 shows details of data sources.

Source Name	**Author**	**Source Reference**
Sankrit Abhyas Portal	Sharat Kotian	http://www.sanskritabhyas.in/en
Panini Pratyayarth kosh	Dr. Gyanprakash Shastri	https://archive.org/details/PaniniPratyayarthaKoshaTaddhitaPrakaranamDr.GyanprakashShastri
KridantaRupaMala	Pandit S. Ramasubba Sastri	https://archive.org/details/KridantaRupaMalaVol.1-5
Sanskrit Hindi Kosh dictionary	V.S. Apte	https://archive.org/details/ SanskritHindiKoshV.S.Apte

Table 5.2: Pratyaya-Kosh Data Sources

This Pratyaya-Kosh corpus consists of *Kridanta* (Primary derivative nouns) and *Taddhitanta* (Secondary Derivative nouns) in a form which can be directly

ingested for machine learning based approaches. Each record is a tuple in the form of (verb stem, *krit* suffix, *Kridanta*) and (noun, *taddhit* suffix, *Taddhitanta*). Corpus size of *Kridanta* padas is 24,757 which includes padas formed by 12 different *krit* suffixes, where as corpus size of *Taddhitanta* padas is 3,088 which are formed by 17 different *taddhit* suffixes. Details of the corpus are given in table 5.3.

***krit* Pratyaya**	**Corpus Size**	***taddhit* Pratyaya**	**Corpus Size**
lyuw	2198	*Ca*	152
anIyar	2198	*QaY*	58
Rvul	2198	*Qak*	124
tumu n	2198	*Rya*	104
tavya	2198	*Rini*	36
tfc	2198	*WaY*	150
ktvA	2198	*Wak*	308
Lyap	2198	*aR*	610
ktavatu	2198	*aY*	246
Kta	2198	*iY*	96
Satf	1687	*Itac*	92
SAnac	1090	*Matup*	168
-	-	*Mayaw*	10
-	-	*Tal*	312
-	-	*Tva*	311
-	-	*yaY*	113
-	-	*Yat*	198

Table 5.3: Pratyaya-Kosh Corpus Details

5.7.2 Derivative Noun (Pada) Analysis Evaluation

Training and test data was taken from Pratyaya-Kosh. In case of *Kridanta* Padas, training data was chosen for 10 *krit* suffixes out of 12 due to data unbalancing issue. *Satf* & *SAnac* suffixes have less data as compared to other *krit* suffixes, hence *Kridanta* Padas were left out corresponding to these 2 suffixes while training

and testing. In case of *Taddhitanta* padas analysis, all suffixes were taken into consideration for training and testing. Analysis of padas (primary derivative nouns & secondary derivative nouns) were done in two ways as pada formation and pada split into its original verb stem & suffix in case of *Kridanta* pada and noun & suffix in case of *Taddhitanta* pada. For analysis of both the cases, pada formation and pada split, total records (tuples) were considered as 21,980 for *Kridanta* and 3088 for *Taddhitanta*. 80% examples used for training & validation (17,584 for *Kridanta*, 2470 for *Taddhitanta*) and remaining 20% examples used for model testing(4396 for *Kridanta* and 618 for *Taddhitanta*). In case of Derivative noun or pada formation, evaluation is based on exact match of whole pada. To evaluate the pada split, both suffix as well as original root stem or noun should be correct to consider it as success. Even if the model confuses between a & A (SLP1 encoding), it is considered a failure. Results from method described above were bench marked with results from other publicly available tools as mentioned in table 5.1. Test set from Pratyaya-Kosh was kept same for comparing with UoH & JNU tools. UoH tool does pada split for both primary as well as secondary derivative nouns where as JNU tool does pada split for only primary derivative nouns. However both these tools does not provide feature for pada formation analysis. Evaluation criteria was kept strict for proposed method as mentioned above while the success criteria for UoH & JNU tool was relaxed in a way if either of original stem verb/noun or suffix is found in the result, it is considered as success.

The comparison is shown in the table 5.5. Every cell in the table 5.4 & 5.5 indicates successful test cases, overall test cases and success percentage.

Model	***Kridanta* Formation Accuracy**	***Taddhitanta* Formation Accuracy**
JNU (Sanskrit *Kridanta* Analyzer)	-	-
UoH (Morphological Generator)	-	-
Proposed Method	3718 / 4396 (84.58 %)	495 / 618 (80.09%)

Table 5.4: Benchmark Results of Derivative noun (Pada) formation

Model	***Kridanta* Split Prediction Accuracy**	***Taddhitanta* Split Prediction Accuracy**
JNU (Sanskrit *Kridanta* Analyzer)	1710 / 4396 (38.9%)	-
UoH (Morphological Generator)	3492 / 4396 (79.43%)	146 / 618 (23.62%)
Proposed Method	3903 / 4396 (88.79 %)	255 / 618 (41.26%)

Table 5.5: Benchmark Results of Derivative noun (Pada) Split

It is evident form results that proposed method improves upon the existing Morphological Analysis tools and methods by a significant margin. In addition, proposed model does not require any external lexicon for analysis and its prediction mechanism works better than dictionary based tools and approaches.

5.8 Summary

In this research work, we propose Pratyaya-Kosh, a benchmark corpus to help researchers new to Sanskrit in building AI based Morphological Analyzer for Sanskrit derivative nouns. Also we propose neural approach for learning derivative noun formation without use of any external resources such as language models, morphological or phonetic analyzers and still manage to outperform existing approaches. In future we intend to extend current work to verb derivative and indeclinable derivative using machine learning methods. Proposed models can be further refined by using additional training data. Benchmark corpus (Pratyaya-Kosh) will be made available on git hub.

CHAPTER 6

Conclusion

This thesis tries to apply standard NLP techniques to Sanskrit and analyzes the results and tries to understand the underlying reasons behind those results. It addresses the problem of *Sandhi* and *Sandhi-Vicceda* by introducing new technique to handle *Sandhi* words with high accuracy.

This thesis introduces a new dataset for *Pratyaya* words and their roots along with a method to create and analyze such words. The suggested method provides better accuracy than any existing methods.

We sincerely hope this work will contribute to the area of Sanskrit NLP by filling the holes in existing Morphological analysis methods and also facilitate further research in this area by other researchers.

6.1 Future Scopes

1. The accuracy of *Sandhi* and *Sandhi-Vicceda* can possibly be improved by using more data. It may also be possible to use context information to improve the accuracy.

2. It is also possible to either use better architectures of the models suggested in this thesis or use new models altogether to get better results for the tasks

accomplished in this thesis. It is certainly possible that improved Neural Network architectures will provide better than the ones suggested in this report. Usage of advanced tools like Transformers[1] and Attention Layer[2] is also likely to be useful.

3. The above possibilities are also applicable to the *Pratyaya* analysis method. We can also benefit from using large existing corpus of *Pratyaya* words and digitizing those using good OCR techniques and some manual annotation using help from Sanskrit scholars.

[1] `https://arxiv.org/abs/1909.06317`
[2] `https://arxiv.org/abs/1706.03762`

References

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

[2] G. Cardona, *Panini: A Survey of Research*. Motilal Banarsidass, 1997.

[3] S. C. Kak, "The paninian approach to natural language processing," *International Journal of Approximate Reasoning*, vol. 1, no. 1, pp. 117 – 130, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0888613X87900077

[4] J. Firth, "A synopsis of linguistic theory 1930-1955," in *Studies in Linguistic Analysis*. Philological Society, Oxford, 1957, reprinted in Palmer, F. (ed. 1968) Selected Papers of J. R. Firth, Longman, Harlow.

[5] S. T. Dumais, "Latent semantic analysis," *Annual Review of Information Science and Technology*, vol. 38, no. 1, pp. 188–230, 2004. [Online]. Available: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440380105

[6] M. Sahlgren. (2015) A brief history of word embeddings. [Online]. Available: https://www.linkedin.com/pulse/brief-history-word-embeddings-some-clarifications-magnus-sahlgren/

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[8] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[9] I. Sh, S. Anand, R. Goyal, and S. Misra, "Representing contexual relations with sanskrit word embeddings," 07 2017, pp. 262–273.

[10] DCS. (2020) Digital corpus of sanskrit. [Online]. Available: http://www.sanskrit-linguistics.org/dcs/index.php

[11] UoH. (2020) A sanskrit computational tool. [Online]. Available: https://sanskrit.uohyd.ac.in/scl/

[12] SansWiki. (2020) Sanskrit wikipedia. [Online]. Available: https://sa.wikipedia.org/wiki

[13] gensim. (2020) Python framework for fast vector space modelling. [Online]. Available: https://pypi.org/project/gensim/

[14] Tensorflow. (2020) Embedding projector. [Online]. Available: https://projector.tensorflow.org/

[15] R. A. Salama, A. Youssef, and A. Fahmy, "Morphological word embedding for arabic," *Procedia Computer Science*, vol. 142, pp. 83 – 93, 2018, arabic Computational Linguistics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918321653

[16] A. A. Markov, "Theory of algorithms," *Journal of Symbolic Logic*, vol. 22, no. 1, pp. 77–79, 1957.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[18] NLTK. (2020) Natural language toolkit. [Online]. Available: https://www.nltk.org/

[19] K. M. K. M. Desiraju Hanumanta Rao. (2020) Valmiki ramayana. [Online]. Available: https://www.valmikiramayan.net/

[20] W. Carey. (2020) Sanskrit bible. [Online]. Available: http://sanskritbible.in

[21] O. R. Hartono. (2020) Bible corpus. [Online]. Available: https://www.kaggle.com/oswinrh/bible

[22] Y. Artzi. (2020) Language models. [Online]. Available: http://www.cs.cornell.edu/courses/cs5740/2017sp/lectures/06-lm.pdf

[23] A. Bharati, A. Kulkarni, V. Sheeba, and R. Vidyapeetha, "Building a wide coverage sanskrit morphological analyzer: A practical approach," 01 2006.

[24] A. Kulkarni and D. Shukl, "Sanskrit morphological analyser: Some issues," 2009.

[25] O. Hellwig, "Improving the morphological analysis of classical Sanskrit," in *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 142–151. [Online]. Available: https://www.aclweb.org/anthology/W16-3715

[26] G. N. Jha, M. Agrawal, Subash, S. K. Mishra, D. Mani, D. Mishra, M. Bhadra, and S. K. Singh, *Inflectional Morphology Analyzer for Sanskrit*. Berlin, Heidelberg: Springer-Verlag, 2009, p. 219–238. [Online]. Available: https://doi.org/10.1007/978-3-642-00155-0_8

[27] V. S. Akshar Bharati, Amba Kulkarni, "Building a wide coverage sanskrit morphological analyzer : A practical approach," in *TheFirst National Symposium on Modelling and Shallow Parsing of Indian Languages, IIT-Bombay*, 2006.

[28] S. Bhardwaj, N. Gantayat, N. Chaturvedi, R. Garg, and S. Agarwal, "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: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://www.aclweb.org/anthology/L18-1712

[29] R. Aralikatte, N. Gantayat, N. Panwar, A. Sankaran, and S. Mani, "Sanskrit sandhi splitting using seq2(seq)2," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 4909–4914. [Online]. Available: https://www.aclweb.org/anthology/D18-1530

[30] O. Hellwig and S. Nehrdich, "Sanskrit word segmentation using character-level recurrent and convolutional neural networks," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2754–2763. [Online]. Available: https://www.aclweb.org/anthology/D18-1295

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

[32] S. V. K. Raja, V. Rajitha, and M. Lakshmanan, "A binary schema and computational algorithms to process vowel-based euphonic conjunctions for word searches," *ArXiv*, vol. abs/1409.4354, 2014.

[33] G. Huet, "A functional toolkit for morphological and phonological processing, application to a sanskrit tagger," *J. Funct. Program.*, vol. 15, pp. 573–614, 07 2005.

[34] A. Kulkarni and D. Shukl, "Sanskrit morphological analyser: Some issues," *Indian Linguistics*, vol. 70, pp. 169–177, 01 2009.

[35] A. Natarajan and E. Charniak, "s^3 - statistical sandhi splitting," in *Proceedings of 5th International Joint Conference on Natural Language Processing*. Chiang Mai, Thailand: Asian Federation of Natural Language Processing, Nov. 2011, pp. 301–308. [Online]. Available: https://www.aclweb.org/anthology/I11-1034

[36] V. Mittal, "Automatic Sanskrit segmentizer using finite state transducers," in *Proceedings of the ACL 2010 Student Research Workshop*. Uppsala, Sweden: Association for Computational Linguistics, Jul. 2010, pp. 85–90. [Online]. Available: https://www.aclweb.org/anthology/P10-3015

[37] A. Krishna, B. Santra, P. Satuluri, S. P. Bandaru, B. Faldu, Y. Singh, and P. Goyal, "Word segmentation in Sanskrit using path constrained random walks," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 494–504. [Online]. Available: https://www.aclweb.org/anthology/C16-1048

[38] V. Reddy, A. Krishna, V. Sharma, P. Gupta, V. M R, and P. Goyal, "Building a word segmenter for Sanskrit overnight," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://www.aclweb.org/anthology/L18-1264

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] P. Kiparsky, "On the architecture of panini's grammar." vol. 5402, 01 2008, pp. 33–94.

[41] N. Murali, R. J. Ramasreee, and K. V. L. N. Acharyulu, "Kridanta analysis for sanskrit," 2014.

[42] G. Huet, "Towards computational processing of sanskrit," 2003.

[43] S. Chandra and G. Jha, "Sanskrit subanta recognizer and analyzer for machine translation," 11 2006.

[44] A. Bharati, A. Kulkarni, V. Sheeba, and R. Vidyapeetha, "Building a wide coverage sanskrit morphological analyzer: A practical approach," 01 2006.

[45] A. Bharati, V. Chaitanya, R. Sangal, and B. Gillon, "Natural language processing: A paninian perspective," 07 2002.

[46] A. Krishna, P. Satuluri, H. Ponnada, M. Ahmed, G. Arora, K. Hiware, and P. Goyal, "A graph based semi-supervised approach for analysis of derivational nouns in sanskrit," in *TextGraphs@ACL*, 2017.

[47] P. Goyal and G. Huet, "Completeness analysis of a sanskrit reader," 01 2013.

[48] A. Krishna and P. Goyal, "Towards automating the generation of derivative nouns in sanskrit by simulating panini," *ArXiv*, vol. abs/1512.05670, 2015.

SLP1 Reference Table

अ a **a**	आ ā **A**	इ i **i**	ई ī **I**	उ u **u**	ऊ ū **U**

ऋ ṛ **f**	ॠ ṝ **F**	ऌ ḷ **x**	ॡ ḹ **X**

ए e **e**	ऐ ai **E**	ओ o **o**	औ au **O**

क् k **k**	ख् kh **K**	ग् g **g**	घ् gh **G**	ङ् ṅ **N**
च् c **c**	छ् ch **C**	ज् j **j**	झ् jh **J**	ञ् ñ **Y**
ट् ṭ **w**	ठ् ṭh **W**	ड् ḍ **q**	ढ् ḍh **Q**	ण् ṇ **R**
		ळ् ḷ **L**	ळ्ह् ḷh **|**	
त् t **t**	थ् th **T**	द् d **d**	ध् dh **D**	न् n **n**
प् p **p**	फ् ph **P**	ब् b **b**	भ् bh **B**	म् m **m**

य् y **y**	र् r **r**	ल् l **l**	व् v **v**
श् ś **S**	ष् ṣ **z**	स् s **s**	ह् h **h**
ः ḥ **H**	ऽ ऱ् **Z**	ऽ ल् **V**	ं ṁ **M**

Source: `https://www.sanskritlibrary.org/help-text.html`

Publications from this Thesis

Conferences

- **Sushant Dave, Arun Kumar Singh, Dr. Prathosh A.P. and Dr. Brejesh Lall**, "Neural Compound-Word (Sandhi) Generation and Splitting in Sanskrit Language" in *8th ACM IKDD CODS and 26th COMAD 2-4 January 2021*.

- **Arun Kumar Singh, Sushant Dave, Dr. Prathosh A.P. and Dr. Brejesh Lall**, "A Benchmark Corpus and Neural Approach for Sanskrit Derivative Nouns Analysis" in *The 16th Conference of the European Chapter of the Association for Computational Linguistics*, 2021.

Author's Biography

Sushant Dave is a MS(Research) candidate from Department of Electrical Engineering, Indian Institute of Technology, Delhi. He completed his B.Tech. degree in Electronics & Communication Engineering from Manipal University.

Contact Information

Permanent Address: Noida, Uttar Pradesh, PIN–201301, India.
Mobile: +91-9654879650
Email: eey157538@ee.iitd.ac.in

Research Interests

Image Processing, NLP, Video Analytics