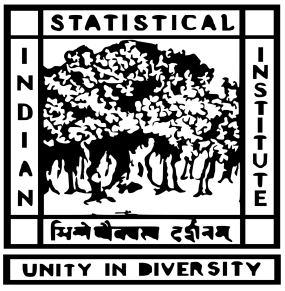
On Lemmatization and Morphological Tagging for Highly Inflected Indic Languages

by

Abhisek Chakrabarty



A thesis submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy in Computer Science

Under the guidance of

Prof. Utpal Garain

Computer Vision and Pattern Recognition Unit

Indian Statistical Institute

203 Barrackpore Trunk Road, Kolkata 700108, India

July 2018

To my parents, Mr. Ahit Kumar Chakrabarty and Mrs. Pronati Chakrabarty.

Abstract

Morphological variations of root words make natural language processing (NLP) a very difficult task for Indic languages. Multiple variants of a root portraying different aspects, such as case, parts of speech, person, tense etc. result in hard computational troubles for developing systems like machine translation, parsing, word sense disambiguation and so on. Thus, word form normalization is a crucial job for further processing of any inflected language. The current thesis mainly focuses on the lemmatization problem which is the process of determining the dictionary root form of a surface word to obtain the meaning or other linguistic properties of textual surface words. The resource scarcity for Indic languages has been initially tackled by building an unsupervised algorithm which is tested on Bengali. Recently developed word embedding technique is also incorporated into the algorithm to exploit the word distributional similarity. Eventually, a supervised deep neural lemmatizer has been developed and evaluated on two Indic (Bengali and Hindi) and seven European languages (Catalan, Dutch, Hungarian, Italian, Latin, Romanian and Spanish). For eight of these languages, the neural lemmatizer defines the current state-of-the-art results. The remaining part of the thesis addresses two other morphological analysis tasks namely, morphological tagging and inflection generation. Morphological tagging involves determining several attributes such as case, degree, gender, POS etc. of an in-context word. As individual word forms in the inflected languages hold considerable linguistic information, so efficiently modeling their morphological properties helps a number of NLP applications. Five Indic languages are chosen for the tagging experiments. The final endeavour targets the problem of inflection generation, the goal of which is to produce the variant of a source word, given the morpho-syntactic descriptions of the target word.

List of Publications

- Chakrabarty, Abhisek, and Utpal Garain. 'BenLem (A Bengali Lemmatizer) and Its Role in WSD.' In ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP) 15.3 (2016): 12. (Content of chapter 2)
- Chakrabarty, Abhisek, Ganguly, Debasis, Roy Choudhury, Sharod, and Utpal Garain. 'Word Embedding based Unsupervised Lemmatization for Highly Inflected Low-Resourced Indic Languages.' Under review in Information Systems Frontiers - Springer. (Content of chapter 3)
- Chakrabarty, Abhisek, Akshay Chaturvedi, and Utpal Garain. 'A Neural Lemmatizer for Bengali.' In Language Resources and Evaluation Conferences (LREC). 2016. (Content of chapter 3)
- Chakrabarty, Abhisek, Onkar Arun Pandit, and Utpal Garain. 'Context Sensitive Lemmatization using Two Successive Bidirectional Gated Recurrent Networks.' In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers). Vol. 1. 2017. (Content of chapter 4)
- Chakrabarty, Abhisek, Akshay Chaturvedi, and Utpal Garain. 'NeuMorph: Neural Morphological Tagging for Low-Resource Languages.' Under review in ACM TALLIP. (Content of chapter 5)
- Chakrabarty, Abhisek, and Utpal Garain. 'ISI at the SIGMORPHON 2017 shared task on morphological reinflection.' In Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection (2017): 66-70. (Content of chapter 5)

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Utpal Garain for his continuous support and encouragement that helped me to complete this work. During the Ph.D tenure, whenever any critical situation came, I got him by my side to overcome that.

Next, I want to thank Dr. Mandar Mitra, one of the best teachers in ISI, for tolerating me and my silly questions.

I am grateful to my labmates - Apurba da, Arjun, Anabik, Akshay and Onkar for helping me with their thoughts and ideas to improve the quality of the thesis.

To all the members of CVPR Unit, especially Dwaipayan, Dipa di, Kripa da, Ayan da, Suchana, Partho da, Chandan da, Jayati, Arundhati, Tanmoy, Sumit, Sankar da, Bhabani di and Manik da, I am indebted for providing me a friendly working environment.

Some of my hostel friends - Ankush, Raju da, Muna, Suman da helped me to remain mentally steady during ups and downs in Ph.D. My heartiest love goes to them.

From my uncle Mr. S. C. Bhoumik, I always got inspiration in pursuing the research work. It gives me pleasure to acknowledge him.

Finally, I want to thank my parents, without whom I could never be here.

Contents

List of Figures

List of Tables

Chapter 1

Background, Problem Statement and Literature Survey

1.1 Introduction

Natural language processing (NLP) for inflectional languages has to encounter an additional challenge as opposed to languages having simpler morphology, e.g. English. This is due to handling of the morphological variations of root words. There exist multiple variants of a root portraying different aspects, such as case, honorific status, parts of speech (POS), person, tense etc. The high level of inflections results in problems for developing many NLP systems, for instance, machine translation (MT), parsing, word sense disambiguation (WSD) and so on. Thus, word form normalization is a crucial job for further processing of any inflected language.

The current thesis mainly focuses on the lemmatization problem along with two other tasks - morphological tagging and inflection generation. Lemmatization is the process to determine the canonical/dictionary/root form of a surface word. In the lexical resources such as dictionary, WordNet etc., the entries are usually roots with their morphological and semantic descriptions. Therefore, to obtain the meaning or other linguistic properties of textual surface words, finding out their lemmas is essential.

Surprisingly, research on lemmatization received little attention from the computational linguistics community. This may be due to the fact that in English, for which substantial

amount of research has been conducted, presence of inflections does not pose much problem and if there is a need of normalizing the word forms, the purpose is well served by using stemming algorithms. Stemming is a way similar to lemmatization producing the common portion of variants but it has several limitations - (*i*) there is no guarantee of a stem to be a legitimate word form, (*ii*) words are considered in isolation. Whereas, a lemma must be a valid form in the language and it may alter on varying contexts for a particular word. Hence, for context sensitive languages i.e. where same inflected word form may come from different sources and can only be disambiguated by considering its neighbouring information, there lemmatization is utmost necessary to handle diverse text processing problems.

As we primarily deal with in-context lemmatization, so it is necessary to choose some morphologically rich and context sensitive languages for experimentation purpose. Major Indic languages belong to this category. The members of the Indo-Aryan language family (Bengali, Hindi, Gujrati etc.) which are originally derived from Sanskrit, show wide morphological diversity. Our research was motivated with the goal to improve the computational processing for them. Initially we start with Bengali which is among the top 10 most widely used languages in the world. About 70% words in Bengali are inflected forms and it is observed that some particular roots produce up to 90 morphological variants (Dash, 2015). In general, 20 or more inflections from a root word are found often. Besides, a significant number of words in Bengali actually belong to several foreign languages including European (Dutch, English, French and Portuguese), Middle Eastern (Arabic and Persian) and the other members of the Indo-Aryan family (Hindi, Sanskrit) as well (Faridee et al., 2009). Different linguistic features such as aspect, modality, person, tense etc. attribute to the presence of the variants. Apart from these features, Bengali is diglossic having literary and standard colloquial styles making the lemmatization task more difficult. Resource scarcity is another big obstacle in building NLP systems for the Indic languages. In the beginning we could not apply the global state-of-the-art lemmatizers on Bengali as they are supervised in nature requiring a large amount of annotated data for training their models. This made us to start with unsupervised techniques. Our first work is a language independent, dictionary based unsupervised lemmatizer that exploits a human readable dictionary, a valid suffix list of the concerned language and a POS tagger to develop the system. Next, in addition to Bengali, experiments are done on Hindi which is the most widely spoken

language in India and ranks within the World's top 10 languages. It is also a member of the Indo-Aryan family and possesses rich morphology. Thereafter we put stress on building a Bengali lemma annotated training dataset and explore the supervised techniques. Eventually we come up with a deep neural lemmatizer and evaluate it on several European languages alongside Bengali and Hindi.

Apart from lemmatization, we address another two morphological tasks: (*i*) morphological tagging and (*ii*) inflection generation. Morphological tagging involves determining several attributes such as case, degree, gender, POS etc. of an in-context word. As individual word forms in the inflected languages hold considerable linguistic information, so efficiently modelling their morphological properties helps a number of NLP applications such as MT, information extraction (IE) etc. Secondly, we target inflection generation, the goal of which is to produce the variant of a root word, given the morpho-syntactic descriptions of the target word. For example, given the source form 'communicate' and the features 'V;3;SG;PRS' (verb, third person singular present form), one has to predict the target form 'communicates'. This problem by nature requires supervision in the form of a set of lemmas and their corresponding features manually tagged to the respective target strings. This is the initial stage of morphological paradigm completion where incomplete inflection tables are to be filled using few complete tables provided as training data. We explore morphological tagging on 5 Indic languages (Hindi, Marathi, Sanskrit, Tamil and Telugu) and 2 severely resource scarce languages (Coptic and Kurmanji). Whereas, experiments for inflection generation are carried out on the CoNLL-SIGMORPHON shared task datasets (Cotterell et al., 2017) that covers 52 languages world-wide. Out of them, 3 languages are Indic (Bengali, Hindi and Urdu).

The rest part of this chapter is arranged as follows. In the following two sections, the previous studies on lemmatization, morphological tagging and inflection generation are discussed. Next we point out the key contributions of this thesis. Finally, the organization of the remaining chapters along with their brief description are provided.

1.2 Foregoing Works on Lemmatization

Previous efforts on developing lemmatizers can be divided into two principle categories - (*i*) rule/heuristics based and (*ii*) statistical learning based methods. The rule-based

approaches are usually not portable to different languages due to lack of generalization property. In this category the initial work was done by Koskenniemi (1984) who introduced a two-level automata model performing morphological transformation between lexical and surface forms of stems. The changes in morphology required to transform one form to the other, are defined by linguistic rules which are implemented using finite-state automata. The rules are deterministic specifying how the surface and lexical forms correspond to each other. In addition, necessary context conditions at character level are considered during the transformation process to realize the rules. Following this work, Plisson et al. (2004) proposed another rule-based approach to word lemmatization. Their work finds the association of suffix pairs to find the lemma of a surface word. Given a suffix pair and an inflected word, the first suffix is to be removed from the word and then adding the second one to it determines the normalized word form. A look-up table is made of words along with their associated suffix pairs from annotated training set. Next, the authors used the look-up table to compare two methods - one is based on control flow of predefined rules and the other is a step-by-step ripple down rules reduction algorithm. However, the common bottleneck in these type of approaches is that the rules vary substantially across languages, which necessitates involvement of language experts.

Later, statistical learning based lemmatizers (Chrupala et al., 2008; Toutanova and Cherry, 2009; Gesmundo and Samardzic, 2012; Müller et al., 2015; Nicolai and Kondrak, 2016) become popular because of their portability to new languages with minimum linguistic intervention. All of them require prior training dataset to learn morphological patterns. These methods can be further categorized depending on whether context of the focus word is considered or not. Among the notable research works those consider in-context lemmatization, Chrupala et al. (2008) presented Morfette that does joint morphological tagging and lemmatization from annotated corpora. Their system is modular and data driven, trained by the Maximum Entropy classifier to predict lemmas and morphological tags. Moreover, it has an additional support to provide probability distribution over sequences of tag-lemma pairs. One thing is to note that the supervised lemmatization methods do not try to classify the lemma of a given word form as it is infeasible due to having a large number of lemmas in a language. Rather, learning the transformation between word-lemma pair is more generalized and it can handle the unknown word forms too. Chrupala et al. (2008) recasts lemmatization as a classification

problem using class labels that represents the transformation from inflections to their respective lemmas. The mapping is defined by the shortest edit script (SES). A SES describes the shortest insertion/deletion instruction sequence that maps a string w to another string w'. An example of SES taken from Chrupala et al. (2008) is provided as follows. Let $< w, w' >$ be <'pidieron', 'pedir'>, then the SES between them is $\{< D, i, 2 >, < I, e, 3 >, < D, e, 5 >, < D, o, 7 >, < D, n, 8 >\}$. Here $< D, i, 2 >$ means deletion of character 'i' at position 2. Same as, $< I, e, 3 >$ denotes insertion of 'e' at position 3. Chrupala et al. (2008) designed the model presuming the suffixal nature of the inflectional morphology i.e. character level changes predominantly occur at the end of the words. Hence, strings are indexed reversely before computing the SESs. In this way the mapping is made generalized so that all different word-lemma pairs which follow unique SES belong to the same class, thus, resulting in the total number of classes to be significantly less. As the maximum entropy classifier allows incorporation of arbitrary number of features, the authors built a language independent and minimalistic feature set to make the feature vector of the focus word in context. Their feature set consists of word form, morpho-tag, suffixes and prefixes of length 1 to 7 and 1 to 5 and spelling pattern of the surface word. Finally, a beam search technique is used to find the most probable sequence of morpho-tag and lemma pairs for a given sentence. Evaluation is done on 3 morphologically rich languages namely, Polish, Romanian and Spanish. Error analysis shows that this method suffers due to the presence of named entities, ambiguity in suffixes and function words, and prefixal morphology.

Another important work by Toutanova and Cherry (2009) introduced a model for joint lemmatization and part-of-speech prediction. This model tries to find the dependencies between the possible POS tags and lemmas for a single word. The intuition behind this approach is that for different words having same lemma, there should be an overlapping between their tag sets. So, if the predictions of these two components (i.e. tagging and lemmatization) can be shared interchangeably, then it would be advantageous for both the tasks. The method exploits a lexicon containing morphologically analyzed words and some unlabeled data for learning the joint model. There are two modules in parallel: (i) a partially-supervised POS tagger and (ii) a string transducer for lemmatization. The POS tagger is trained using both the morphological lexicon and the unlabeled text. Whereas, only the lexicon is sufficient to train the lemmatizer but it can also use the unlabeled portion. The lemmatizer is basically a discriminative character transducer

which takes a surface word as input along with optional POS tag. Inspired by the model of Jiampojamarn et al. (2008) for letter to phoneme conversion, the transducer implements dynamic programming algorithm for monotone phrasal decoding. The algorithm uses three main feature categories- context, transition, and vocabulary (root list) features. Each of the features are assigned linear weights using an averaged perceptron for structure prediction (Collins, 2002). However, an additional burden is there for building the feature vectors. In the training set together with the gold word-lemma pairs, the step-by-step derivations from words to their corresponding lemmas are also required. As an example, for 'living' to 'live', the following steps ('l'→'l' ;'i'→'i' ;'v'→'v' ;'ing'→'e') should be there in the training data. Finally, the joint probability distribution of tag sets and lemmas for all words in the lexicon is learnt using log linear model over the feature vectors. The model is tested on English, Bulgarian, Czech, and Slovene. Experimental results reveal that these two tasks are mutually dependent and sharing information between them is beneficial for performance improvement.

Following Chrupala et al. (2008), Gesmundo and Samardzic (2012) took a similar approach to formalize lemmatization as a tagging task. They encode word to lemma transformation in a single label thus making the task trainable with any supervised sequence tagging model. In the paper, the authors reasonably argued that considering the problem straight as lemma tagging must be avoided due to two facts - (*i*) the tag set size would be comparable to the size of vocabulary in the language and (*ii*) common morphological transformations that are required for tackling the unseen words would not be addressed. The words are mapped to lemmas by introducing label set. A label is a tuple consisting of four parameters: lengths of the prefix and suffix to be removed from the source word and the new prefix and suffix to be added with it. For a particular language the set of unique labels are extracted from training data. Compared to the model of Chrupala et al. (2008), this method can handle both prefixal and suffixal changes in the transformation process. Training is accomplished by the bi-directional tagger with guided learning proposed in (Shen et al., 2007). The feature set of the model contains the surrounding words and their predicted lemmas, POS tag of the focus word, prefix and suffix of the focus word etc. The lemmatizer is evaluated on eight European languages namely, Czech, English, Estonian, Hungarian, Polish, Romanian, Serbian and Slovene. The paper demonstrates empirically that in spite of not using POS information, high quality lemmatization can be achieved.

Among the in-context statistical lemmatizers, the most remarkable one was developed by Müller et al. (2015). They proposed Lemming, which is a joint morphological tagging and lemmatization system that operates on raw surface words at sentence level. In contrast to the previously stated works, the distinguishing factors of Lemming are as follows. The Transformation process between word to lemma is shaped into a generic tree structure. The tree is referred as edit tree which encodes every morphological changes (i.e. addition, removal and substitution of character strings at pre, in and post position of the source word) and naturally, can handle the unseen words. At first, the tag sequence is modeled using Marmot Tagger (Mueller et al., 2013) and next, tagging components and lemmatization are combined into a tree structured conditional random field (CRF). Given a word-lemma pair, an exhaustive set of features is incorporated into the model. The feature set contains primarily four types of features. They are - (*i*) edit tree features including the edit tree between the source word and the target lemma, the pair of tree and the source word, affix of the source word up to maximum length 10 (*ii*) alignment features that accounts the character level alignment of the source word and its lemma (*iii*) lemma features which covers the affix of the lemma and the lemma itself (*iv*) dictionary features checking whether the lemma is present in the lexical database. In this joint model, two weight vectors are learnt from the training data. The first one considers the features of the focus word and the second one is for the tagging components. Eventually, the weighted feature vectors are added to formulate the combined tag-lemma probability. In the paper, the authors report the experimental results on six languages - English, German, Czech, Hungarian, Latin and Spanish. For all the mentioned languages, Lemming sets the new state-of-the-art.

A different strategy was taken by Nicolai and Kondrak (2016) that used inflection tables for stemming and lemmatization of isolated words. Their model is a supervised and discriminative string transducer. Though supervised models need a large amount of annotated training data, but the problem is overcome by exploiting semi structured and crowd-sourced partial inflection tables available from the world wide web. Initially, a word to stem model is attempted. To train the transducer, a set of aligned word-stem pairs is required. The alignment tells where to insert the boundary markers which divide the source words into stems and affixes. The aligned word-stem pairs are extracted from the inflection tables in an unsupervised way. Since several inflection tables show same morphological patterns, so each word is associated with an abstract tag sequence

consisting of <STEM> and <AFFIX> tags. Then the M2M aligner designed by Jiampojamarn et al. (2007) is employed for boundary alignment. The principle of the M2M aligner is based on expectation maximization algorithm to determine the maximum probable alignment between character sequences and the tags. Also, the aligner learns the stems within a single inflection table and the affixes across multiple tables. Once the aligned source and target pairs are made, the transduction model is trained by adapting the DIRECTL+ tool (Jiampojamarn et al., 2010). Next, using the word to stem model three different lemmatization methods are explored. In the first method, a source word is segmented into stem by the trained transducer. Next its stem to lemma transformation is learnt by another character level aligner. The second one is a slight variation of the first model. Here instead of stem to lemma aligner, stemma to lemma alignment is used. The Stemmed version of a lemma is called stemma. Word to stem model is converted to word to stemma model. The final version directly learns the transformation between source word to lemma. For experimentation, four European languages are chosen. They are - English, Dutch, German and Spanish. Experimental results outperform two baselines Lemming (Müller et al., 2015) and Morfette (Chrupala et al., 2008).

1.2.1 Previous Works on Bengali and Hindi

Since the major focus of this thesis is on the Indic languages (especially Bengali and Hindi), this section revisits the previous efforts on developing lemmatizers for them. It would not be an overstatement to say that there have been little works on lemmatization so far. Researchers working on the Indic languages mostly followed the way in which English is processed and as a result no good lemmatization algorithm is available for such languages. Previous studies on the Indic languages put stress on developing stemmers, mostly in the context of information retrieval (IR) (Majumder et al., 2007; Pandey and Siddiqui, 2008; Dolamic and Savoy, 2010; Loponen and Järvelin, 2010; Loponen et al., 2013; Paik and Parui, 2011; Paik et al., 2011; Sarkar and Bandyopadhyay, 2012a,b; Paik et al., 2013; Ganguly et al., 2007). In Bengali, the notable works in morphological analysis are done by (Bhattacharya et al., 2005; Dasgupta and Ng, 2006; Faridee et al., 2009; Senapati and Garain, 2012; Bhattacharyya et al., 2014). The study in (Bhattacharya et al., 2005) describes the procedures of inflectional morphology synthesis of the Bengali noun, pronoun and verb systems. It is basically the reverse process of lemmatizing the

textual surface words i.e. given a root word along with its morphological attributes, the system synthesizes the corresponding inflected form. The authors exhaustively sort out the inflection patterns for the root words belonging to individual categories. For example, they found that there are 160 different inflected forms of Bengali verb roots which are categorized into 24 classes based on the vowels of the last two syllables. Suffixes that can appear after a root, are differentiated on the basis of tense, aspect, person and modality. Irregular patterns are handled separately. As an inflection is formed often by adding multiple suffixes to a root word, so the ordering of the suffixes and their concatenation rules are determined precisely. The whole system is embedded into a finite state machine (FSM) where the verb roots are represented as regular expressions. Following the same way, different FSMs are built for the noun and pronoun systems. The bottlenecks of this work are that the words are considered in isolation. Moreover, it's a process of inflection synthesis, not lemmatization.

Dasgupta and Ng (2006) did an unsupervised morphological analysis for segmentation of Bengali words into the constituent morphemes as prefix, stem and suffix. The method has two steps. Firstly, by exploiting a large vocabulary of the language, frequent affixes are induced. In the second step, using the induced affixes, a new word is segmented. The induction procedure computes the frequency of words and sub-words in the vocabulary to identify the valid morphemes. Then, the morphemes are categorized into stems and affixes by counting two different metrics. One is the number of different words to which each affix is attached. Another one counts the number of different affixes attached to a stem to measure its generative strength. Finally segmentation for a new word is done by arranging it as a sequence of prefixes, stems and suffixes. The method basically works as a stemmer and does not consider contextual information.

Faridee et al. (2009) proposed another rule based Bengali morphological analyzer using finite state machine. Only standard colloquial Bengali words are handled by the FSM, though Bengali has multiple dialects. The rules are taken from different grammar books and Wikipedia. The analyzer sets six basic inflection rules for verbs depending on tense, aspect, modality and person. All total, 59 different inflections from a root verb are identified. For noun roots, their animacy levels are considered to generate the variant forms. A significant point to note here is that the lemmas returned by the analyzer are not as the standard forms given in the dictionary thus making it difficult to map them

with the lexical resources. The authors choose the present indefinite, second person informal form as the lemma.

In the work of Bhattacharyya et al. (2014), a trie-based lemmatization method was introduced. This method only considers suffix based morphology. Initially a trie is built taking words from WordNet. Then for an input word, the trie is navigated following the unicode characters present in the input word. Two different searching strategies are taken after the navigation ends. The first one returns all WordNet words having maximum prefix match with the input words. The second one backtracks up to the n levels previous to the maximum matched prefix where n is given by the user. In case there are more than one candidates returned from the trie, the candidates are sorted in ascending order according to the length. The method does not consider the context of the target word and the evaluation method considers that a surface word is correctly lemmatized if the appropriate lemma occurs within the top 10 candidates.

In Hindi morphology, the works by (Bögel et al., 2007; Goyal and Lehal, 2008; Kumar et al., 2012; Paul et al., 2013) are important. Bögel et al. (2007) developed a morphological analyzer for Hindi and Urdu. As these two languages are structurally almost identical, in spite having two distant writing systems (Hindi is from left to right and Urdu is the reverse of it), the analyzer builds a common ASCII transliteration system for both the languages. Then, from the ASCII text, the reverse mappings to both Hindi and Urdu are designed. The authors discussed the various morphological challenges in building the transliterator - such as tokenization problem in Urdu, identification of frequent reduplicated words etc.

Goyal and Lehal (2008) built a morphological analyzer cum generator for Hindi. It has been created as a part of a translation system from Hindi to Punjabi. The analyzer analyzes a word and returns its constituent morphemes and their categories. Whereas, the generator generates the inflection from a given stem/root and its morphological attributes. The authors divide the Hindi vocabulary in different classes based on basic parts of speech like Noun, pronoun, verb and adjective. Thereafter, every class is grained into their respective attributes. For example, the verb class contains six attributes such as gender, number, person, tense, aspect and modality. Each of the attributes are further segregated according to various features to make the complete paradigm of the word class. The whole system takes the form a huge database where all the possible

variants for every root word are stored. However, the authors agreed that the software designed to handle the database is not user friendly and difficult to handle. Following Goyal and Lehal (2008), Kumar et al. (2012) also develop an analyzer cum generator. To design the system, a dictionary of root words, a file containing manually designed rules for handling morphology, and another dictionary of irregular word patterns are required. The method builds a finite state morphological transducer that makes use of the rule files and the root lexicon. Irregular words are treated separately. To build the transducer, the authors take the help of SFST tool (Schmid, 2006).

Paul et al. (2013) proposed a rule based Hindi lemmatization. At first, 112 suffixes are manually produced from a corpus of 20,000 sentences. Next the rules are generated for extracting suffixes from inflected words and if necessary, then adding new characters to it. When an input word is encountered, it is first searched in the database. If the word exists there, it is returned as the output. Otherwise, the rules are used for suffix elimination from the word. After that, if the truncated word is a legitimate root found in the database, the process stops. Else, new characters are added to it to obtain a root.

1.3 Literature Survey on Morphological Tagging and Inflection Generation

Morphological tagging is a relatively new and less explored research area in NLP where the words in a sentence are analyzed for finding their morphological properties. In this field, the important works were done by (Mueller et al., 2013; Buys and Botha, 2016; Heigold et al., 2016, 2017; Cotterell and Heigold, 2017). Mueller et al. (2013) proposed a higher-order CRF based tagger that is capable of doing full POS+MORPH analysis. The objective function of their model is modified than that of the normal higher-order CRFs to make the model faster as well as applicable to bigger tagset. The tagger was evaluated on 6 languages - Arabic, Czech, English, German, Hungarian and Spanish. Buys and Botha (2016) developed an indirectly supervised cross-lingual tagger which needs aligned bitext for projecting the tags. The projection from a high-resource language sets constraints for the possible tags for a given word in the low-resource languages. The authors proposed a discriminative embedding-based tagging model with rank-based learning. The experimentation was done on 11 European languages. The

works of Heigold et al. (2016, 2017) are the first character-level neural morphological taggers. They explored different neural architectures such as convolutional neural network (CNN) (Krizhevsky et al., 2012), long-short-term-memory (LSTM) (Graves, 2013), bidirectional long-short-term-memory (BLSTM) to embed the words into character-level vectors. Next, the character-vectors in a sentence are fed to another BLSTM to classify the tags. Along with the character-vectors, pre-trained word vectors are also used as supplement in their model. In their experiments, 14 languages were taken as the references. Yu et al. (2017) used CNN both for character-level embeddings of words and tag classification. Their model has a character composition model and a context encoding model. The initial CNN builds features using character n-grams and the second level CNN considers word n-grams to exploit contextual information. Very recently, Cotterell and Heigold (2017) proposed the transfer learning framework for cross-lingual, character-level neural tagging. This model learns the character-level embeddings over multiple languages, which allows sharing of common syntactic features. Next, a joint loss function is optimized combining both high-resource and low-resource taggers. Multiple softmax architectures are introduced to learn language-specific properties. Evaluation of the cross-lingual model is done on 18 languages taken from 4 language families.

In this thesis, we conduct morphological tagging experiments on 5 Indic languages - Hindi, Marathi, Sanskrit, Tamil and Telugu and on 2 severely resource-poor languages namely Coptic and Kurmanji. Research in morphology for the 5 concerning Indic languages has not advanced much beyond the rule-based approaches. The previously built morphological analyzers in Hindi (Goyal and Lehal, 2008; Kumar et al., 2012; Kanuparthi et al., 2012; Kumar and Kaur, 2013; Rastogi and Khanna, 2014; Kumar et al., 2017), Marathi (Bapat et al., 2010; Dabre et al., 2012, 2013; Ravishankar and Tyers, 2017) and Sanskrit Bharati et al. (2006); Jha et al. (2009) rely on word segmentation into constituent morphemes. For Tamil and Telugu, Lushanthan et al. (2014); Sunitha and Kalyani (2009) proposed rule-based analyzers. An unsupervised word segmentation approach was proposed by Kumar et al. (2015) using adaptor grammars for Dravidian languages. Later, Kumar et al. (2017) released a training corpus to segment the words to be used as features for POS tagging. The research described above mainly focus on the word segmentation rather than the tagging task.

Research in inflection generation has recently drawn the attention of the NLP community. The task finds the changes in the structure of a word depending on its morpho-syntactic properties, which enables insight to leverage morphology for high-level NLP problems. In this area, several studies have been published so far. Among them, the initial efforts using traditional machine learning techniques such as semi-Markov model, discriminative string transduction etc. were reported by Durrett and DeNero (2013); Hulden et al. (2014); Nicolai et al. (2015). All these works explored on Czech, Dutch, Finnish, French, German and Spanish. Recently Cotterell et al. (2016, 2017) have started the series of shared tasks on universal morphological reinflection. In the year 2016, the problem statement was to identify the reinflection of an already inflected word form. For ten languages including Arabic, Finnish, Georgian, German, Navajo, Russian, Spanish, and Turkish, the morphological datasets were released. Out of nine diverse participating systems, the sequence-to-sequence learning based neural models (Aharoni et al., 2016; Kann and Schütze, 2016) produced the best result. In 2017, the task covers 52 languages world-wide and for this time also, neural strategies by Bergmanis et al. (2017); Makarov et al. (2017); Kann and Schütze (2017) became the top performers. Apart from the mentioned studies, the works of Kann et al. (2017a,b) also introduced multi-source sequence-to-sequence network and cross-lingual transfer learning framework.

1.4 Contribution of the Thesis

As discussed in the previous section, there are several pitfalls of the previous research works on lemmatization for major Indic languages. Some methods are based on language specific rules, hence cannot cope up with the dynamic characteristics of the languages. As new words come into the languages, it may demand inclusion of new rules and their analysis. Under this circumstance, simple language independent and less resource contingent methods are highly needed to tackle the wide morphological variations. We introduce minimum resource dependent, generalized and easily implementable methods which build the initial ground for advanced research on Indic languages NLP. At first, an unsupervised lemmatization algorithm is developed that lemmatizes surface words considering their contextual information. A machine readable dictionary, a POS tagger and a valid suffix list of the concerned language are the only requirements of the model. The orthographically similar roots for an input word are extracted from the

dictionary using the string distance measure proposed by Majumder et al. (2007). Then the extracted roots are pruned using POS of the input word and the valid suffixes to determine the lemma. The semantic similarity between the surface word and the root words is evaluated using Lesk based word sense disambiguation algorithm (Lesk, 1986; Banerjee and Pedersen, 2002) that considers the context and the sense definitions provided in the dictionary. The algorithm is applied on Bengali and further its effectiveness is tested for Bengali WSD. As there was no such lemmatization and WSD test datasets in Bengali, two small datasets are built for evaluation purpose. Lemmatization as a pre-processing task empirically proves beneficial for WSD.

Next the previous method is extended with the following modifications. Instead of finding the orthographic similarity between two words by the string distance measure of YASS stemmer (Majumder et al., 2007), a novel graph based distance is introduced to choose the potential roots of a surface word. In addition, We make use of the recent development in the area of machine learning for NLP i.e. word embedding technique which has been established as remarkably effective in semantic processing tasks (Mikolov et al., 2013a,b). Semantic similarity between two words is measured by the cosine distance between their corresponding vector embeddings. The main rationale behind using the word embedding is that as a root and its variants are semantically related, so their corresponding vectors in the embedded space should have high cosine similarity. This method is tested on Bengali and Hindi. The proposed graph based orthographic distance gains better lemmatization accuracy than the string similarity based measure.

Following the advancement of word embedding technique, application of neural network based models in NLP tasks has been immensely popular. Following the trend, we focus on designing efficient neural architecture for lemmatization. In this area, our first experiment is carried out building a small neural network to generate lemma for an inflected word. Embeddings of the focus word along with its contextual neighbours are given input to the network. Cosine distance is used to measure the loss and the weights are updated using standard back-propagation rule. The model is tested on Bengali. However, satisfactory result is not obtained indicating that a huge amount of training data is needed to efficiently induce the lemmas.

Our final lemmatization model uses deep recurrent neural network. In our previous approach, the problem was formulated as a lemma generation task, for which the target

domain of the network becomes huge. Now the goal is changed as to classify the proper transformation that applied on a surface word, generates the appropriate lemma. This strategy reduces the number of classes by a huge margin. The transformation between an inflection to its lemma is represented by an edit tree which encodes all necessary string operations (insertion, deletion and substitution) within it. Thus, the problem is formulated as to classify the appropriate edit tree for a given surface word. A two level bi-directional gated recurrent model has been developed to lemmatize sentences. Compared to the state-of-the-art supervised lemmatizers, the proposed model does not need feature definitions and other morphological attributes except the gold lemmas. To run the model on Bengali, a lemma and POS annotated Bengali training data is built that consists of $1,702$ sentences having $20,257$ words. As Bengali is a resource scarce language, this dataset will definitely help the subsequent research on Bengali NLP. Apart from Bengali and Hindi, the model is evaluated on 7 European languages and it over-performs two state-of-the-art baselines namely Lemming by Müller et al. (2015) and Morfette by Chrupala et al. (2008) on all the languages except Bengali.

Next, we target two different problems - (i) morphological tagging and (ii) inflection generation. In morphological tagging, the state-of-the-art methods of (Heigold et al., 2016, 2017; Cotterell and Heigold, 2017) used the following common model architecture. At first, character-level embeddings of words are generated using either convolutional/recurrent models to make their syntactic representations. Then, for a sentence, its constituent word vectors are passed through another RNN to predict their morphological tags. So far the tagging problem has been formulated mostly as a single label classification task where all the morphological attributes of a particular word are jointly considered as a single tag. Although representing every tag as the concatenation of universal key:value pairs is a standard annotation scheme (Sylak-Glassman et al., 2015), it suffers from a major drawback. In a supervised system, only those tags that are present in the training data provide the possible morphological analysis for any arbitrary word and hence the missing legitimate combinations of the component attributes are not addressed. In this plight, we make the following key changes to the generic neural tagging model: (i) Instead of targeting the problem as a single label classification, we view it as a multi-label classification framework where each label corresponds to a universal morphological key and the distinct values of a key stand for the number of classes under the respective label. (ii) We start with the rationale that, to predict

the morphological features of a word, considering its longer context such as the whole sentence is not justified. Hence, to design the tag classification module, we propose a convolutional neural network (CNN) based architecture that captures the local context of the focus word where the context length is a model parameter. Though in the foregoing work of Yu et al. (2017), CNN is used for tag classification, but their proposed model considers relatively longer context by taking seven words on both sides of the focus word. However, we find that with the increase of the context length, tagging accuracy decreases monotonically. (*iii*) Our model is evaluated on 3 languages from Indo-Aryan family (Hindi, Marathi and Sanskrit), 2 languages from Dravidian family (Tamil and Telugu) and 2 severely resource scarce languages (Coptic and Kurmanji). The last two are non-Indic but considered for experimentation to test the model in highly resource-poor condition. (*iv*) Following the recent work of Cotterell and Heigold (2017), we incorporate the cross-lingual, character-level transfer learning strategies in our experiments. The BLSTM-BLSTM model described in their paper is compared with our proposed BLSTM-CNN model using language-universal and language-specific softmax architectures. Under each language family, the member languages are jointly trained at character level to share the common syntactic features amongst them. Additionally we explore the impact of using the distributional similarity of words along with the character-level embeddings for morphological tagging. Though in the literature, Inoue et al. (2017) explored multi-label learning for Arabic morphological tag classification, our work is the first of its kind for Indic languages.

Finally, this thesis addresses the problem of morphological inflection generation. We participated in the CoNLL-SIGMORPHON 2017 shared task on morphological reinflection (Cotterell et al., 2017). The system submitted in the competition resembles to our deep neural lemmatizer that determines the edit tree for an input inflected word. As the task asks for supervised methods, training datasets are provided for 52 languages in which a group of words along with the morpho-syntactic features are tagged with the respective target forms. At first we extract the unique edit trees between the source and the target strings. These extracted trees become the class labels for our network which is trained on the <word,features $\rightarrow$ edit tree> pairs. The character level embeddings of source words are generated using a BLSTM and the obtained embeddings together with the features are passed to a dense network to classify the edit trees. For every

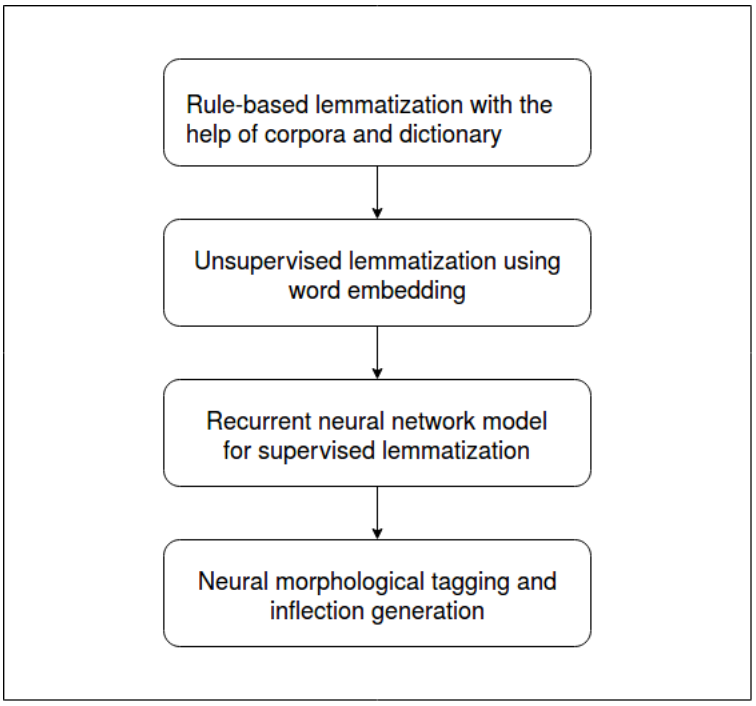


FIGURE 1.1: Overview of the thesis.

language, 3 different training sets are provided for high, medium and low resource conditions respectively. It is experimentally found that our proposed system performs most effectively in high resource scenario.

We pictorially represent the integration of the whole study in Figure 1.1.

1.5 Thesis Organization

In this section, the remaining chapters of the thesis are briefly summarized.

Chapter 2. In the second chapter a dictionary-based lemmatization algorithm is presented and it is evaluated for Bengali. Effectiveness of the algorithm for Bengali WSD is also investigated. One of the key challenges for computer processing of highly inflected languages is to deal with the frequent morphological variations of the root words appearing in the text. Therefore, a lemmatizer is essential for developing NLP tools for such languages. In this experiment, Bengali which is the national language of Bangladesh and the second most popular language in the Indian subcontinent has been taken as a reference. In order to design the Bengali lemmatizer (named as BenLem), possible transformations through which surface words are formed from lemmas are studied so

that appropriate reverse transformations can be applied on a surface word to get the corresponding lemma back. BenLem is found to be capable of handling both inflectional and derivational morphology in Bengali. It is evaluated on a set of 18 news articles taken from the FIRE Bengali News Corpus (Majumder et al., 2010) consisting of 3,342 surface words (excluding proper nouns) and found to be 81.95% accurate. The role of the lemmatizer is then investigated for Bengali WSD. A set of 10 highly polysemous Bengali words is considered for sense disambiguation. The FIRE corpus and a collection of Tagore's short stories are considered for creating the WSD dataset. Different WSD systems are considered for this experiment and it is noticed that BenLem improves the performance of all the WSD systems and the improvements are statistically significant.

Chapter 3. The third chapter describes an unsupervised and language independent lemmatization method. Two highly inflected as well as low-resourced Indic languages namely, Bengali and Hindi are chosen for experimentation. To find the orthographic similar roots for an input surface word, a new trie searching strategy is introduced to find the potential candidates from dictionary headwords. We design different versions of the proposed method exploring the impact of expensive resources such as part of speech (POS) tags and semantic information of the root words on lemmatization performance. Additionally, for each version, two variants are proposed without/with incorporating word embedding technique. Experimental results show that in absence of any expensive resource, use of word embedding improves the accuracy but use of POS information along with the word embedding cannot yield any betterment. For Bengali and Hindi, our proposed method produces 82.70 and 73.49 percent accuracies respectively and achieves the new state-of-the-art in unsupervised lemmatization for the two languages.

We further propose a rudimentary neural lemmatization network which is language independent and supervised in nature. To handle the words in a neural framework, word2vec method of Mikolov et al. (2013a) is used to represent words as vectors. The proposed lemmatizer makes use of contextual information of the surface word to be lemmatized. Given a word along with its contextual neighbours as input, the model is designed to produce the lemma of the concerned word as output. A new network architecture is introduced, that permits only dimension specific connections between the input and the output layer of the model. For this work, Bengali is taken as the reference language. Evaluation method shows that the proposed lemmatizer achieves 69.57% accuracy and outperforms the simple cosine similarity based baseline.

Chapter 4. We introduce a composite deep neural network architecture for supervised and language independent context sensitive lemmatization. The proposed method considers the task as to identify the correct edit tree representing the transformation between a word-lemma pair. To find the lemma of a surface word, we exploit two successive bidirectional gated recurrent structures - the first one is used to extract the character level dependencies and the next one captures the contextual information of the given word. The key advantages of our model compared to the state-of-the-art lemmatizers such as Lemming (Müller et al., 2015) and Morfette (Chrupala et al., 2008) are - (i) it is independent of human decided features (ii) except the gold lemma, no other expensive morphological attribute is required for joint learning. We evaluate the lemmatizer on 9 languages - Bengali, Catalan, Dutch, Hindi, Hungarian, Italian, Latin, Romanian and Spanish. It is found that except Bengali, the proposed method outperforms Lemming and Morfette on the other languages. To train the model on Bengali, we develop a large gold lemma annotated dataset. The dataset (having $1,702$ sentences with a total of $20,257$ word tokens) and the implementation are publicly available[1], which is an additional contribution of this work.

Chapter 5. In the fifth chapter we explore several convolutional and recurrent neural network architectures for morphological tagging task. The task involves predicting morphological tag of an in-context word. We hypothesize that the tag of a word is dependent only on its local context instead of the entire sentence. In this light, usefulness of convolution operation for predicting the tags is explored. Our work also models morphological tagging as a multitask learning framework across keys. The experiments are conducted in 5 Indic languages, namely, Hindi, Marathi, Sanskrit, Tamil and Telugu, and 2 severely resource-scarce languages (Coptic and Kurmanji). The proposed model is compared with previous state-of-the-art and achieves insightful results. We further explore enriching the word embeddings by concatenating character level embeddings with distributional word vectors. This significantly improves the performance of the proposed model on the five Indic languages.

Next we present a LSTM-based system for morphological inflection generation. Given an input word and morpho-syntactic descriptions, the problem is to classify the proper edit tree that, applied on the input word, produces the target form. The proposed method does not require human defined features and it is language independent also. Moreover,

[1] http://www.isical.ac.in/~utpal/resources.php

no external data is used for data augmentation. We participate with our system in the SIGMORPHON 2017 shared task competition on morphological reinflection (Cotterell et al., 2017). From the results on the test sets, it is found that the proposed model beats the baseline on 15 out of the 52 languages in high resource scenario. But its performance is poor when the training set size is medium or low.

Finally, chapter 6 concludes the thesis and discusses the future scope for the related research.

Chapter 2

BenLem: an Unsupervised Lemmatizer and its Role in WSD

2.1 Introduction

This chapter introduces BenLem - a language independent lemmatization algorithm developed for Bengali, which is considered as a highly inflected, low-resourced Indic language. The algorithm requires a machine readable dictionary, a POS tagger and a valid suffix list of the concerned language. In designing the lemmatizer, we try to capture the possible transformations through which the surface words are generated from their respective lemmas. However, these transformations can be hardly defined by a single rule. They are broadly categorized into 4 cases as follows with the examples in Bengali: (*i*) the surface word can be obtained by adding a valid suffix to its lemma (e.g. 'করার'[1]/karaara = 'করা'/karaa + 'র'/ra) (*ii*) subtraction of a valid suffix from the lemma may result in the surface word (e.g. 'কর'/kara = 'করা'/karaa - 'া'/aa) (*iii*) unlike any direct addition or subtraction, their combination (at first, subtraction and then addition) may produce the surface word (e.g. 'করছেন'/karchhen = ['করা'/karaa - 'া'/aa] + 'ছেন'/chhen) (*iv*) None of the above transformations works (e.g. 'অভ্যাস'/abhyaas → 'অভ্যেসের'/abhyeser, 'পাকা'/paakaa → 'পেকে'/peke). So, if the above transformations can be determined in an unambiguous manner then lemmas can be found by applying corresponding reverse transformations on the surface words. BenLem works on this

[1]Throughout the thesis, we used the "Indian languages TRANSliteration" (https://www.aczoom.com/itrans/online/) for presenting the Unicode characters.

philosophy and additionally considers contextual information such as the sense of the surface word. The following section presents the algorithm.

2.2 The BenLem Algorithm

Let w be a surface word to be lemmatized. Two operations are defined on w as follows:

- $CON(w)$ denotes the bag of words in the current context of w.
- $POS(w)$ denotes the part of speech of the word w in the current context.

BenLem requires two resources: (i) a valid suffix list of the language and (ii) a dictionary.

Bengali Suffix List: For preparing the suffix list in Bengali, we follow the work by Paik and Parui (2011) where the set of all suffixes of length n ($n = 1, 2, ...$) is generated from the lexicon (of size more than 400K as available from the FIRE Bengali News Corpus[2]) by grouping words that share same suffix and the number of words in a group becomes the frequency of the corresponding suffix. A suffix is a valid suffix of the language if its frequency in the lexicon is larger than a certain cut-off threshold. The authors showed that suffixes selected purely based on their frequencies in a lexicon have high recall but very low precision. Although most of the valid suffixes in Bengali belong to the selected set, several invalid suffixes are also part of this set. So, we put manual effort (one linguist was involved) to remove the invalid suffixes to get a set of valid Bengali suffixes. Let $S = \{s_1, s_2, ..., s_m\}$ be the set of valid suffixes. In our experiment, S contains 264 entries. However, this list is not exhaustive and with the availability of new corpora, it can be updated.

Bengali Dictionary: We use the Bengali dictionary available from the University of Chicago[3]. There are 47,189 distinct headwords in the dictionary. The dictionary is organized in the following format. Let $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ be the set of all headwords present in the dictionary. The structure of each headword d_i is as follows: $< d_i, (c_1, pos_1, pos_2, \ldots), (c_2, pos_1, pos_2, \ldots), \ldots, (c_k, pos_1, pos_2, \ldots) >$, where the word d_i has k different senses represented by $(c_1, c_2, \ldots, c_k)$ and for each sense c_i, all possible

[2] http://fire.irsi.res.in/fire/static/data
[3] http://dsal.uchicago.edu/dictionaries/list.html

parts of speech are denoted by $(pos_1, pos_2, \ldots)$. Each c_i is actually a short description of a particular sense and hence, treated as a sense bag (i.e. a bag of words representing a particular sense). The following operations are defined on this dictionary.

- $POS_{dic}(d_i)$ returns the set of all the parts of speech that d_i can have.
- $POS_{sense}(d_i, c_j)$ returns the set of all the parts of speech of the sense c_j for the dictionary word d_i.
- $OverlapScore(d_i, c_j, w)$ gives the number of words common between c_j of d_i and $CON(w)$ if $POS(w) \in POS_{sense}(d_i, c_j)$ or 0, otherwise.
- $MaxOverlapScore(d_i, w)$ is the maximum among the numbers returned by $OverlapScore(d_i, c_j, w)$ $\forall j$.

BenLem algorithm makes use of one another important component, namely a distance measure as explained below.

Distance Measure: If w_1 and w_2 be two words, then $Dist(w_1, w_2)$ denotes the string distance between w_1 and w_2 and to define the function, $Dist$, we have selected the distance measure used by Majumder et al. (2007) which rewards long matching prefixes and penalizes an early mismatch as follows. Let the length of two strings X and Y be $n+1$ (where n is a positive integer and if strings are of unequal length, null characters are added at the end of the shorter string to make them equal) and let m denotes the position of the first mismatch between X and Y (i.e. $x_0 = y_0$, $x_1 = y_1, \ldots, x_{m-1} = y_{m-1}$, but $x_m \neq y_m$). Now, $Dist$ is defined as follows: $Dist(X, Y) = \frac{n-m+1}{m} \times \sum_{i=m}^{n} \frac{1}{2^{i-m}}$ if $m > 0$, ∞ otherwise. The rationale behind choosing this distance measure is that for suffixing languages like Bengali, morphologically related words typically share a common prefix. So for the two words w_1 and w_2, if $Dist(w_1, w_2)$ is less than a threshold δ, then we can say that they may be morphologically related.

2.2.1 The Lemmatization Algorithm

Input: A surface word w, $CON(w)$ and $POS(w)$.

Output: the lemma l of w.

1. If $\exists\ d_i\ \in \mathcal{D}$, such that w equals to d_i and $POS(w)\ \in\ POS_{dic}(d_i)$, then w is taken as the lemma l. Else, go to step 2.

2. Find the set D_p ($D_p\ \subset\ \mathcal{D}$), such that $Dist(w, d_p) < \delta$ and $POS(w)\ \in\ POS_{dic}(d_p)$ $\forall\ d_p\ \in\ D_p$. Here D_p is the set of potential dictionary headwords that are close to w as well as their possible parts of speech also include the part of speech of w in the current context. If $D_p = \emptyset$, then w is returned as the lemma l. Otherwise, go to step 3(a).

3. (a) For each $d_p\ \in\ D_p$, if $\exists\ s_x\ \in\ S$ such that $w = d_p\ \pm\ s_x$, then d_p is added to a set D'_p. If $D'_p = \emptyset$, then execute step 3(b); else, go to step 3(c).

   (b) For each $d_p\ \in\ D_p$, if $\exists\ s_y, s_z\ \in\ S$, such that $w = (d_p\ -\ s_y)\ +\ s_z$, then d_p is put in the set D'_p.

   (c) If $D'_p = \emptyset$, then go to step 4. Otherwise, D'_p is the set of possible lemmas of the word w. Return $d_q \in D'_p$ as the lemma l, such that $Dist(w, d_q) \leq Dist(w,$ $d_r),\ \forall\ d_r\ \in\ D'_p$.

4. When none of the previous transformations could produce the lemma, the last step examines if any potential headword in D_p has a sense similar to that of w in the current context. It returns $d_s \in D_p$ as the lemma l if $MaxOverlapScore(d_s, w) \geq MaxOverlapScore(d_p, w)\ \forall\ d_p \in D_p$. If more than one such d_s is found, then the one having the minimum distance with w measured by $Dist$ is returned as the lemma. If $MaxOverlapScore$ is zero for all the headwords in D_p, then w itself is returned as the lemma.

We present examples of 4 surface words along with their contexts, which are successfully resolved by different steps of BenLem algorithm respectively.

1. <Target word: 'ছেলে'/chhele; Context: 'ঘরের ছেলে/chhele'> — 'ছেলে'/chhele is a dictionary headword and its part of speech in the given context (i.e. POS('ছেলে'/chhele)) is noun. As noun belongs to the set POS_{dic}('ছেলে'/chhele), so the target word form itself is considered as the lemma by Step 1.

2. <Target word: 'দলে'/dale; Context: 'দলে/dale ভারী'> — 'দল'/dala is the appropriate lemma of 'দলে'/dale in the present context. POS('দলে'/dale) is noun which belongs

to POS_{dic}(‘দল’/dala). It is found that ‘দল’/dala + ‘ে’/e produces ‘দলে’/dale where ‘ে’/e is a valid suffix. Hence, in this case, Step 3(a) resolves the surface word.

3. <Target word: ‘পারলেন’/paarlen; Context: ‘তিনি করতে পারলেন/paarlen’> — In this context, the right lemma of ‘পারলেন’/paarlen is ‘পারা’/paaraa. POS(‘পারলেন’/paarlen) is verb and it belongs to POS_{dic}(‘পারা’/paaraa). ‘পারা’/paaraa − ‘া’/aa + ‘লেন’/len produces ‘পারলেন’/paarlen. Both ‘া’/aa and ‘লেন’/len are valid suffixes in Bengali. Therefore, this case is resolved by Step 3(b).

4. <Target word: ‘এগিয়ে’/egiYe; Context: ‘সামনে/saamne এগিয়ে/egiYe যাওয়া/JaaoYaa’> — ‘এগনো’/egano is the appropriate lemma of ‘এগিয়ে’/egiYe in the given context. One of the dictionary senses of ‘এগনো’/egano is ‘সামনে/saamne যাওয়া/JaaoYaa’. POS(‘এগিয়ে’/egiYe) is verb that belongs to POS_{sense}(‘এগনো’/egano, ‘সামনে/saamne যাওয়া/JaaoYaa’). Step 4 would be fruitful here as there are two overlapping words (‘সামনে’/saamne and ‘যাওয়া’/JaaoYaa) between the context and the sense of the lemma.

2.3 Experimental Setup

The experiments consider several aspects starting from developing an annotated dataset for evaluating the lemmatizer to the development of Lesk based Bengali WSD systems and then preparing a sense annotated dataset for evaluating the WSD systems and finally, investigating the role of the lemmatizer for improving WSD accuracy. Initially, the lemmatizer is evaluated and the different stages of the lemmatization algorithm (BenLem) are analyzed. BenLem makes use of a parameter called δ used by the distance function $Dist$ (Step 2 of the algorithm in Section 2.2.1). At first, we discuss how the value of this parameter is found and the subsequent sections discuss about the preparations of the datasets, the Bengali WSD systems and evaluation of the lemmatizer and the WSD systems.

2.3.1 Computation of δ

To determine the value of δ, the threshold used by the distance function $Dist$, we initially took the longest suffix from the suffix list. In our suffix list, the longest suffix is

‘ইতেছিলাম’/itechhilaam. Then we calculate the value of the *Dist* function between each surface word ending with ‘ইতেছিলাম’/itechhilaam and its corresponding lemma and note the maximum distance. In this experiment, we find this value to be 7.96875 (between ‘খাইতেছিলাম’/khaaitechhilaam and ‘খাওয়া’/khaaoYaa). So, we choose 7.97 as the value of δ for our experiments. This relatively high threshold allows many (noisy) dictionary words but hardly misses a viable candidate. Many of these noisy candidates are later pruned based on the matching of parts of speech (refer Step 2 of the algorithm in Section 2.2.1).

2.3.2 Datasets for Evaluation

To evaluate the performance of the proposed lemmatization algorithm, we prepare a lemma-annotated dataset which is built using 18 newspaper articles selected randomly from the FIRE Bengali News corpus. Altogether, there are 6,314 space separated tokens out of which 3,342 are either dictionary words or their morphological variations. The remaining tokens are either proper nouns or punctuation marks which were not considered for lemmatization. All the surface words in this dataset are manually tagged with their parts of speech (in the given context). For the 3,342 tokens, the corresponding gold lemmas are also given. We take linguist's help for doing this POS and lemma annotation. As only one linguist was involved, we have not got any chance for measuring the inter-annotator agreement.

To investigate the role of our lemmatizer for WSD, we have selected ten highly polysemous words from the dictionary. Text pieces are selected from the FIRE Bengali News Corpus and the collection of Tagore's short stories[4]. The chosen words, the number of senses for these words in the dictionary and the number of instances (frequency), sense entropy and the number of the most frequent sense of each of the ten words in the WSD test dataset are given in the Table 2.1. The words for which disambiguation is attempted are manually lemmatized and sense tagged in the dataset. Sense entropy of the words is computed as follows. If a word w has n senses defined in the dictionary, then the probability p_i of a sense c_i of w is calculated as the ratio of the number of times w is used in sense c_i to the total number of instances (frequency) of w in the dataset. Next, the summation $[\sum_{i=1}^{n}(p_i \times \log \frac{1}{p_i})]$ gives the sense entropy of w.

[4] http://www.rabindra-rachanabali.nltr.org/

Words	# Senses	Frequency	Sense Entropy	# Most Freq Sense
'করা' (karaa)[to do/to work/...]	25	204	1.73	115
'দেওয়া' (deoYaa)[to give/to offer/...]	34	80	2.63	20
'কাটা' (kaataa)[to cut/to break/...]	24	69	1.47	23
'রাখা' (raakhaa)[to keep/to obey/...]	26	58	2.26	12
'তোলা' (tolaa)[to lift/to raise/...]	31	56	1.69	21
'খোলা' (kholaa)[to unfold/to start/...]	19	52	1.68	18
'যোগ' (Joga)[to add/to join/...]	16	52	1.44	18
'পড়া' (pa.Daa)[to read/to fall/...]	38	41	2.26	12
'চলা' (chalaa)[to walk/to continue/...]	20	35	1.96	12
'ধরা' (dharaa)[to catch/to hold/...]	39	20	1.78	6

TABLE 2.1: Words Chosen for Disambiguation, Their Number of Senses in the Dictionary and Frequency, Sense Entropy and Number of the Most Frequent Sense in the Test Data

2.3.3 The WSD Systems

For disambiguation of the ten polysemous words as given in the Table 2.1, we have chosen the Simplified Lesk (SL) method (Kilgarriff and Rosenzweig, 2000) and the Lesk's Dictionary based Disambiguation (LDD) method (Manning and Schütze, 1999). Note that SL only measures the overlap between the context bag and each sense bag of the word to be disambiguated and returns the sense that has maximum overlap with the context bag. Whereas, LDD tries to expand the context bag by adding the dictionary definitions of the words present in the context. Next, it finds the intersection between this elaborated context bag and each sense bag of the target word and finally selects the sense with the maximum intersection score as the winner sense. To investigate the role of lemmatization in WSD, both SL and LDD methods are executed separately for each of the ten selected words. Firstly they are run without lemmatizing the context bag and the sense bag of the concerned word and next, these two bags are lemmatized before applying the methods. Furthermore, the above experiments have been repeated for each word with the incorporation of the Bengali WordNet[5] to expand the sense definitions of the ten words by adding the glosses taken from the WordNet. For a word, all of its dictionary senses are often not present in this WordNet. For the ten words, we manually map the available WordNet senses to the similar senses in the dictionary. When the WordNet is used in WSD, a dictionary sense is expanded by adding the corresponding (if any) gloss from the WordNet.

[5] http://www.isical.ac.in/~lru

2.4 Experimental Results

2.4.1 Evaluation Measures

To evaluate the performance of the proposed lemmatization algorithm, we compute direct accuracy by measuring the ratio of the number of surface word tokens for which correct lemmas are produced to the total number of surface word tokens given as input to the lemmatizer. For evaluation of the WSD systems, we measure *coverage*, *precision*, *recall* and *F1-score* of the systems (Navigli, 2009). The definitions of the above four measures are given below.

Let $W = \{w_1, \ldots, w_n\}$ be a WSD test set. For each word $w_i \in W$, let $Sense_D(w_i)$ be the set of all senses of w_i defined in the dictionary D. We define an "answer" function A_{gold} which returns the appropriate sense of a word in its present context i.e. $A_{gold}(w_i) \in Sense_D(w_i)$ represents the gold sense of w_i. Another "answer" function $A_{automatic}$ ($A_{automatic}(w_i) \in Sense_D(w_i) \cup \{\epsilon\}$) is introduced that gives the sense of a word assigned by an automatic WSD system[6]. Here ϵ denotes the case where the automatic WSD system cannot assign a sense. Now, *coverage* C is defined as follows.

$$C = \frac{\#\ answers\ provided}{\#\ total\ answers\ to\ provide} = \frac{\mid \{i \in \{1, \ldots, n\} \ :\ A_{automatic}(w_i) \neq \epsilon\} \mid}{n}$$

Precision P of a WSD system is the fraction of correct answers given by the WSD system.

$$P = \frac{\#\ correct\ answers\ provided}{\#\ \ answers\ provided} = \frac{\mid \{i \in \{1, \ldots, n\} \ :\ A_{automatic}(w_i) = A_{gold}(w_i)\} \mid}{\mid \{i \in \{1, \ldots, n\} \ :\ A_{automatic}(w_i) \neq \epsilon\} \mid}$$

Recall R is the fraction of the number of correct answers given by the WSD system over the total number of answers to be given.

$$R = \frac{\#\ correct\ answers\ provided}{\#\ total\ answers\ to\ provide} = \frac{\mid \{i \in \{1, \ldots, n\} \ :\ A_{automatic}(w_i) = A_{gold}(w_i)\} \mid}{n}$$

[6]We assume here that for a word to be disambiguated, only a single sense of that word is appropriate in its present context and the automatic WSD system also assigns a single sense from dictionary to that word. However, this notation can be extended to assignment of multiple senses.

	Step 1	Step 3(a)	Step 3(b)	Step 4	Remaining
Word tokens resolved	49.61%	27.20%	12.26%	4.39%	6.52%
Accuracy	96.99%	83.82%	69.03%	6.80%	34.86%

TABLE 2.2: Percentage of Word Tokens Resolved by Each Step of BenLem Algorithm in the Manually POS-tagged Test Dataset

	Step 1	Step 3(a)	Step 3(b)	Step 4	NNP/SYM	Remaining
Word tokens resolved	45.12%	26.63%	13.70%	5.60%	0.42%	8.53%
Accuracy	95.23%	76.07%	60.04%	5.34%	7.14%	43.16%

TABLE 2.3: Percentage of Word Tokens Resolved by Each Step of BenLem Algorithm in the Automatically POS-tagged Test Dataset

F1-score is the weighted harmonic mean of *precision* and *recall*. It is useful to judge the performance of a WSD system when *coverage* is less than 100%.

$$F1 = \frac{2PR}{P + R}$$

2.4.2 Lemmatization Accuracy

BenLem has been tested on the POS and lemma-annotated dataset as described in Section 2.3.2. The container sentence is considered as the context for a word token for which the lemma is to be found. Evaluation result shows that BenLem gives 81.95% accuracy for 3,342 word tokens. The algorithm may need minimum one to maximum four steps to process the words for finding out their corresponding lemmas. The percentage of the word tokens resolved by each step of the algorithm are given in the Table 2.2. Accuracy of each step in finding the correct lemma is shown in the second row of the Table 2.2. Initially, all words are passed through Step 1 that finds the lemmas for 49.61% tokens and saves the additional processing effort needed in the subsequent steps for them. It shows one important characteristics of the language (more specifically, the characteristics of the test set) that only searching for a token in the dictionary and matching its POS with that of the dictionary word can decide lemmas for about half of the Bengali words. Since this step resolves the highest percentage of the total number of surface words, accuracy of this step is quite vital for the overall efficiency of the lemmatizer. Experimental result shows that Step 1 is 96.99% accurate. The errors in Step 1 occur due to the fact that in Bengali, when a lemma is suffixed with a valid suffix and the resulting surface word is another lemma. For example, 'কুল'/kula + 'টা'/taa = 'কুলটা'/kulataa. Both 'কুল'/kula and 'কুলটা'/kulataa are valid root words and 'টা'/taa is a

valid suffix in Bengali. If 'কুলটা'/kulataa is to be lemmatized by BenLem, it will always return 'কুলটা'/kulataa as the lemma which is wrong. We analyze our dictionary and suffix list and found that there are 4788 <dictionary headword, valid suffix> pairs such that when the dictionary headword is suffixed, another dictionary headword is produced. However the high accuracy of Step 1 reveals that a good dictionary and an efficient POS tagger make the lemmatization task in Bengali easy.

The tokens for which Step 1 does not find any lemma but the set D_p (in Step 2) is not empty are passed to the subsequent steps. Step 3(a) carries the next major load and finds lemmas for about 27.20% of the total words. This shows that addition (or deletion) of a valid suffix to (from) a dictionary headword generates about one-fourth of the inflected words in Bengali. This step produces 16.18% errors which occur due to the situation when two different lemmas are suffixed with two different valid suffixes to produce same surface word. If this situation occurs, BenLem will always select that lemma which has the shorter distance with the surface word according to the distance measure $Dist$. For example, 'কুল'/kula + 'টাকে'/taake = 'কুলটাকে'/kulataake = 'কুলটা'/kulataa + 'কে'/ke. Both 'টাকে'/taake and 'কে'/ke are valid suffixes in Bengali. If 'কুলটাকে'/kulataake is given as input to BenLem, always 'কুলটা'/kulataa will be returned as the lemma as it has shorter distance with 'কুলটাকে'/kulataake than that with 'কুল'/kula. We search in the FIRE Bengali corpus and find 5299 distinct such surface words (about 1%) each of which can be produced by two different dictionary headwords suffixed with two different valid suffixes. When this processing is not adequate to find the lemma, Step 3(b) is invoked that explores whether deletion of a valid suffix (from a dictionary root) followed by addition of another valid suffix would result in the surface word. This step finds lemmas for about 12.26% of the total word tokens and produces 30.97% errors. The errors in this step come from the situation when deletion of a valid suffix from a root word and then addition of another valid suffix produce a surface word but the root is not the appropriate lemma for the produced surface word. For example, 'হেরা'/heraa is a root word and 'হেরা'/heraa - 'া'/aa + 'ে'/e generates 'হেরে'/here but for the surface word 'হেরে'/here, the appropriate lemma is 'হারা'/haaraa. Steps 3(a) and 3(b) give 83.82% and 69.03% accuracy respectively further revealing the importance of the list of valid suffixes for Bengali lemmatization. In cases where more than one headword in D'_p is found having the shortest distance with the target surface word, the smallest one among them according to the lexicographically sorted order is selected as the lemma.

The last step, i.e. Step 4 tries to find the lemmas for the remaining words (for which none of the previous three steps could find the lemmas) using sense of the word in question. About 4.39% word tokens are resolved by this step and for such words, the corresponding lemmas are found with 6.80% accuracy. This shows that finding the lemma of a word by understanding its sense in the context requires further investigation to improve the accuracy of Step 4. If there is more than one root word with the highest $MaxOverlapScore$ as well as having equal distance with the surface word, then the lexicographically smallest one is chosen as the lemma. The 'Remaining' column in the Table 2.2 gives the statistics of the words for which (i) either D_p is empty in Step 2 (ii) or $MaxOverlapScore$ is zero for all the headwords of D_p in Step 4. For those words, BenLem accepts a word form itself as the lemma. 34.86% of the lemmas found in this way are correct, which reveals that these correct lemmas are not covered by the dictionary used in the experiment. Because if they were present in the dictionary, they would have been resolved by Step 1.

2.4.3 Effect of POS Tagger

To study the effect of POS tagging on lemmatization performance, BenLem is tested further on the lemmatization dataset (consisting of 18 newspaper articles from the FIRE Bengali News corpus having 6,314 word tokens) where all proper nouns and punctuation symbols have been manually tagged and an automatic POS tagger is used to tag the remaining 3,342 word tokens. The proper names and punctuations are manually tagged to avoid machine errors as these tokens are not considered for the present evaluation of the lemmatizer. To configure a POS tagger for Bengali, we retrain the Stanford POS tagger[7] using datasets available from the Linguistic Data Consortium (LDC), the University of Pennsylvania; ICON 2007 NLP Tool Contest data for shared task competition on POS tagging and chunking[8]; and the Indian Institute of Technology, Kharagpur. In these datasets, there are total 10,416 sentences having 138,847 word tokens. The POS tagger used in this experiment is tested on the manually POS annotated lemmatization dataset of total 6,314 word tokens and it is found to be 79.18% accurate on that dataset. For the automatically POS tagged 3,342 word tokens, overall accuracy of BenLem is found to be 75.46%. The percentage of the word tokens resolved by each step of BenLem

[7] http://www.nlp.stanford.edu/software/
[8] http://ltrc.iiit.ac.in/icon/2013/nlptools/prev-contests.php

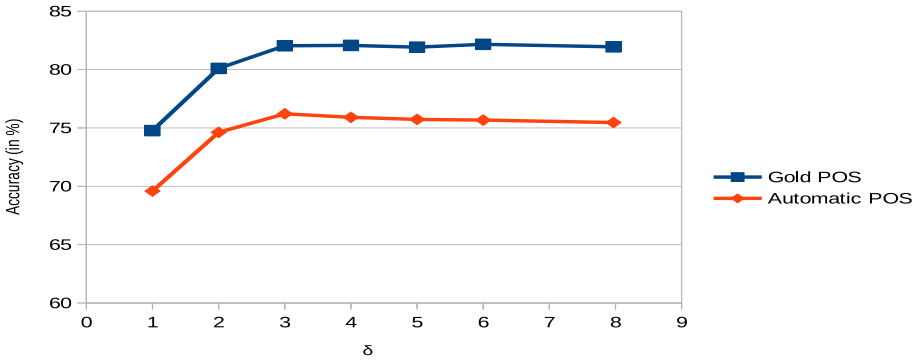


FIGURE 2.1: δ vs. accuracy plot.

and stepwise accuracy in finding the lemmas are given in the Table 2.3. It is found that the results presented in the Tables 2.2 and 2.3 are highly correlated. We have calculated the Pearson's correlation coefficient between the percentage of word tokens resolved by Step 1 to Step 4 of BenLem using gold and automatic POS tags and the value is found to be 0.9994. For the stepwise accuracy, the value of the above coefficient is 0.9844. There is one extra field in the Table 2.3, i.e., NNP/SYM. It is to be noted that 0.42% of the 3,342 word tokens are tagged as either proper nouns or punctuation symbols wrongly by the automatic POS tagger and they remain unprocessed by BenLem. It has been noted that for 7.14% of these words, the word forms themselves are the lemmas.

2.4.4 Effect of δ on Lemmatization Performance

A post-experiment analysis has been done to verify whether the current approach taken to determine the value of δ (described in Section 2.3) is adequate or not. For this, we have conducted a set of experiments (on both the gold and the automatic POS tagged lemmatization datasets) using different values of $\delta = 1, 2, 3, 4, 5, 6$ and 7.97. On the gold POS tagged dataset, $\delta = 6$ gives the highest accuracy of 82.16% whereas, on the automatic POS tagged dataset, 76.21% is the best accuracy obtained at $\delta = 3$. We further plot the (δ vs. Accuracy) curves for the experiments on both gold and automatic POS tagged dataset (given in the Figure 2.1). In the range of $\delta = 1$ to 3, both the curves are strictly increasing showing a positive correlation between δ and the lemmatization accuracy. From $\delta = 3$ to 7.97, the curves are almost parallel to the x-axis which indicates that changing the value of δ affects the accuracy very little. This analysis shows that selecting a relatively higher value of δ is acceptable.

Step 1	Step 2	Step 3(a)	Step 3(b)	Step 4	Remaining
7.46	137.44	1914.52	2178.97	2217.72	1225.44

TABLE 2.4: Per Token Speed Analysis for Each Step of BenLem (figures are in milliseconds)

2.4.5 Speed Analysis

We have analyzed the time requirement for each step of BenLem. Average running time (in milliseconds) for word tokens that are resolved by Step 1 to Step 4, are given in the Table 2.4. The specifications of the system in which the experiment has been carried out are as follows. Random Access Memory (RAM): 4 GB; Processor: Intel(R) Core(TM)2 Duo; Frequency: 2.93 GHz; Cache Size: 3072 KB. All the units given here express their standard meanings. Table 2.4 shows that Steps 1 and 2 are quite fast but Steps 3 and 4 are much more time consuming. Analysis reveals that handling the suffix list is the cause behind the time requirement of Step 3. If the size of the suffix list is increased, more time will be required for completion of that step. However, we do not attempt to do any code optimization at this stage and represented all the dictionary headwords and valid suffixes in list structures but to speed up the algorithm, more efficient data structures can be used (may be trie representations for the dictionary headwords and valid suffixes), which will save the processing time of Steps 1 to 3. As Step 4 attempts to do sense matching, we are not surprised to note that it is taking about 2 seconds. Though it is difficult to reduce the amount of time taken by this step but it does not affect the overall processing time of the algorithm significantly as only a small part of total word tokens are processed at this step.

2.4.6 Qualitative Error Analysis

The major reasons behind the errors made by BenLem are as follows: (i) in Bengali, there are many compound words like 'আগ্রহপ্রকাশের'/aagrahaprakaasher, 'অভয়বাণীর'/abhaYabaa Niir etc., which BenLem fails to handle because the dictionary does not cover all the compound words (ii) some infrequent but valid suffixes in Bengali are not present in the suffix list (iii) the dictionary does not contain all possible parts of speech for the morphological variants of some headwords. For example, *noun* is the only possible part of speech of the headword 'আইন'/aain but one of its morphologically derived forms 'আইনি'/aaini is used as *adjective* which is not mentioned in the dictionary (iv) due to

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	34.80%	11.26%	3.92%	5.82%
1.2	SL	NO/YES	47.55%	11.34%	5.40%	7.30%
2.1	SL	YES/NO	90.69%	30.27%	27.45%	28.80%
2.2	SL	YES/YES	100%	33.82%	33.82%	33.82%
3.1	LDD	NO/NO	99.01%	3.47%	3.43%	3.45%
3.2	LDD	NO/YES	100%	7.35%	7.35%	7.35%
4.1	LDD	YES/NO	99.50%	44.33%	44.11%	44.22%
4.2	LDD	YES/YES	100%	54.90%	54.90%	54.90%

TABLE 2.5: WSD results for 'করা'/karaa: to do/to work/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	50%	5%	2.50%	3.33%
1.2	SL	NO/YES	58.75%	4.25%	2.50%	3.15%
2.1	SL	YES/NO	95%	5.26%	5%	5.12%
2.2	SL	YES/YES	98.75%	11.39%	11.25%	11.32%
3.1	LDD	NO/NO	98.75%	0%	0%	-
3.2	LDD	NO/YES	100%	1.25%	1.25%	1.25%
4.1	LDD	YES/NO	100%	10%	10%	10%
4.2	LDD	YES/YES	100%	18.75%	18.75%	18.75%

TABLE 2.6: WSD results for 'দেওয়া'/deoYaa: to give/to offer/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	21.74%	0%	0%	-
1.2	SL	NO/YES	56.52%	0%	0%	-
2.1	SL	YES/NO	100%	2.90%	2.90%	2.90%
2.2	SL	YES/YES	100%	1.45%	1.45%	1.45%
3.1	LDD	NO/NO	100%	0%	0%	-
3.2	LDD	NO/YES	100%	20.29%	20.29%	20.29%
4.1	LDD	YES/NO	100%	5.79%	5.79%	5.79%
4.2	LDD	YES/YES	100%	1.45%	1.45%	1.45%

TABLE 2.7: WSD results for 'কাটা'/kaataa: to cut/to break/...

globalization, many English words like 'রেজাল্ট'/result, 'প্র্যাকটিস'/practice, 'পারফর্ম'/perform etc. are being used frequently in Bengali and they are gradually finding their places in the Bengali vocabulary list. Words that come from different languages cannot be resolved by BenLem (v) examples of 'কুলটা'/kulataa, 'কুলটাকে'/kulataake and 'হেরে'/here (discussed in Section 2.4.2) also point out the reasons for errors in Steps 1, 3(a) and 3(b) of BenLem. Apart from all of the above reasons, the coverage of the dictionary and the accuracy of the POS tagger used for lemmatization are two very important factors that influence the overall performance of BenLem significantly.

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	74.14%	4.65%	3.45%	3.96%
1.2	SL	NO/YES	86.20%	8%	6.89%	7.40%
2.1	SL	YES/NO	98.27%	5.26%	5.17%	5.21%
2.2	SL	YES/YES	98.27%	8.77%	8.62%	8.69%
3.1	LDD	NO/NO	100%	8.62%	8.62%	8.62%
3.2	LDD	NO/YES	100%	12.07%	12.07%	12.07%
4.1	LDD	YES/NO	100%	15.51%	15.51%	15.51%
4.2	LDD	YES/YES	100%	15.51%	15.51%	15.51%

TABLE 2.8: WSD results for 'রাখা'/raakhaa: to keep/to obey/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	50.0%	0%	0%	-
1.2	SL	NO/YES	64.28%	0%	0%	-
2.1	SL	YES/NO	100%	5.36%	5.36%	5.36%
2.2	SL	YES/YES	100%	16.07%	16.07%	16.07%
3.1	LDD	NO/NO	87.5%	0%	0%	-
3.2	LDD	NO/YES	100%	1.78%	1.78%	1.78%
4.1	LDD	YES/NO	100%	23.21%	23.21%	23.21%
4.2	LDD	YES/YES	100%	33.92%	33.92%	33.92%

TABLE 2.9: WSD results for 'তোলা'/tolaa: to lift/to raise/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	9.61%	0%	0%	-
1.2	SL	NO/YES	28.85%	0%	0%	-
2.1	SL	YES/NO	100%	3.85%	3.85%	3.85%
2.2	SL	YES/YES	100%	7.70%	7.70%	7.70%
3.1	LDD	NO/NO	65.38%	14.70%	9.61%	11.63%
3.2	LDD	NO/YES	98.07%	25.49%	25%	25.24%
4.1	LDD	YES/NO	100%	9.61%	9.61%	9.61%
4.2	LDD	YES/YES	100%	9.61%	9.61%	9.61%

TABLE 2.10: WSD results for 'খোলা'/kholaa: to unfold/to start/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	17.30%	0%	0%	-
1.2	SL	NO/YES	23.07%	0%	0%	-
2.1	SL	YES/NO	100%	0%	0%	-
2.2	SL	YES/YES	100%	1.92%	1.92%	1.92%
3.1	LDD	NO/NO	98.07%	0%	0%	-
3.2	LDD	NO/YES	100%	0%	0%	-
4.1	LDD	YES/NO	100%	17.31%	17.31%	17.31%
4.2	LDD	YES/YES	100%	21.15%	21.15%	21.15%

TABLE 2.11: WSD results for 'যোগ'/Joga: to add/to join/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	53.66%	27.27%	14.63%	19.04%
1.2	SL	NO/YES	65.85%	25.92%	17.07%	20.58%
2.1	SL	YES/NO	95.12%	17.94%	17.07%	17.50%
2.2	SL	YES/YES	100%	2.44%	2.44%	2.44%
3.1	LDD	NO/NO	97.56%	2.50%	2.44%	2.47%
3.2	LDD	NO/YES	100%	19.51%	19.51%	19.51%
4.1	LDD	YES/NO	100%	4.88%	4.88%	4.88%
4.2	LDD	YES/YES	100%	4.88%	4.88%	4.88%

TABLE 2.12: WSD results for 'পড়া'/pa.Daa: to read/to fall/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	37.14%	23.07%	8.57%	12.49%
1.2	SL	NO/YES	54.28%	21.04%	11.43%	14.81%
2.1	SL	YES/NO	94.29%	0%	0%	-
2.2	SL	YES/YES	100%	0%	0%	-
3.1	LDD	NO/NO	100%	5.70%	5.70%	5.70%
3.2	LDD	NO/YES	100%	11.44%	11.44%	11.44%
4.1	LDD	YES/NO	100%	11.44%	11.44%	11.44%
4.2	LDD	YES/YES	100%	17.14%	17.14%	17.14%

TABLE 2.13: WSD results for 'চলা'/chalaa: to walk/to continue/...

ID.	Algorithm	WordNet/BenLem	Coverage	Precision	Recall	F1-Score
1.1	SL	NO/NO	55%	0%	0%	-
1.2	SL	NO/YES	90%	0%	0%	-
2.1	SL	YES/NO	100%	5%	5%	5%
2.2	SL	YES/YES	100%	0%	0%	-
3.1	LDD	NO/NO	95%	0%	0%	-
3.2	LDD	NO/YES	100%	10%	10%	10%
4.1	LDD	YES/NO	100%	0%	0%	-
4.2	LDD	YES/YES	100%	0%	0%	-

TABLE 2.14: WSD results for 'ধরা'/dharaa: to catch/to hold/...

2.4.7 Role of BenLem in WSD

In order to explore the role of lemmatization in WSD, we conduct the following eight types of experiments. Two algorithms namely SL and LDD are used with or without using WordNet giving four variations. Each variation is further tested with or without using BenLem giving eight different experiments.

- ID 1.1 and 1.2: SL without and with BenLem (no use of WordNet).
- ID 2.1 and 2.2: SL, WordNet is used without and with BenLem.

- ID 3.1 and 3.2: LDD without and with BenLem (no use of WordNet).
- ID 4.1 and 4.2: LDD, WordNet is used without and with BenLem.

We have used BenLem to lemmatize the sense bags and the context bags of the ten polysemous words (as in the Table 2.1). For lemmatization, the retrained POS tagger (discussed in Section 2.4.3) is used to POS tag the words present in the sense bags and the context bags. In order to choose the context size for execution of SL and LDD methods, we have taken 3-sentences context (1 sentence before and after the focus sentence and the focus sentence itself) which is common in WSD. It is to be noted here that for the lemmatization experiment, the container sentence of the word to be lemmatized was considered as the context window. WSD results are presented in the Tables 2.5 to 2.14. In each of those tables, the rows with IDs X.1 (X varies from 1 to 4) correspond to the WSD systems without BenLem and the rows with IDs X.2 correspond to the WSD systems with BenLem. There are 40 such pairs (four for each of the ten words). If results in each such pair are compared, we see the usefulness of BenLem. For 22 (out of 40) pairs, BenLem improves coverage (statistically significant at p-value < 0.01 in a two-tailed paired t-test); for 23 pairs, the recall is improved (statistically significant at p-value < 0.01) upon using BenLem. Highest improvement in coverage is from 34.80% to 47.55% (Row IDs 1.1 vs. 1.2 in the Table 2.5) whereas maximum improvement in recall is from 44.11% to 54.90% (Row IDs 4.1 vs. 4.2 in the Table 2.5). Note that there are 204 instances of the word 'করা'/karaa in the test set. An example is provided here to show why lemmatization is effective for WSD. Consider the context given below and the gold sense of the word 'চলা'/chalaa for that context.

- Context: 'তাঁর কথায়, বাণিজ্যিক ভাবে যে সব অভিযানের আয়োজন করা হয়/haYa, সেগুলিতেই মৃত্যুর ঘটনা বেশি ঘটছে/ghatchhe। কে টু র হাতছানিতে এই দফায় যাঁরা প্রাণ হারালেন, তাঁরা কতটা প্রশিক্ষিত ছিলেন, কিংবা বিরূপ প্রকৃতি এর জন্য কতটা দায়ী এ সব নিয়ে বিতর্ক চলতেই/chaltei<Disambiguate> থাকবে/thaakbe। তবে তালিকায় তাঁদের নামের পাশে থাকবে/thaakbe ছোট্ট একটি তারার মতো চিহ্ন। '
- Gold Sense: 'ক্রমাগত হতে/hate বা ঘটতে/ghatte থাকা/thaakaa।'

'চলতেই'/chaltei is an inflected form of 'চলা'/chalaa. Before lemmatization, there is no common word present between the context and the gold sense but after lemmatization, we

find three distinct words, namely 'হওয়া'/haoYaa, 'ঘটা'/ghataa and 'থাকা'/thaakaa, common between them. 'হওয়া'/haoYaa is the lemma of 'হয়'/haYa and 'হতে'/hate; 'ঘটা'/ghataa is the lemma of 'ঘটছে'/ghatchhe and 'ঘটতে'/ghatte; and 'থাকা'/thaakaa is the lemma of 'থাকবে'/thaakbe. Hence, the score of the gold sense for the context becomes greater showing the importance of lemmatization for WSD.

In the Table 2.14, use of lemmatization does not improve WSD results significantly (BenLem degrades recall in case of Row IDs 2.1 vs. 2.2 and improves same in case of Row IDs 3.1 vs. 3.2) as there are few (20) instances of the word 'ধরা'/dharaa in the dataset and the word has the highest number of senses (39 different senses) defined in the dictionary. For certain cases, we get 0% precision and recall values indicating that no common word is found between the context bag and the sense bags. This happens especially when the dictionary senses of a word is small in length. This problem is alleviated when we use WordNet which can provide longer sense bags through glosses. For example, consider the Row IDs 1.1 and 1.2 vs. Row IDs 2.1 and 2.2 in the Table 2.10. Here, when SL was executed without using WordNet, it produced 0% precision and recall. But the same method gives non zero precision and recall values when Wordnet is used to add the glosses from it.

There are four instances (out of 40), where use of BenLem degrades the WSD performance. For Row IDs 2.1 vs. 2.2 and 4.1 vs. 4.2 in the Table 2.7, Row IDs 2.1 vs. 2.2 in the Table 2.12 and Row IDs 2.1 vs. 2.2 in the Table 2.14, the recall values are degraded because of BenLem. For recall, maximum degradation is from 17.07% to 2.44% (Row IDs 2.1 vs. 2.2 in the Table 2.12). We have further investigated the reasons for such degradations. Presence of lemmatization errors is one major reason. Combination of the WordNet sense bags with the dictionary sense bags makes the resultant sense bags bigger. The WSD performance is degraded if many of the words in this bigger sense bag (though chances are less) are unfortunately affected (at the same time) by lemmatization errors. We are presenting a particular example below where doing lemmatization hurts WSD performance. Consider the context given below and the gold sense of the word 'ধরা'/dharaa for this context and one other sense of 'ধরা'/dharaa.

- Context: 'শোয়েব এবং আসিফ দুজনেই ভারতে গত চ্যাম্পিয়ন্স ট্রফির সময় ডোপ পরীক্ষায় ধরা পড়েছিলেন। সম্প্রতি দুবাইয়ে মাদকদ্রব্য নিয়ে ধরা/dharaa<Disambiguate> পড়ে আটক/aatak হয়েছিলেন আসিফ।

আই পি এল চেয়ারম্যান ললিত মোদী রাতের দিকে স্বীকার করে/kare নেন/nen এশীয় বোলারের ডোপ পরীক্ষায় ব্যর্থ হওয়ার খবর।'

- Gold Sense: 'কারও ইচ্ছার বিরুদ্ধে তাকে আটক/aatak করা/karaa।'
- Other Sense: 'নির্ধারণ বা স্থির করা/karaa কোনও কথা ইত্যাদি জেনে নেওয়া/neoYaa।'

Before lemmatization, 'আটক'/aatak is the only common word between the gold sense and the context and there is no common word between the other sense and the context. So, according to SL method, gold sense is the winner sense as it has higher co-occurrence score with the context. After lemmatization, the common words between the gold sense and the context are 'আটক'/aatak and 'করা'/karaa as 'করা'/karaa is the lemma of the context word 'করে'/kare. As 'নেওয়া'/neoYaa is the lemma of 'নেন'/nen, hence after lemmatization 'নেওয়া'/neoYaa and 'করা'/karaa are the two common words between the context and the other sense of 'ধরা'/dharaa. Thus, after lemmatization, both the gold sense and the other sense have the same number of overlapped words with the context bag resulting the failure of SL method to determine the winner sense.

In Bengali, some of the frequent words are sometimes present in the sense bags and in the context bags in various inflected forms. A few examples of those words and their multiple inflections are <Root: `করা'/karaa; Inflections: 'করে'/kare, 'কর'/kara, ...>, <Root: `নেওয়া'/neoYaa; Inflections: 'নিন'/nin, 'নেন'/nen, ...>, <Root: `যাওয়া'/JaaoYaa; Inflections: 'যেয়ে'/JeYe, 'যেতে'/Jete, ...> etc. They may or may not bear the principal meaning of the context and it is very hard to determine when they should be removed from the context and when not. In the WSD algorithms where co-occurrence count between the sense bag and the context bag is considered as the score of a sense, it may happen that two different inflected forms of such words (e.g. `নেওয়া'/neoYaa, `যাওয়া'/JaaoYaa, etc.) are present in the context but not bearing its principal meaning as well as present in the sense bag of a sense which is not the gold sense for that context. Before lemmatization, those differently inflected forms will not add value to the score of that sense but after lemmatization they will get replaced by their unique lemma and add a count to the score of that sense raising its chance to be a wrong winner.

We have further compared the WSD accuracy with respect to the random sense baseline (Navigli, 2009) for the selected ten words. In this method, let k be the number of senses of a target word, w_t. A random integer is generated from the range [1, k].

Words	'করা'/karaa	'দেওয়া'/deoYaa	'কাটা'/kaataa	'রাখা'/raakhaa	'তোলা'/tolaa
Random Sense Baseline	2.45%	1.25%	1.45%	5.17%	1.78%
Best Recall Achieved	54.90%	18.75%	20.29%	15.51%	33.92%
Words	'খোলা'/kholaa	'যোগ'/Joga	'পড়া'/pa.Daa	'চলা'/chalaa	'ধরা'/dharaa
Random Sense Baseline	1.92%	1.92%	2.44%	2.86%	0%
Best Recall Achieved	25.24%	21.15%	20.58%	14.81%	10%

TABLE 2.15: Random Sense Baseline Results of Selected ten Words

Let i be this random integer. Next, all the instances of w_t in the test data are tagged with the sense i. The results are given in the Table 2.15. Comparison between the random sense recall and the best recall achieved for each word in our WSD experiments shows that the present research produces results which are far better than what we could achieve by following a random sense pick-up method. It is to be noted that the random sense baseline gives accuracy less than (1/number of senses) for nine out of the ten words (except for 'রাখা'/raakhaa). The reason behind this is as follows. Under uniform distribution of all the senses in the test data, the probability of success for the random sense baseline is (1/number of senses). From the statistics given in the Table 2.1, it is clear that in the WSD dataset used for this experiment, the senses of the target words are not uniformly distributed and that is the cause behind the poor performance of the random sense pick-up method. Actually, seven out of the ten polysemous words get 1 accurate prediction with the random sense baseline. They are `দেওয়া'/deoYaa, `কাটা'/kaataa, `তোলা'/tolaa, `খোলা'/kholaa, `যোগ'/Joga, `পড়া'/pa.Daa and `চলা'/chalaa. The words `করা'/karaa, `রাখা'/raakhaa and `ধরা'/dharaa get 5, 3 and 0 accurate prediction, respectively.

It would have been better if we could compare the WSD accuracy with the most frequent sense baseline which is also known as the first sense baseline (Navigli, 2009). This baseline method requires a ranking of all the senses of the target word based on their frequencies in a sense annotated corpus. The particular sense having the highest frequency in that corpus gets the first rank. Finally, all the occurrences of the target word in the test dataset are tagged with the first sense from this rank list. For example, in English WordNet (Fellbaum, 1998), senses of the same word are ranked based on their frequencies in the SemCor corpus (Miller et al., 1994). Hence, for comparing the WSD accuracy with the most frequent sense baseline, a sense annotated training corpus is needed from which the senses of the target word can be ranked. Unfortunately, there is no such corpus available in Bengali and thus, in this present work, we are unable to report the comparison results with respect to the most frequent sense baseline.

2.5 Summary

This chapter addresses two relatively less explored NLP issues for Bengali namely, lemmatization and WSD. Effect of lemmatization on WSD is also investigated. Annotated datasets for evaluating these tools have been prepared. Though both the datasets prepared for this work are small in size, but they, in their present form, would definitely help the respective research community. The methods and the experimental results presented here too form a nice base for doing further research in the related topic.

Another significant by-product of this research is the investigation of the role of the newly developed Bengali WordNet for WSD. In spite of having a relatively low coverage, the WordNet largely improves WSD results over the use of dictionary only. The present version of the WordNet contains 29, 882 synsets. For the ten words for which the current WSD systems have been tested, it is observed that the WordNet contains fewer senses compared to the senses present in the dictionary. For instance, the word 'ধরা'/dharaa has 8 senses in the WordNet out of the 39 senses present in the dictionary. However, the glosses from the WordNet are found to be quite helpful for WSD compared to the sense definitions (and sometimes glosses) present in the dictionary. Therefore, though the current version of the Bengali WordNet is linguistically not very rich, still its positive contribution is nicely demonstrated by the WSD results. Therefore, it can be expected that with the addition of more linguistic richness to the WordNet, we would be able to do better WSD in Bengali.

While designing BenLem, we have carefully kept the algorithm less dependent on language based resources. Low resource requirement and language independent features would help the algorithm to be extended for other inflected languages including several major languages. It is to be noted that the qualities of the resources used (e.g. coverage of the dictionary, accuracy of the POS tagger) have significant impact on the accuracy of our proposed lemmatization algorithm.

Chapter 3

Word Embedding for Lemmatization

3.1 Introduction

This chapter presents a method that attempts to overcome the shortcomings of BenLem algorithm. The proposed method is language independent and unsupervised in nature. Compared to BenLem, the following changes have been incorporated into the new algorithm.

(i) Previously, the orthographical similar roots of an input surface word were selected from dictionary using the string distance measure of YASS stemmer (Majumder et al., 2007). The set of roots having distance less than a threshold with the input word are considered as the potential lemmas. However, it is difficult to rightly determine the threshold for new languages. A relatively high distance value allows many noisy candidates under consideration. Hence, a graph based distance measure is introduced in the new method to capture the syntactic similarity which significantly reduces the number of potential roots. Analogous to the approach of Bhattacharyya et al. (2014), our method initially builds a trie taking all roots from lexical knowledge-base such as a dictionary and then exploits a new searching strategy to pick a set of potential roots from the trie. Next, the obtained set is further filtered by valid suffixes in the language to get more refined candidates. Finally, the candidate semantically nearest with the input word, is nominated as the lemma.

(*ii*) Semantic proximity between a potential root and the target surface word is appraised from two different aspects: The word embedding technique (Mikolov et al., 2013a,b) is exploited to measure the semantic relatedness. Similarity between two words is determined by the cosine distance between their corresponding vector embeddings. The reason behind using the distributional similarity is that as a root and its variants are semantically close, so their corresponding vectors in the embedded space should have high cosine similarity. Additionally, dictionary sense bags of roots are utilized to determine if the contextual sense of the surface word matches with any of the senses of a potential root available in the dictionary.

(*iii*) Along with Bengali, Hindi is included for experimentation, which is the most widely spoken language in India and ranks within the World's top 10 languages. The next section describes the algorithm.

3.2 The Trie based Lemmatization Method

Let w be the surface word to be lemmatized. The operations $CON(w)$ and $POS(w)$ denote the contextual words of w and its part of speech in the context respectively. Similar to BenLem, the proposed algorithm makes use of a dictionary $\mathcal{D}$ and a valid suffix list S of the concerned language. All the operations $POS_{dic}(d_i)$, $POS_{sense}(d_i, c_j)$, $Overlap(d_i, c_j, w)$ and $MaxOverlap(d_i, w)$ express their standard meaning (as explained in chapter 2). Another operation $MCPL(d_i, w)$ is defined to return the length of the maximum common prefix between d_i and w.

In addition to a dictionary and a valid suffix list, the algorithm requires a text corpus to get the vector embeddings of the words in the concerned language. The corpus is fed to the recurrent neural network based language model developed by Mikolov et al. (2013a). Let $\vec{w'}$ be the embedded vector for any arbitrary word w'.

For lemmatization, two aspects play crucial role. If l be the lemma of w, then $POS(w)$ should belong to $POS_{dic}(l)$ and sense of w in the current context should match with one of the senses of l provided in the lexical knowledge base. For example, let w = 'আসছে'/Aschhe and $CON(w)$ = 'সে/se বাড়ি/bA.Di আসছে/Aschhe'. The appropriate lemma of w for the given context is 'আসা'/AsA which is a polysemous root word in Bengali. $POS(w)$ is verb that belongs to the set POS_{dic}('আসা'/AsA). Here, w is used in the

sense 'to come' which is one of the multiple senses of 'আসা'/AsA. However, for a resource scarce language, an efficient POS tagger is usually not available as a sufficient number of manually POS annotated sentences are required to build it. Preparing a sense inventory having well explained glosses of root words with good example sentences also demands extensive human effort. The prepared suffix list is not regarded as an expensive resource because no manual effort is required to build it. Based on the resource requirements, three versions of our lemmatization algorithm are proposed as follows.

Version 1: does not use POS and sense.

Version 2: uses only POS.

Version 3: uses both POS and sense.

In the next subsection, we describe two trie searching methods to find out the potential roots for w from the set $\mathcal{D}$. Depending on the resource requirements, two different methods are designed. The first one is applicable when $POS(w)$ is known and hence it is used by version 2 and 3. Version 1 uses the second method as it does not exploit $POS(w)$.

3.2.1 Potential Roots Searching in Trie

At first, all root words taken from $\mathcal{D}$ are arranged in a trie. Each node in the trie corresponds to a unicode character of the language and the nodes which represent the last character of any root word are marked as *final* nodes. The remaining nodes are marked as *non-final* nodes. For a final node f, let $root(f)$ be the root word represented by the path from the initial node to f. To find the lemma of w, the trie is navigated starting from the initial node and navigation ends when either w is completely found in the trie or after traversing some portion of w, there is no path for navigation. So, for either of these situations, let the node where navigation ends be called as the *end* node. Next, the methods are presented one by one.

3.2.1.1 Method 1: Trie Search with POS

Input: w and $POS(w)$. **Output:** A set $\mathcal{R}$ containing the potential root(s) to be the lemma of w.

Step 1. Let $\mathcal{F}$ be the set of *final* nodes present in the path from the initial node to the *end* node. If $\mathcal{F} = \emptyset$, then go to step 2. Otherwise, for each $f \in \mathcal{F}$, if $POS(w) \in$

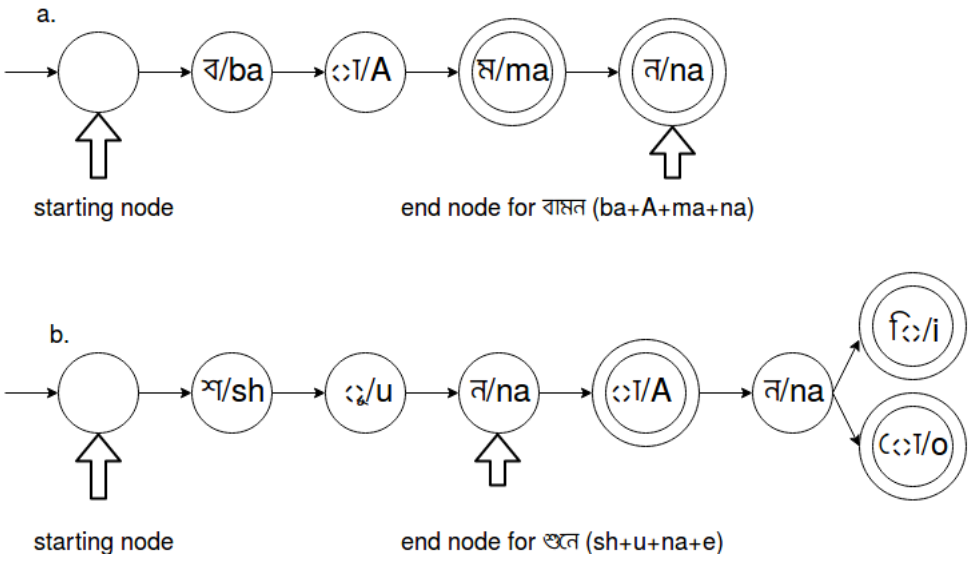


FIGURE 3.1: Examples of two sample cases where steps 1 and 2 of the method 1 respectively select the potential roots.

$POS_{dic}(root(f))$, put f in a new set $\mathcal{F}'$. If, $\mathcal{F}' = \emptyset$, then go to step 2. Otherwise, select the *final* node f' from $\mathcal{F}'$ such that f' is closest to the *end* node among all the members of $\mathcal{F}'$. We measure the distance between two nodes by the number of edges between them present in the trie. Finally put $root(f')$ in the set $\mathcal{R}$ and the searching process terminates.

Step 2. If either of the sets $\mathcal{F}$ and $\mathcal{F}'$ is empty at step 1, then this step is executed. Find the *final* node f'' in the trie which is closest to the *end* node as well as satisfies the constraints that (i) $POS(w) \in POS_{dic}(root(f''))$; and (ii) between w and $root(f'')$, there exists a common prefix. Next, put $root(f'')$ in the set $\mathcal{R}$. If more than one *final* node is found at the closest distance satisfying the above constraints, then put all of their corresponding root words in the set $\mathcal{R}$ and the searching process terminates.

Figure 3.1 presents two sample cases where the steps 1 and 2 of the above method are illustrated. In this figure, the double circled nodes refer to the *final* nodes. In case a, let w = 'বামন'/(ba+A+ma+na) which itself is a Bengali root word. In the path from the starting node to the *final* node of w, another root 'বাম'/(ba+A+ma) is present. Let us assume here that $POS(w)$ is noun which also belongs to the set POS_{dic}('বাম'/bAma). As the node 'ন'/na is the *end* node for w as well as the closest *final* node to the *end* node (distance 0), so $\mathcal{R}$ will contain only 'বামন'/bAmana. In case b, we choose w = 'শুনে'/(sh+u+na+e). The trie contains 3 *final* nodes 'া'/A, 'ি'/i and 'ো'/o representing the roots 'শুনা'/shunA, 'শুনানি'/shunAni and 'শুনানো'/shunAno

respectively. As there is no *final* node found in the path from the starting node to the *end* node of w, step 2 is applied here. The word w is a verb inflection derived from the root 'শুনা'/shunA which is also the closest root to the *end* node of w (distance 1). Whereas, the remaining roots are at a distance 3 from the *end* node. Hence, $\mathcal{R}$ will contain only the appropriate candidate.

3.2.1.2 Method 2: Trie Search without POS

Input: w and a threshold radius rad. **Output:** A set $\mathcal{R}$ containing the potential root(s) to be the lemma of w.
Step 1. Find the *end* node for w in the trie and then obtain the set $\mathcal{F}$ containing the *final* nodes present in the path from the initial node to the *end* node. Now, keeping the *end* node as center, find all root words in the trie which share a common prefix with w and have their corresponding *final* nodes within the distance rad from the *end* node. Next, put them in the set $\mathcal{F}'$. Let $\mathcal{F}'' = \mathcal{F} \cup \mathcal{F}'$. Finally, for each $f'' \in \mathcal{F}''$, put $root(f'')$ in the set $\mathcal{R}$ and searching ends.

Method 2 is parameterized by a radius rad. Basically, both steps 1 and 2 of the method 1 are incorporated here as we do not get the chance to prune the candidates using POS information. The examples in Figure 3.1 show that the lemma can be either in the direct path from the starting node to the *end* node or in the outside of the direct path. So, initially all the root words which are proper substrings of w are selected. Next, those for which the length of uncommon part with w is within the value of rad, are also considered as the potential roots. For case b in Figure 3.1, if $rad = 1$, then $\mathcal{R}$ will contain 'শুনা'/shunA only. If rad is increased to 3, then 'শুনানি'/shunAni and 'শুনানো'/shunAno will also be included in $\mathcal{R}$.

3.2.2 Filtering using Frequent Suffixes

Next, the root words in $\mathcal{R}$ are further filtered by the set of frequent suffixes S. The filtering method is as follows. For each $r \in \mathcal{R}$, if $\exists\ s_x \in S$ such that $w = r \pm s_x$, then r is put in a set $\mathcal{R}'$. If $\mathcal{R}' = \emptyset$ then if $\exists\ s_y, s_z \in S$, such that $w = (r - s_y) + s_z$ then r is put in the set $\mathcal{R}'$. $\mathcal{R}'$ is the set of potential lemmas of w.

In the major Indic languages, often a surface word can be obtained by adding/subtracting a frequent suffix to/from its lemma (e.g. 'করার'/karAr = 'করা'/karA + 'র'/ra ; 'লগাই'/lagAI = 'লগা'/lagA + 'ই'/I ; 'কর'/kar = 'করা'/karA - 'া'/A ; 'হো'/ho = 'হোনা'/honA - 'ানা'/A). Sometimes, first subtraction of a valid suffix and then addition of another valid suffix to the lemma may produce the surface word (e.g. 'করলাম'/karlAm = 'করা'/karA - 'া'/A + 'লাম'/lAm ; 'অগলে'/agle = 'অগলা'/aglA - 'া'/A + 'ে'/e). Also, there exist such inflected words for which none of the above transformations can map them to their respective lemmas (e.g. 'পেকে'/peke → 'পাকা'/pAkA ; 'গয়া'/gayA → 'জা'/jA). Next, we will present the details of the three versions of the proposed lemmatization algorithm.

3.2.3 Workflows of Different Versions of the Lemmatization Algorithm

Input : w and radius rad
Output: The lemma l of w
Execute Method 2 of trie searching taking w and rad as inputs, and obtain $\mathcal{R}$;
if $\mathcal{R} = \emptyset$ *or* $w \in \mathcal{R}$ **then**
  return w as the lemma l;
end
Filter $\mathcal{R}$ with the suffix set S and get $\mathcal{R}'$;
if $\mathcal{R}' = \emptyset$ **then**
  return w as the lemma l;
end
/* variant 1.1 (without word embedding) */
$l = \arg\max_{r'} MCPL(r', w),\ \forall r' \in \mathcal{R}'$;
return l;
/* variant 1.2 (with word embedding) */
if $\forall r' \in R',\ \vec{r'}$ *exists* **then**
  $l = \arg\max_{r'}\ cos\ sim(\vec{r'}, \vec{w}),\ \forall r' \in \mathcal{R}'$;
  return l;
else
  $l = \arg\max_{r'} MCPL(r', w),\ \forall r' \in \mathcal{R}'$;
  return l;
end

Algorithm 1: Version 1 of the Proposed Lemmatization Algorithm.

We design two variants for each of the versions 1 to 3 without/with using vector embeddings of words, to explore the effectiveness of word vectors for lemmatization, thus producing 6 different variations. For a version X, $X.1$ is used to refer to the variant that does not use the word embeddings and $X.2$ is for that which uses it (X varies from 1 to 3). Table 3.1 categorically presents the resource requirements of each version together with their variants.

		POS	Sense	Embedding
Version 1	Variant 1.1	×	×	×
	Variant 1.2	×	×	✓
Version 2	Variant 2.1	✓	×	×
	Variant 2.2	✓	×	✓
Version 3	Variant 3.1	✓	✓	×
	Variant 3.2	✓	✓	✓

TABLE 3.1: Resource requirements of version $1-3$ of the lemmatization algorithm together with their variants

Algorithm 1 presents the detailed stepwise workflow of version 1. It takes w and rad as the input parameters. Initially, method 2 is used to construct the set $\mathcal{R}$ which consists of early potential candidates and then $\mathcal{R}$ is filtered by the suffix set S to obtain $\mathcal{R}'$. If any of the sets $\mathcal{R}$ and $\mathcal{R}'$ is empty or $\mathcal{R}$ contains w (i.e. w is a root word), then w is returned as the lemma of itself. Next, variant 1.1 selects that root from $\mathcal{R}'$ as the lemma, which one shares maximum common prefix with w among all the members of $\mathcal{R}'$. In case of variant 1.2, the root from $\mathcal{R}'$ having the highest cosine similarity with w in the embedded vector space, is selected as the lemma. Note that in Indic languages, a significant number of root words found in lexical resources are hardly used and therefore, their occurrences are not found in a corpus from where the word vectors are learnt. So, if any such word appears in $\mathcal{R}'$, then its corresponding vector is not available and it might happen that the very word is the lemma of w. In this situation, we do not want to lose any viable candidate from $\mathcal{R}'$ and therefore, apply $MCPL$ operation with w to get the lemma from $\mathcal{R}'$. Version 2 of the lemmatization algorithm is the same as version 1 except for the trie searching method i.e. instead of method 2, method 1 is applied here for searching in the trie. Thus, version 2 accepts w and $POS(w)$ as the input parameters for lemmatization.

Version 3 is outlined by algorithm 2. This version makes use of both POS information and sense inventory. The additional processing module added to it deals with the words in $\mathcal{R}$ if $\mathcal{R}'$ is empty. In version 2, w itself is returned as the lemma for this situation, but here we further explore the roots present in $\mathcal{R}$. The variant 3.1 basically applies Lesk method on $\mathcal{R}$ which picks the headword having maximum cooccurrence score between its lexical glosses and $CON(w)$. If for all the members of $\mathcal{R}$, no cooccurrence is found with $CON(w)$, then w is considered as the lemma. Variant 3.2 exploits the maximum cosine similarity score to choose the lemma of w from $\mathcal{R}$. If for any root word in $\mathcal{R}$, the

Input : w, $POS(w)$ and $CON(w)$
Output: The lemma l of w
Execute Method 1 of trie searching taking w and $POS(w)$ as inputs, and obtain $\mathcal{R}$;
if $\mathcal{R} = \emptyset$ *or* $w \in \mathcal{R}$ **then**
  return w as the lemma l;
end
Filter $\mathcal{R}$ with the suffix set S and get $\mathcal{R}'$;
if $\mathcal{R}' = \emptyset$ **then**
  /* variant 3.1 (without word embedding) */
  if $\forall r \in \mathcal{R},\ MaxOverlap(r, w) = 0$ **then**
    return w as the lemma l;
  else
    $l = \arg\max_r MaxOverlap(r, w)\ \forall r \in \mathcal{R}$;
    return l;
  end
  /* variant 3.2 (with word embedding) */
  if $\forall r \in R,\ \vec{r}$ *exists* **then**
    $l = \arg\max_r\ cos\ sim(\vec{r}, \vec{w}),\ \forall r \in \mathcal{R}$;
    return l;
  else
    if $\forall r \in \mathcal{R},\ MaxOverlap(r, w) = 0$ **then**
      return w as the lemma l;
    else
      $l = \arg\max_r MaxOverlap(r, w)\ \forall r \in \mathcal{R}$;
      return l;
    end
  end
end
/* variant 3.1 (without word embedding) */
$l = \arg\max_{r'} MCPL(r', w),\ \forall r' \in \mathcal{R}'$;
return l;
/* variant 3.2 (with word embedding) */
if $\forall r' \in R',\ \vec{r'}$ *exists* **then**
  $l = \arg\max_{r'}\ cos\ sim(\vec{r'}, \vec{w}),\ \forall r' \in \mathcal{R}'$;
  return l;
else
  $l = \arg\max_{r'} MCPL(r', w),\ \forall r' \in \mathcal{R}'$;
  return l;
end

Algorithm 2: Version 3 of the Proposed Lemmatization Algorithm.

corresponding embedded vector is not found, then Lesk is applied to choose the lemma. Again, for zero cooccurrence, w is returned as the lemma of its own.

3.3 Experimentation

To evaluate the proposed lemmatization algorithm for Bengali, we use the BenLem dataset[1]. For Hindi, the development data of the shared task on dependency parsing in

[1] http://www.isical.ac.in/~utpal/resources.php

		$rad = 1$	$rad = 2$	$rad = 3$
Bengali	var 1.1	70.92%	70.65%	70.07%
	var 1.2	**71.81%**	71.72%	71.12%
Hindi	var 1.1	71.34%	71.16%	71.05%
	var 1.2	**71.68%**	71.51%	71.41%

TABLE 3.2: Lemmatization accuracy obtained using version 1

		$rad = 1$	$rad = 2$	$rad = 3$
Bengali	$\mathcal{R} = \emptyset$	1.82/18.03	0.62/23.80	0.30/50.00
	$w \in \mathcal{R}$	53.32/90.40	53.32/90.40	53.32/90.40
	$\mathcal{R}' = \emptyset$	4.25/54.22	3.98/54.13	3.68/52.03
	$\mathcal{R}' \neq \emptyset$	40.60/49.45	42.07/47.86	42.70/46.39
Hindi	$\mathcal{R} = \emptyset$	1.04/48.30	0.22/68.75	0.14/65.00
	$w \in \mathcal{R}$	67.92/91.07	67.92/91.07	67.92/91.07
	$\mathcal{R}' = \emptyset$	6.83/55.95	7.03/56.28	6.80/58.31
	$\mathcal{R}' \neq \emptyset$	24.20/21.30	24.83/20.90	25.14/20.42

TABLE 3.3: % of words resolved in each case and case-wise accuracy (in %) using variant 1.1

COLING'12 is used. The numbers of evaluated word tokens are 3,342 and 14,142 for Bengali and Hindi respectively. In the present work, proper nouns are not evaluated as they may not be morphological variations of the root words available in lexical resources. To obtain the roots and their sense bags in Bengali and Hindi, freely available dictionaries[2] are utilized. We employ FIRE Bengali and Hindi news corpus[3] for the extraction of the frequent suffixes and the vector embeddings of words. In our Bengali and Hindi suffix lists, there are 1,070 and 970 suffixes respectively. Note that in chapter 2, after the extraction of valid Bengali suffixes using n-gram statistics from FIRE corpus, the list was manually curated. In this current experiment no manual effort has been given for refining the suffix lists for both Bengali and Hindi to avoid language specific intervention as much as possible. The word2vec tool (Mikolov et al., 2013a) is trained to produce the word vectors of 200 dimensions. Following the work in Mikolov et al. (2013a), continuous bag of words (cbow) architecture with negative sampling is used for training. We evaluate the lemmatization performance by measuring the direct accuracy which gives the ratio of the number of correctly lemmatized tokens to the total number of tokens given as input to the lemmatizer. In the next subsection, the experimental results and their analysis are presented.

[2] http://dsal.uchicago.edu/dictionaries/
[3] http://fire.irsi.res.in/fire

3.3.1 Results

Accuracy of the version 1 of the proposed lemmatization algorithm on the Bengali and Hindi test datasets are given in Table 3.2. We run our experiments with different values of $rad = 1, 2$ and 3. From Table 3.2 it is evident that irrespective of the variants, lemmatization accuracy degrades with the increasing value of rad. To investigate its reason, the results of variant 1.1 are further analyzed in detail assigning rad to different values. While lemmatizing a word (say w) using variant 1.1, four exclusive cases may arise – (i) $\mathcal{R} = \emptyset$; (ii) $w \in \mathcal{R}$; (iii) $\mathcal{R}' = \emptyset$; and (iv) $\mathcal{R}' \neq \emptyset$. For the first three cases, w itself is returned as the lemma and for the last one, $MCPL$ operation is performed to choose the lemma from $\mathcal{R}'$. Table 3.3 presents the percentage of word tokens resolved in each of the above four cases for varying rad as well as the case-wise accuracy in finding the lemmas. For a particular case, the percentage of tokens resolved through it and its accuracy are provided together in the corresponding entry of Table 3.3 separated by a '/' delimiter. With the increasing value of rad, smaller percentage of words are resolved through the first case and for a particular input word, $\mathcal{R}$ contains more number of noisy roots extracted from the trie, which raises the possibility to select a wrong candidate as the lemma. That is why the accuracy of the fourth case drops when rad is increased.

Another important observation from Table 3.2 is that throughout the experiments, variant 1.2 always performs better than variant 1.1. They only vary in handling the case corresponding to $\mathcal{R}' \neq \emptyset$. In this situation, variant 1.2 chooses the lemma either by using cosine similarity or if, there exists any such root in $\mathcal{R}'$ for which the corresponding vector does not exist, then following the first variant, $MCPL$ operation is applied to choose the lemma from $\mathcal{R}'$. Table 3.4 shows the percentage of words resolved in the fourth case of variant 1.2 and its accuracy to find the lemma. Compared to the results in Table 3.3, overall performance of variant 1.2 is better than 1.1 when $\mathcal{R}' \neq \emptyset$. So, we come to the conclusion that under without POS setup, use of cosine similarity between word vectors is advantageous than $MCPL$ operation to choose the lemma from $\mathcal{R}'$.

We present the experimental results obtained by executing versions 2 and 3 of the proposed lemmatization algorithm in Table 3.5. As these versions require the contextual POS of the target word, so all the experiments are carried on both the gold and the automatic POS tagged datasets. For the automatic POS tagging, the Stanford log linear

		$rad = 1$	$rad = 2$	$rad = 3$
Bengali	*cos sim*	36.60/53.72	35.61/54.96	35.22/54.46
	$MCPL$	4/32.83	6.46/25.46	7.48/22.40
Hindi	*cos sim*	14.96/33.41	14.48/34.45	14.30/34.12
	$MCPL$	9.24/5.35	10.39/5.40	10.84/5.67

TABLE 3.4: % of words resolved and accuracy (in %) using cosine similarity and $MCPL$ when $\mathcal{R}' \neq \emptyset$ in variant 1.2

		Gold POS	Automatic POS
Bengali	var 2.1	**82.70%**	75.61%
	var 2.2	81.95%	74.89%
	var 3.1	**82.70%**	75.58%
	var 3.2	80.55%	72.59%
	BenLem	**81.24%**	**74.12%**
Hindi	var 2.1	**73.49%**	62.80%
	var 2.2	73.46%	62.80%
	var 3.1	69.42%	59.54%
	var 3.2	68.40%	58.68%
	BenLem	**68.50%**	**58.77%**

TABLE 3.5: Lemmatization accuracy obtained using version 2 ans 3

POS tagger[4] (Toutanova et al., 2003) is retrained on Bengali and Hindi. For Bengali, the training data is taken from the Linguistic Data Consortium (LDC), The University of Pennsylvania, ICON'07 NLP Tool Contest data for shared task competition on POS tagging and chunking and the Indian Institute of Technology, Kharagpur. For Hindi, we use the COLING'12 training dataset for the shared task on dependency parsing. The Bengali tagger is evaluated on the BenLem lemmatization dataset producing 79.18% accuracy and the Hindi tagger is found to be 87.84% accurate on the present reference test set for lemmatization.

Table 3.5 shows that adding the POS information sufficiently improves the lemmatization accuracy compared to version 1. A very important finding to note here is that in most of the times, variant 2.1 achieves higher accuracy than variant 2.2. Similar to the variants of version 1, the working procedures of these two variants differ from each other when $\mathcal{R}' \neq \emptyset$. For this situation, the first variant only employs $MCPL$ operation and the other one selects that particular root from $\mathcal{R}'$, which bears maximum cosine similarity with the target input word.

Table 3.6 shows the percentage of words resolved in the fourth case and the accuracy in finding the lemmas for the variants 2.1 and 2.2. So, in contrast to the results obtained

[4] http://nlp.stanford.edu/software/tagger.shtml

			gold POS	automatic POS
Bengali	var 2.1	$MCPL$	44.37/71.61	46.65/61.97
	var 2.2	*cos sim*	40.63/71.06	41.68/62.74
		$MCPL$	3.74/57.60	4.97/40.97
Hindi	var 2.1	$MCPL$	23.62/25.90	22.88/26.14
	var 2.2	*cos sim*	16.19/36.82	15.48/37.89
		$MCPL$	7.43/1.52	7.40/1.52

TABLE 3.6: % of words resolved and accuracy (in %) to find the lemmas when $\mathcal{R}' \neq \emptyset$ in version 2

using version 1, it is reflected that if POS information is included to search the potential roots from the trie, then using $MCPL$ is more beneficial than using the cosine similarity score to select the appropriate root from $\mathcal{R}'$. The additional POS helps to find more refined candidates from the trie, on which simple $MCPL$ operation is enough to choose the right lemma. Table 3.5 also provides the lemmatization accuracy on the test datasets obtained using BenLem algorithm which is so far the state of the art unsupervised lemmatizer for Indic languages. For both Bengali and Hindi, the method proposed here produces better results compared to BenLem.

In version 3, we exploit semantic matching when $\mathcal{R}' = \emptyset$. For this situation, version 1 and 2 return the input word itself as the lemma. The underlying rationale is that, sense matching between the context of the target word and the senses of the roots in $\mathcal{R}$ may help to determine the right lemma. We consider the container sentence of the target word as its context. Variant 3.1 employs the basic Lesk (Lesk, 1986) based approach whereas, variant 3.2 uses the word embedding based cosine similarity to determine the root having maximum semantic similarity with the input word. Except for the variant 3.1 on the Bengali gold POS tagged data, in all other experiments accuracy is poor than version 2. We conduct the comparative analysis between version 2 and 3 when $\mathcal{R}' = \emptyset$ and provide the results in Table 3.7. The results show that for both of the variants 3.1 and 3.2, accuracy does not improve using semantics revealing that simple Lesk method or word embedding based cosine similarity cannot capture the right sense here and hence, fail to determine the appropriate lemma.

		Gold POS	Automatic POS
Bengali	ver 2	5.86/45.92	7.63/46.66
	var 3.1	5.86/45.92	7.63/46.28
	var 3.2	5.86/21.94	7.63/16.47
Hindi	ver 2	9.26/58.85	8.02/54.97
	var 3.1	9.26/14.80	8.02/14.36
	var 3.2	9.26/4.20	8.02/3.70

TABLE 3.7: % of words resolved and accuracy (in %) to find the lemmas when $\mathcal{R}' = \emptyset$ in version 2 and 3

3.4 An Initial Neural Lemmatization

Due to the immense popularity of neural network applications in NLP tasks (Collobert et al., 2011; Mikolov et al., 2013a), a neural based approach for lemmatization is attempted next. To tackle the lemmatization problem from neural network paradigm, the following challenge appears. If it is considered as a classification task, then the number of classes equals to the number of roots present in the language thus making the task practically impossible. We rather model lemmatization as a task of lemma transduction. To handle the words in a neural framework, word embedding technique is used to represent words as vectors. Given a word along with its context represented in an array of vectors as input, the model is designed in such a way so that the output vector should correspond to the appropriate lemma of the input surface word. Initially the network is trained on a set of <surface word with context, lemma> mapped pairs to learn its parameters and then the trained model is tested to find the lemma of unknown surface words. Throughout the experiment, we set 3 as the input length i.e. combination of the previous word, the surface word and the next word together constitutes the context. However, the proposed model can be extended to support arbitrary context length. The next section describes the network model.

3.4.1 The Network Model

Our proposed model is a feedforward neural network as shown in Figure 3.2. All words are handled by their corresponding vector embeddings and we use the word2vec tool (Mikolov et al., 2013a) to obtain the 200-dimensional word vectors. For training of the recurrent neural model of word2vec, we use continuous bag of words (cbow) architecture and in the output layer of the model, negative sampling is used.

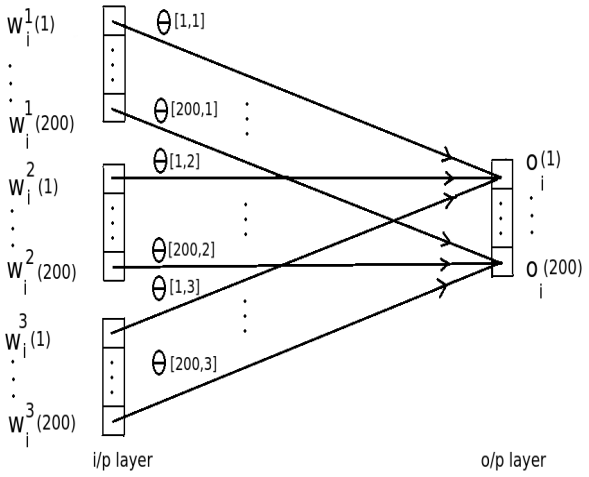


FIGURE 3.2: Architecture of the proposed neural lemmatization model.

Consider n training samples present in the training set and (w_i^1, w_i^2, w_i^3) be the i^{th} training sample representing the respective vectors of the preceding word, the target surface word and the succeeding word. Let o_i be the output for the i^{th} sample. For an arbitrary word vector w, its j^{th} dimension is denoted by $w(j)$. The input layer of the model accepts an array of three contextual word vectors and hence, consists of total 600 nodes and the output layer contains total 200 nodes corresponding to the transduced lemma vector. The network architecture is designed in a novel way. Instead of making all to all connections between the input and the output layer, we propose a dimension specific architecture of the network. Since every dimension of a word vector preserves a specific characteristic of the corresponding word, so we restrict a particular dimension of the output to be attributed only from that same respective dimensions of the three input word vectors. That is, $\forall\ j = 1, \ldots, 200$, $o_i(j)$ is influenced by $w_i^1(j), w_i^2(j)$ and $w_i^3(j)$ only. So, there are total 600 (200×3) synaptic connections present in the network. Let $\theta \in \mathbb{R}^{200\times 3}$ be the weight matrix where $\theta[j,k]$ is the weight of the connection connecting $o_i(j)$ and $w_i^k(j)\ \forall\ j = 1, \ldots, 200$ and $\forall\ k = 1, 2, 3$. Here we use batch mode learning to obtain the optimum θ. To make the model work as a transducer, instead of choosing a bounded function like sigmoidal or hyperbolic tangent, we select the identity function for activation i.e.

$$o_i(j) = \sum_{k=1}^{3} w_i^k(j) \times \theta[j,k],\ \forall\ j = 1, \ldots, 200. \tag{3.1}$$

Figure 3.2 shows the schematic diagram of the proposed network model. We train our model in the following way. Our main objective is to increase the cosine similarity between o_i and the desired word vector for i^{th} training example, say d_i. d_i corresponds

to the vector of that particular dictionary root word which is the appropriate lemma for i^{th} training instance. We define the cost function for the i^{th} training example J_i as follows

$$J_i = 1 - cosine\_sim(d_i, o_i) \tag{3.2}$$

where $cosine\_sim(d_i, o_i)$ denotes the cosine of the angle between d_i and o_i. The overall cost function J is the average of J_i over all n training examples. From equation (1) and (2) it can be derived that for the i^{th} training example,

$$\frac{\partial J_i}{\partial \theta[j,k]} = -\frac{w_i^k(j)}{\|d_i\|\|o_i\|}\left[d_i(j) - \frac{d_i \cdot o_i}{\|o_i\|^2}\left(o_i(j)\right)\right] \quad \forall\ j = 1, \ldots, 200\ ;\ \forall\ k = 1, 2, 3 \tag{3.3}$$

where $d_i \cdot o_i$ denotes the dot product between d_i and o_i and $\|\bullet\|$ denotes the Euclidean norm operation. We minimize the cost function J using batch gradient descent with a learning rate of 0.1. $\theta[j,k]$ for the next iteration is updated by the following rule.

$$\theta_{next}[j,k] = \theta_{previous}[j,k] - \frac{0.1}{n}\sum_{i=1}^{n}\frac{\partial J_i}{\partial \theta[j,k]} \tag{3.4}$$

The process to obtain the lemma of a test sample is as follows. If there are m root words $r_1, r_2, \ldots, r_m$ present in the dictionary and o is the output vector of the neural model for the input test sample, then choose r_i as the lemma where $r_i \leftarrow \underset{r_j}{\text{argmax}}\ cosine\_sim(vec(r_j), o)\ \forall\ j = 1$ to m. The function vec is defined as follows. Taking a word as an argument, vec returns the corresponding embedded vector.

3.5 Experimentation

3.5.1 Datasets Preparation

The above model is evaluated for Bengali. To train the neural model, a huge amount of training sample is needed, for which lot of time as well as manual labour is required. Thus, for training and testing purpose, we generate samples in Bengali in a semi-automatic way involving minimal human intervention. Each sample consists of a surface word accompanied with its contextual neighbours. The process of creating the training dataset consists of two parts: 1. generating the training samples

for which transformation rules from roots to inflections follow regular patterns (e.g. where simple addition/subtraction of a frequent suffix to/from a root produces a surface word) 2. generating the irregular samples for which no well-formed transformation rule works. In Bengali, usually a root word and its morphological variants are both orthographically and semantically similar (e.g. root; inflections: 'করা'/karaa; 'করার'/karaar, 'করলাম'/karlaam). In cases of irregular samples, only semantic similarity exists between a lemma and its variants (e.g. root; inflections: 'থাকা'/thaakaa; 'ছিলে'/chhile, 'ছিল'/chhila). To generate the regular samples, we follow the rationale that for suffixing languages like Bengali, "morphologically-related words typically share a long common prefix" (Paik et al., 2011). Initially, all the roots having length[5] greater than or equal to 6, are mined from the Bengali digital dictionary available with the Chicago university[6]. Next, for every root word, its 10 nearest words in the vector space are considered (cosine similarity is taken as the distance measure) and among them, only those are selected for which the overlapping prefix length with the root is at least 6. Following the process, we are able to generate a set of <token; lemma> mapped pairs which mutually have very high syntactic and semantic association and thus, these mappings are mostly context invariant. To generate the contexts of the inflections in the mapped set, we search for their instances in the FIRE corpus and when an instance is found, the surrounding context is extracted. There are 9,536 regular samples in the training dataset. To generate the irregular samples, at first 120 different irregular word forms are crafted and out of them, those ones which have association with fixed lemmas irrespective of context, are spotted and their contexts are randomly picked up from the FIRE corpus. The particular word forms that have varying lemmas depending on the context, are handled specially and their combinations with different context-lemma pairs are manually arranged. In this way, we get 9,623 irregular samples. Both the regular and the irregular samples are put in the training data with almost equal proportion to make the dataset balanced. In total, there are 19,159 training samples. The test dataset is prepared by taking samples randomly from the FIRE corpus. Each sample in the training and test data consists of 3 adjacent words appearing in the text where the middle one is the target word for lemmatization. For all the samples, the respective gold lemmas of the target words are also provided.

[5]Length of a word is measured in terms of the number of unicode character/s present in that word.
[6]http://dsal.uchicago.edu/dictionaries/list.html

	Accuracy
Proposed Lemmatizer	69.57%
Baseline Method	68.20%

TABLE 3.8: Lemmatization accuracy.

3.5.2 Results

We use 10-fold cross validation on the training data to obtain the optimum θ. The proposed lemmatization method is evaluated on the test dataset by computing the direct accuracy i.e. the fraction of the total number of surface words which are correctly lemmatized. To choose a suitable baseline, we calculate the respective cosine similarities of the embedded vector of every surface word in the test dataset with the vectors of all dictionary root words and select that root as the lemma for which the corresponding vector possesses maximum similarity with the surface word vector. Table 3.8 presents the experimental results. Our proposed neural lemmatizer beats the simple cosine similarity based baseline by a 1.37% margin out of $2,126$ instances i.e. 29 more instances are correctly lemmatized by our method. However, the overall accuracy is not very high reflecting that much more amount of training data is needed to efficiently induce the lemmas.

3.6 Summary

In this chapter, we propose the following key changes in BenLem algorithm. The string similarity measure of Majumder et al. (2007) to find the potential root words is replaced by a novel trie search strategy. Based on the availability of POS information for the input surface word, two different searching techniques are devised. Note that as opposed to the string distance threshold parameter needed in BenLem, the 'trie search with POS' strategy does not require any parameter and hence, is more generalized. Another notable aspect is to plug-in word embedding into lemmatization algorithm and investigate its role on the performance. In cases where expensive resources are not used, word embedding improves the accuracy for both Bengali and Hindi but its combination with POS degrades the performance. A sophisticated way of integrating the word vectors along with other linguistic information may help the purpose.

This chapter also presents an initial neural network model for supervised lemmatization. The proposed model takes an input word along with its adjacent left and right neighbours as context and transduces the lemma of the middle word. All the words are represented by their corresponding vector embeddings. To reduce the network parameters, we propose a dimension specific architecture. The loss of the model is measured using cosine distance and the weights are updated by the standard backpropagation rule. The model is tested on Bengali. However, the obtained accuracy is not very impressive which indicates that it is not worthy to design a lemma transduction model due to the huge number of lemmas present in the language. Rather, capturing the word-lemma transformation patterns is much more reasonable.

Chapter 4

Context Sensitive Lemmatization Using Gated Recurrent Networks

4.1 Introduction

So far we put our focus mainly on developing unsupervised knowledge-based lemmatization methods. Lastly a small neural model has been built, which, taking word vectors as input, induces lemmas. Experimental result shows that this type of supervised approach needs a huge amount of training data, which is a clear bottleneck. Thus, in this chapter, a novel network architecture is proposed, that does not function as a lemma generator. Rather, it classifies the transformation between a surface word to the corresponding lemma.

The key contributions of this work are as follows. We address context sensitive lemmatization introducing a two-stage bidirectional gated recurrent neural network (BGRNN) architecture. Our model is a supervised one that needs lemma tagged continuous text to learn. Its two most important advantages compared to the state-of-the-art supervised models (Chrupala et al., 2008; Toutanova and Cherry, 2009; Gesmundo and Samardzic, 2012; Müller et al., 2015) are - (*i*) there is no need to define hand-crafted features such as the word form, presence of special characters, character alignments, surrounding words etc. (*ii*) parts of speech and other morphological attributes of the surface words are not required for joint learning. Additionally, unknown word forms are also taken care of as the transformation between word-lemma pair is learnt, not the lemma itself. We

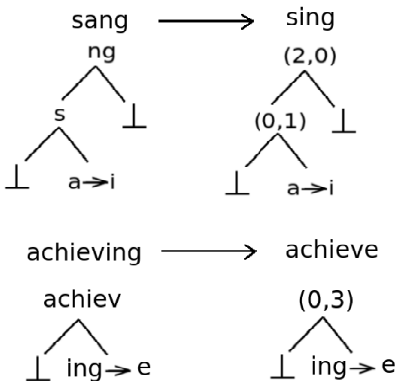


FIGURE 4.1: Edit trees for the word-lemma pairs 'sang-sing' and 'achieving-achieve'.

exploit two steps learning in our method. At first, characters in the words are passed sequentially through a BGRNN to get a syntactic embedding of each word and then the outputs are combined with the corresponding semantic embeddings. Finally, mapping between the combined embeddings to word-lemma transformations are learnt using another BGRNN.

For the present work, the model is assessed on nine languages having diverse morphological variations. Out of them, two (Bengali and Hindi) are Indo-Aryan, six (Catalan, Dutch, Italian, Latin, Romanian and Spanish) are Indo-European and the last one (Hungarian) is Uralic. To evaluate the proposed model on Bengali, a lemma annotated continuous text has been developed. As so far there is no such standard large dataset for supervised lemmatization in Bengali, the prepared one would surely contribute to the respective NLP research community. For the remaining languages, standard datasets are used for experimentation. Experimental results reveal that our method outperforms Lemming (Müller et al., 2015) and Morfette (Chrupala et al., 2008) on all the languages except Bengali.

4.2 Method

It is noteworthy here that the supervised lemmatization methods (Chrupala et al., 2008; Gesmundo and Samardzic, 2012; Müller et al., 2015) do not try to classify the lemma of a given word form as it is infeasible due to having a large number of lemmas in a language. Rather, learning the transformation between word-lemma pair is more generalized and it can handle the unknown word forms too. Several representations of word-lemma

transformation have been introduced so far such as shortest edit script (SES), label set, edit tree by Chrupala et al. (2008), Gesmundo and Samardzic (2012) and Müller et al. (2015) respectively. Following Müller et al. (2015), we consider lemmatization as the edit tree classification problem. An edit tree embeds all the necessary edit operations within it i.e. insertions, deletions and substitutions of strings required throughout the transformation process. Figure 4.1 depicts two edit trees that map the inflected English words *'sang'* and *'achieving'* to their respective lemmas *'sing'* and *'achieve'*. For generalization, edit trees encode only the substitutions and the length of prefixes and suffixes of the longest common substrings (LCS). For example, in Figure 4.1 the root node of the edit tree between *achieving-achieve* contains the tuple (0,3) as the prefix and suffix length of their LCS (*achiev*) in the source string are 0 and 3 respectively. Note that the leaf node contains the substitution *ing* to *e* as there exists no common substring. Initially, all unique edit trees are extracted from the associated surface word-lemma pairs present in the training set. The extracted trees refer to the class labels in our model. So, for a test word, the goal is to classify the correct edit tree which, applied on the word, returns the lemma.

Next, the architecture of the proposed neural lemmatization model is described. It is evident that for morphologically rich languages, both syntactic and semantic knowledge help in lemmatizing a surface word. Nowadays, it is a common practice to embed the functional properties of words into vector representations. Despite the word vectors prove very effectual in semantic processing tasks, they are modelled using the distributional similarity obtained from a raw corpus. Morphological regularities, local and non-local dependencies in character sequences that play deciding roles to find the lemmas, are not taken into account where each word has its own vector interpretation. This issue is addressed by incorporating two different embeddings into our model. Semantic embedding is achieved using word2vec (Mikolov et al., 2013a,b), which has been empirically found highly successful. To devise the syntactic embedding of a word, the following work of Ling et al. (2015) is considered, that uses compositional character to word model using bidirectional long-short term memory (BLSTM) network. In our experiments, different gated recurrent cells such as LSTM (Graves, 2013) and GRU (Cho et al., 2014), are explored. The next subsection describes the module to construct the syntactic vectors by feeding the character sequences into BGRNN architecture.

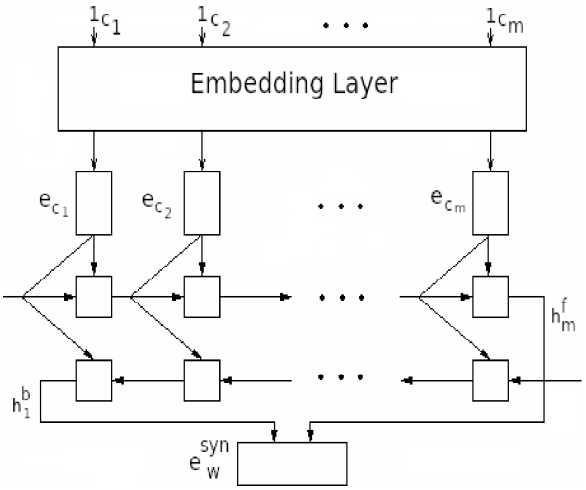


FIGURE 4.2: Syntactic vector composition for a word.

4.2.1 Forming Syntactic Embeddings

Our goal is to build syntactic embeddings of words that capture the similarities in morphological level. Given an input word w, the target is to obtain a d dimensional vector representing the syntactic structure of w. The procedure is illustrated in Figure 4.2. At first, an alphabet of characters is defined as C. w is represented as a sequence of characters $c_1, \ldots, c_m$ where m is the word length and each character c_i is defined as a one hot encoded vector $\mathbf{1}_{c_i}$, having one at the index of c_i in the alphabet C. An embedding layer is defined as $\mathbf{E}_c \in \mathbb{R}^{d_c \times |C|}$, that projects each one hot encoded character vector to a d_c dimensional embedded vector. For a character c_i, its projected vector $\mathbf{e}_{c_i}$ is obtained from the embedding layer $\mathbf{E}_c$, using this relation $\mathbf{e}_{c_i} = \mathbf{E}_c \cdot \mathbf{1}_{c_i}$ where '$\cdot$' is the matrix multiplication operation.

Given a sequence of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$ as input, a LSTM cell computes the state sequence $\mathbf{h}_1, \ldots, \mathbf{h}_m$ using the following equations:

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{V}_f \mathbf{c}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{V}_i \mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
&\quad + \mathbf{i}_t \odot tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{V}_o \mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t),
\end{aligned}
$$

Whereas, the updation rules for GRU are as follows

$$\begin{aligned}
\mathbf{z}_t &= \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z) \\
\mathbf{r}_t &= \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r) \\
\mathbf{h}_t &= (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} \\
&\quad + \mathbf{z}_t \odot tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h),
\end{aligned}$$

σ denotes the sigmoid function and $\odot$ stands for the element-wise (Hadamard) product. Unlike the simple recurrent unit, LSTM uses an extra memory cell $\mathbf{c}_t$ that is controlled by three gates - input ($\mathbf{i}_t$), forget ($\mathbf{f}_t$) and output ($\mathbf{o}_t$). $\mathbf{i}_t$ controls the amount of new memory content added to the memory cell, $\mathbf{f}_t$ regulates the degree to which the existing memory is forgotten and $\mathbf{o}_t$ finally adjusts the memory content exposure. $\mathbf{W}$, $\mathbf{U}$, $\mathbf{V}$ (weight matrices), $\mathbf{b}$ (bias) are the parameters.

Without having a memory cell like LSTM, a GRU uses two gates namely update ($\mathbf{z}_t$) and reset ($\mathbf{r}_t$). The gate, $\mathbf{z}_t$ decides the amount of update needed for activation and $\mathbf{r}_t$ is used to ignore the previous hidden states (when close to 0, it forgets the earlier computation). So, for a sequence of projected characters $\mathbf{e}_{c_1}, \ldots, \mathbf{e}_{c_m}$, the forward and the backward networks produce the state sequences $\mathbf{h}_1^f, \ldots, \mathbf{h}_m^f$ and $\mathbf{h}_m^b, \ldots, \mathbf{h}_1^b$ respectively. Finally, the syntactic embedding of w, denoted as $\mathbf{e}_w^{syn}$, is obtained by concatenating the final states of these two sequences.

$$\mathbf{e}_w^{syn} = [\mathbf{h}_1^b, \mathbf{h}_m^f]$$

4.2.2 Model

The final integrated model is depicted in Figure 4.3. For a word w, let $\mathbf{e}_w^{sem}$ denotes its semantic embedding obtained using word2vec. Both the vectors, $\mathbf{e}_w^{syn}$ and $\mathbf{e}_w^{sem}$ are concatenated together to shape the composite representation $\mathbf{e}_w^{com}$ which carries the morphological and distributional information within it. Firstly, for all the words present in the training set, their composite vectors are generated. Next, they are fed sentence-wise into the next level of BGRNN to train the model for the edit tree classification task. This second level bidirectional network accounts the local context in both forward and backward directions, which is essential for lemmatization in context sensitive languages.

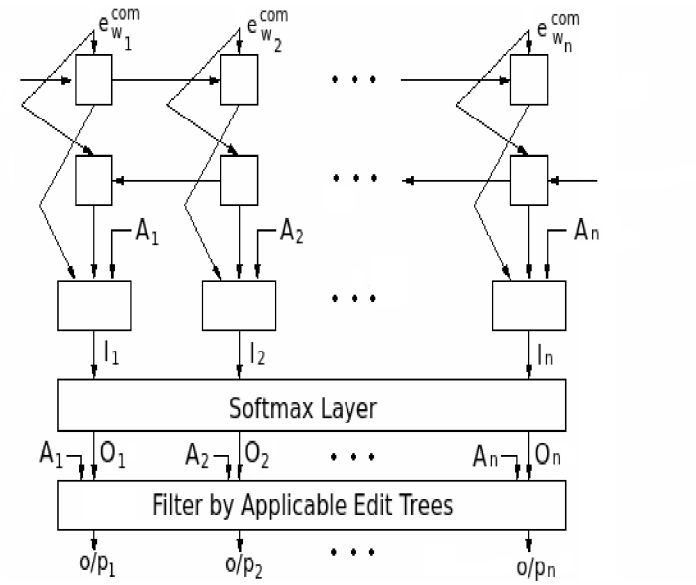


FIGURE 4.3: Second level BGRNN model for edit tree classification.

Let, $\mathbf{e}^{com}_{w_1}, \ldots, \mathbf{e}^{com}_{w_n}$ be the input sequence of composite vectors to the BGRNN model, representing a sentence having n words $w_1, \ldots, w_n$. For the i^{th} vector $\mathbf{e}^{com}_{w_i}$, $\mathbf{h}^f_i$ and $\mathbf{h}^b_i$ denote the forward and backward states respectively carrying the informations of $w_1, \ldots, w_i$ and $w_i, \ldots, w_n$.

4.2.2.1 Incorporating Applicable Edit Trees Information

One aspect that has not been taken into account so far, is that for a word all unique edit trees extracted from the training set are not applicable as this would lead to incompatible substitutions. For example, the edit tree for the word-lemma pair 'sang-sing' depicted in Figure 4.1, cannot be applied on the word 'achieving'. This information is prior before training the model i.e. for any arbitrary word, we can sort out the subset of unique edit trees from the training samples in advance, which are applicable on it. In general, if all the unique edit trees in the training data are set as the class labels, the model will learn to distribute the probability mass over all the classes which is a clear-cut bottleneck. In order to alleviate this problem, a novel strategy is taken so that for individual words in the input sequence, the model will learn, to which classes, the output probability should be apportioned.

Let $T = \{t_1, \ldots, t_k\}$ be the set of distinct edit trees found in the training set. For the word w_i in the input sequence $w_1, \ldots, w_n$, its applicable edit trees vector is defined as $\mathbf{A}_i = (a^1_i, \ldots, a^k_i)$ where $\forall j \in \{1, \ldots, k\}$, $a^j_i = 1$ if t_j is applicable for w_i, otherwise 0.

Hence, $\mathbf{A}_i$ holds the information regarding the set of edit trees to concentrate upon, while processing the word w_i. $\mathbf{A}_i$ is combined together with $\mathbf{h}_i^f$ and $\mathbf{h}_i^b$ for the final classification task as following,

$$\mathbf{l}_i = softplus(\mathbf{L}^f \mathbf{h}_i^f + \mathbf{L}^b \mathbf{h}_i^b + \mathbf{L}^a \mathbf{A}_i + \mathbf{b}_l),$$

where 'softplus' denotes the activation function $f(x) = ln(1 + e^x)$ and $\mathbf{L}^f, \mathbf{L}^b, \mathbf{L}^a$ and $\mathbf{b}_l$ are the parameters trained by the network. At the end, $\mathbf{l}_i$ is passed through the softmax layer to get the output labels for w_i.

To pick the correct edit tree from the output of the softmax layer, the prior information $\mathbf{A}_i$ is used. Instead of choosing the class that gets the maximum probability, the maximum over the classes corresponding to the applicable edit trees is selected. The idea is expressed as follows. Let $\mathbf{O}_i = (o_i^1, \ldots, o_i^k)$ be the output of the softmax layer. Instead of opting for the maximum over $o_i^1, \ldots, o_i^k$ as the class label, the highest probable class out of those corresponding to the applicable edit trees, is picked up. That is, the particular edit tree $t_j \in T$ is considered as the right candidate for w_i, where

$$j = \text{argmax}_{j' \in \{1,\ldots,k\} \ \wedge \ a_i^{j'} = 1} \ o_i^{j'}$$

In this way, we cancel out the non-applicable classes and focus only on the plausible candidates.

4.3 Experimentation

Out of the nine reference languages, initially four of them (Bengali, Hindi, Latin and Spanish) are chosen for in-depth analysis. An exhaustive set of experiments is conducted-such as determining the direct lemmatization accuracy, accuracy obtained without using applicable edit trees in training, measuring the model's performance on the unseen words etc. on these four languages. Later five more languages are considered (Catalan, Dutch, Hungarian, Italian and Romanian) mostly for testing the generalization ability of the proposed method. For these additional languages, only the lemmatization accuracy is presented in section 4.3.2.

	# Sentences	# Word Tokens
Bengali	1,702	20,257
Hindi	36,143	819,264
Latin	15,002	165,634
Spanish	15,984	477,810

TABLE 4.1: Dataset statistics of the 4 languages.

Datasets: As Bengali is a low-resourced language, a relatively large lemma annotated dataset is prepared for the present work using Tagore's short stories collection[1] and randomly selected news articles from miscellaneous domains. One linguist took around 2 months to complete the annotation which was checked by another person and differences were sorted out. Out of the 91 short stories of Tagore, the value of (# tokens / # distinct tokens) is calculated for each story. Based on this value (lower is better), top 11 stories are selected. The news articles[2] are crafted from the following domains: animal, archaeology, business, country, education, food, health, politics, psychology, science and travelogue. In Hindi, the COLING'12 shared task data for dependency parsing and Hindi WSD health and tourism corpora[3] (Khapra et al., 2010) are combined together[4]. For Latin, the PROIEL treebank (Haug and Jøhndal, 2008) is used and for Spanish, the training and development datasets of CoNLL'09 (Hajič et al., 2009) shared task on syntactic and semantic dependencies are merged. The dataset statistics are given in Table 4.1. Lemmatization performance is measured by the direct accuracy which is the ratio of the number of correctly lemmatized words to the total number of input words. The experiments are performed using 4 fold cross validation technique i.e. the datasets are equi-partitioned into 4 parts at sentence level and then each part is tested exactly once using the model trained on the remaining 3 parts. Finally, the average accuracy over 4 fold is reported.

Induction of Edit Tree Set: Initially, distinct edit trees are induced from the word-lemma pairs present in the training set. Next, the words in the training data are annotated with their corresponding edit trees. Training is accomplished on this edit tree tagged text. Figure 4.4 plots the growth of the edit tree set against the number of word-lemma samples in the four languages. With the increase of samples, the size of

[1] www.rabindra-rachanabali.nltr.org
[2] http://www.anandabazar.com/
[3] http://www.cfilt.iitb.ac.in/wsd/annotated_corpus/
[4] The Bengali and Hindi datasets are available in http://aclweb.org/anthology/P/P17/

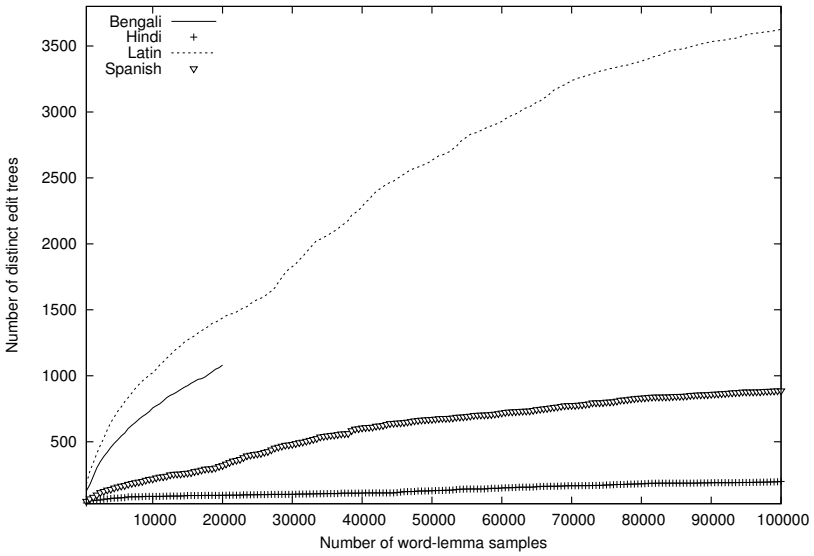


FIGURE 4.4: Increase of the edit tree set size with the number of word-lemma samples.

	Bengali	Hindi	Latin	Spanish
BLSTM-BLSTM	90.84/91.14	94.89/**94.90**	89.35/89.52	97.85/97.91
BGRU-BGRU	90.63/90.84	94.44/94.50	89.40/**89.59**	98.07/**98.11**
Lemming	**91.69**	91.64	88.50	93.12
Morfette	90.69	90.57	87.10	92.90

TABLE 4.2: Lemmatization accuracy (in %) without/with restricting output classes.

edit tree set gradually converges revealing the fact that most of the frequent transformation patterns (both regular and irregular) are covered by the induction process. From Figure 4.4, morphological richness can be compared across the languages. When convergence happens quickly i.e. at relatively less number of samples, it evidences that the language is less complex. Among the four reference languages, Latin stands out as the most intricate, followed by Bengali, Spanish and Hindi.

Semantic Embeddings: The distributional word vectors for Bengali and Hindi are obtained by training the word2vec model on FIRE Bengali and Hindi news corpora[5]. Following the work by Mikolov et al. (2013a), continuous-bag-of-words architecture with negative sampling is used to get 200 dimensional word vectors. For Latin and Spanish, the embeddings released by Bamman and Smith (2012)[6] and Cardellino (2016)[7] are used respectively.

[5] http://fire.irsi.res.in/fire
[6] http://www.cs.cmu.edu/~dbamman/latin.html
[7] http://crscardellino.me/SBWCE/

	Bengali	Hindi	Latin	Spanish
BLSTM-BLSTM	86.46/89.52	94.34/94.52	85.70/87.35	97.39/97.62
BGRU-BGRU	86.39/88.90	93.84/94.04	85.49/86.87	97.51/97.73

TABLE 4.3: Lemmatization accuracy (in %) without using applicable edit trees in training.

Syntactic Representation: From the statistics of word length versus frequency from the datasets, it is found out that irrespective of the languages, longer words (have more than 20-25 characters) are few in numbers. Based on this finding, each word is limited to a sequence of 25 characters. Smaller words are padded null characters at the end and for the longer words, excess characters are truncated out. So, each word is represented as a 25 length array of one hot encoded vectors which is given input to the embedding layer that works as a look up table producing an equal length array of embedded vectors. Initialization of the embedding layer is done randomly and the embedded vector dimension is set to 10. Eventually, the output of the embedding layer is passed to the first level BGRNN for learning the syntactic representation.

Hyper Parameters: There are several hyper parameters in our model such as the number of neurons in the hidden layer ($\mathbf{h}_t$) of both first and second level BGRNN, learning mode, number of epochs to train the models, optimization algorithm, dropout rate etc. Experiments are carried out with different settings of these parameters and the particular parameter values are reported where optimum results are achieved. For both the bidirectional networks, number of hidden layer neurons is set to 64. Online learning is applied for updation of the weights. Number of epochs varies across languages to converge the training. It is maximum for Bengali (around 80 epochs), followed by Latin, Spanish and Hindi taking around 50, 35 and 15 respectively. Throughout the experiments, we set the dropout rate as 0.2 to prevent over-fitting. Different optimization algorithms like AdaDelta (Zeiler, 2012), Adam (Kingma and Ba, 2014), RMSProp (Dauphin et al., 2015) are explored. Out of them, Adam yields the best result. We use the categorical cross-entropy as the loss function in our model.

Baselines: To compare our method Lemming[8] and Morfette[9] are chosen as the baselines. Both the model jointly learns lemma and other morphological tags in context. Lemming uses a 2nd-order linear-chain CRF to predict the lemmas whereas, the current

[8] http://cistern.cis.lmu.de/lemming/
[9] https://github.com/gchrupala/morfette

version of Morfette is based on structured perceptron learning. As POS information is a compulsory requirement of these two models, the Bengali data is manually POS annotated. For the other languages, the tags were already available. Although this comparison is partially biased as the proposed method does not need POS information, but the experimental results show the effectiveness of our model. There is an option in Lemming and Morfette to provide an exhaustive set of root words which is used to exploit the dictionary features i.e. to verify if a candidate lemma is a valid form or not. To make the comparisons consistent, we do not exploit any external dictionary in our experiments.

4.3.1 Results

The lemmatization results are presented in Table 4.2. The proposed model is explored with two types of gated recurrent cells - LSTM and GRU. As there are two successive bidirectional networks - the first one for building the syntactic embedding and the next one for the edit tree classification, so basically we deal with two different models BLSTM-BLSTM and BGRU-BGRU. Table 4.2 shows the comparison results of these models with Lemming and Morfette. In all cases, the average accuracy over 4 fold cross validation on the datasets is reported. For an entry 'x/y' in Table 4.2, x denotes the accuracy without output classes restriction, i.e. taking the maximum over all edit tree classes present in the training set, whereas y refers to the accuracy when output is restricted in only the applicable edit tree classes of the input word. Except for Bengali, the proposed models outperform the baselines for the other three languages. In Hindi, BLSTM-BLSTM gives the best result (94.90%). For Latin and Spanish, the highest accuracy is achieved by BGRU-BGRU (89.59% and 98.11% respectively). In the Bengali dataset, Lemming produces the optimum result (91.69%) beating its closest performer BLSTM-BLSTM by 0.55%. It is to note that the training set size in Bengali is smallest compared to the other languages (on average, $16,712$ tokens in each of the 4 folds). Overall, BLSTM-BLSTM and BGRU-BGRU perform equally good. For Bengali and Hindi, the former model is better and for Latin and Spanish, the later yields more accuracy. Throughout the experiments, restricting the output over applicable classes improves the performance significantly. The maximum improvements we get are: 0.30% in Bengali using BLSTM-BLSTM (from 90.84% to 91.14%), 0.06% in Hindi using BGRU-BGRU (from 94.44% to 94.50%), 0.19% in Latin using BGRU-BGRU (from 89.40% to 89.59%) and 0.06% in

Bengali	Hindi	Latin	Spanish
27.17	5.25	15.74	7.54

TABLE 4.4: Proportion of unknown word forms (in %) present in the test sets.

	Bengali	Hindi	Latin	Spanish
BLSTM-BLSTM	71.06/72.10	87.80/88.18	60.85/61.63	88.06/88.79
BGRU-BGRU	70.44/71.22	88.34/88.40	60.65/61.52	91.48/92.25
Lemming	74.10	90.35	57.19	58.89
Morfette	70.27	88.59	47.41	57.61

TABLE 4.5: Lemmatization accuracy (in %) on unseen words.

	Bengali	Hindi	Latin	Spanish
BLSTM-BLSTM	56.16/66.26	87.42/88.41	49.80/56.05	86.22/87.97
BGRU-BGRU	59.45/66.84	87.19/88.26	50.24/55.35	86.74/88.49

TABLE 4.6: Lemmatization accuracy (in %) on unseen words without using applicable edit trees in training.

Spanish using BLSTM-BLSTM (from 97.85% to 97.91%). To compare between the two baselines, Lemming consistently performs better than Morfette (the maximum difference between their accuracies is 1.40% in Latin).

Effect of Training without Applicable Edit Trees: The impact of applicable edit trees in training is also investigated. To see the effect, our model is trained without giving the applicable edit trees information as input. In the model design, the equation for the final classification task is changed as follows,

$$\mathbf{l}_i = softplus(\mathbf{L}^f \mathbf{h}_i^f + \mathbf{L}^b \mathbf{h}_i^b + \mathbf{b}_l),$$

The results are presented in Table 4.3. Except for Spanish, BLSTM-BLSTM outperforms BGRU-BGRU in all the other languages. As compared with the results in Table 4.2, for every model, training without applicable edit trees degrades the lemmatization performance. In all cases, BGRU-BGRU model gets more affected than BLSTM-BLSTM. Language-wise, the drops in its accuracy are: 1.94% in Bengali (from 90.84% to 88.90%), 0.46% in Hindi (from 94.50% to 94.04%), 2.72% in Latin (from 89.59% to 86.87%) and 0.38% in Spanish (from 98.11% to 97.73%).

One important finding to note in Table 4.3 is that irrespective of any particular language

and model used, the amount of increase in accuracy due to the output restriction on the applicable classes is much more than that observed in Table 4.2. For instance, in Table 4.2 the accuracy improvement for Bengali using BLSTM-BLSTM is 0.30% (from 90.84% to 91.14%), whereas in Table 4.3 the corresponding value is 3.06% (from 86.46% to 89.52%). These outcomes signify the fact that training with the applicable edit trees already learns to dispense the output probability to the legitimate classes over which, output restriction cannot yield much enhancement.

Results for Unseen Word Forms: Next, the lemmatization performance is analyzed on those words which were absent in the training set. Table 4.4 shows the proportion of unseen forms averaged over 4 folds on the datasets. In Table 4.5, we present the accuracy obtained by our models and the baselines. For Bengali and Hindi, Lemming produces the best results (74.10% and 90.35%). For Latin and Spanish, BLSTM-BLSTM and BGRU-BGRU obtain the highest accuracy (61.63% and 92.25%) respectively. In Spanish, our model gets the maximum improvement over the baselines. BGRU-BGRU beats Lemming with 33.36% margin (on average, out of 9,011 unseen forms, 3,005 more tokens are correctly lemmatized). Similar to the results in Table 4.2, the results in Table 4.5 evidences that restricting the output in applicable classes enhances the lemmatization performance. The maximum accuracy improvements due to the output restriction are: 1.04% in Bengali (from 71.06% to 72.10%), 0.38% in Hindi (from 87.80% to 88.18%) using BLSTM-BLSTM and 0.87% in Latin (from 60.65% to 61.52%), 0.77% in Spanish (from 91.48% to 92.25%) using BGRU-BGRU.

Further, performance of our models trained without the applicable edit trees information is investigated on the unseen word forms. The results are given in Table 4.6. As expected, for every model, the accuracy drops compared to the results shown in Table 4.5. The only exception that we find out is in the entry for Hindi with BLSTM-BLSTM. Though without restricting the output, the accuracy in Table 4.5 (87.80%) is higher than the corresponding value in Table 4.6 (87.42%), but after output restriction, the performance changes (88.18% in Table 4.5, 88.41% in Table 4.6) which reveals that only selecting the maximum probable class over the applicable ones would be a better option for the unseen word forms in Hindi.

Effects of Semantic and Syntactic Embeddings in Isolation: To understand the impact of the combined word vectors on the model's performance, the accuracy is

	Sem. Embedding	Syn. Embedding
Bengali	90.76/91.02	86.61/86.82
Hindi	94.86/94.86	91.24/91.25
Latin	88.90/89.09	85.31/85.49
Spanish	97.95/98	96.07/96.10

TABLE 4.7: Results (in %) obtained using semantic and syntactic embeddings separately.

	# Sentences	# Word Tokens
Catalan	14,832	474,069
Dutch	13,050	197,925
Hungarian	1,351	31,584
Italian	13,402	282,611
Romanian	8,795	202,187

TABLE 4.8: Dataset statistics of the 5 additional languages.

	Catalan	Dutch	Hungarian	Italian	Romanian
BLSTM-BLSTM	97.93/**97.95**	93.20/**93.44**	91.03/**91.46**	96.06/**96.09**	94.25/**94.32**
Lemming	89.80	86.95	87.95	92.51	93.34
Morfette	89.46	86.62	86.52	92.02	94.13

TABLE 4.9: Lemmatization accuracy (in %) for the 5 languages.

measured experimenting with each one of them separately. While using the semantic embedding, only distributional word vectors are used for edit tree classification. On the other hand, to test the effect of the syntactic embedding exclusively, output from the character level recurrent network is fed to the second level BGRNN. The results are presented in Table 4.7. For Bengali and Hindi, experiments are carried out with the BLSTM-BLSTM model as it gives better results for these languages compared to BGRU-BGRU (given in Table 4.2). Similarly for Latin and Spanish, the results obtained from BGRU-BGRU are reported. From the outcome of these experiments, use of semantic vector proves to be more effective than the character level embedding. However, to capture the distributional properties of words efficiently, a huge corpus is needed which may not be available for low resourced languages. In that case, making use of syntactic embedding is a good alternative. Nonetheless, use of both types of embedding together improves the result.

4.3.2 Experimental Results for Another Five Languages

As mentioned earlier, five additional languages (Catalan, Dutch, Hungarian, Italian and Romanian) are considered to test the generalization ability of the method. The datasets for these languages are taken from the UD Treebanks[10] (Nivre et al., 2017). For each language, we merge the training and development data together and perform 4 fold cross validation on it to measure the average accuracy. The dataset statistics are shown in Table 4.8. For experimentation, we use the pre-trained semantic embeddings released by Bojanowski et al. (2017). Only BLSTM-BLSTM model is explored and it is compared with Lemming and Morfette. The hyper parameters are kept same as described previously except for the number of epochs needed for training across the languages. For Hungarian, around 80 epochs are needed to converge the training and for the remaining languages maximum 10 epochs are run. Table 4.9 shows the results. For all the languages, BLSTM-BLSTM outperforms Lemming and Morfette. The maximum improvement over the baselines we get is for Catalan (beats Lemming and Morfette by 8.15% and 8.49% respectively). Similar to the results in Table 4.2, restricting the output over applicable classes yields consistent performance improvement.

4.3.3 Overall Comparison of the Proposed Lemmatization Algorithms on Bengali Datasets

In this subsection, performance of the three lemmatizers proposed so far (BenLem, the trie-based lemmatizer and the deep neural lemmatizer) are compared. Firstly we run BenLem and the trie-based lemmatizer on the larger Bengali dataset (of size $20,257$ tokens) prepared to evaluate our deep neural lemmatization model. All the lexical resources (dictionary etc.) and parameters required to run them are kept same as previously used. Table 4.10 shows the accuracy obtained by the two methods. While experimenting with the trie-based lemmatizer, 3 best results obtained by using different variants of it are presented. They are - (*i*) variant 2.1 (using only POS). (*ii*) variant 2.2 (using POS and embedding). (*iii*) variant 3.1 (using POS and dictionary sense bag but not embedding). Note that BenLem and the trie-based lemmatizer cannot lemmatize proper nouns as these two methods are dependent on the POS tags given in the dictionary. In case a proper noun comes, the surface word form itself is considered as

[10] http://universaldependencies.org/

		With NNPs	Without NNPs
BenLem		69.13	69.38
Trie-based	var 2.1	72.84	73.37
lemmatizer	var 2.2	72.04	72.50
	var 3.1	72.85	73.38

TABLE 4.10: Accuracy (in %) of BenLem and trie-based lemmatizer on the larger Bengali dataset.

	With NNPs	Without NNPs
Deep neural lemmatizer	89.56	83.97

TABLE 4.11: Accuracy (in %) of the proposed deep neural lemmatizer on the smaller Bengali dataset.

the lemma of its own. Hence we provide both accuracies (without/with including the proper nouns for evaluation) produced by them on the dataset. Out of 20,257 tokens, there are 1,418 named entities.

Another set of results is presented in Table 4.11 showing the performance of the deep neural lemmatizer on the smaller sized BenLem dataset (6,314 tokens in total). On this dataset, BenLem and the trie-based lemmatizer were evaluated only on 3,342 tokens which are not named entities. As the proposed neural lemmatizer can handle all types of surface words, so we evaluate it on the full dataset. For the sake of comparison among all three methods, accuracy excluding the proper nouns is also reported. From Tables 4.10 and 4.11 it is evident that in presence of a moderate size good quality training data, supervised approaches are better than the unsupervised ones.

Comparative Analysis of BenLem, Trie-based lemmatizer and Deep Neural Lemmatizer with Examples: We further provide some example cases with analysis to show the superiority of the trie-based lemmatizer over BenLem and next, that of the neural lemmatizer over the previous two methods.

1. <Target word: 'গায়ে'/gaaye> — 'গায়ে'/gaaye is a surface word and its lemma is 'গা'/gaa. BenLem selects two root words 'গা'/gaa and 'গায়েন'/gaayena after the suffix checking step. From 'গায়ে'/gaaye, we get 'গা'/gaa with subtraction of the suffix 'য়ে'/ye and 'গায়েন'/gaayena with addition of the suffix 'ন'/na. Now, the string distance function used in BenLem returns the value 1.5 between 'গায়ে'/gaaye and 'গা'/gaa and 0.25 between 'গায়ে'/gaaye and 'গায়েন'/gaayena. Hence, always

‘গায়েন’/gaayena will be selected as the root by BenLem which is wrong. In the trie-based algorithm, ‘গায়েন’/gaayena will not be selected because between the *start* node and *end* node in the trie, ‘গা’/gaa is the only root word present. The same situation arises for the surface word-lemma pairs ‘মিলে-মিল, বিশ্বকে-বিশ্ব’ etc. If a surface word is a proper substring of a root which is not the lemma of the surface word, then in this case BenLem may end up to a wrong candidate. But the trie-based method does not suffer from the problem as it always gives priority to the roots having smaller length than the surface words.

2. <Target word: ‘বলেছেন’/balechhen> —‘বলা’/balaa is the lemma of this surface word. BenLem selects two roots ‘বলা’/balaa and ‘বলানো’/balaano after the suffix checking step. ‘বলেছেন’/balechhen - ‘েছেন’/echhen + ‘া’/aa = ‘বলা’/balaa and ‘বলেছেন’/balechhen - ‘েছেন’/echhen + ‘ানো’/aano = ‘বলানো’/balaano. Both the words ‘বলা’/balaa and ‘বলানো’/balaano have distance 3.75 from the surface word ‘বলেছেন’/balechhen>. Therefore, only the semantic checking step is left to determine the proper root. In the other hand, the trie-based algorithm will select ‘বলা’/balaa as from the *end* node, the traversal length required to reach the root is less than that for ‘বলানো’/balaano. Here, the root node finding strategy is more effective compared to the distance measure used in BenLem.

The proposed deep neural lemmatizer performs better than both BenLem and the trie-based lemmatizer in a number of cases. They are listed below.

1. The proper nouns, irregular word forms are not resolved by the rule-based/ unsupervised methods. Whereas, the supervised one can lemmatize the proper nouns as well as irregular word forms if the corresponding edit trees exist in the training set.

2. If the string distance/edge traversal distance between a root and its variant is very high (e.g. ‘মারা’/maraa-‘মেরেছেন’/merechhen, ‘দেওয়া’/deoyaa-‘দিল’/dila, ‘নেওয়া’/neoyaa-‘নিল’/nila etc.), then there is a chance of failure in BenLem/trie-based method as a wrong candidate having smaller distance with the surface word may be the winner. But, it is empirically found that the proposed BGRNN model successfully lemmatize those words by assigning highest probability to the appropriate edit tree.

4.4 Summary

This chapter presents a recurrent neural network model for context sensitive lemmatization. The proposed approach is supervised and needs lemma tagged continuous text for training. Instead of generating the lemma of an input surface word, the network classifies the appropriate transformation to induce the lemma. This makes it capable of handling the unseen word forms. Its another two major advantages are - (*i*) except the gold lemma, no additional morphological attribute such as POS is required to train the model and (*ii*) there is no need to define hand-crafted features. In this work, both character-level and distributional similarity-based word embeddings are exploited. We evaluate the model on 2 Indic (Bengali and Hindi) and 7 European languages. The results have been compared with two state-of-the-art baseline systems. It is found that the proposed lemmatizer outperforms the baselines on 8 out of 9 reference languages except Bengali for which the amount of training data is least.

Chapter 5

Morphological Tagging and Inflection Generation

5.1 Introduction

This chapter mainly deals with the task of morphological tagging to determine various attributes such as case, degree, gender, number etc. of an in-context word. Like other traditional natural language processing (NLP) applications e.g. language modelling and part of speech (POS) tagging, morphological tagging is seen as a sequence modelling problem and following the trend, several recurrent neural network (RNN) based methods have been developed for the task so far (Heigold et al., 2016, 2017; Cotterell and Heigold, 2017). These approaches have the following common model architecture. At first, character-level embeddings of words are generated using convolutional/recurrent models to make their syntactic representations. Then, for a sentence, its constituent word vectors are passed through another RNN to predict their morphological tags. Note that in the previous methods, the tagging problem is formulated as a single label classification task where all the morphological attributes of a particular word are jointly considered as a single tag. As an example, the tag 'Pos=ADJ|Case=Acc|Num=Sing' denotes accusative, singular adjective form. This way of representing every tag as the concatenation of universal key:value pairs suffers from a major drawback. Let 'Gen=Masc|Num=Sing', 'Gen=Fem|Num=Sing' and 'Gen=Masc|Num=Plur' be the only Gender;Number combinations present in the training set. As 'Gen=Fem|Num=Plur' tag is missing there, so

the system cannot predict it despite the occurrence of the individual attribute values 'Feminine' and 'Plural'. Thus, only the particular tags present in the training set gives the possible morphological analysis for any arbitrary word and hence the missing valid combinations of the component attributes are not addressed.

In this present work, the following changes are incorporated to the generic neural tagging model. We configure the tagging task as a multi-label classification task where each label corresponds to a universal morphological key and the distinct values of a key stands for the number of classes under the respective label. It is also hypothesized that, to predict the morphological features of a word, considering only its local context is sufficient. This leads us to build a convolutional neural network (CNN) based tagging model. The embedded word vectors of a sentence are given as input to a CNN layer having kernel size equal to the local context length which is a model parameter. Finally, outputs of the CNN are passed through softmax to classify the tags. Next, following Cotterell and Heigold (2017), we include the cross-lingual, character-level transfer learning strategies in our experiments. Finally, the distributional word vectors are used along with the character-level embeddings in the tagging model.

Our model has been evaluated on two different families of Indic languages which exhibit high morphological richness. In the first group, three languages are taken from the Indo-Aryan family. They are Hindi, Marathi and Sanskrit. The second group consists of two languages - Tamil and Telugu belonging to the Dravidian family. Besides having diverse morphology, all the other languages excluding Hindi are resource scarce. Furthermore, we have also considered two severely resource-scarce languages namely Coptic and Kurmaji for experimentation purpose.

5.2 Method

As discussed in section 5.1, a morphological tag t consists of universal key-value pairs i.e. $t = [k_1 = v_1 | k_2 = v_2 | \ldots]$. Given an input sentence w_1^N having tags t_1^N, the conditional distribution of tags given the input sentence is formulated as

$$p(t_1^N | w_1^N) = \prod_{i=1}^{N} p(t_i | w_{i-C}^{i+C}) \tag{5.1}$$

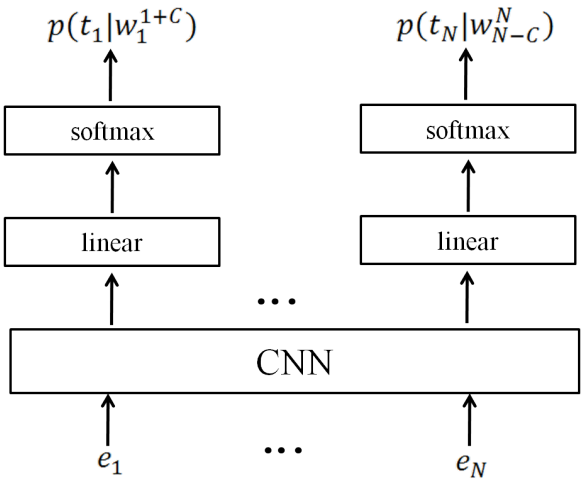


FIGURE 5.1: BLSTM-CNN tagging model as multitask learning.

where the tag t_i depends on the words present in the local context of word w_i (denoted by w_{i-C}^{i+C}) instead of the entire sentence. This is in contrast to the previous works of Heigold et al. (2016); Cotterell and Heigold (2017) where the tag t_i is dependent on the entire sentence. In this paper, we refer to the architecture proposed by Cotterell and Heigold (2017) as BLSTM-BLSTM model since BLSTM is used to generate word embedding as well as positional embedding. In our proposed model, words are embedded using final states of a character-level BLSTM network i.e.

$$e_i = BLSTM(c_1^{l_i}) \tag{5.2}$$

where e_i is the word embedding, l_i denotes the number of characters present in the word w_i and $c_1, \ldots, c_{l_i}$ are the character embeddings obtained from a look-up table.

The word embeddings obtained from BLSTM are arranged in a matrix M of shape $N \times D$ where N is the number of words in the sentence and D is the word embedding size. We use CNN to generate the positional embeddings q_i. Formally,

$$q_i = CNN(M) \tag{5.3}$$

The number of filters and the size of each filter in CNN are F and $(2C + 1) \times D$ respectively where C denotes the left and right context length of the focus word. The filter stride length is set to D for considering the next word. Hence, CNN ensures that

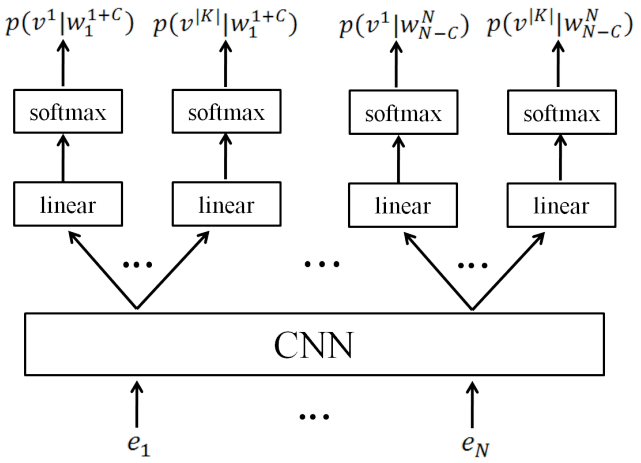


FIGURE 5.2: BLSTM-CNN tagging model as multitask learning across Keys. $p(v^k|w_{i-C}^{i+C})$ denotes the probability distribution of values for key $k \in K$ and word w_i.

the positional embeddings q_i is dependent only on the local context of the word w_i. The probability distribution over tag t_i is given by

$$p(t_i) = softmax(Wq_i + b) \tag{5.4}$$

Here W is the weight matrix of the time-distributed dense layer and b is the bias. We refer to our proposed network as BLSTM-CNN model. Figure 5.1 depicts the architecture of it.

5.2.1 Multitask Learning across Languages

Morphological tagging across languages can be posed as multitask learning problem as described in (Cotterell and Heigold, 2017). We trained our BLSTM-CNN model on two different frameworks namely, language-universal softmax and language-specific softmax.

- **Language-Universal Softmax:** In this framework, the number of possible tags is the union of tags present in the languages in consideration. Language universal softmax signifies that only one softmax layer is used to generate probability distribution over tags. The character level embeddings are changed by adding a special character c_{lan-id} to the start and end position of each word. c_{lan-id} acts as the

language identifier. Thus,

$$e_i^{lan} = BLSTM(c_{lan-id}, c_1^{l_i}, c_{lan-id}) \tag{5.5}$$

- **Language-Specific Softmax:** In this framework, given the positional embedding, a separate softmax layer is used for every language to obtain the probability distribution over tags present in the language.

$$p(t_i|lan) = softmax(W_{lan}q_i + b_{lan}) \tag{5.6}$$

W_{lan} and b_{lan} are the language-specific time-distributed weight matrix and bias respectively.

Let $\mathcal{T}$ denotes the set of morphological tags present in the training set. Given an input sentence w_1^N with tags t_1^N, the loss function for both the frameworks is given as

$$L = \frac{1}{|\mathcal{T}|}\sum_{i=1}^{N}[(log\,(1 + p(t_i|w_{i-C}^{i+C})) - log\,2)^2 + \sum_{\substack{t\in\mathcal{T}\\ t\neq t_i}} log^2\,(1 + p(t|w_{i-C}^{i+C}))] \tag{5.7}$$

5.2.2 Multitask Learning across Keys

Task Definition: Instead of treating morphological tagging as a classification problem at tag level, we form it as classification at key level. This allows the system to generalize the unseen tags during testing. Accordingly in this framework, the problem of morphological tagging is considered as a multitask learning problem across universal keys (gender, part-of-speech, case etc.). Let K denotes the universal morphological key set and for a particular morphological key $k \in K$, let V^k denote the set of possible values. Given an input sentence w_1^N with tags t_1^N, the loss function for this framework is given as

$$\begin{aligned} L = \sum_{i=1}^{N}\sum_{k\in K}\frac{1}{|V^k|}\sum_{v\in V^k}([(k,v) \notin t_i] log^2\,(1 + p(v|w_{i-C}^{i+C})) + \\ [(k,v) \in t_i](log\,(1 + p(v|w_{i-C}^{i+C})) - log\,2)^2) \end{aligned} \tag{5.8}$$

	train	dev	test
Hindi	13,304/281,057	1,659/35,217	1684/35,430
Marathi	373/2,997	46/440	47/412
Sanskrit	100/695	50/275	40/236
Tamil	400/6,328	80/1,263	120/1,983
Telugu	1,051/5,082	131/662	146/721

TABLE 5.1: Dataset statistics (# sentences/# words) of the 5 languages.

Hindi	Marathi	Sanskrit	Tamil	Telugu
939	260	202	200	14

TABLE 5.2: # of unique tags in the training sets of the 5 languages.

where [.] denotes Iverson bracket. For a particular morphological key $k \in K$, the probability distribution over V^k is given by

$$p(v|w_{i-C}^{i+C}) = softmax(W_k q_i + b_k) \tag{5.9}$$

Hence, we design a separate softmax layer for each $k \in K$. W_k and b_k denotes key-specific weight matrix and bias respectively. The model is shown in Figure 5.2.

5.2.3 Using Distributional Word Vectors

So far we have discussed different variations of the character-level neural tagging model where only the character streams in words are embedded to predict the tags. Additionally, the impact of distributional word vectors is explored along with the character level embedding on morphological tagging as they are empirically found to be efficient to capture the linguistic regularities in languages (Mikolov et al., 2013b). So the character-level embedding e_i is concatenated with the distributional word vector of the word w_i obtained from a large corpus to form the composite embedding that captures syntactic as well as semantic information. In our experiments, we use the pre-trained word vectors provided by Bojanowski et al. (2017). Using this composite word representation, all the variant models i.e. BLSTM-BLSTM and BLSTM-CNN with language-universal, language-specific and key-specific softmax layers are explored to make an exhaustive comparison between character-level and composite tagging models.

	BLSTM-BLSTM	BLSTM-CNN
Hindi	**73.36/87.18**	72.62/86.10
Marathi	31.31/48.37	**52.43/67.75**
Sanskrit	12.29/12.29	**15.67/42.24**
Tamil	29.76/50.11	**49.37/67.87**
Telugu	71.15/71.15	**82.11/82.11**

TABLE 5.3: Accuracy/F1 score of two models (in %) for single language setting.

5.3 Experimentation

Datasets: For experimentation on the 5 reference Indic languages - Hindi, Marathi, Sanskrit, Tamil and Telugu, the Universal Dependencies datasets (Nivre et al., 2017) are used. In the annotated files, 4^{th} and 6^{th} columns are joined to make the tags for the surface words. The dataset statistics and the number of unique morphological tags in the training set of each language are given in Tables 5.1 and 5.2 respectively. Though Hindi has a large amount of data, the other languages are truly resource-scarce as evident by the statistics. Note that for cross-lingual character-level morphological transfer, the languages grouped together should bear the same script. For example, in the work of Cotterell and Heigold (2017), the language families selected (Romance, Northern Germanic, Slavic and Uralic) share common alphabets. Regarding the languages considered in the present work, Hindi, Marathi and Sanskrit have common Devanagari script but for Tamil and Telugu, there are disjoint character alphabets (Tamil has unicode values from 0B80 to 0BFF and Telugu ranges from 0C00 to 0C7F). We overcome this problem by using the transliterated versions of the Tamil and Telugu datasets into Roman, which makes it possible to share character-level similarities across the two languages.

	BLSTM-BLSTM	BLSTM-CNN
Hindi	**79.35/92.06**	72.52/88.32
Marathi	31.07/50.03	**45.87/67.76**
Sanskrit	7.20/27.47	**15.68/45.55**
Tamil	25.44/48.28	**49.69/67.65**
Telugu	66.43/66.57	**83.49/83.63**

TABLE 5.4: Accuracy/F1 score of two models (in %) for multitask learning across keys.

Hyper Parameters: Our models have several hyper parameters such as the number of neurons at the output of the character-level BLSTM, the hidden layer size of the second

level BLSTM, local context length of the CNN layer, optimizer, dropout value etc. The character-embedded vector size is set to 128 and for the second level BLSTM, the number of hidden layer nodes are 256. In order to choose the optimum local context length of the CNN, we run our experiments with the context length from 1 to 7. It has been observed that with the increase of the context length, tagging accuracy decreases monotonically. Hence for all of the experiments with CNN model, the local context length is set to 1. We also found that compared to the cross-entropy loss, the mean squared logarithmic error function (Chollet et al., 2015) given in equations 5.7 and 5.8 produces higher accuracy. The values of batch size and dropout rate are kept 16 and 0.2 respectively. Out of the different optimizers such as AdaDelta (Zeiler, 2012), Adam (Kingma and Ba, 2014), Nadam (Dozat, 2016) and RMSProp (Dauphin et al., 2015), Nadam yields the highest accuracy. All the experiments are done using Keras (Chollet et al., 2015).

Evaluation Metrics: Two different measures are taken for evaluating the tagging performance. The first one is average per word accuracy i.e. the ratio of the number of correctly tagged words to the total number of words in the test data. The models of multitask learning across keys are evaluated as follows. For a test word w, assessment is done only on those particular keys that exist in the gold tag of w. w is said to be correctly tagged if for all the keys in its gold tag, the system predicts the correct values.

The second measure is per feature F1 score[1] as used by Buys and Botha (2016); Cotterell and Heigold (2017). It calculates the tagging accuracy at key level and gives partial credit when a subset of keys is properly classified. For each distinct universal key, its F1 score is defined as the ratio of the number of words for which the key is correctly predicted to the number of words for which the key is present in their respective gold tags. Finally, the key-specific F1 scores are averaged over all keys.

5.3.1 Results

The experimental results for single language setting (i.e. training and testing are done on the same language) are provided in Table 5.3 for the 5 reference Indic languages. Table 5.4 shows the results for the experiments with multitask learning across keys.

[1]The conventional definition of F1 score is different as the harmonic mean of precision and recall. We use the term here following the previous works of *Buys and Botha (2016); Cotterell and Heigold (2017)*

	BLSTM-BLSTM		BLSTM-CNN	
	Univ.	Spec.	Univ.	Spec.
Hindi	74.56/88.34	**75.42/89.04**	72.21/85.53	72.51/86.39
Marathi	45.14/62.42	**48.79/66.85**	48.05/65.27	48.54/62.82
Sanskrit	10.17/38.38	15.68/45.30	10.17/35.28	**16.52/47.10**
Tamil	30.77/51.68	33.18/54.10	**49.72/67.32**	48.11/65.14
Telugu	70.60/71.01	75.45/75.45	79.75/79.75	**81.14/81.14**

TABLE 5.5: Accuracy/F1 score (in %) for multitask learning across languages with language-universal and language-specific softmax.

We compare the performance of our proposed BLSTM-CNN model with the state-of-the-art BLSTM-BLSTM model of Cotterell and Heigold (2017). The results in Tables 5.3 and 5.4 exhibit that under each experimental setting, BLSTM-CNN outperforms BLSTM-BLSTM for all the languages except Hindi. Moreover, there is a considerable performance gain for the 3 languages (Marathi, Tamil and Telugu). Comparison between Tables 5.3 and 5.4 also shows that the approach of considering the tagging task as multitask learning across keys proves to be beneficial than learning over the tags. Except Marathi, for all the other languages, accuracies of the best models (shown in bold) increase. For Hindi, Sanskrit, Tamil and Telugu, increase in accuracies are 5.99%, 0.01%, 0.32% and 1.38% respectively. If assessment is done based on F1 score, then also the learning across keys approach establishes its superiority. Out of the best models of 5 languages, F1 reduces (by 0.22%) in Tamil only.

In Table 5.5, the results for multitask learning across languages are provided. The languages belonging to the same family are trained together. The terms 'Univ' and 'Spec' stand for the language-universal and the language-specific softmax architectures respectively. Between these two different architectures of joint training, language-specific softmax achieves better results. Out of the 10 comparisons in Table 5.5, except for BLSTM-CNN in Tamil, it overperforms the language-universal architecture for all other cases. Even for Tamil, the performance drop is marginal (1.61% and 2.18% reduction in accuracy and F1). In comparison between Tables 5.3 and 5.5, it is found that joint training of related languages is mostly useful than the single language setting. While using BLSTM-BLSTM, the cross-lingual learning strategy produces better results for all the 5 languages. Whereas, in case of using BLSTM-CNN, only Sanskrit and Tamil get benefited by the joint learning scheme. Overall, multitask learning across languages gives the highest accuracy for Hindi, Sanskrit and Tamil.

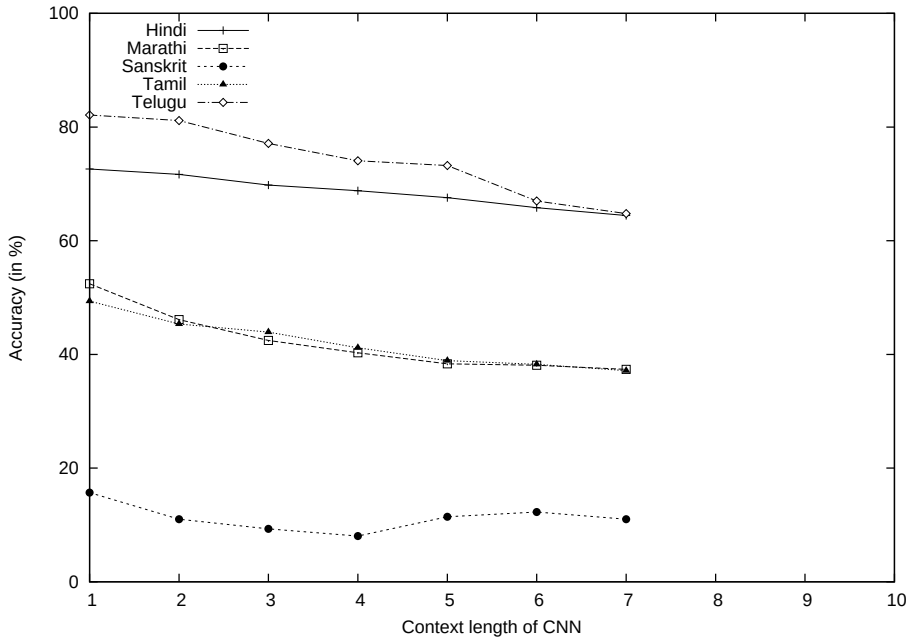


FIGURE 5.3: CNN context length vs. accuracy plot.

	BLSTM-BLSTM	BLSTM-CNN
Hindi	80.43/89.34	**81.63/93.39**
Marathi	**64.32/78.24**	58.25/72.87
Sanskrit	**22.03/54.62**	20.34/52
Tamil	**76.47/86.05**	70.49/82.07
Telugu	**89.59/89.59**	89.45/89.45

TABLE 5.6: Accuracy/F1 score of two models (in %) for single language setting using distributional word vectors.

Effect of CNN Context Length on Accuracy: We further study the effect of CNN context length on the tagging accuracy for the 5 reference Indic languages. Following the experiments of Yu et al. (2017), the accuracy on the test sets are measured by varying the context length from 1 to 7. Figure 5.3 shows the plots. For each of the 5 languages, best accuracy is obtained at length 1 i.e. when only the left and right adjacent words of the focus word are considered as the context. With the increase of the context length, the graphs are mostly decreasing (except Sanskrit) which indicates that considering a smaller context in CNN is advantageous for the tagging task.

Results using Distributional Word Vectors: Next we analyze the models that use distributional word vectors along with character-level embeddings. Tables 5.6, 5.7 and 5.8 show the results under single language setting, multitask learning across keys

	BLSTM-BLSTM	BLSTM-CNN
Hindi	**87.44/95.69**	81.96/93.54
Marathi	**62.62/80.20**	55.58/76.01
Sanskrit	**27.97/60.38**	23.73/56.41
Tamil	**72.55/85.74**	66.57/81.88
Telugu	**91.12/91.26**	88.07/88.21

TABLE 5.7: Accuracy/F1 score of two models (in %) for multitask learning across keys using distributional word vectors.

	BLSTM-BLSTM		BLSTM-CNN	
	Univ.	Spec.	Univ.	Spec.
Hindi	86.36/95.50	**86.40/95.53**	80.77/92.54	81.02/92.67
Marathi	58.98/75.66	**60.68/76.44**	46.12/60.57	51.21/66.06
Sanskrit	16.10/46.42	**20.76/52.55**	3.39/23.02	10.59/36.84
Tamil	**76.22/86.07**	75.61/85.22	67.57/79.94	69.08/80.66
Telugu	**90.98/90.98**	90.15/90.15	87.79/87.79	87.65/87.65

TABLE 5.8: Accuracy/F1 score (in %) for multitask learning across languages with language-universal and language-specific softmax using distributional word vectors.

and multitask learning across languages respectively. It is clearly evident from the results that combining the character-level embeddings with distributional word vectors significantly improves the performance under all the settings. In contrast to the results obtained using only character-level embeddings, in this case, BLSTM-BLSTM substantially outperforms BLSTM-CNN for all the 5 languages. In Table 5.6, for Hindi only, BLSTM-CNN (Acc. 81.63%) beats BLSTM-BLSTM (Acc. 80.43%). In Tables 5.7 and 5.8, throughout BLSTM-BLSTM performs better. Comparison between Tables 5.6 and 5.7 shows that learning across keys improves the best model accuracy (shown in bold) for Hindi (Acc. from 80.43% to 87.44%), Sanskrit (Acc. from 22.03% to 27.97%) and Telugu (Acc. from 89.59% to 91.12%). Though accuracy decreases in Marathi (from 64.32% to 62.62%) but there F1 increases (from 78.24% to 80.20%). To analyze the performances of learning over single language and across-languages strategies reported in Tables 5.6 and 5.8 respectively, it is found that their results are comparable. For Hindi and Telugu, cross-lingual learning achieves better results (Acc. 86.40% and 90.98%) whereas for Marathi and Sanskrit, the single language learning is preferable (Acc. 64.32% and 22.03%) over across-languages.

Experiments on Severely Resource Scarce Languages: Additionally we choose two severely resource scarce languages - Coptic and Kurmanji, for experimentation.

	train	dev	test
Coptic	364/9,273	41/910	39/1,046
Kurmanji	591/8,058	71/960	72/1,000

TABLE 5.9: Dataset statistics (# sentences/# words) of Coptic and Kurmanji.

	BLSTM-BLSTM	BLSTM-CNN
Coptic	56.59/79.13	**62.72/86.39**
Kurmanji	43/58.03	**66.50/78.07**

TABLE 5.10: Accuracy/F1 score of two models (in %) for single language setting.

	BLSTM-BLSTM	BLSTM-CNN
Coptic	75.62/76.49	**84.13/84.49**
Kurmanji	34.80/56.71	**54.40/75.31**

TABLE 5.11: Accuracy/F1 score of two models (in %) for multitask learning across keys.

Coptic is a northern Afro-Asiatic language mostly spoken in Egypt. It is agglutinative in nature and consists of many inflectional and derivational variants of root words. Different morphological properties such as gender, mood, number, tense etc. are featured by the common prefixes. Each noun form in the language carries gender information and the pronouns and the verbs also have gradations which represent diverse morphological properties. Whereas, Kurmanji is a Kurdish language spoken in Turkey, Iran, Iraq and Syria. It is the most prevalent language among the Kurdish languages. There are several dialectal varieties of it e.g. Northwestern, Southwestern, Northern, Southern, Southeastern and Anatolian. By the phrase 'severely resource scarce' we mean that the distributional word vectors are unavailable for these two languages and hence, character level embedding is the only choice. For both the languages, experiments are done on the Universal Dependencies datasets. In Kurmanji, only test set was made available by the developers. Therefore, this data is split into 8:1:1 ratio for training, development and testing. Table 5.9 shows the dataset statistics. The number of unique tags present in the training sets of Coptic and Kurmanji are 54 and 196 respectively. We run BLSTM-BLSTM and BLSTM-CNN for single language setting and for multitask learning across keys keeping the same set of hyper parameters as used for the Indic languages. The results are presented in Tables 5.10 and 5.11. Throughout the experiments, BLSTM-CNN significantly outperforms BLSTM-BLSTM. Use of multitask learning across keys improves the accuracy for Coptic (21.41% improvement with BLSTM-CNN) but for

Kurmanji, single language setting produces the best result.

Inferences: We draw the following inferences from the experimental results reported above.

1. BLSTM-BLSTM performs better than BLSTM-CNN when we combine character-level embeddings with distributional word vectors. However, large corpora is needed to generate high quality word vectors, which might not be feasible for many severely resource-scarce languages. For such languages, since only character-level embeddings are an option, BLSTM-CNN is a better approach.

2. When using only character-level embeddings, multitask learning across languages gives significant advantage over single language setting to BLSTM-BLSTM. However, with addition of distributional vectors, the cross-lingual learning does not prove to be effective.

3. Multitask learning across keys enables the system to generalize over unseen tags. However, from the results we find that it performs comparably with the single language setting, which demands experimentation on more languages.

4. Throughout the experiments, language-specific softmax yields better performance to language-universal softmax.

5.4 Morphological Inflection Generation

Our final work in the thesis deals with the task of morphological inflection generation. The goal in this task is to produce the variant of a source word, given the morpho-syntactic descriptions of the target word. We participated in the SIGMORPHON 2017 shared task of inflection generation (Cotterell et al., 2017) which is divided into 2 sub-tasks. Task 1 demands to inflect the isolated word forms based on labelled training data. For example, given the source form 'communicate' and the features 'V;3;SG;PRS' (third person, singular number, present verb form), one has to predict the target form 'communicates'. Whereas, in task 2, partially filled incomplete paradigms are to be completed using a restricted number of full paradigms. For each of the tasks, 3 separate training files are given per language, which differ in size (low/medium/high), in order to

analyze systems' generalization ability in low, medium and high resource situations. The competition considers 52 languages. For each language, a finite set of morphological tags are provided, from which the target inflections are taken. Evaluation is done separately under each of the three different training sets. To make the shared task competition fair, use of external resources are forbidden for the main competition track. However, for those systems which make use of external monolingual corpora, a list of approved external corpora selected from the Wikipedia text dumps are provided.

Currently we target the task 1 introducing a LSTM network to handle the inflection generation problem. Our model is related to the encoder-decoder based approaches such as (Aharoni et al., 2016; Faruqui et al., 2016; Kann and Schütze, 2016), but the main difference is that the proposed network is not designed to generate sequence of characters as output. Rather, the problem is composed as to classify the transformation pattern required to convert a source form to its target form. Our goal is to model such a system which receives an input word and the morphological tags and returns the proper transformation that induces the target word. The source-target transformation is accomplished using edit tree (Chrupala et al., 2008; Müller et al., 2015). Initially all edit trees are extracted from the labelled pairs in the training data and then the distinct candidates from them are marked as the class labels. The character sequence of the input word is passed through the LSTM network for encoding the syntactic properties and finally, the encoded representation is jointly trained with the input tags to classify the correct edit tree. Primarily the SIGMORPHON competition demanded systems that exploit only the training data provided and hence, our system does not use external mono-lingual corpora such as the Wikipedia dumps. From the results obtained from the test datasets, it shows that the proposed method is resource intensive. When the training size is high, it achieves over the baseline system on 15 out of the 52 languages. But on medium and low amount of training data, the performance is poor beating the baseline on 5 and 4 languages respectively. Next, we describe our system.

5.4.1 The System Description

The architecture of our system is presented in Figure 5.4. It is similar to the deep neural lemmatizer described in chapter 4. At first, the LSTM network is utilized to make a syntactic embedding of the source word, that captures the morphological regularities.

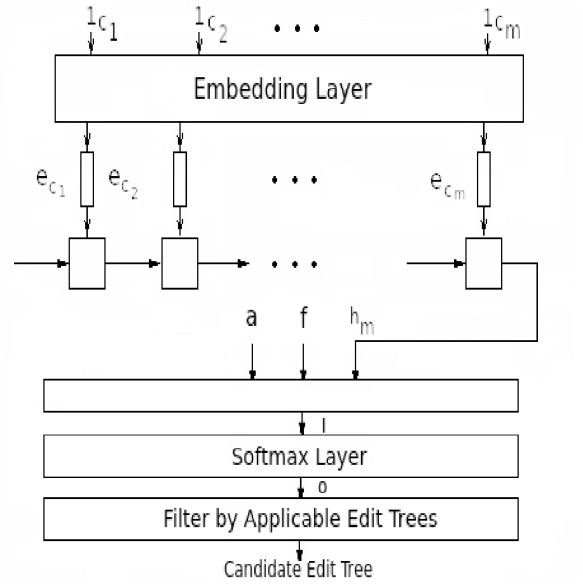


FIGURE 5.4: The model architecture.

A character alphabet of the concerned language is defined as C. Let the input word w consists of the character sequence $c_1, \ldots, c_m$ where the word length is m and each character c_i is represented as a one hot encoded vector $\mathbf{1}_{c_i}$. The particular dimension of $\mathbf{1}_{c_i}$ referring to the index of c_i in the alphabet C, is set to one and the other dimensions are made zero. $\mathbf{1}_{c_1}, \ldots, \mathbf{1}_{c_m}$ are passed to an embedding layer $\mathbf{E}_c \in \mathbb{R}^{d_c \times |C|}$, which projects them to d_c dimensional vectors $\mathbf{e}_{c_1}, \ldots, \mathbf{e}_{c_m}$, by doing the operation $\mathbf{e}_{c_i} = \mathbf{E}_c \cdot \mathbf{1}_{c_i}$ where '$\cdot$' denotes matrix multiplication. When the sequence of vectors $\mathbf{e}_{c_1}, \ldots, \mathbf{e}_{c_m}$ is given to the LSTM network, it computes the state sequence $\mathbf{h}_1, \ldots, \mathbf{h}_m$. Eventually, the final state $\mathbf{h}_m$ is considered as the encoded representation of w.

In addition to the source word, we have morpho-syntactic features in hand to predict the target form. From the training data, all distinct features are sorted out to make a feature dictionary F. For a training sample, the given features are mapped to $|F|$ dimensional feature vector $\mathbf{f} = (f_1, \ldots, f_{|F|})$ where $f_i = 1$ if the i^{th} feature in the dictionary is present in the input features, otherwise f_i is set to 0. Thus, $\mathbf{f}$ becomes a numeric representation of the input features for the present training sample.

Another important point is that, for any arbitrary input word, all unique edit trees in the training data are not applicable due to incompatible substitutions. In spite of all unique edit trees are set as the class labels, few of them are applicable for an input word to the model. To sort out this issue, we take the same strategy of constructing

the vector of applicable edit trees as described in chapter 4 to put the information over which classes the model should distribute the output probability mass while training.

Let $T = \{t_1, \ldots, t_k\}$ be the distinct edit trees set extracted from the training data. For the input word w, its applicable edit trees vector is defined as $\mathbf{a} = (a_1, \ldots, a_k)$ where $\forall j \in \{1, \ldots, k\}$, $a_j = 1$ if t_j is applicable for w, otherwise 0. Hence, $\mathbf{a}$ holds the applicable edit tree information for w. Finally, we combine the LSTM output $\mathbf{h}_m$, feature vector $\mathbf{f}$ and applicable tree vector $\mathbf{a}$ together for the edit tree classification task as following,

$$\mathbf{l} = softplus(\mathbf{L}^h\mathbf{h}_m + \mathbf{L}^f\mathbf{f} + \mathbf{L}^a\mathbf{a} + \mathbf{b}),$$

where 'softplus' is the activation function $f(x) = ln(1 + e^x)$ and $\mathbf{L}^h, \mathbf{L}^f, \mathbf{L}^a$ and $\mathbf{b}$ are the network parameters. Next, $\mathbf{l}$ is passed through the softmax layer to get the output labels for w.

To pick the maximum probable edit tree for an input word, the prior information about applicable classes is exploited. Let $\mathbf{o} = (o_1, \ldots, o_k)$ be the output of the softmax layer. The particular edit tree $t_j \in T$ is considered as the right candidate, where

$$j = \text{argmax}_{j' \in \{1,\ldots,k\} \wedge a_{j'}=1} o_{j'}$$

In this way, we choose the maximum probable class over the applicable classes only.

5.4.2 Experimentation

Datasets: We use the SIGMORPHON-2017 datasets (Cotterell et al., 2017) for experimentation. For each of the 52 languages, 3 different training sets (high, medium and low) are provided varying on the number of training samples. Each training sample comprises a pair of word and its morphosyntatic features tagged with the respective target form. In the high, medium and low training sets, there are 10,000, 1,000 and 100 samples respectively. Both the development and test sets contain 1,000 word-feature samples for which the target forms are to be found. Evaluation is done by measuring

Resource Condition	Language-Training Set Size	Our Model's Acc. (%)	Baseline Acc. (%)
High	albanian-high	79.1	78.9
	arabic-high	60.2	50.7
	armenian-high	89.5	87.2
	dutch-high	90.2	87.0
	georgian-high	94.4	93.8
	hebrew-high	71.1	54.0
	hindi-high	99.1	93.5
	hungarian-high	77.5	68.5
	icelandic-high	78.2	76.3
	irish-high	60.6	53.0
	italian-high	91.1	76.9
	khaling-high	56.0	53.7
	russian-high	86.1	85.7
	turkish-high	73.5	72.6
	urdu-high	98.2	96.5
Medium	english-medium	91.1	90.9
	french-medium	73.9	72.5
	hebrew-medium	46.6	37.5
	italian-medium	75.7	71.6
	scottish-gaelic-medium	62.0	48.0
Low	albanian-low	21.6	21.1
	danish-low	61.9	58.4
	khaling-low	6.1	3.1
	serbo-croatian-low	24.2	18.4

TABLE 5.12: Our model's performance on the test datasets for those languages where it beats the baseline system.

the direct accuracy i.e. the fraction of the total number of samples present in the test set, for which the target forms are accurately predicted.

Baseline: The shared task provides a baseline system which is the implementation of Liu and Mao (2016). The system initially makes an alignment between the source and the target forms by using the minimum-cost edit path. After that the every aligned pair is broken into prefix, stem and suffix parts and string substitution rules are designed using the segmented parts. Finally the most frequent rules under prefix and suffix changing categories associated with a particular morpho-syntactic feature combination is applied on the test word.

Model Parameters: For all the 52 languages, we limit each word length to maximum 25 characters. Null characters are padded to the smaller words at the end and for words having more than 25 characters, extra characters are omitted. We represent each

Resource Condition	Language-Training Set Size	Our Model's Acc. (%)	Baseline Acc. (%)
High	albanian-high	79.4	78.1
	arabic-high	60	47.7
	armenian-high	89.2	89.1
	dutch-high	88	86.8
	georgian-high	94.6	94
	hebrew-high	72.3	55.8
	hindi-high	99.1	94
	hungarian-high	75.8	71.1
	icelandic-high	80.6	76.1
	irish-high	62.4	54.3
	italian-high	91.6	79.9
	khaling-high	56.2	53.8
	russian-high	85.6	82
	turkish-high	74.4	72.9
	urdu-high	97.4	95.8
Medium	georgian-medium	90.4	90
	hebrew-medium	47.3	40
	italian-medium	78.2	73.8
	scottish-gaelic-medium	68	52
Low	danish-low	60.2	59.8
	khaling-low	5	3.9
	serbo-croatian-low	26.7	21.3
	welsh-low	17	15

TABLE 5.13: Our model's performance on the development datasets for those languages where it beats the baseline system.

character as 25 length sequence of one hot encoded character vectors that are passed to the embedding layer. The output dimension of the embedding layer is set as the length of the one hot encoded character vectors i.e. $|C|$, size of the character alphabet of the concerned language.

Hyper parameters of the model are given as follows. The number of neurons in the hidden layer of LSTM is set to 64 for all languages. We apply online learning in our model. Number of epochs and the dropout rate are set to 150 and 0.2 respectively. We use 'Adagrad' (Duchi et al., 2011) optimization algorithm for training. Categorical cross-entropy function is used to measure the loss in our model.

5.4.3 Results

As published by the SIGMORPHON 2017 shared task competition organizers, our method overperforms the baseline system on 15 out of the 52 languages in high resource configuration for the test sets. Whereas, in medium and low resource situations separately, it beats the baseline on 5 and 4 languages respectively. The results are provided in Table 5.12, which shows that the proposed method is resource intensive.

We also provide our model's performance on the development datasets in Table 5.13. The results are quite similar to the results given in Table 5.12. When the training size is high, the proposed model beats the baseline on 15 languages. For medium and low resource scenario, it achieves over the baseline on 4 languages only. Furthermore, the results on the test datasets where our model cannot beat the baseline system are provided in Tables 5.14 and 5.15.

5.5 Summary

In this chapter, we mainly target the task of morphological tagging for five Indic and two severely resource scarce languages. Different neural network models such as BLSTM-BLSTM and BLSTM-CNN are tried to handle the problem. A number of experiments are carried out under three different learning setup namely single language setting, multitask learning across keys and across languages. Initially all the experiments are done using only character-level embeddings. Then distributional word vectors are combined with it to exploit semantic information. Experimental results reveal that when only character-level embeddings are used, BLSTM-CNN model with a smaller context performs better than BLSTM-BLSTM. But after addition of distributional word vectors, BLSTM-BLSTM outperforms BLSTM-CNN. Though the use of word vectors considerably improves the accuracy all through, still generating word vectors of high quality remains a challenge for severely resource-scarce languages.

Further, the problem of morphological inflection generation is attempted, which demands to produce the proper inflected form of a word given its target morpho-syntactic features. A LSTM-based neural model is proposed that frames the problem as edit tree classification task. The developed system takes part in the SIGMORPHON 2017

shared task competition. From the results it is found that when the training set size is medium or low, the model's performance is poor. But under high resource scenario, it outperforms the baseline on fifteen languages out of which two are Indic (Hindi and Urdu).

Resource Condition	Language-Training Set Size	Our Model's Acc. (%)	Baseline Acc. (%)
High	basque-high	0.0	5.0
	bengali-high	76.0	81.0
	bulgarian-high	80.9	88.8
	catalan-high	92.0	95.5
	czech-high	83.0	89.6
	danish-high	82.3	87.8
	english-high	93.3	94.7
	estonian-high	75.5	78.0
	faroese-high	64.3	74.1
	finnish-high	52.4	78.2
	french-high	80.8	81.5
	german-high	79.4	82.4
	haida-high	57.0	67.0
	kurmanji-high	93.0	93.0
	latin-high	21.1	47.6
	latvian-high	77.9	92.1
	lithuanian-high	43.5	64.2
	lower-sorbian-high	85.0	86.4
	macedonian-high	90.8	92.1
	navajo-high	23.3	37.8
	northern-sami-high	28.5	64.0
	norwegian-bokmal-high	83.6	91.0
	norwegian-nynorsk-high	66.4	76.9
	persian-high	71.3	79.0
	polish-high	83.4	88.0
	portuguese-high	96.1	98.1
	quechua-high	93.1	95.4
	romanian-high	77.3	79.8
	serbo-croatian-high	77.4	84.6
	slovak-high	82.4	83.3
	slovene-high	85.7	88.9
	sorani-high	54.9	63.6
	spanish-high	88.4	90.7
	swedish-high	82.7	85.4
	ukrainian-high	84.9	85.4
	welsh-high	62.0	69.0
Medium	albanian-medium	53.9	66.3
	arabic-medium	34.6	42.1
	armenian-medium	58.2	72.7
	basque-medium	1.0	2.0
	bengali-medium	53.0	76.0
	bulgarian-medium	62.3	72.8
	catalan-medium	78.1	84.3
	czech-medium	67.6	81.5
	danish-medium	76.8	78.1
	dutch-medium	60.5	73.2
	estonian-medium	34.8	62.9
	faroese-medium	52.0	60.6
	finnish-medium	20.5	43.7
	georgian-medium	91.1	92.0
	german-medium	62.5	72.1
	haida-medium	28.0	56.0
	hindi-medium	80.6	85.9
	hungarian-medium	40.6	42.3
	icelandic-medium	53.9	60.4
	irish-medium	39.5	44.0
	khaling-medium	16.4	17.9
	kurmanji-medium	88.0	89.1
	latin-medium	14.5	37.6
	latvian-medium	62.7	85.7
	lithuanian-medium	20.7	52.2
	lower-sorbian-medium	69.9	70.8
	macedonian-medium	76.0	83.6
	navajo-medium	14.4	33.5
	northern-sami-medium	11.8	37.0
	norwegian-bokmal-medium	78.0	80.7
	norwegian-nynorsk-medium	59.6	61.1
	persian-medium	41.2	62.3
	polish-medium	59.7	74.0
	portuguese-medium	87.7	93.4
	quechua-medium	36.2	70.3
	romanian-medium	60.9	69.4
	russian-medium	69.4	75.9
	serbo-croatian-medium	55.5	64.5
	slovak-medium	69.9	72.3
	slovene-medium	69.2	82.2
	sorani-medium	23.0	51.7
	spanish-medium	71.4	84.7
	swedish-medium	73.0	75.7
	turkish-medium	27.1	32.9
	ukrainian-medium	65.2	72.8
	urdu-medium	80.2	87.5
	welsh-medium	32.0	56.0

TABLE 5.14: Our model's performance on the test datasets for those languages where it cannot beat the baseline system.

Resource Condition	Language-Training Set Size	Our Model's Acc. (%)	Baseline Acc. (%)
Low	arabic-low	12.4	21.8
	armenian-low	16.4	35.8
	basque-low	0.0	2.0
	bengali-low	23.0	50.0
	bulgarian-low	24.8	30.2
	catalan-low	49.0	55.9
	czech-low	19.9	39.3
	dutch-low	35.9	53.6
	english-low	73.5	80.6
	estonian-low	8.8	21.5
	faroese-low	25.7	30.0
	finnish-low	6.5	15.4
	french-low	39.1	61.8
	georgian-low	33.9	70.5
	german-low	37.3	54.3
	haida-low	19.0	32.0
	hebrew-low	15.3	24.7
	hindi-low	16.5	29.1
	hungarian-low	14.9	21.0
	icelandic-low	21.9	30.3
	irish-low	20.5	30.3
	italian-low	27.2	41.1
	kurmanji-low	69.4	82.8
	latin-low	8.9	16.0
	latvian-low	30.8	64.2
	lithuanian-low	13.7	23.3
	lower-sorbian-low	26.2	33.8
	macedonian-low	34.9	52.1
	navajo-low	7.9	19.0
	northern-sami-low	6.0	16.2
	norwegian-bokmal-low	56.6	67.8
	norwegian-nynorsk-low	43.5	49.6
	persian-low	11.3	24.5
	polish-low	18.9	41.3
	portuguese-low	32.2	63.6
	quechua-low	11.0	16.4
	romanian-low	21.7	44.8
	russian-low	23.8	45.6
	scottish-gaelic-low	42.0	44.0
	slovak-low	23.6	42.4
	slovene-low	32.9	49.0
	sorani-low	8.6	19.3
	spanish-low	35.8	57.1
	swedish-low	45.5	54.2
	turkish-low	7.3	14.1
	ukrainian-low	31.5	43.9
	urdu-low	30.5	31.7
	welsh-low	22.0	22.0

TABLE 5.15: Our model's performance on the test datasets for those languages where it cannot beat the baseline system.

Chapter 6

Conclusion and Future Scope

This thesis mainly deals with the task of lemmatization as well as two other morphological operations for highly inflected Indic languages. To start with, we choose Bengali which is the second most widely spoken language in India as well as highly rich in morphology. Due to the unavailability of standard training dataset in Bengali, we could not initially use the globally popular state-of-the-art supervised lemmatizers and thus, concentrate on unsupervised techniques. Our first endeavour in this area is to develop a knowledge-based, unsupervised lemmatization algorithm (named as BenLem) that makes use of a dictionary, a POS tagger and a suffix list in the concerned language. The algorithm is easy to port for similar languages. Primarily a small lemma tagged Bengali dataset is built for evaluation of our proposed lemmatizer. Further we show the effectiveness of lemmatization in Bengali WSD systems. Ten highly polysemous words are chosen for experimentation. Empirically it is found that prior lemmatization of the sense and the context bags of the target word improves the performance of the WSD systems. Moreover, the role of Bengali WordNet is investigated for WSD.

Next, our initial lemmatization algorithm is modified by incorporating the following changes. Firstly, to determine the potential lemmas for an input contextual word, formerly used string similarity-based measure is replaced by a novel trie searching technique. Secondly, word embeddings are utilized to find the most appropriate root from the plausible candidates. On the basis of resource requirements, three different experimental settings are designed. For each of the settings, two different variants are developed depending on whether word vectors are used or not. In addition to Bengali,

this time Hindi is included for experimentation. It is found from the experimental results that in absence of an efficient POS tagger, word embeddings are useful to find the correct lemma. But having a good POS tagger in hand, a prefix matching technique is better.

Thereafter, inspired by the success of word vectors in capturing the linguistic regularities, we initially develop a shallow neural network model for lemma transduction. Given a context of three consecutive words as input, the proposed model generates the lemma of the middle word. All words are represented by their corresponding vector embeddings. The particular root in the dictionary is returned as the lemma with which the transduced vector possesses maximum cosine similarity. The model is supervised and needs training data of lemma tagged in-context words. We test the model on Bengali. However, it cannot produce a good accuracy reflecting that a very large amount of data is required to train the model.

We eventually come up with a deep gated recurrent model for supervised lemmatization. The proposed model is language independent as well as context sensitive i.e. it lemmatizes sentential words. To find the lemma, the proper word-lemma transformation pattern is classified. Hence this approach is much more generalized and can manage out-of-vocabulary words. Our model consists of two successive bidirectional recurrent networks. The major edges of the proposed lemmatizer are - no feature definition is necessary and except the gold lemma form, no other morphological tag is required for training. The model is assessed over nine languages. Out of them two are Indo-Aryan, Six are Indo-European and last one is Uralic. The results beat the state-of-the-art for all the languages except Bengali. Another salient contribution of this work is to develop a large gold lemma and POS annotated Bengali dataset which will definitely help the respective NLP research community.

Regarding lemmatization, a number of future research directions exist. They are listed below.

- There are chances to improve the current state-of-the-art of supervised lemmatization systems. Our proposed deep neural lemmatizer cannot produce the best result in Bengali for which the amount of training data is relatively smaller than that of the other languages considered for experimentation. Also for unseen words, its performance is worse than Lemming (Müller et al., 2015) in Bengali and Hindi. So far,

we did not try the recently advanced neural networks e.g. sequence-to-sequence model (Sutskever et al., 2014), model with attention mechanism (Bahdanau et al., 2014) etc. for lemmatization purpose. They can be explored later.

- Integration of a lemmatization module in higher level NLP systems is a vast area to work. For different semantic processing tasks such as WSD, MT, question answering (QA) etc., where to know the meaning of surface words is needed, there use of a lemmatizer can significantly boost the performance. A preliminary work on WSD for Bengali shows that a more intensive research is needed to achieve good performance for WSD. This is indeed an important research problem.

- Replacing stemming by lemmatization in the preprocessing step of information retrieval can be a good experimental study. In general, the terms present in the query set and the document collection are normalized by stemming before retrieval starts. This is done in order to reduce the error caused by the presence of morphological inflections. If instead of stemmer, a lemmatizer is used to handle the variant word forms, then investigating how the retrieval performance changes would give more insight to the effective way of word form normalization.

- Although the supervised lemmatizers produce very high accuracy as evident by our experiments, still there is a demand of unsupervised ones for resource scarce languages. In these languages, building a sufficient amount of training data is very expensive as well as highly time consuming. Because of that, using unsupervised lemmatizers is the only option there. In chapter 3, we have seen that the combination of word embedding with POS degrades the accuracy of the trie-based lemmatizer compared to when only POS information is used. An effective way of exploiting word embedding can improve the state-of-the-art of unsupervised lemmatization.

Apart from lemmatization, our research includes the problems of morphological tagging and inflection generation. In morphological tagging, the objective is to capture different properties of an in-context word. Previously this task has been formulated as a single label classification problem where all the attributes are combinedly represented as a single tag. As this approach suffers from the inability to handle the missing valid combinations of the component attributes, we devise a multi-label classification strategy for tagging. Additionally, instead of considering the whole sentences, a CNN-based

model is proposed, that takes into account the local context of the target word. We carry out the experiments on five Indic languages and two severely resource poor non-Indic languages. Initially, only character-level word embeddings are given input to our model. After that the distributional word vectors are added to the syntactic vectors to improve the model performance. Experimental outcomes show that in absence of the distributional word vectors, the proposed BLSTM-CNN model performs better than the state-of-the-art BLSTM-BLSTM model. Next, we discuss the scope of future research for this task.

- BLSTM-CNN performs better than BLSTM-BLSTM in our experiments when only character-level word vectors are used. However, to make a definitive conclusion regarding this, more languages should be explored.
- Character-level, cross-lingual learning produces notable performance improvement over single language setting for BLSTM-BLSTM. However, with addition of distributional word vectors with character vectors, the cross-lingual learning cannot beat the single language setting. One reason behind it may be that we did not jointly train the distributional word vectors using parallel corpora of the related languages. That is why the distributional vectors are not sharing the cross-lingual semantic information and when used for cross-lingual learning, degrades the performance.
- Although from the model design perspective, multitask learning across keys can manage unseen tags, but empirically it is found comparable with the single language setting. An in-depth experimental study is required to find the reason behind it.

Finally, for the inflection generation problem, we participated in the CoNLL-SIGMORPHON 2017 shared task (Cotterell et al., 2017). The task covers 52 languages world-wide and asks for supervised systems. Three different training environments are provided depending on the size of the training data. Our system consists of a character-level LSTM model designed to classify the transformation between the input and the target word forms. Though the proposed model performs relatively better under high resource scenario, still its performance is poor when the size of the training data is medium or low.

Thus, building an effective model for the task under low resource scenario remains a challenge. Proper integration of external corpora is another research goal.

Bibliography

Aharoni, R., Goldberg, Y., and Belinkov, Y. (2016). Improving sequence to sequence learning for morphological inflection generation: The biu-mit systems for the sigmorphon 2016 shared task for morphological reinflection. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 41–48.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Bamman, D. and Smith, D. (2012). Extracting two thousand years of latin from a million book library. *J. Comput. Cult. Herit.*, 5(1):2:1–2:13.

Banerjee, S. and Pedersen, T. (2002). An adapted lesk algorithm for word sense disambiguation using wordnet. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 136–145. Springer.

Bapat, M., Gune, H., and Bhattacharyya, P. (2010). A paradigm-based finite state morphological analyzer for marathi. In *Proceedings of the 1st Workshop on South and Southeast Asian Natural Language Processing*, pages 26–34.

Bergmanis, T., Kann, K., Schütze, H., and Goldwater, S. (2017). Training data augmentation for low-resource morphological inflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 31–39. Association for Computational Linguistics.

Bharati, A., Kulkarni, A. P., and Sheeba, V. (2006). Building a wide coverage sanskrit morphological analyser: A practical approach. In *The First National Symposium on Modelling and Shallow Parsing of Indian Languages, IIT-Bombay*.

Bhattacharya, S., Choudhury, M., Sarkar, S., and Basu, A. (2005). Inflectional morphology synthesis for bengali noun, pronoun and verb systems. *Proc. of NCCPB*, 8.

Bhattacharyya, P., Bahuguna, A., Talukdar, L., and Phukan, B. (2014). Facilitating multi-lingual sense annotation: Human mediated lemmatizer. In Orav, H., Fellbaum, C., and Vossen, P., editors, *Proceedings of the Seventh Global Wordnet Conference*, pages 224–231, Tartu, Estonia.

Bögel, T., Butt, M., Hautli, A., and Sulger, S. (2007). Developing a finite-state morphological analyzer for urdu and hindi. *Finite State Methods and Natural Language Processing*, page 86.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL*, 5:135–146.

Buys, J. and Botha, J. A. (2016). Cross-lingual morphological tagging for low-resource languages. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1954–1964, Berlin, Germany. Association for Computational Linguistics.

Cardellino, C. (2016). Spanish Billion Words Corpus and Embeddings.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Chollet, F. et al. (2015). Keras. https://github.com/keras-team/keras.

Chrupala, G., Dinu, G., and Genabith, J. v. (2008). Learning morphology with morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Cotterell, R. and Heigold, G. (2017). Cross-lingual character-level neural morphological tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 748–759, Copenhagen, Denmark. Association for Computational Linguistics.

Cotterell, R., Kirov, C., Sylak-Glassman, J., Walther, G., Vylomova, E., Xia, P., Faruqui, M., Kübler, S., Yarowsky, D., Eisner, J., and Hulden, M. (2017). Conll-sigmorphon 2017 shared task: Universal morphological reinflection in 52 languages. *CoRR*, abs/1706.09031.

Cotterell, R., Kirov, C., Sylak-Glassman, J., Yarowsky, D., Eisner, J., and Hulden, M. (2016). The sigmorphon 2016 shared task—morphological reinflection. In *Proceedings of the 2016 Meeting of SIGMORPHON*, Berlin, Germany. Association for Computational Linguistics.

Dabre, R., Amberkar, A., and Bhattacharyya, P. (2012). Morphological analyzer for affix stacking languages: A case study of Marathi. In *Proceedings of COLING 2012: Posters*, pages 225–234, Mumbai, India. The COLING 2012 Organizing Committee.

Dabre, R., Amberkar, A., and Bhattacharyya, P. (2013). A way to break them all: A compound word analyzer for marathi. *ICON*.

Dasgupta, S. and Ng, V. (2006). Unsupervised morphological parsing of bengali. *Language Resources and Evaluation*, 40(3-4):311–330.

Dash, N. S. (2015). A descriptive study of bengali words.

Dauphin, Y. N., Vries, H. d., and Bengio, Y. (2015). Equilibrated adaptive learning rates for non-convex optimization. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 1504–1512, Cambridge, MA, USA. MIT Press.

Dolamic, L. and Savoy, J. (2010). Comparative study of indexing and search strategies for the hindi, marathi, and bengali languages. 9(3):11:1–11:24.

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Durrett, G. and DeNero, J. (2013). Supervised learning of complete morphological paradigms. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1185–1195, Atlanta, Georgia. Association for Computational Linguistics.

Faridee, A. Z. M., Tyers, F. M., et al. (2009). Development of a morphological analyser for bengali. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pages 43–50. Universidad de Alicante. Departamento de Lenguajes y Sistemas Informáticos.

Faruqui, M., Tsvetkov, Y., Neubig, G., and Dyer, C. (2016). Morphological inflection generation using character sequence to sequence learning. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 634–643, San Diego, California. Association for Computational Linguistics.

Fellbaum, C. (1998). *WordNet: An electronic lexical database*. MIT Press.

Ganguly, D., Leveling, J., and Jones, G. J. F. (2007). Dcu@morpheme extraction task of fire-2012: Rule-based stemmers for bengali and hindi. In *Post-Proceedings of the 4th and 5th Workshops of the Forum for Information Retrieval Evaluation*, FIRE '12 & '13, pages 12:1–12:5, New York, NY, USA. ACM.

Gesmundo, A. and Samardzic, T. (2012). Lemmatisation as a tagging task. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 368–372, Jeju Island, Korea. Association for Computational Linguistics.

Goyal, V. and Lehal, G. S. (2008). Hindi morphological analyzer and generator. In *2008 First International Conference on Emerging Trends in Engineering and Technology*, pages 1156–1159.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, Colorado. Association for Computational Linguistics.

Haug, D. T. and Jøhndal, M. (2008). Creating a parallel treebank of the old indo-european bible translations. In *Proceedings of the Second Workshop on Language Technology for Cultural Heritage Data (LaTeCH 2008)*, pages 27–34.

Heigold, G., Neumann, G., and van Genabith, J. (2016). Neural morphological tagging from characters for morphologically rich languages. *CoRR*, abs/1606.06640.

Heigold, G., Neumann, G., and van Genabith, J. (2017). An extensive empirical evaluation of character-based morphological tagging for 14 languages. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 505–513, Valencia, Spain. Association for Computational Linguistics.

Hulden, M., Forsberg, M., and Ahlberg, M. (2014). Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 569–578, Gothenburg, Sweden. Association for Computational Linguistics.

Inoue, G., Shindo, H., and Matsumoto, Y. (2017). Joint prediction of morphosyntactic categories for fine-grained arabic part-of-speech tagging exploiting tag dictionary information. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 421–431, Vancouver, Canada. Association for Computational Linguistics.

Jha, G. N., Agrawal, M., Mishra, S. K., Mani, D., Mishra, D., Bhadra, M., Singh, S. K., et al. (2009). Inflectional morphology analyzer for sanskrit. In *Sanskrit computational linguistics*, pages 219–238. Springer.

Jiampojamarn, S., Cherry, C., and Kondrak, G. (2008). Joint processing and discriminative training for letter-to-phoneme conversion. In *Proceedings of ACL-08: HLT*, pages 905–913, Columbus, Ohio. Association for Computational Linguistics.

Jiampojamarn, S., Cherry, C., and Kondrak, G. (2010). Integrating joint n-gram features into a discriminative training framework. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 697–700, Los Angeles, California. Association for Computational Linguistics.

Jiampojamarn, S., Kondrak, G., and Sherif, T. (2007). Applying many-to-many alignments and hidden markov models to letter-to-phoneme conversion. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 372–379, Rochester, New York. Association for Computational Linguistics.

Kann, K., Cotterell, R., and Schütze, H. (2017a). Neural multi-source morphological reinflection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 514–524, Valencia, Spain. Association for Computational Linguistics.

Kann, K., Cotterell, R., and Schütze, H. (2017b). One-shot neural cross-lingual transfer for paradigm completion. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1993–2003, Vancouver, Canada. Association for Computational Linguistics.

Kann, K. and Schütze, H. (2016). Single-model encoder-decoder with explicit morphological representation for reinflection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 555–560, Berlin, Germany. Association for Computational Linguistics.

Kann, K. and Schütze, H. (2017). The lmu system for the conll-sigmorphon 2017 shared task on universal morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 40–48. Association for Computational Linguistics.

Kanuparthi, N., Inumella, A., and Sharma, D. M. (2012). Hindi derivational morphological analyzer. In *Proceedings of the Twelfth Meeting of the Special Interest Group on Computational Morphology and Phonology*, SIGMORPHON '12, pages 10–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

Khapra, M., Kulkarni, A., Sohoney, S., and Bhattacharyya, P. (2010). All words domain adapted wsd: Finding a middle ground between supervision and unsupervision. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1532–1541, Uppsala, Sweden. Association for Computational Linguistics.

Kilgarriff, A. and Rosenzweig, J. (2000). English senseval: Report and results. In *LREC*, volume 6, page 2.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koskenniemi, K. (1984). A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics*, pages 178–181, Stanford, California, USA. Association for Computational Linguistics.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kumar, A., Cotterell, R., Padró, L., and Oliver, A. (2017). Morphological analysis of the dravidian language family. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 217–222, Valencia, Spain. Association for Computational Linguistics.

Kumar, A., Padró, L., and Oliver, A. (2015). Learning agglutinative morphology of indian languages with linguistically motivated adaptor grammars. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 307–312.

Kumar, D., Singh, M., and Shukla, S. (2012). FST based morphological analyzer for hindi language. *CoRR*, abs/1207.5409.

Kumar, V. and Kaur, R. G. (2013). *Paradigm based Hindi morphological analyzer*. PhD thesis.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual*

International Conference on Systems Documentation, SIGDOC '86, pages 24–26, New York, NY, USA. ACM.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luis, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

Liu, L. and Mao, L. J. (2016). Morphological reinflection with conditional random fields and unsupervised features. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 36–40.

Loponen, A. and Järvelin, K. (2010). A dictionary and corpus independent statistical lemmatizer for information retrieval in low resource languages. In *Multilingual and Multimodal Information Access Evaluation*, pages 3–14. Springer.

Loponen, A., Paik, J. H., and Järvelin, K. (2013). *UTA Stemming and Lemmatization Experiments in the FIRE Bengali Ad Hoc Task*, pages 258–268. Springer Berlin Heidelberg, Berlin, Heidelberg.

Lushanthan, S., Weerasinghe, A. R., and Herath, D. L. (2014). Morphological analyzer and generator for tamil language. In *2014 14th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 190–196.

Majumder, P., Mitra, M., Pal, D., Bandyopadhyay, A., Maiti, S., Pal, S., Modak, D., and Sanyal, S. (2010). The fire 2008 evaluation exercise. *ACM Transactions on Asian Language Information Processing (TALIP)*, 9(3):10:1–10:24.

Majumder, P., Mitra, M., Parui, S. K., Kole, G., Mitra, P., and Datta, K. (2007). Yass: Yet another suffix stripper. *ACM Trans. Inf. Syst.*, 25(4).

Makarov, P., Ruzsics, T., and Clematide, S. (2017). Align and copy: Uzh at sigmorphon 2017 shared task for morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 49–57. Association for Computational Linguistics.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA. Curran Associates Inc.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Miller, G. A., Chodorow, M., Landes, S., Leacock, C., and Thomas, R. G. (1994). Using a semantic concordance for sense identification. In *Proceedings of the workshop on Human Language Technology*, pages 240–243. Association for Computational Linguistics.

Mueller, T., Schmid, H., and Schütze, H. (2013). Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA. Association for Computational Linguistics.

Müller, T., Cotterell, R., Fraser, A., and Schütze, H. (2015). Joint lemmatization and morphological tagging with lemming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2268–2274, Lisbon, Portugal. Association for Computational Linguistics.

Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Comput. Surv.*, 41(2):10:1–10:69.

Nicolai, G., Cherry, C., and Kondrak, G. (2015). Inflection generation as discriminative string transduction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 922–931, Denver, Colorado. Association for Computational Linguistics.

Nicolai, G. and Kondrak, G. (2016). Leveraging inflection tables for stemming and lemmatization. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1138–1147, Berlin, Germany. Association for Computational Linguistics.

Nivre, J., Agić, Ž., Ahrenberg, L., Aranzabe, M. J., Asahara, M., Atutxa, A., Ballesteros, M., Bauer, J., Bengoetxea, K., Bhat, R. A., Bick, E., Bosco, C., Bouma, G., Bowman, S., Candito, M., Cebiroğlu Eryiğit, G., Celano, G. G. A., Chalub, F., Choi, J., Çöltekin, Ç., Connor, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Diaz de Ilarraza, A., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eli, M., Erjavec, T., Farkas, R., Foster, J., Freitas, C., Gajdošová, K., Galbraith, D., Garcia, M., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., Gonzáles Saavedra, B., Grioni, M., Grūzītis, N., Guillaume, B., Habash, N., Hajič, J., Hà Mỹ, L., Haug, D., Hladká, B., Hohle, P., Ion, R., Irimia, E., Johannsen, A., Jørgensen, F., Kaşıkara, H., Kanayama, H., Kanerva, J., Kotsyba, N., Krek, S., Laippala, V., Lê Hồng, P., Lenci, A., Ljubešić, N., Lyashevskaya, O., Lynn, T., Makazhanov, A., Manning, C., Mărănduc, C., Mareček, D., Martínez Alonso, H., Martins, A., Mašek, J., Matsumoto, Y., McDonald, R., Missilä, A., Mititelu, V., Miyao, Y., Montemagni, S., More, A., Mori, S., Moskalevskyi, B., Muischnek, K., Mustafina, N., Müürisep, K., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikolaev, V., Nurmi, H., Ojala, S., Osenova, P., Øvrelid, L., Pascual, E., Passarotti, M., Perez, C.-A., Perrier, G., Petrov, S., Piitulainen, J., Plank, B., Popel, M., Pretkalniņa, L., Prokopidis, P., Puolakainen, T., Pyysalo, S., Rademaker, A., Ramasamy, L., Real, L., Rituma, L., Rosa, R., Saleh, S., Sanguinetti, M., Saulīte, B., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shakurova, L., Shen, M., Sichinava, D., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Smith, A., Suhr, A., Sulubacak, U., Szántó, Z., Taji, D., Tanaka, T., Tsarfaty, R., Tyers, F., Uematsu, S., Uria, L., van Noord, G., Varga, V., Vincze, V., Washington, J. N., Žabokrtský, Z., Zeldes, A., Zeman, D., and Zhu, H. (2017). Universal dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.

Paik, J. H., Mitra, M., Parui, S. K., and Järvelin, K. (2011). Gras: An effective and efficient stemming algorithm for information retrieval. *ACM Trans. Inf. Syst.*, 29(4):19:1–19:24.

Paik, J. H. and Parui, S. K. (2011). A fast corpus-based stemmer. 10(2):8:1–8:16.

Paik, J. H., Parui, S. K., Pal, D., and Robertson, S. E. (2013). Effective and robust query-based stemming. *ACM Trans. Inf. Syst.*, 31(4):18:1–18:29.

Pandey, A. K. and Siddiqui, T. J. (2008). An unsupervised hindi stemmer with heuristic improvements. In *Proceedings of the second workshop on Analytics for noisy unstructured text data*, pages 99–105. ACM.

Paul, S., Joshi, N., and Mathur, I. (2013). Development of a hindi lemmatizer. *International Journal of Computational Linguistics and Natural Language Processing*, 2(5):380–384.

Plisson, J., Lavrac, N., Mladenic, D., et al. (2004). A rule based approach to word lemmatization. *Proceedings of IS-2004*, pages 83–86.

Rastogi, M. and Khanna, P. (2014). Development of morphological analyzer for hindi. *International Journal of Computer Applications*, 95(17).

Ravishankar, V. and Tyers, F. M. (2017). Finite-state morphological analysis for marathi. In *Proceedings of the 13th International Conference on Finite State Methods and Natural Language Processing (FSMNLP 2017)*, pages 50–55.

Sarkar, S. and Bandyopadhyay, S. (2012a). Fire 2012 working notes: Morpheme extraction task using mulaadhaar–a rule-based stemmer for bengali. In *Working Notes for the FIRE 2012 Workshop*.

Sarkar, S. and Bandyopadhyay, S. (2012b). On the evolution of stemmers: A study in the context of bengali language. *International Journal of Computational Linguistics and Natural Language Processing*, 1.

Schmid, H. (2006). *A Programming Language for Finite State Transducers*, pages 308–309. Springer Berlin Heidelberg, Berlin, Heidelberg.

Senapati, A. and Garain, U. (2012). Bangla morphological analyzer using finite automata: Isi@ fire met 2012. In *Working Notes for the FIRE 2012 Workshop*.

Shen, L., Satta, G., and Joshi, A. (2007). Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767, Prague, Czech Republic. Association for Computational Linguistics.

Sunitha, K. and Kalyani, N. (2009). A novel approach to improve rule based telugu morphological analyzer. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 1649–1652. IEEE.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Sylak-Glassman, J., Kirov, C., Yarowsky, D., and Que, R. (2015). A language-independent feature schema for inflectional morphology. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 674–680, Beijing, China. Association for Computational Linguistics.

Toutanova, K. and Cherry, C. (2009). A global model for joint lemmatization and part-of-speech prediction. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 486–494, Suntec, Singapore. Association for Computational Linguistics.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technologies (NAACL:HLT)-Volume 1*, pages 173–180.

Yu, X., Falenska, A., and Vu, N. T. (2017). A general-purpose tagger with convolutional neural networks. *CoRR*, abs/1706.01723.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.