Towards developing tools for Indian Languages using Deep Learning

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computational Linguistics by Research

by

Pravallika Etoori
201325096
e.pravallika@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
October 2019

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "**Towards developing tools for Indian Languages using Deep Learning**" by **Pravallika Etoori**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. Radhika Mamidi

To My beloved Parents

Acknowledgments

I am indebted to all the amazing people in my life. Firstly, I would like to thank my parents and brother for being my pillars of strength. They have been supportive throughout my journey and believed in me with all my decisions. Their unconditional love during my lowest and the highest points kept me going to achieve my goals. Without their tremendous support, this wouldn't have been possible. I thank my family - grandparents, uncles, aunts - for always supporting me, loving me, and taking care of me when I was away from my parents.

I am grateful to my advisor Dr. Radhika Mamidi for guiding me throughout this work. She has given me the freedom to explore different areas of research and always encouraged to try different topics and approaches. I am glad I could get that holistic experience. I would like to extend my immense gratitude to Dr. Manoj Chinnakotla because of whom this topic took form. Despite being away from campus, he has been available whenever I needed guidance. I would also like to thank Prof. Dipti Misra Sharma and Prof. Manish Srivastava for teaching some wonderful courses which proliferated my interest in this field and inspired me to pursue work in it.

I am thankful to IIIT-H for giving me opportunity to learn, explore, and grow. Five years of IIIT life has given me the most amazing friends and life-long relations. Thanks is a small word for Tinku Monish, my best friend, without whom I cannot imagine my college life. He has been with me throughout - all my ups and downs and helped me survive the most critical times. I am grateful to Soujanya, who has not only been my best friend but also a great inspiration. She has always been there for me from even before college life has begun to this day. I have always looked up to and counted on Hari Narayana who is more of a friend than senior. Thanks to Aditya Srinivas for being an amazing caring friend and for always lending a patient ear for me. From roommate to close friend, Divya Jyothi (DJ) has always been a great supportive friend. Thanks to Madhuri, my CLD mate, for all the theoretical discussions and pep talks. I am thankful to my flatmate Srishti for being supportive in my stressful times of thesis completion. Last but not at all the least, my best friend Sanjana Reddy has always been my go-to person and my happy place even though we have been in long-distance friendship.

I am eternally grateful to all these people and many other people whose names might not be here but have been in my life during college and in the journey of this work.

Abstract

Extensive research is being carried out in the field of Natural Language Processing (NLP) in the context of Indian languages. Spelling correction, word segmentation, and grammar checking are the fundamental problems in NLP. The aim of these problems is to identify noise in the data and correct it. These tools are important for many NLP applications like web search engines, text summarization, sentiment analysis, machine translation etc.

Many methods have been developed for these problems for English, which usually exploit linguistic resources like parsers, large amounts of real world data etc. making it difficult to adapt them to other languages. For these problems, deep learning models have also been implemented for English. These models use parallel data of noisy and correct mappings from different sources as training data for automatic correction tasks. Indian languages are resource-scarce and do not have such parallel data due to low volume of queries and non-existence of such prior implementations. Spell correction is very crucial for applications like web search engines. In speech recognition and optical character recognition (OCR), word boundaries are not often comprehensible. Grammar checking is important to clean up the corpus before using it in any application. It is also useful as an aid for second language users and for applications like automatic essay grading. In applications like speech-to-text, handwritten text-to-text, and Machine Translation, applying spell checking, word segmentation, and grammar checking on the output will result in improved accuracy.

Most of the existing language processing systems for Indian languages are rule-based and language specific. Majority of the works have a dictionary-based approach and check the errors against the dictionary. Due to lack of availability of training data for machine learning models, we create synthetic datasets using language rules.

In this work, we present approaches to build automatic language correction systems for Hindi. We have proposed deep learning models for spelling correction, word segmentation, and grammar checking. The proposed approaches are applicable to any resource-scarce language. A comparative evaluation for each of these three problems shows that the proposed models are competitive with the existing correction techniques for Indian languages.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spelling correction, word segmentation, and grammar checking are some of the fundamental but crucial problems in the field of Natural Language Processing (NLP). Spell checking is the process of checking misspellings in a text. Word segmentation is the process of separating wrongly fused words in a text with word boundaries at correct positions. Grammar checking is the process of verifying text for grammatical, including morphological and syntactical, correctness. These methods are applied on data to improve the performance of NLP applications like web search engines, machine translation, text summarization, automatic essay grading etc.

Apart from the usage in NLP applications, these tools are necessary for another important reason. Second language speakers of a language commit many mistakes. Native speakers and many people who claim to be fluent in a language often fail to be proficient in their writing skills due to lack of grammatical knowledge. Grammar checkers are considered as a type of foreign language writing aid which non-native speakers can use to proofread their writings as such programs endeavor to identify syntactical errors [16].

In linguistics, corpus can be defined as a large collection of texts. Data corpora are critical to develop various applications [55]. Developing datasets in different languages for different types of applications is important in the field of NLP.

Examples of spelling errors, word segmentation errors, and grammatical errors in Hindi are shown below along with their corresponding English translations. The error parts in the examples can be seen underlined in each example:

Example 1.1	अलीबाबा ओर चालिस छोर	*Misspelling*
	अलीबाबा और चालीस चोर	*Correct spelling*
	Alibaba and forty thieves	*English translation*

Example 1.2	यहाँ एकपेड़ था।	*Incorrect word boundary*
	यहाँ एक पेड़ था।	*Correct word boundary*
	There was a tree here.	*English translation*

Example 1.3	मैं मेरा काम करता हूँ।	*Ungrammatical*
	मैं अपना काम करता हूँ।	*Grammatical*
	I do my own work.	*English translation*

Hindi as a language

Hindi belongs to the Indo-Iranian branch of Indo-European language family [21]. Linguistically, it is not different from Urdu. It is natively spoken by approximately 366 million speakers (*source*: Ethnologue, www.ethnologue.com), mainly in the central and northern part of India, but also in Bangladesh, Nepal, the United Kingdom and many other countries. In addition, Hindi is also used as a second language or a lingua franca by many Indians in non-Hindi speaking regions. Like many other Asian languages, though Hindi follows a Subject-Object-Verb (SOV) word order, it has relatively free word order. The word order flexibility increases the number of options available for expressing information structure, significantly complicating the means by which the incoming signal can be decoded [57].

1.1 Motivation

1.1.1 Tools for Indian Languages

India being a multilingual nation, research is being carried out extensively in the field of NLP in the context of Indian languages. The language technology and resources are of critical importance in present India. The linguistic scenario in India is complex and there is a huge need for building software systems for Indian languages [34]. There have been initiatives to develop corpora for Indian languages [66, 35, 6]. There has been very little focus on developing language processing tools and there are not many recent developments in this area. All the available tools for Indian languages like spell checkers and grammar checkers are based on dictionary and rule-based methods (Chapter 2).

1.1.2 Deep Learning Techniques

Recently, deep learning techniques have made it possible to develop end-to-end systems that are more generic in the set of tasks they can handle as they do not require hand-crafted features. With the large availability of data in English language, many NLP systems are being built for English using deep learning. These techniques have been successful in producing state-of-the-art results in various domains [76].

Thus, the current scenario of available language processing tools for Indian languages and the success of deep learning techniques in various NLP problems motivated us to attempt spell checking, word segmentation, and grammar checking problems for Indian languages using deep learning. The goal was to present techniques which are suitable for resources-scarce languages and can be extended to any language. We also aimed to present techniques for creating datasets to pursue these problems with data driven approaches. In the recent times, there has been a huge focus on developing applications like sentiment analysis, machine translation, dialogue systems, text summarization, recommendation systems etc. for Indian languages using statistical methods. Clean data is very important for the performance of these approaches [36]. In this work, we present techniques which have shown promising results for spell checking, word segmentation, and grammar checking of Hindi. The presented methodologies can be applied to other Indian languages.

1.2 Thesis Overview

In this work, we approached three problems of NLP namely Spell Checking, Word Segmentation, and Grammar Checking for Hindi. A good amount of research has been carried out in the past for these problems. There have been a lot of recent advances in these areas for English. But every language differs in their linguistic structure and morphology. While, English is a fixed word order language, Hindi and many other Indian languages are free word order languages. The morphological richness of the language also increases from English to Hindi (an Indo-Aryan language) and further to Telugu (a Dravidian language). We discuss in detail each of these problems in the context of Indian languages. This work also presents how deep learning techniques can be applied to achieve good results despite data unavailability.

The performance of sequence-to-sequence (seq2seq) models [71, 17] has been impressive for a variety of tasks in recent times. They follow encoder-decoder architecture. These models have presented great results for neural machine translation, a critical NLP problem. In this work, we thoroughly explore this model architecture. We find that this type of models are very powerful when trained on large dataset of input and output sequences.

For spell checking, we collected dataset of words and phrases for the Indian languages, Hindi and Telugu. Then, for both of these languages, we created a set of rules indicating the possible spelling errors at character-level. For every instance in data, we generated all possible misspellings by applying the created rules. Thus, we obtained a dataset of noisy and correct word pairs. We used this synthetic dataset to train an end-to-end sequence-to-sequence model.

For word segmentation, we considered a corpus of Hindi sentences. After preprocessing the corpus, for every sentence we generated a noisy sentence with word fusings, that is, a sentence with word

segmentation error. We did this by dropping spaces in between words in every sentence. Using the obtained dataset of erroneous and correct sentence pairs, we trained a sequence-to-sequence model.

For grammar checking, we have performed a detailed study of the linguistic structure of Hindi. We created a set of rules which contains occurrences violating the above mentioned language norms of Hindi. For every sentence in the Hindi sentence corpus, we checked if the sentence follows any of the created set of violations. We have collected these ungrammatical sentences. Then, we randomly extracted same number of grammatical sentences from the corpus. Using this dataset of ungrammatical and grammatical sentences, we built a binary classifier using machine learning which identifies if a given Hindi sentence is grammatical or ungrammatical.

We evaluated all these models against the baseline methods for each problem. We also experimented with other deep learning and machine learning models. We performed a detailed analysis on the results obtained. The results show that the proposed method for each problem is competitive with the existing techniques.

1.3 Our contributions

In this thesis, we present, to the best of our knowledge, the first ever work which deals with language processing problems in Hindi using Deep Learning. We consider the problems - spell checking, word segmentation, and grammar checking. All the datasets are made publicly available[1]. Our contributions in this work can be summarized as follows:

- Parallel dataset of misspelled Hindi words and phrases and their corrections. We also present a similar dataset for Telugu.
- A dataset for Word Segmentation in Hindi which contains pairs of sentences with word segmentation errors and corresponding sentence with word boundaries in correct positions.
- For the problem of grammar checking, we present two types of datasets:
  - A dataset of equal number of ungrammatical and grammatical sentences which can be availed for binary classification
  - A dataset containing pairs of ungrammatical sentence and its corresponding grammatical sentence.
- *seq2seq* model with encoder and decoder as Long Short Term Memory (LSTM), a type of Recurrent Neural Network (RNN) for Spell Checking in Hindi and Telugu.
- LSTM-LSTM *seq2seq* model which addresses the problem of word segmentation in Hindi

[1] `https://github.com/PravallikaRao/ToolsForIL`

- Theoretical review of Hindi grammar
- *Logistic Regression* binary classifier for grammar checking in Hindi to predict if a given sentence is grammatical or ungrammatical.

1.4 Thesis Organization

This thesis is divided into 7 chapters. The current chapter gives the introduction about the problems that are being addressed and showcases the motivation behind the choice of problems in this work. Chapter 2 discusses the relevant work that was done in the past in the domain. Chapter 3 explains in detail the architecture of deep learning models used for spell checking and word segmentation. Chapter 4 discusses the approach, experiments and results for Spell Checking. Chapter 5 discusses the methodology for word segmentation problem. Chapter 6 explains the major types of grammatical errors in Hindi and proposes an automatic grammar checking approach. In chapter 7, we present the conclusion and possible extensions of this research work.

Chapter 2

Related Work

2.1 Spell Checking

The beginning of spell checker design dates back to 1961 when Les Earnest saw it necessary to include the first spell checker that accessed a list of 10,000 acceptable words. The first spell checkers were verifiers instead of correctors. They offered no suggestions for incorrectly spelled words. This was helpful for typos but it was not so helpful for logical or phonetic errors. The challenge the developers faced was the difficulty in offering useful suggestions for misspelled words. This requires reducing words to a skeletal form and applying pattern-matching algorithms. Later, various spell checkers which suggest a correction have been developed. Significant work has been done in the field of Spell checking for Indian languages. There are spell-checkers available for Indian languages like Hindi, Marathi, Bengali, Telugu, Tamil, Oriya, Malayalam, Punjabi.

Dixit et al. [22] designed a rule-based spell-checker for Marathi which is a major Indian Language. This is the first initiative for morphology-based spell checking for Marathi. A spellchecker architecture and implementation for first level suffixes based on morphological analysis and rules of orthography was presented. Suggestions for words found to be incorrect are provided by considering the words three constituents, which are root, stem forming suffix and case marker or postposition. This method gave promising results but is completely dependant on the vocabulary and rules list.

A spell-checker is designed for Telugu [61], an agglutinating Indian language which has a very complex morphology. This spell-checker is based on Morphological Analysis and Sandhi splitting rules. It consists of two parts: a set of routines for scanning the text (Morphological Analyzer and sandhi splitting rules) and identifying valid words, and an algorithm for comparing the unrecognized words and word parts against a known list of variantly spelled words and word parts.

Another Hindi Spell-checker [67] uses a dictionary with word, frequency pairs as language model. Error detection is done by dictionary look-up. Error correction is performed using Damerau-Levenshtein

edit distance and n-gram technique. These candidates are ranked by sorting in increasing order of edit distance. Words at same edit distance are sorted in order of their frequencies.

HINSPELL [68] is a spell-checker designed for Hindi which implements a hybrid approach. Error is detected by dictionary look-up. Error correction is done by using Minimum Edit Distance technique where the closest words in the dictionary to the error word are obtained. These obtained words are given priority using a weightage algorithm and Statistical Machine Translation (SMT).

Ambili et al. [4] designed a Malayalam spell-checker that detects the error by a dictionary look-up approach and error correction is done through n-gram based technique. If a word is not present in the dictionary, it is identified as an error and n-gram based technique corrects error by finding similarity between words and computing a similarity coefficient.

Recently, Ghosh and Kristensson [25] proposed a Deep Learning model for text correction and completion in keyboard decoding for English. Unlike all traditional text correction and completion keyboard decoders which train on billions of words and train over extensive period of time, this model trains on a small vocabulary of 50000 words and the training is completed within a day. This is a first attempt at text correction using Deep Neural Networks which gave promising results.

Sakaguchi et al. [64] approached the problem of Spell Correction using semi-character Recurrent Neural Networks (scRNN) on English data. This was inspired by the robust word recognition mechanism known in psycholinguistics literature as the Cmabrigde Uinervtisy effect. They have also demonstrated a similarity between scRNN and human word recognition mechanisms, by showing that scRNN replicates a psycholinguistics experiment about word recognition difficulty in terms of the position of jumbled characters.

Studies of spell checking techniques for Indian Languages [42, 29] show that the existing spell-checkers have two major steps: Error detection and error correction. Error detection is done by dictionary look-up. Error correction consists of two steps: the generation of candidate corrections and the ranking of candidate corrections. The most studied spelling correction algorithms are: edit distance, similarity keys, rule-based techniques, n-gram-based techniques, probabilistic techniques, neural networks, and noisy channel model. All of these methods can be thought of as calculating a distance between the misspelled word and each word in the dictionary or index. The shorter the distance the higher the dictionary word is ranked.

While there have been a few attempts to design spell-checkers for English and a few other languages using Machine Learning, to the best of our knowledge, no such prior work has been attempted for Indian languages.

2.2 Word segmentation

In most of the languages, white space is a common word boundary but in Asian languages like Chinese, Thai, and Myanmar, the word boundary information must be inferred from morphological, syntactic, and statistical analysis. Hindi is written in Devanagari script with white space as a word boundary.

A rule-based word segmentation technique [48] has been proposed in the domain of medicine. This approach takes into account syntactic, semantic, and grammatical rules. The approach developed in this paper is a pragmatic way to rapidly increase the lexico-semantic part of medical dictionaries.

In 2006, Gambell and Yang [23] presented a detailed review of the literature on word segmentation. This analysis in this review is performed from an interesting point of view of how computational models would scale up in a realistic setting of language acquisition. They have performed experiments using Statistical Learning and Algebraic learning. They have obtained surprising observation that Algebraic learning yielded best word segmentation results. Based on these experiments, they have conjectured that algebraic learning is a reasonable strategy for word segmentation.

Chinese word segmentation is a difficult problem that has received a lot of attention. A Chinese word segmentation bake-off [70] has been held, as word segmentation is an important and difficult problem in Chinese language processing. Various approaches ranging from dictionary-based to statistical to neural networks have been presented.

Lehal presented statistical techniques for word segmentation handling space omission [45] and space insertion [46] errors for Urdu. These systems are part of the larger system designed for transliteration of Urdu text to Hindi. Statistical language modeling of both Urdu and Hindi languages in development of the space omission system. A two-stage system for detecting the space insertion problem. In the first stage, Urdu grammar rules have been applied to decide if the adjacent Urdu words have to be joined. In case these rules give a definite answer, then the system stops at that stage only. A hybrid approach is employed in the second stage to incorporate Urdu and Devanagari unigram and bigram probabilities to make the decision

Sindhi being a cursive ligature based Persio-Arabic script, is quite complex and rich having large number of characters in its script with all characters having multiple glyphs based on its position in the text. For Sindhi word segmentation, Mahar et al. [49] and Bhatti et al. [8] proposed lexicon-based approaches for preprocessing in Sindhi NLP applications. These works present the process of tokenizing Sindhi text into individual words for corpus building and creating word repository for Sindhi Spell, grammar checker and other NLP applications.

Li et al. [47] shows the impact of word segmentation on the performance of name tagging for Chinese and Japanese. They have experimented with dictionary based and Conditional Random Field (CRF) word segmenters and found that the CRF-based segmenter outperforms the dictionary based segmenter. Similarly, word segmentation will have an impact on other NLP applications which make use of data and their performance might improve when word segmentation is performed on the erroneous data.

Word segmentation is one of the crucial steps in speech synthesis and character recognition. Swaranjali [60], an isolated word recognition system for speech has been implemented for identifying distinct words in speech input. Hanmandlu and Agrawal [30], Garg et al. [24], Bhujade and Meshram [10] have presented techniques for segmentation of Hindi handwritten text.

A survey of state-of-the-art machine learning models in the context of NLP [37] presents various approaches used for word segmentation in different languages. These approaches include Hidden Markov Model (HMM), Conditional Random Field (CRF), Support Vector Machine (SVM), Maximum Entropy model for languages like English, Arabic, Chinese, Thai etc. It is found that no work has been done for word segmentation of Hindi or any other Indian language using machine learning.

Most of the works on word segmentation for Hindi are in the Speech recognition and character recognition fields rather than in NLP. Hence, we address this problem from NLP point of view. To the best of our knowledge, this is the first work using Deep Learning for word segmentation of Hindi text.

2.3 Grammar checking

Grammar checking being a crucial task, multiple grammar checkers have been developed for various languages using different techniques. But, for Indian languages we find only a few grammar checkers. In the past, grammar checking techniques have been proposed for the Indian languages: Hindi, Punjabi, and Bangla.

A grammar checker for Bangla and English was designed by Alam et al. [3] using statistical approach. This technique is based on n-gram analysis of words and POS tags. The probability of the POS tag sequence of the sentence is measured. If the probability value is greater than zero, then it is considered as a grammatical sentence, else ungrammatical.

A rule-based Punjabi grammar checker [26] has been developed which performs morphological analysis using a full-form lexicon. It uses rule-based systems for POS tagging and phrase chunking. When this system detects a grammatically incorrect sentence, it outputs the suggested correction and detailed description of the reason for grammatical error.

Bopche and Dhopavakar [11] proposed a rule-based system for Hindi grammar checking which utilizes a full-form lexicon of morphology analysis. This system matches a Parts of speech (POS) tagged input Hindi sentence against a set of correct grammatical patterns and then outputs if the sentence is grammatically correct or not.

For training machine learning models, relevant data is very important. An approach for automatic creation of Bangla error corpus [44] for training and evaluation of grammar checker systems has been presented. The procedure begins with automatic creation of large number of erroneous sentences from a set of grammatically correct sentences. Miłkowski [51] presented several approaches to automatic or semiautomatic development of symbolic rules for grammar checkers from the information contained in corpora. These can be availed to create corpus for grammar checking and thus, minimizing the manual rule generation.

Recently, attempts have been made to address grammar checking process using statistical techniques. A neural network architecture [72] has been designed to identify the grammatical errors in Statistical Machine Translation (SMT) results. Tezcan et al. proposed a Recurrent Neural Network (RNN) architecture for word-level detection of grammatical errors in SMT that utilizes monolingual features in context windows of surface and syntactic n-grams. This method relies on POS, morphological, and dependency information of the MT output and uses multi-hot encoding to represent the morphosyntactic properties of words as word vectors. This approach has been evaluated on English-Dutch and English-German Machine Translation data.

In 2018, Soni and Thakur [69] presented a systematic review of grammar checking techniques for English. They have presented how the systems have evolved over the past decade. This review concludes that approaches can be classified into three categories namely: Rule based technique, Machine learning based technique, and Hybrid technique. Each technique has its own advantages and limitations. Rule based techniques are best suited for language learning but rule designing is a laborious task. Machine learning alleviates this labor but it is dependent on the size and type of the corpus used. Hybrid technique combines the best of both techniques but each part of the hybrid technique should be implemented according to its suitability.

Various surveys [52, 9, 41] have been done on the existing grammar checking techniques in various languages. Main types of techniques for grammar checking are syntax-based, statistical, and rule-based. Manchanda et al. [50] discusses the advantages and disadvantages of each of these techniques. These studies show that grammar checkers are available for very few Indian languages and the majority of these techniques are rule-based.

Recently, Rozovskaya and Roth [62], Sakaguchi et al. [65], Chollampatt and Ng [18] presented approaches based on Machine Learning and Deep Learning for Grammatical Error Correction of English.

All of these approaches, especially deep models, typically require large amounts of annotated or labeled data. The unavailability of such data for Indian languages makes it difficult to implement such approaches.

2.4 Summary

In this chapter, we try to present the diverse nature of research done in the past for the problems of Spell Checking, Word Segmentation, and Grammar Checking. We present works done for Indian languages as well as recent advances in other languages like English. Works have been done for each of these tasks using rule-based, lexicon based, and statistical techniques. For Indian languages, not much work has been done for these tasks using Machine Learning techniques. In our work, we propose Deep Learning approaches for the tasks Spell Checking, Word Segmentation, and Grammar Checking in the context of Indian Languages, mainly Hindi.

Chapter 3

Model Architecture

In this chapter, we present the architecture and functionality of sequence-to-sequence models. These models are used for building the auto-correction tools described in the chapters 4-6.

3.1 Sequence-to-sequence Model

Sequence-to-sequence (seq2seq) models [71, 17] have enjoyed great success in a variety of tasks such as machine translation, speech recognition, image captioning, and text summarization. A basic sequence-to-sequence model consists of two neural networks: an encoder that processes the input and a decoder that generates the output. It aims to map a fixed length input with a fixed length output where the length of the input and output may differ. This model has shown great potential in input-output sequence mapping tasks like machine translation. An input side encoder captures the representations in the data, while the decoder gets the representation from the encoder along with the input and outputs a corresponding mapping to the target language.

Coupled with the attention mechanism, seq2seq models have become de-facto standard in generation tasks. The encoder Recurrent Neural Network (RNN) embeds each of the source characters into vectors exploiting the hidden states computed by the RNN. The decoder RNN predicts the next character based on its current hidden state, previous character, and also the context vector c_i, computed by the attention model. By letting the decoder have an attention mechanism [5], the encoder is relieved from the burden of having to encode all information in the source sequence into a fixed-length vector. With attention, the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly. The attention mechanism computes a fixed-size vector that encodes the whole input sequence based on the sequence of all the outputs generated by the encoder as opposed to the plain encoder-decoder model which looks only at the last state generated by the encoder for all the slices of the decoder.

Fig. 3.1. shows an overview of seq2seq model. While the core architecture remains common, there are many configurable options such as number of layers, Convolutional Neural Network (CNN) vs.

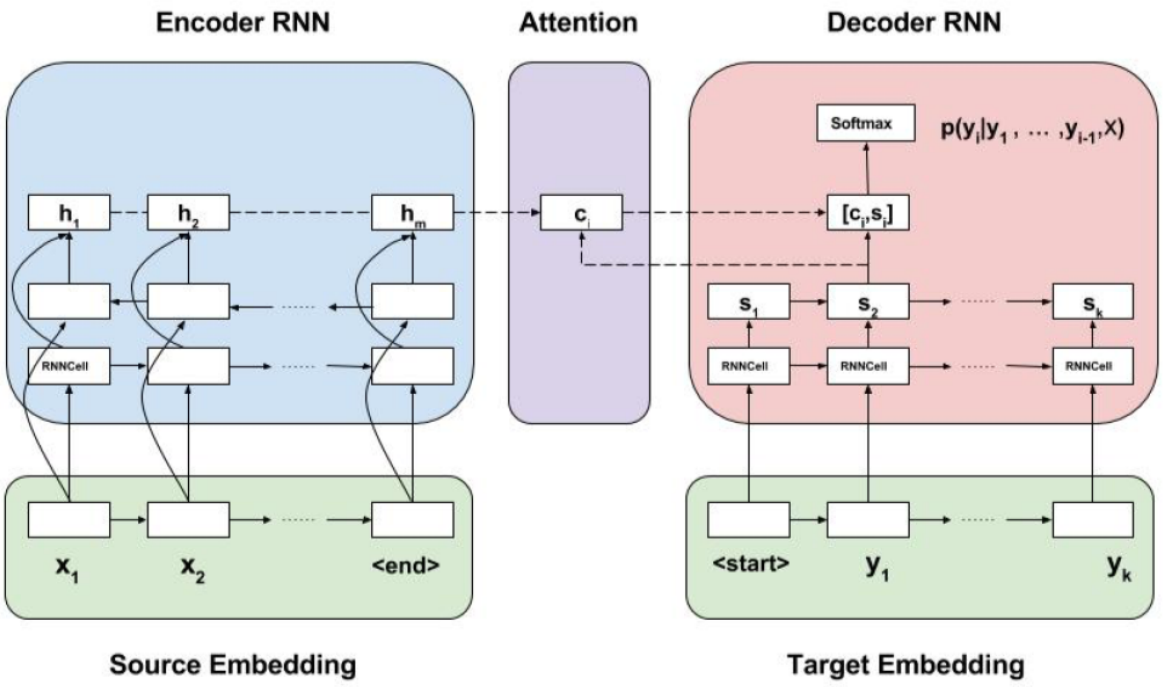

Figure 3.1 Vanilla Seq2Seq architecture with attention mechanism (source [13]).

RNN for encoder and decoder, unidirectional vs. bidirectional encoders, additive vs. multiplicative attention mechanism, Adam vs. Stochastic Gradient Descent (SGD) vs. Momentum optimizer etc.

3.2 Encoder-Decoder Architecture

An encoder-decoder architecture consists of two main parts:

Encoder takes the input data, captures the representations in the data, and trains on it. Then, the final state produced from the encoder part of the model is passed as the initial state to the decoder. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.

Decoder takes the final state of last layer from the encoder. The decoder then uses this internal representation to generate the output.

Cho et al. [17] proposed a RNN Encoder-Decoder framework that consists of two RNNs. One RNN encodes a sequence of symbols into a fixed length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of this model are jointly trained to maximize the conditional probability of a target sequence given a source sequence. From

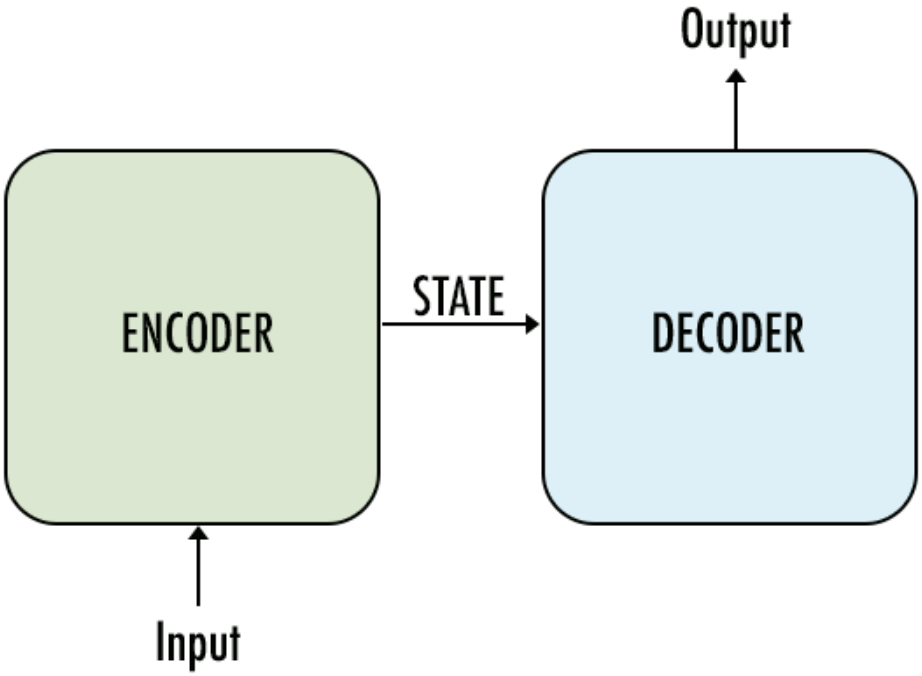

Figure 3.2 A simple representation of encoder decoder architecture.

the above description of Encoder and Decoder, we can understand that the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems which can now be solved using such architecture.

3.3 RNN

The Recurrent Neural Network (RNN) [63, 74] is a natural generalization of feed-forward neural networks to sequences. It is an artificial neural network which processes behaviour of dynamic temporal sequences using an internal state. RNN provides the ability to predict current state label based on previous states. It has a hidden state **h** and an optional output **y**. On a sequence of inputs ($x_1, \ldots, x_T$), at time *t*, the hidden state of RNN is updated by the equation:

$$h_t = f(h_{t-1}, x_t) \tag{3.1}$$

where, *f* is a non-linear activation function which can be logistic, sigmoid, or anything more complex

Given a sequence of inputs ($x_1, \ldots, x_T$), a standard RNN computes a sequence of outputs ($y_1, \ldots, y_T$) by iterating the following equation:

$$y_t = W^{yh} h_t \tag{3.2}$$

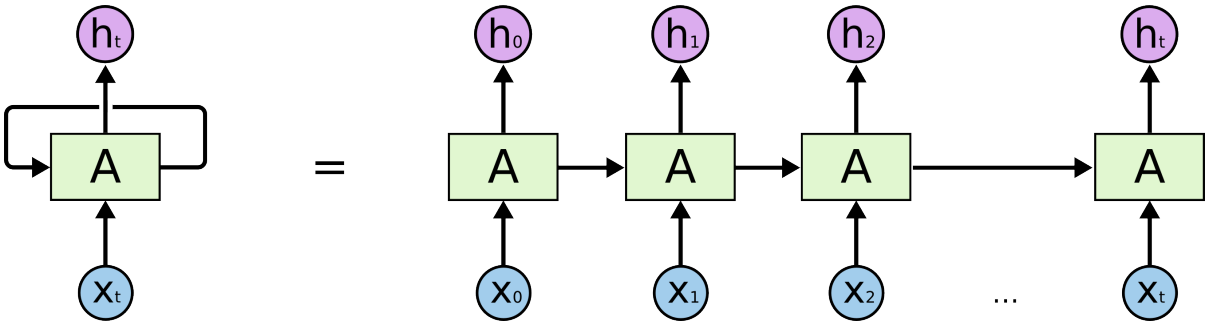

Figure 3.3 A Recurrent Neural Network, with a hidden state that is meant to carry pertinent information from one input item in the series to others.

The RNN can easily map sequences to sequences whenever the alignment between the inputs the outputs is known ahead of time. In fact, recurrent neural networks, long short-term memory networks [33], and gated recurrent neural networks [20] have become standard approaches in sequence modelling and transduction problems such as language modelling and machine translation.

RNNs struggle to cope with long-term dependency in the data due to vanishing gradient problem [32], especially when trying to learn long-term dependencies. Gates facilitate the learning of longer term temporal relationships [33]. Gating, as used in gated recurrent units (GRUs) [17] and long short-term memory (LSTM) networks [33], has become common-place in recurrent architectures. Furthermore, in the presence of noise in the input signal, gates can protect the cell state from undesired updates, thereby improving overall stability and convergence.

Thus, LSTM is an RNN which handles the vanishing gradient problem of traditional RNN, and is suitable for handling text sequences. GRU is a gating mechanism in RNNs which is like LSTM with a forget gate.

Chapter 4

Spell Checking

4.1 Introduction

Spelling correction is important for many of the potential Natural Language Processing (NLP) applications such as text summarization, sentiment analysis, machine translation [7]. Automatic spelling correction is crucial in search engines as spelling mistakes are very common in user-generated text. Many websites have a feature of automatically giving correct suggestions to the misspelled user queries in the form of *Did you mean?* suggestions or automatic corrections. Providing suggestions makes it convenient for users to accept a proposed correction without retyping or correcting the query manually. This task is approached by collecting similar intent queries from user logs [31, 75, 2]. The training data is automatically extracted from event logs where users re-issue their search queries with potentially corrected spelling within the same session. Example query pairs are *(house lone, house loan), (ello world, hello world), (mobilephone, mobile phone)*. Thus, large amounts of data is collected and models are trained using techniques like Machine Learning, Statistical Machine Translation etc.

The task of spelling correction is challenging for resource-scarce languages. In this chapter, we consider Indian languages, Hindi and Telugu, because of their resource scarcity. Due to lesser query share, we do not find the same level of parallel alteration data from logs. We also do not have many language resources such as Parts of Speech (POS) Taggers, Parsers etc. to linguistically analyze and understand these queries.

Due to lack of relevant data, we create synthetic dataset using highly probable spelling errors and real world errors in Hindi and Telugu given by language experts. Similarly, synthetic dataset can be created for any resource-scarce language incorporating the real world errors. Deep learning techniques have shown enormous success in sequence to sequence mapping tasks [71]. Most of the existing spell-checkers for Indian languages are implemented using rule-based techniques [42].

4.2 Our contributions

In this work, we approach the spelling correction problem for Indian languages with Deep learning. The proposed model can be employed for any resource-scarce language. We propose a character based Sequence-to-sequence text Correction Model for Indian Languages which trains end-to-end.

Our main contributions in this chapter are summarized as follows:

- We propose a character based recurrent sequence-to-sequence architecture with a Long Short Term Memory (LSTM) encoder and a LSTM decoder for spelling correction of Indian languages.
- We create synthetic datasets of noisy and correct word mappings for Hindi and Telugu by collecting highly probable spelling errors and inducing noise in clean corpus.
- We evaluate the performance of the proposed model by comparing with various approaches such as Statistical Machine Translation (SMT), rule-based methods, and various deep learning models, for this task.

4.3 Creation of Parallel Corpus

4.3.1 Data source

Due to lack of data with error patterns in Indian languages, we have built a synthetic dataset that the proposed model is trained on. Initially, we create data lists for Hindi and Telugu. For this, we have extracted a corpus of most frequent Hindi words[1] and most frequent Telugu words[2]. We have also extracted Hindi movie names and Telugu movie names of the movies released between the years 1930 and 2018 from Wikipedia which constitute phrases in the data lists. Thus, the Hindi and Telugu data lists consist of words and phrases consisting maximum of five words.

4.3.2 Protocol for data creation

For each data instance in the data list, multiple noisy words are generated by introducing error. The type of errors include insertion, deletion, substitution of one character, and word fusing. Spaces between words are randomly dropped in phrases to simulate the word fusing problem. The list of errors for Hindi and Telugu is created by collecting the highly committed spelling errors users make in each of these languages. We created this error list from linguistic resources and with help from language experts. The language experts analyzed Hindi and Telugu usage and listed the most probable errors. These errors are based on observations on real data and lexicon of Hindi and Telugu. Thus, the synthetic datasets are made as close as possible to real world user-generated data.

[1] https://ltrc.iiit.ac.in/download.php

[2] https://en.wiktionary.org/Frequency_lists/Telugu

Correct word	Noisy words
प्रेम कहानी (prem kahaanii)	प्रेमा कहानी (prema kahaanii) प्रेम कहनी (prem kahanii) प्रेमकहानी (premkahaanii) प्रेम कहानि (prem kahaani) प्रैम कहानी (praim kahaanii)

Table 4.1 Example of noisy words generation for a Hindi word along with their corresponding transliterations.

Language	High Frequency Words	Movie names	Size of parallel corpus
Hindi	15000	3021	108587
Telugu	10000	3689	92716

Table 4.2 Details of the synthetic datasets for Hindi and Telugu.

Table 4.1 shows the example of generation of noisy words corresponding to a correct word considering a Hindi word. Thus, the pairs of noisy word and original word constitute the parallel data for training. Table 4.2 gives the details about the size of the synthetic datasets for Hindi and Telugu.

4.4 Model Implementation

We have implemented the model architecture described in 3. All the code is written in Python using tf-seq2seq [13], a general-purpose RNN encoder-decoder framework for Tensorflow [1] deep learning library version 1.0.1. Both the encoder and the decoder are jointly trained end-to-end on the synthetic datasets we created. The model has a learning rate of 0.001, batch size of 100, sequence length of 50 (characters) and number of training steps 10,000. The size of the encoder LSTM cell is 256 with one layer. The size of the decoder LSTM cell is 256 with two layers. We use Adam optimization [38] for training the model. We apply Bahdanau attention [5] on the decoder. The character embedding dimension are fixed to 256 and the dropout rate to 0.8.

4.5 Experiments

We performed experiments with the proposed model and other models using synthetic datasets which we created for the Indian languages: Hindi and Telugu. Hindi is the most prominent Indian language

and the third most spoken language in the world. Telugu is the most widely spoken Dravidian language in the world and third most spoken native language in India.

We perform experiments on various models. The datasets are divided into train, dev, test partitions randomly in the ratio 80:10:10 respectively. In all our results, the models learn over the train partition, get tuned over the dev partition, and are evaluated over the test partition.

4.5.1 Baseline Methods

We train an SMT model using Moses [39] on Hindi and Telugu synthetic datasets. This is our main baseline model. The standard set of features is used, including a phrase model, length penalty, jump penalty and a language model. This SMT model is trained at the character-level. Hence, the system learns mappings between character-level phrases. Moses framework allows us to easily conduct experiments with several settings and compare with the proposed model.

Other baselines are character based sequence-to-sequence attention models with encoder-decoder as: CNN-GRU and GRU-GRU. All the models compared in this set of experiments look at batch sizes of 100 inputs and a maximum sequence size of 50 characters with a learning rate of 0.001 and run for 10000 steps. Throughout, the size of GRU cell is 256 with one layer. The CNN consists of 5 filters with sizes varying in the range of [2,3,4]. These multiple filters of particular widths produce the feature map, which is then concatenated and flattened for further processing.

4.6 Results and Analysis

Table 6.3 shows the accuracies reported by the proposed model and SMT methods. To check if Moses performs better on larger training data, we increased the size of Hindi synthetic dataset to 20567 and performed SMT. The accuracy value was 62% which is almost equivalent to the accuracy on original synthetic dataset. Our model outperforms Moses, an SMT technique by a margin. In Table 6.3, we find that the proposed model performs better than other sequence-to-sequence models with different convolutional and recurrent encoder-decoder combinations. These accuracies are on test set over the entire sequence. Thus, the results show that the proposed model performs better than all other baseline models. Our results support the conclusion by Britz et al. [14] that LSTMs outperform GRUs.

The rule-based spell-checker for Hindi, HINSPELL [68] reported an accuracy of 77.9% on a data of 870 misspelled words randomly collected from books, newspapers and people etc. The data used in [68] and the HINSPELL system are not publicly available. Hence, we implemented HINSPELL using Shabdanjali dictionary[3] consisting of 32952 Hindi word. This system when tested on our Hindi synthetic dataset, gave an accuracy is 72.3%. This accuracy being lower than the original HINSPELL

[3]https://ltrc.iiit.ac.in/download.php

accuracy can be accounted to larger size of testing data and inclusion of out-of-vocabulary words. Thus, our model outperforms HINSPELL by reporting an accuracy of 85.4%.

In Table 4.4, we have shown predictions given by the proposed model on few Hindi inputs. The results show that errors are contextually learned by the model. It can also be seen that the model has learned the word fusing problem. Also, the error detection step is not required separately as our model trains end-to-end and learns the presence of noise in the input based on context. The advantage of the proposed model over rule-based techniques is that it can handle out of vocabulary words as it is trained at character level. For example, the last entry in the Table 4.4, is the English word *bomb* spelled in Hindi. The model has learned it and corrected when misspelled in Hindi as *bombh*. Dictionary based techniques fail in such cases. The model fails in few cases when it corrects one character of the misspelled word instead of other like the example in the fourth row of 4.4.

Model	**Hindi (%)**	**Telugu (%)**
Moses (SMT)	62.8	64.7
CNN-GRU	74.4	79.7
GRU-GRU	77.6	84.3
LSTM-LSTM (Proposed model)	**85.4**	**89.3**

Table 4.3 Accuracy of spelling correction on Hindi and Telugu synthetic datasets given by Moses, character-based deep learning models (CNN-GRU and GRU-GRU), and the proposed model.

The slightly higher accuracy of our model on Telugu data than on Hindi data might be due to the fact that the highly probable spelling errors used in data creation are slightly less in number for Telugu when compared to Hindi. This can be handled for any language with more errors by increasing the size of dataset which includes enough data instances capturing each kind of error.

4.7 Summary

In this chapter, we proposed a model for automatic spelling correction in Indian languages which employs a recurrent sequence-to-sequence attention model to learn the spelling corrections from noisy and correct word pairs. We created a parallel corpus of noisy and correct spellings for training by introducing spelling errors in correct words. We validated the proposed model on these synthetic datasets

Input	Prediction	Correct Output
राजसि (raajasi)	राजसी (raajasii)	राजसी (raajasii)
दोआँखें (dhoaankhei)	दो आँखें (dho aankhei)	दो आँखें (dho aankhei)
अंस (ans)	अंश (ansh)	अंश (ansh)
दशावतर (dhashaavatar)	दशवतर (dhashavatar)	दशावतार (dhashaavataar)
बॉम्भ (baambh)	बॉम्ब (baamb)	बॉम्ब (baamb)

Table 4.4 Qualitative evaluation of predictions by the proposed model on few Hindi inputs along with expected outputs and corresponding transliterations.

created for Hindi and Telugu. We implemented spelling correction using Moses, as SMT system as a baseline model. We evaluated our system against existing techniques for Indian languages and showed favorable results.

The proposed model can be used in applications like search engines as we have shown that it automatically corrects the input text in Indian languages. Most of the deep learning models train on billions of data instances. On the contrary, our model trains on a dataset of less than a million parallel instances and gives competitive results. This shows that our approach can be used for automatic spelling correction of any resource-scarce language.

Chapter 5

Word Segmentation

5.1 Introduction

Word segmentation is a basic Natural Language Processing (NLP) task of parsing a sentence of concatenated text into words delimited with word boundaries. Often, the terms word splitting and word segmentation are interchangeably used in the field of NLP. Automatic word segmentation is one of the most important preprocessing steps in various applications dealing with noisy text. In speech recognition and optical character recognition (OCR), word boundaries are not often comprehensible. In tasks like speech-to-text, handwritten text-to-text, and Machine Translation, using word segmentation on the output will result in improved accuracy. In most of the NLP applications, tokenization, which is the primary step performed on data, occurs at word level. If the text is noisy and the words are concatenated, the error gets carried from the initial step of application. Initializing the process with word segmentation will parse the text into correct word tokens and will help in improving the performance of NLP applications.

Hindi is a language spoken primarily in Indian subcontinent. Being a ubiquitous language, it comes as no surprise that there are many NLP applications being developed in Hindi. In recent advances, various works have been done on user-generated text which is prone to typing errors. Hence, word splitting in Hindi is vital for these applications.

5.2 Our contributions

In this chapter, we present a technique for automatic word segmentation for Hindi using deep learning approach. For resource-scarce languages like Hindi, the available tools like Part of Speech (POS) taggers, parsers etc. are prone to some error. We do not make use of any such resources in our method which avoids the error carried by them. Our approach is completely data-dependent. Due to the unavailability of data with word splitting errors, we synthetically create a dataset by fusing words in the Hindi sentences corpus. The approach proposed can be implemented for word segmentation of any language given suitable data and model parameters.

Our main contributions in this chapter include,

- A character-based sequence-to-sequence model with Recurrent Neural Network (RNN) encoder and decoder for Hindi word segmentation
- A synthetic Hindi parallel dataset of sentences with word segmentation errors and their corresponding correct sentences
- Analytical evaluation of our model

5.3 Creation of Parallel Corpus

5.3.1 Data source

We have used the Hindi monolingual corpus[1] from English-Hindi parallel corpus [43]. This corpus has been created from various sources like Wikipedia, Judicial domain, Health domain etc. This dataset consists of 44 million Hindi sentences. Since this dataset was created by compilation of data from mostly official websites, it is presumed that the text follows norms of Hindi language and is not noisy. From this data, we used 200,000 sentences and created synthetic dataset for the model as explained in Subection 5.3.2. We divide the dataset into train, dev, test sets randomly in the ratio 70:15:15. Thus, the size of our training set is 140,000 sentences and that of dev and test sets is 30,000 sentences each.

5.3.2 Protocol for synthetic data creation

Hindi language is written in Devanagari script with space as the word boundary. We have created a synthetic parallel data for training the proposed seq2seq model. In this parallel data, the sources are Hindi sentences with some concatenated words and targets are the sentences properly spaced words. As an initial step we cleaned the data to remove all non-unicode characters, extra spaces, single word sentences, and punctuation. For every sentence in the data, we dropped 90% of the spaces at random positions to generate word segmentation errors. We have not dropped all the spaces in the sentences to preserve the space character in the vocabulary and to keep the dataset close to real-world errors. Table 5.1 shows the examples of how we create synthetic data from the Hindi dataset. Thus, the erroneous data and correct data constitute the parallel dataset for training.

5.4 Model Implementation

We implement an LSTM-LSTSM seq2seq model following the methodology described in 3. For implementing the model, we used the open source tf-seq2seq framework[2] [13] built over Tensorflow

[1] `http://www.cfilt.iitb.ac.in/iitb_parallel/`
[2] `https://github.com/google/seq2seq`

Original Sentence (target)	Generated sentence (source)
अच्छा लडका (achchaa ladkaa) (good boy)	अच्छालडका (achchaaladkaa) (goodboy)
क्या आपका बच्चा बीमार है (kyaa aapkaa bacchaa biimaar hai) (what your boy ill is)	क्या आपकाबच्चाबीमार है (kyaa aapkaabacchaabiimaar hai) (what yourboyill is)

Table 5.1 Examples of generation of word segmentation errors in Hindi sentences along with corresponding transliteration and translation of words.

[1]. All the code is written in Python. The encoder and decoder are jointly trained on our synthetic dataset.

Our model consists of a two layers bidirectional encoder (one layer in each direction), and a two layer decoder with Bahdanau [5] attention mechanism. We use 256-unit LSTM cells for both the encoder and decoder and apply dropout of 0.8 at the input of each cell. The model is trained with batch size of 100 and over 10,000 training steps. We train using the Adam optimizer [38] with epsilon of 0.0000008 and a fixed learning rate of 0.001 without decay. The embedding dimensionality is set to 256.

5.5 Results and Analysis

Gated Recurrent Units (GRU) are gating mechanism in RNNs which is like LSTM with a forget gate. We have also performed experiments with GRU as the neural network in encoder/decoder. Along with the proposed LSTM-LSTM model, we implemented character-based seq2seq models with encoder-decoder as GRU-GRU, GRU-LSTM, LSTM-GRU for evaluating the performance. Our results support the conclusion [13] that LSTMs perform better than GRUs.

For evaluation, we measure the precision, recall, and F1 score for word tokens between the predicted and expected output. Our model, LSTM-LSTM seq2seq, has the best F1 score of 0.93. A rule-based or a lexicon based approaches may perform better but they will not be able to handle out of dictionary words.

Table 5.2 shows the F1 scores obtained by training seq2seq models with different RNN combinations for encoder and decoder.

Model	P	R	F1
GRU-GRU	0.89	0.79	0.84
GRU-LSTM	0.9	0.87	0.88
LSTM-LSTM (Proposed model)	**0.96**	**0.9**	**0.93**

Table 5.2 Evaluation metrics: Precision (P), Recall (R) and F1 score (F1) on our synthetic Hindi dataset when trained on various Deep Learning models

Input	Prediction	Expected Output
कामहो गया (kaamho gayaa)	काम हो गया (kaam ho gayaa)	काम हो गया (kaam ho gayaa)
गाने बजरहेथे (gaane bajrahethe)	गाने बज रहे थे (gaane baj rahe the)	गाने बज रहे थे (gaane baj rahe the)
फानआन है (faanaan hai)	फान आन है (faan aan hai)	फान आन है (faan aan hai)
होदिलकेजहॉमें (hodilkijahaamei)	होदिल के जहॉ में (hodil ki jahaa mei)	हो दिल के जहॉ में (ho dil ki jahaa mei)

Table 5.3 Qualitative evaluation of predictions by our model on few Hindi inputs along with expected outputs and corresponding transliterations.

Table 5.3 shows a few examples of the output of our model. The model works even for sentences with few number of words like phrases which makes it applicable in search engines for automatically correcting search queries. In the third example, फान and आन are the transliterated Hindi words for the English words *fan* and *on* respectively. This example illustrates that the proposed model successfully segments the transliterated words. In the fourth example, the model segments the concatenated sentence into four words whereas the expected sentence has five words. Such errors can be handled and the model performance can be improved by increasing the training data and including more real word segmentation errors in the data.

5.6 Summary

In this chapter, we have presented a deep learning approach for automatic word segmentation for Hindi, an Indian language. This model architecture can be extended to any language provided appropriate data and model parameters. We have created synthetic parallel corpus for training the sequence-to-sequence model. The experimental results show that segmentation accuracy is primarily correlated with segmentation frequency as well as the order of occurrence of characters in the language. Even if the correlation between orthographic tokens and words is not perfect, using white space as delimiters is critical to word segmentation.

For space-delimited languages, high accuracy can be obtained even with relatively small training sets, while more training data is required to obtain high segmentation accuracy for languages without spaces. We apply a minimal number of language-specific settings to attain high segmentation accuracy. To the best of our knowledge, this is the first time word segmentation for an Indian language is approached using neural networks. Our model achieved an F1 score of 0.93.

Chapter 6

Grammar Checking

6.1 Introduction

In linguistics, grammar is the set of structural rules governing the composition of clauses, phrases, and words in a natural language. Grammar also refers to the study of such rules and this includes phonology, morphology, and syntax, often complemented by phonetics, semantics, and pragmatics. A grammar checker, in computing terms, is a program that attempts to verify written text for grammatical correctness. In Natural Language Processing (NLP), there have been many efforts towards developing grammar checking techniques for various languages. The task of grammar checking can broadly be divided into two steps: (i) Identifying if a sentence is grammatical or ungrammatical (ii) Suggesting the correction for ungrammatical sentences. In this work, we present a detailed taxonomy of grammatical errors in Hindi. Also, we propose a binary classification model to identify if a given Hindi sentence is grammatical or ungrammatical.

6.2 Our contributions

In this work, we approach the grammar checking problem for Hindi. The proposed methodology can be employed for any resource-scarce language.

Our main contributions in this work are summarized as follows:

- We present a detailed taxonomy of grammatical errors in Hindi that helped us in formulating rules in the creation of synthetic dataset.
- We create synthetic dataset of grammatical and ungrammatical sentences for Hindi by collecting grammatical errors and inducing noise in clean corpus.
- We create a dataset of pairs of ungrammatical and grammatical sentences.
- We propose a binary classifier for grammar checking of Hindi.
- We evaluate the performance of the proposed model by comparing with various other approaches.

6.2.1 Taxonomy of Hindi grammatical errors

The grammatical errors can be mainly classified into two categories - Morphological and Syntactical [40]. Following are the broad categories indicating the rules of Hindi language and the most probable error areas.

6.2.1.1 Morphological errors

Morphological errors are plausible in nouns, pronouns, adjectives, verbs, adverbs, particles, connectives, and interjections.

Postpositions: The postposition ने (*ne*) is used with subject noun phrases usually with transitive verbs in the past tense. The verb agrees with the object. The usage of *ne* follows some exceptions for particular verbs.

Example 6.1	मैं पत्र लिखा। (mei pathr likha)	**ungrammatical**
	मैंने पत्र लिखा। (mei***ne*** pathr likha)	**grammatical**

Example 6.2	वह कपड़े धोए। (***vah*** kapde dhoe)	**ungrammatical**
	उसने कपड़े धोए। (***usne*** kapde dhoe)	**grammatical**

The postposition को (*ko*) is used in different types of sentences and is placed after nouns. It is optional when used with object nouns which are followed by conjunct verbs with an adjective or adverb and the verb. It is not used with time and place adverbials.

Example 6.3	वह आज को आयेगा। (vah aaj ***ko*** aayega)	**ungrammatical**
	वह आज आयेगा। (vah aaj aayega)	**grammatical**

Example 6.4	वे ऊपर को पहुँचे। (ve uupar ***ko*** pahunche)	**ungrammatical**
	वे ऊपर पहुँचे। (ve uupar pahunche)	**grammatical**

Noun-noun compounds: Copulative compounds, also known as co-compounds, are composed of semantically-related nouns. Each noun behaves as an independent constituent in the sense that each

may be separately inflected for gender and number, though not for a postposition. Members of some compounds occur in a fixed order.

Example 6.5	पिता माता (pitha maatha)	**ungrammatical**
	माता पिता (maatha pitha)	**grammatical**

Example 6.6	नीच ऊँच (niich uunch)	**ungrammatical**
	ऊँच नीच (uunch niich)	**grammatical**

Particles: The particle भी (*bhi*) is used with different types of nouns in the direct or oblique case. It immediately follows a noun in the direct case and the postposition in the oblique case. *bhi* cannot be used between a noun and a postposition. It is also not used in vocative constructions. In the oblique form of the indefinite pronouns, the particle *bhi* is placed after the postpositions.

Example 6.7	घर भी में गर्मी है। (ghar ***bhi me*** garmi hai)	**ungrammatical**
	घर में भी गर्मी है। (ghar ***me bhi*** garmi hai)	**grammatical**

Example 6.8	सोहन भी आओ! (Sohan ***bhi*** aao!)	**ungrammatical**
	सोहन आओ! (Sohan aao!)	**grammatical**

Example 6.9	आप किसी भी को बुलाइए। (aap kisi ***bhi ko*** bulaaie)	**ungrammatical**
	आप किसी को भी बुलाइए। (aap kisi ***ko bhi*** bulaaie)	**grammatical**

Verbs: In Hindi, verbs carry a lot of information like gender, number, person, tense, aspect, and modality. There are two types of verbs: main and auxiliary. There are three types of main verbs: simple verbs, conjunct verbs, and compound verbs. A simple verb may consist of one main verb and person, gender, number, tense, and aspect markers. In the compound verb construction, the person, gender, number, and aspect markers are taken by the explicators/operators, and in the conjunct verbal construction they are taken by the verb element. The verb होना (*hona*) meaning *to be* is used as a copula in simple predicative sentences, as well as an auxiliary in different types of verbal constructions.

6.2.1.2 Syntactical errors

Syntactical errors occur in Hindi due to mistakes in structure of phrases, structure of clauses, and sentence construction.

Noun phrases: A noun phrase is defined as a nominal head preceded by one or more modifiers. It also serves as a nucleus of a postpositional phrase. It may function as a subject or object (indirect or direct) predicative complement or as a direct object of a postposition. A noun or a pronoun can be the minimum constituent of a noun phrase. There are certain co-occurrence restrictions. Indefinite determiners do not co-occur with ordinals. Similarly, the multiplicatives do not co-occur with collective or measure quantifiers. There are other usage constraints on modifiers. For example, the combination of indefinite determiners and cardinal quantifiers is possible; the combination of an indefinite determiner and a demonstrative pronoun in not allowed.

Example 6.10	कोई वह बच्चा यह काम नहीं कर सकता। (koi ***vah*** bachcha yah kaam nahi kar saktha)	**ungrammatical**
	कोई बच्चा यह काम नहीं कर सकता। (koi bachcha yah kaam nahi kar saktha)	**grammatical**

Postpositional phrases: A postpositional phrase is defined as a noun phrase followed by an oblique case marker and a postposition. Time adverbials take case markers as well as postpositions. The use of the direct forms of the time adverbials सवेरा (*savera*) meaning *morning* and शाम (*shaam*) meaning *evening* make them ungrammatical.

Example 6.11	अजित शाम काम कर्ता है। (Ajit shaam kaam kartha hai)	**ungrammatical**
	अजित शाम को काम कर्ता है। (Ajit shaam ***ko*** kaam kartha hai)	**grammatical**

Adjectival Phrases: There are two types of adjectives: those which do not take a complement, and those which do take a complement. Adjectives like मैला (dirty) do not take a complement, whereas adjectives like तैयार (ready) do take it. The latter type of adjectives with their complements occur attributively.

Example 6.12	कपड़े धोने के लिये तैयार लड़का। (kapde dhone ke liye ***thaiyaar ladka***)	**ungrammatical**
	कपड़े धोने के लिये लड़का तैयार है। (kapde dhone ke liye ***ladka thaiyaar hai***)	**grammatical**

Subordinate clauses: Subordinate clauses are of two types: finite and non-finite. Finite clauses normally have the same sentence structure as main clauses. Sometimes they may precede the main clause due to the consideration of focus.

Example 6.13	कि वह आयेगा मुझे आशा है। (***ki*** vah aayega mujhe aasha hai)	**ungrammatical**
	मुझे आशा है कि वह आयेगा। (mujhe aasha hai ***ki*** vah aayega)	**grammatical**

Relative Clauses: There are two types of relative clause constructions: finite and nonfinite participial relative clauses. The finite relative clauses maintain full sentence structures with subject verb agreement and are very common. Participial relative clauses exhibit the non-finite form of the verb. The former is more explicit than the latter. The former type is also labeled as the real relative clause

Direct	Singular	जो (*jo*)
	Plural	जो (*jo*)
Oblique	Singular	जिस (*jis*)
	Plural	जिन (*jin*)

Table 6.1 Relative markers in Hindi along with their English transliterations

Interrogative: In non-equational copular interrogative sentences, all the elements like the accompanier, locative, and time adverbial except the verb may be questioned. The copular verb cannot be deleted. But, it's often deleted in spoken informal language usage.

Example 6.14	किताब कहाँ? (kithaab kahaan?)	**ungrammatical**
	किताब कहाँ है? (kithaab kahaan ***hai***?)	**grammatical**

In equational copular interrogative sentences, either the subject noun phrase or the predicate nominal can be questioned. The demonstrative pronoun used as a subject cannot be questioned.

Example 6.15	यह पर्दा है (yah pardha hai)	**statement**
	क्या पर्दा है? (***kya*** pardha hai)	**ungrammatical question**
	यह क्या है? (yah ***kya*** hai)	**grammatical question**

Reflexives: Reflexivity is expressed by the use of agentive reflexive pronouns. This term is used to distinguish between the possessive reflexive *apna* and non-possessive reflexive *apne aap* meaning *self*. The reflexive *apne aap* represents the main reflexive pronoun, which when followed by a postposition, has the oblique form *apne*.

Masculine	Singular	अपना (*apna*)
	Plural	अपने (*apne*)
Feminine	Singular	अपनी (*apni*)
	Plural	अपनी (*apni*)

Table 6.2 Reflexives in Hindi along with their English transliterations

Example 6.16	मैं मेरा कमरा साफ कर रहा हूँ। (mei ***mera*** kamra saaf kar raha hu)	**ungrammatical**
	मैं अपना कमरा साफ कर रहा हूँ। (mei ***apna*** kamra saaf kar raha hu)	**grammatical**

Comparison: Comparison is usually expressed by sentential, phrasal, and morphological strategies. When two sentences are joined, the identical elements in the second conjunct are usually deleted. Whereas forward deletion is possible, backward deletion is not.

Example 6.17	अमित उतना नहीं है जितना उसका भाई चालाक है। (Amit uthna nahi hai ***jithna*** uska bhai chaalaak hai)	**ungrammatical**
	अमित उतना चालाक नहीं है जितना उसका भाई है। (Amit uthna chaalaak nahi hai ***jithna*** uska bhai hai)	**grammatical**

Co-ordination: Sentence coordination is marked mainly by the use of the conjunction morphemes और (*aur*) meaning *and*, या (*yaa*) meaning *or*, and magar magar/pr par/ikMtu kintu but. The unity of

the conjoined phrase cannot be distorted, and this unity is expressed only by coordination and not by accompaniment.

Example 6.18	बेटा घर और पिता आये। (beta ghar ***aur*** pitha aaye)	**ungrammatical**
	बेटा और घर पिता आये। (beta ***aur*** ghar pitha aaye)	**ungrammatical**
	बेटा और पिता घर आये। (beta ***aur*** pitha ghar aaye)	**grammatical**

6.3 Creation of Parallel Corpus

6.3.1 Data source

Kunchukuttan et al. [43] presented a English-Hindi parallel corpus[1] for machine translation. We use the Hindi monolingual corpus from this data which contains 44 million Hindi sentences from Wikipedia, BBC news, Health domain, Tourism domain, Judicial domain etc. We preprocess this data to remove the sentences containing any English words and sentences with less than 3 words. After preprocessing, we obtained a corpus of 14 million Hindi sentences.

6.3.2 Protocol for data creation

Using the grammatical errors mentioned in 6.2.1, we created a set of rules. These rules contain phrases which constitute the grammatical errors in the sentences and their corresponding corrections. We apply these rules on the preprocessed Hindi dataset and extract the ungrammatical sentences. Since the Hindi monolingual corpus was created by compilation of data from mostly official websites, we obtain a very small number of sentences (around 2500) which are ungrammatical. Then, we create two types of datasets:

- We randomly extract same number of grammatical sentences. Thus, we have a dataset of grammatical and ungrammatical sentences that can be used for training binary classification models.

- For every sentence in the extracted ungrammatical sentences, we check which grammar violation(s) among the set of violations is present in it. Then, we generate the corresponding grammatical sentence. Thus, we obtain a dataset of ungrammatical and grammatical sentence pairs. This data can be used for training sequence-to-sequence models.

[1] http://www.cfilt.iitb.ac.in/iitb_parallel/

6.4 Methodology

6.4.1 Model description

We propose a binary classification approach which predicts if a given Hindi sentence is grammatically correct or not. We represent each sentence as a vector of word n-grams and Part of Speech (POS) tag n-grams. We train our model on these feature vectors created for the sentences in the dataset. We performed experiments on different machine learning and deep learning models.

6.4.1.1 Sequence-to-sequence model

Similar to the proposed methodologies for spell checking and word segmentation, we attempted to train a seq2seq model on the dataset of ungrammatical and grammatical pairs we have created. We encountered two problems in this approach. Firstly, the dataset was small and was not sufficient for training a deep learning model. Second, unlike the models implemented for spell checking and word segmentation which were at character-level, the model for grammar checking is supposed to be trained at word-level to learn the morphological and syntactical errors. This resulted in a huge vocabulary (of words in the training data) size during seq2seq model training. The computational resources available for us were not sufficient to accommodate this data. These issues resulted in failure of seq2seq model training. In the scenario with bigger dataset and enough computational resources, this approach (Chapter 3) will give promising results for grammar checking.

6.4.2 Training details

We implement the Logistic Regression model [27] using *scikit-learn* [58] library in Python. The size of the dataset is around 5000 with equal number of grammatical and ungrammatical sentences. We generate word and POS tag n-gram feature vectors with n value ranging from 1 to 3. We use L_1 regularization to the model. L_1 regularized logistic regression is often used for feature selection, and has been shown to have good generalization performance in the presence of many irrelevant features [54].

6.5 Experiments

We have implemented some other machine learning models and a deep learning along with the proposed model and compared their performance. These included Support Vector Machine (SVM) [15], Stochastic Gradient descent (SGD) [12] Classifier, and Sequential model. We train these models over different types of input features: bag-of-ngrams of words, bag-of-ngrams of POS tags, and a combination of word and POS tag n-grams.

We use Sequential model [28] for this task. Sequential model is a linear stack of layers. It can be configured by adding any number of layers like *Dense, Dropout, Activation, Recurrent, Convolutional* etc. The model can also be configured with optimizer and loss function.

We implement a simple Sequential model with three layers: a Dense layer with 64 units and *relu* recurrent activation, a Dropout layer with dropout rate of 0.5, and another Dense layer with 1 unit.

- *Dense* layer is a regular densely-connected Neural Network (NN) layer. Dense layer implements the operation:

  $$output = activation(dot(input, kernel) + bias)$$

  where *activation* is the element-wise activation function passed as the activation argument, *kernel* is a weights matrix created by the layer, and *bias* is a bias vector created by the layer (only applicable if *use_bias* is True.

- *Dropout* layer applies dropout to the input. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

We perform experiments on various models. The datasets are divided into train, dev, test partitions randomly in the ratio 75:15:15 respectively. In all our results, the models learn over the train partition, get tuned over the dev partition, and are evaluated over the test partition.

6.6 Results and Analysis

In the table 6.3, we have shown the precision, recall, and F1 scores obtained in different experiments. The best results for each type of features are highlighted. It can be observed that the results by every model have improved when the features changed from word n-grams to POS tag n-grams to combination of both. Also, there is very less difference among performance metrics of different models.

In table 6.4, we present a few examples of prediction by our model. We observe that the model is able to identify morphological and syntactical incorrectness in the input. Also, in the third example, we have shown a Hindi sentence containing a transliterated word for English word *bomb* and the model predicts correctly. This shows that the proposed model is not dependent on a fixed Hindi lexicon and is tuned on the dataset it is trained on.

6.7 Summary

In this chapter, we proposed a binary classification model which identifies if a Hindi sentence is grammatical or ungrammatical. Through our experiments we show how taking Part of speech information into account improved the performance. We have also compared the performance of different binary

Model	Input	Precision	Recall	F1
Linear SVM	word n-grams	76.7	83.2	79.8
	POS n-grams	89.6	89.8	89.7
	(word+POS) n-grams	94.4	94	94.2
SGDClassfier	word n-grams	77.4	88.6	82.6
	POS n-grams	86.7	93.7	90.1
	(word+POS) n-grams	90.2	**94.8**	92.5
Sequential	word n-grams	75.2	77.6	76.4
	POS n-grams	98.1	83	89.9
	(word+POS) n-grams	**96.8**	91.4	94.1
Logistic Regression (Proposed model)	word n-grams	76.8	87.8	81.9
	POS n-grams	93	92.4	92.7
	(word+POS) n-grams	95	**94.8**	**94.9**

Table 6.3 Performance metrics reported by different models developed for grammar checking on Hindi dataset we have created.

Input	Prediction	Expected output
मैं आगे को बढ़ती रही। (mei aage **ko** badthi rahi)	ungrammatical	ungrammatical
मैं मेरे जीवन के लिये आभारी हूँ। (**mei mere** jiivan ke liye aabhaari hu)	ungrammatical	ungrammatical
उसने बॉम्ब लगाया। (usne bomb lagaaya)	grammatical	grammatical

Table 6.4 Qualitative evaluation of predictions by our model on few Hindi input sentences along with expected outputs and corresponding transliterations.

classifiers. Since real-world data for grammar checking in Hindi is scarcely available, we have proposed an approach for creation of synthetic dataset for this purpose. Also, we have presented a study on important and most frequent types of grammatical errors in Hindi along with examples.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we present, to the best of our knowledge, the first ever work for developing language processing tools for Hindi using Deep Learning. We explored the basic but crucial Natural Language Processing (NLP) problems namely Spell Checking, Word Segmentation, and Grammar Checking. We use the IIT-Bombay Hindi monolingual corpus to create the synthetic datasets for each of these tasks.

For approaching the problem of Spell Checking, we have created synthetic dataset. This is a parallel corpus containing pairs of misspelled and correct words. We have proposed a sequence-to-sequence (seq2seq) model which takes a misspelled word as input and outputs the corrected word. This proposed seq2seq model follows encoder-decoder architecture with LSTM as encoder and decoder. We evaluated the performance of this model against some baseline methods. To verify if this approach works for other languages, we created synthetic dataset for Telugu and implemented the same approach for Telugu. The model showed promising results for both Hindi and Telugu.

Similar to the Spell Checking approach, we proposed a seq2seq model for Word Segmentation in Hindi. We created a synthetic dataset containing pairs of noisy and correct sentences. Noisy sentences are the ones with incorrect word boundaries. We implement other baseline models and compare the performance of our model. We find that the LSTM-LSTM seq2seq model outperforms other approaches. This is a first attempt for word segmentation in Hindi from NLP perspective.

In this work, we also address the problem of Grammar Checking in Hindi. We present the types of grammatical errors in this language in detail. Using this theoretical information, we create a set of rules and identify ungrammatical sentences from Hindi monolingual corpus. From this, we create two types of datasets. One is a dataset of equal number of ungrammatical and grammatical sentences useful for binary classification. Other is a parallel corpus containing pairs of ungrammatical and their corresponding grammatical sentences. We propose a Sequential binary classifier which predicts if a

given Hindi sentence is ungrammatical or grammatical. We implement other models and evaluate the performance of the proposed model.

7.2 Future Work

The proposed approaches are initial attempts to develop deep learning models for Spell Checking, Word Segmentation, and Grammar Checking tasks. These approaches have obtained results that are competitive with the existing techniques. There is still a lot of scope to improve on the presented work.

The spell correction model presently deals with spelling corrections at word level. One potential improvement would be to change the training data from words and phrases to sentences. This will help in achieving context based spelling correction. The spelling correction model can be extended to a text correction and completion model. Changing the decoder from character-level to word-level will add the functionality of auto completion. This will improve the scope of the model in various applications. The word segmentation model can be extended to handle hyphenation, punctuation, acronyms and abbreviations, and other special cases. The grammar checking model can be extended to perform error correction along with error identification. The parallel data of ungrammatical and grammatical sentences presented in this work can be increased and used to train sequence-to-sequence model.

The synthetic datasets created for these tasks can be improved by collecting noisy data from different platforms like social media, blogs etc. and introducing these real world errors into clean corpus. This will improve the performance of the models on user-generated data. Further, for proper evaluation of the model, the models should be tested on real world user generated parallel data. There is a lack of availability of such data for Indian languages. Hence, there is a necessity to extract this kind of data.

The presented methodologies are applicable to any language. Hence, attempts should be made to create data and develop processing tools for all Indian languages. Given the present state of extensive research happening in the field of code-mixing of Indian languages and English, it would be an interesting extension to use the proposed methodology for code-mixed data. Further, a single pre-processing pipeline can be developed integrating spell checking, word segmentation, and grammar checking techniques.

"Correct language is the prerequisite for correct living."

- Socrates

"If language is not correct, then what is said is not what is meant; if what is said is not what is meant, then what must be done remains undone; if this remains undone, morals and art will deteriorate; if justice goes astray, the people will stand about in helpless confusion. Hence there must be no arbitrariness in what is said. This matters above everything."

- Confuscius

Appendix A

Experimental Settings and Model Parameters

In this section, we describe in detail the experimental settings and model parameters used for our experiments. All the codebase is developed in Python.

A.1 Sequence-to-sequence models

For training the seq2seq models, we have used *tf-seq2seq* [14] package. We applied a bidirectional RNN encoder with LSTM cell and attention decoder with LSTM cell. We used *Bahdanau* attention layer and *Adam* optimizer. We trained the models on Graphic Processing Units (GPUs) provided by IIIT-H. These are Nvidia GeForce GTX 1080 Ti GPUs which provide 14336 CUDA cores, and 44 GB of GDDR5X VRAM.

A.2 Logistic Regression

For grammar checking, we have implemented a logistic regression model for binary classification. We used *scikit-learn*[1] for this purpose. We experimented with different parameter values. We apply a L_1 regularization to the model.

A.3 Sequential model

For grammar checking, we trained a *Sequential* model which is a linear stack of layers. For the implementation of the Sequential model, we use Keras [19]. Keras is a high-level neural networks API, written in Python which was developed with a focus on enabling fast experimentation. The Sequential model is trained on the dataset we created. We used Root Mean Square Propagation (RMSprop) optimizer [73] on our model. Our model is trained with loss function as Binary cross entropy [53]. Keras provides the functionality to add *Dense, Dropout, and Activation* layers to the model. For the im-

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

plemeted Sequential model, we added a *Dense* layer with *relu* activation, a *Dropout* layer with dropout rate of 0.5, and another *Dense* layer. We used *RMSProp* optimizer and *binary cross entropy* loss.

A.4 Moses

We trained *Moses*[2], a Statistical Machine Translation (SMT) model for the evaluation of proposed seq2seq models. Moses requires *GIZA++* [56] for word alignment of the parallel corpus. The standard set of features is used, including a phrase model, length penalty, jump penalty and a language model. This SMT model is trained at the character-level. Hence, the system learns mappings between character-level phrases.

A.5 SVM and SGDClassifier

For the evaluation of grammar checking model, we implemented supervised learning models - Support Vector Machine (SVM) and Stochastic Gradient Descent (SGD) Classifier. We used *scikit-learn* [59] library. We experimented linear and Radial Basis Function (RBF) kernels in SVM. For SGDClassfier, we used *hinge* loss and *L2* regularization which is a penalty added to the loss function that shrinks model parameters towards the zero vector.

A.6 Data preprocessing tools

In spelling correction, for applying the rules to create misspellings from correct spellings, we converted the data from unicode to WX notation[3] (transliteration scheme for representing Indian languages in ASCII). This allows easy alteration of the spellings. For this purpose, we have used Indic Converter tool-kit[4]. For cleaning the corpus, we use tokenizer. In grammar checking, for feature creation we extract Part of Speech (POS) tags of the data. For this, we have used Natural language tool-kit for Indian languages[5].

[2] http://www.statmt.org/moses/
[3] http://en.wikipedia.org/wiki/WX_notation
[4] https://github.com/irshadbhat/python-converter-indic
[5] https://bitbucket.org/iscnlp

Related Publications

1. Pravallika Etoori, Manoj Chinnakotla, and Radhika Mamidi.
   Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning.
   Proceedings of ACL 2018, Student Research Workshop at Melbourne, Australia.

2. Pravallika Etoori and Radhika Mamidi.
   A Deep Learning Approach for Hindi Word Segmentation.
   Proceedings of 20th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2019) at La Rochelle, France.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

[2] Farooq Ahmad and Grzegorz Kondrak. 2005. Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 955–962. Association for Computational Linguistics.

[3] Md Alam, Naushad UzZaman, Mumit Khan, et al. 2007. N-gram based statistical grammar checker for bangla and english.

[4] Ambili, Panchami KS, and Neethu Subash. 2016. Automatic error detection and correction in malayalam. *International Journal of Science Technology and Engineering(IJSTE)*, 3(2).

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[6] Akanksha Bansal, Esha Banerjee, and Girish Nath Jha. 2013. Corpora creation for indian language technologies–the ilci project. In *the sixth Proceedings of Language Technology Conference (LTC 13)*.

[7] Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*.

[8] Zeeshan Bhatti, Imdad Ali Ismaili, Waseem Javaid Soomro, and Dil Nawaz Hakro. 2014. Word segmentation model for sindhi text. *American Journal of Computing Research Repository*, 2(1):1–7.

[9] Nivedita S Bhirud, RP Bhavsar, and BV Pawar. Asurvey of grammar checkers for natural languages. *Computer Science & Information Technology*, page 51.

[10] Ms Vaishali G Bhujade and MCM Meshram. 2014. A technique for segmentation of handwritten hindi text. *Int. J. Eng. Res. Technol*, 3:1491–1495.

[11] Lata Bopche and Gauri Dhopavakar. 2012. Rule based grammar checking system for hindi. *Journal of Information Systems and Communication*, 3(1):45.

[12] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[13] D. Britz, A. Goldie, T. Luong, and Q. Le. 2017. Massive Exploration of Neural Machine Translation Architectures. *ArXiv e-prints*.

[14] Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

[15] Christopher JC Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167.

[16] Flora Ramírez Bustamante and Fernando Sánchez León. 1996. Gramcheck: A grammar and style checker. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 175–181. Association for Computational Linguistics.

[17] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[18] Shamil Chollampatt and Hwee Tou Ng. 2018. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[19] François Chollet et al. 2015. Keras. `https://keras.io`.

[20] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[21] Bernard Comrie. 2009. *The world's major languages*. Routledge.

[22] Veena Dixit, Satish Dethe, and Rushikesh K Joshi. 2005. Design and implementation of a morphology-based spellchecker for marathi, an indian language. *ARCHIVES OF CONTROL SCIENCE*, 15(3):301.

[23] Timothy Gambell and Charles Yang. 2006. Word segmentation: Quick but not dirty. *Unpublished manuscript*.

[24] Naresh Kumar Garg, Lakhwinder Kaur, and MK Jindal. 2010. Segmentation of handwritten hindi text. *International Journal of Computer Applications*, 1(4):22–26.

[25] Shaona Ghosh and Per Ola Kristensson. 2017. Neural networks for text correction and completion in keyboard decoding. *arXiv preprint arXiv:1709.06429*.

[26] Mandeep Singh Gill, Gurpreet Singh Lehal, and Shiv Sharma Joshi. 2008. A punjabi grammar checker. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*.

[27] Joshua Goodman. 2004. Exponential priors for maximum entropy models. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*.

[28] Antonio Gulli and Sujit Pal. 2017. *Deep Learning with Keras*. Packt Publishing Ltd.

[29] Neha Gupta and Pratistha Mathur. 2012. Spell checking techniques in nlp: a survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(12).

[30] M Hanmandlu and Pooja Agrawal. 2005. A structural approach for segmentation of handwritten hindi text. In *International conference on cognition and recognition*, pages 589–597.

[31] Saša Hasan, Carmen Heger, and Saab Mansour. 2015. Spelling correction of user search queries through statistical machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 451–460.

[32] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

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

[34] Girish Nath Jha. 2003. Current trends in indian languages technology. *Language in India*, 3:12.

[35] Girish Nath Jha. 2010. The tdil program and the indian langauge corpora intitiative (ilci). In *LREC*.

[36] Vaishali Kalra and Rashmi Aggarwal. Importance of text data preprocessing & implementation in rapidminer.

[37] Wahab Khan, Ali Daud, Jamal A Nasir, and Tehmina Amjad. 2016. A survey on the state-of-the-art machine learning models in the context of nlp. *Kuwait journal of Science*, 43(4).

[38] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[39] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

[40] Omkar N Koul. 2008. *Modern Hindi Grammar*. Dunwoody Press Springfield, USA.

[41] Neethu S Kumar and LP Supriya. 2018. Survey on grammar checking and correction using deep learning for indian languages.

[42] Rakesh Kumar, Minu Bala, and Kumar Sourabh. 2018. A study of spell checking techniques for indian languages. *JK Research Journal in Mathematics and Computer Sciences*, 1(1).

[43] Anoop Kunchukuttan, Pratik Mehta, and Pushpak Bhattacharyya. 2017. The iit bombay english-hindi parallel corpus. *arXiv preprint arXiv:1710.02855*.

[44] Bibekananda Kundu, Sutanu Chakraborti, and Sanjay Kumar Choudhury. 2012. Combining confidence score and mal-rule filters for automatic creation of bangla error corpus: grammar checker perspective. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 462–477. Springer.

[45] Gurpreet Lehal. 2010. A word segmentation system for handling space omission problem in urdu script. In *Proceedings of the 1st Workshop on South and Southeast Asian Natural Language Processing*, pages 43–50.

[46] Gurpreet Singh Lehal. 2009. A two stage word segmentation system for handling space insertion problem in urdu script. *World Academy of Science, Engineering and Technology*, 60:321–324.

[47] Haibo Li, Masato Hagiwara, Qi Li, and Heng Ji. 2014. Comparison of the impact of word segmentation on name tagging for chinese and japanese. In *LREC*, pages 2532–2536.

[48] Christian Lovis, Pierre-André Michel, Robert Baud, and Jean-Raoul Scherrer. 1995. Word segmentation processing: a way to exponentially extend medical dictionaries. *Medinfo*, 8(pt 1):28–32.

[49] JA Mahar, H Shaikh, and GQ Memon. 2012. A model for sindhi text segmentation into word tokens. *Sindh University Research Journal-SURJ (Science Series)*, 44(1).

[50] Blossom Manchanda, Vijay Anant Athavale, and Sanjeev kumar Sharma. 2016. Various techniques used for grammar checking. *International Journal of Computer Applications & Information Technology*, 9(1):177.

[51] Marcin Miłkowski. 2012. Automating rule generation for grammar checkers. *arXiv preprint arXiv:1211.6887*.

[52] Misha Mittal, Dinesh Kumar, and Sanjeev Kumar Sharma. 2016. Grammar checker for asian languages: A survey. *International Journal of Computer Applications & Information Technology*, 9(1):163.

[53] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. 2014. Large-scale multi-label text classificationrevisiting neural networks. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer.

[54] Andrew Y Ng. 2004. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM.

[55] Peter Norvig. 2009. Natural language corpus data. *Beautiful data*, pages 219–242.

[56] Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

[57] Umesh Patil, Gerrit Kentner, Anja Gollrad, Frank Kügler, Caroline Féry, and Shravan Vasishth. 2008. Focus, word order and intonation in hindi. *Journal of South Asian Linguistics*, 1(1).

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[59] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

[60] Tarun Pruthi, Sameer Saksena, and Pradip K Das. 2000. Swaranjali: Isolated word recognition for hindi language using vq and hmm. In *international conference on multimedia processing and systems (ICMPS)*, pages 13–15.

[61] Uma Maheshwar Rao. 2011. Telugu spell-checker. *International Telugu Internet Conference Proceedings*.

[62] Alla Rozovskaya and Dan Roth. 2016. Grammatical error correction: Machine translation and classifiers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2205–2215.

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088):533.

[64] Keisuke Sakaguchi, Kevin Duh, Matt Post, and Benjamin Van Durme. 2017. Robsut wrod reocginiton via semi-character recurrent neural network. In *AAAI*, pages 3281–3287.

[65] Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. 2017. Grammatical error correction with neural reinforcement learning. *arXiv preprint arXiv:1707.00299*.

[66] Rajeev Sangal and Dipti Misra Sharma. 2004. Creating language resources for nlp in indian languages. In *Proc. Int. Conf. on Crossing the Digital Divide Shaping Technologies to Meet Human Needs*, pages 1–13.

[67] Amit Sharma and Pulkit Jain. 2013. Hindi spell checker. *Indian Institute of Technology Kanpur*.

[68] Harsharndeep Singh et al. 2015. Design and implementation of hinspell-hindi spell checker using hybrid approach. *International Journal of Scientific Research and Management*, 3(2).

[69] Madhvi Soni and Jitendra Singh Thakur. 2018. A systematic review of automated grammar checking in english language. *arXiv preprint arXiv:1804.00540*.

[70] Richard Sproat and Thomas Emerson. 2003. The first international chinese word segmentation bakeoff. In *Proceedings of the second SIGHAN workshop on Chinese language processing-Volume 17*, pages 133–143. Association for Computational Linguistics.

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

[72] Arda Tezcan, Véronique Hoste, and Lieve Macken. 2017. A neural network architecture for detecting grammatical errors in statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, 108(1):133–145.

[73] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[74] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[75] W John Wilbur, Won Kim, and Natalie Xie. 2006. Spelling correction in the pubmed search engine. *Information retrieval*, 9(5):543–564.

[76] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75.