Universal Dependency Parsing of Hindi-English Code-switching

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

MS
in
Computer Science and Engineering by Research

by

Irshad Ahmad Bhat
201407664
irshad.bhat@research.iiit.ac.in

Intrnational Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2018

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Universal Dependency Parsing of Hindi-English Code-switching" by Irshad Ahmad Bhat, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Manish Shrivastava

To My Parents

Acknowledgments

First and foremost, I would like to thank Prof. Manish Shrivastava for being an outstanding mentor. I am grateful for his guidance, and especially for providing me consistent feedback while allowing me enough freedom to grow as a researcher. It has been a great pleasure working with him. I look forward to more collaborations with him in the future. Besides my advisor, I would like to thank Prof. Dipti Misra Sharma. I appreciate all her contributions of time and ideas to make my MS experience productive and stimulating.

Moreover, I would like to thank my teachers (the unsung heroes), Prof. B. Yegnanarayana, Prof. C. V. Jawahar, Dr. Kannan Srinathan and Prof. Bapi Raju for instilling in me a passion for research and other life lessons.

I would also like to thank my colleagues who are (or were) part of the Language Technology group here at IIIT-Hyderabad: Riyaz Ahmad Bhat, Ratish Puduppully, Himanshu Sharma, Naman Jain, Maaz Nomani, Aniruddha Tammewar, Pruthwik Mishra, Himani Chaudhry, Pratibha Rani, Juhi Tandon, Silpa Kanneganti, Vandan Mujadia and others. Special thanks go to Vaishali Pal, Saket Maheshwary, Nayyar Ali, Shastri Vaishampayan, Chao Prakash Borparta, Prateek Saxena and many other graduate students for sharing the joys and sorrows of MS journey and being toghether during the tough periods!

I would like to thank my friends from my hometown who, every now and then, helped me take some time off from my MS schedule. Thank you Muzammil, Aameer, Atif, Aqib, Abid, Shakir, Khursheed and other friends for giving me some joyful moments which deflated the work pressure.

Most importantly, I would like to thank my family, especially my parents and my brothers for all their love and encouragement. This adventure was only possible due to their enormous support and trust in me. I sincerely thank them for their love and everyday prayers to God for my successful life.

Thank you very much, everyone!

Abstract

Code-switching is a phenomenon of mixing grammatical structures of two or more languages under varied social constraints. The code-switching data differ so radically from the benchmark corpora used in NLP community that the application of standard technologies to these data degrades their performance sharply. Unlike standard corpora, these data often need to go through additional processes such as language identification, normalization and/or back-transliteration for their efficient processing. In this thesis, we investigate these indispensable processes and other problems associated with syntactic parsing of code-switching data and propose methods to mitigate their effects. In particular, we study dependency parsing of code-switching data of Hindi and English multilingual speakers from Twitter. We present a treebank of Hindi-English code-switching tweets under Universal Dependencies scheme and propose domain adaptation techniques to efficiently leverage monolingual syntactic annotations and the annotations from the Hindi-English code-switching treebank. Firstly, we propose modifications to the parsing models which are trained only on the Hindi and English monolingual treebanks. We have shown that code-switching texts can be efficiently parsed by the monolingual parsing models if they are intelligently manipulated. Against an informed monolingual baseline, our parsing strategies are at-least 10 LAS points better. Secondly, we propose a neural stacking model for parsing that efficiently leverages part-of-speech tag and syntactic tree annotations in the code-switching treebank and the monolingual Hindi and English treebanks. We also present normalization and back-transliteration models with a decoding process tailored for code-switching data. Our neural stacking models achieve an accuracy of 90.53% for POS tagging and 80.23% UAS and 71.03% LAS for dependency parsing. Results show that our neural stacking parser is 1.5% LAS points better than the augmented parsing model and our decoding process improves results by 3.8% LAS points over the first-best normalization and/or back-transliteration.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Code Mixing vs Code Switching . . . 1
1.2 Matrix Language and Embedded Language . . . 3
1.3 Background Study . . . 3
1.4 Parsing Code-Switching Hindi-English: Challenges and Issues . . . 4
1.5 Contributions of the Thesis . . . 5
1.6 Related Publications . . . 6
1.7 Thesis Overview . . . 7

2 General Background . . . 9
2.1 Dependency Parsing . . . 9
2.2 Parsing framework . . . 11
2.2.1 Transition-based Dependency Parsing . . . 11
2.2.2 Oracle . . . 14
2.2.2.1 Dynamic Oracle Training . . . 15
2.2.3 Non-Projectivity . . . 15

3 Hindi-English Treebanking: Grammar Formalism and Annotation Procedure . . . 17
3.1 Introduction . . . 17
3.2 Universal Dependency Grammar . . . 18
3.2.1 Dependency Relations and Labels . . . 18
3.2.2 Annotation Procedure . . . 21
3.3 Summary . . . 23

4 Priliminary Tasks: Language Identification and Back-Transliteration/Normalization . . . 24
4.0.1 Language Identification . . . 24
4.0.1.1 Hyperparameters . . . 26
4.0.2 Normalization and Back-transliteration . . . 26
4.0.2.1 Hyperparameters . . . 29
4.0.2.2 Extraction of Transliteration Pairs . . . 30
4.1 Summary . . . 31

5 Adapting Monolingual Parsing Models to Code-switching Data . . . 32
5.1 Parsing Strategies . . . 32
5.1.1 Monolingual . . . 32

5.1.2 Multilingual . . . 33
5.1.3 Multipass . . . 34
5.2 Experimental Setup . . . 36
5.2.1 Parsing Models . . . 37
5.2.2 POS Models . . . 37
5.2.3 Word Representations . . . 38
5.3 Experiments and Results . . . 39
5.4 Conclusion . . . 40

6 Leveraging Monolingual and Code-switching treebanks using Neural Stacking . . . 42
6.1 Dependency Parsing . . . 42
6.1.1 Base Models . . . 42
6.1.2 Tagger network . . . 43
6.1.3 Parser Network . . . 44
6.1.4 Stacking Models . . . 45
6.2 Experiments . . . 47
6.2.1 Hyperparameters . . . 47
6.2.1.1 Word Representations . . . 47
6.2.1.2 Hidden dimensions . . . 48
6.2.1.3 Learning . . . 48
6.3 Results . . . 48
6.3.1 Pipeline vs Stack-prop . . . 50
6.3.2 Significance of normalization . . . 50
6.3.3 Monolingual vs Cross-lingual Embeddings . . . 51
6.4 Summary . . . 51

7 Summary and Future Work . . . 53
7.1 Conclusion . . . 53
7.2 Future Research Directions . . . 54

List of Figures

Figure Page

2.1 Dependency tree of Example sentence 3. 10
2.2 Transition sequence for Example sentence 3 based on Arc-eager algorithm. 13

3.1 Few examples trees our Hindi-English Code-Mixed dependency treebank. 22

4.1 Language identification network 25
4.2 Synthetic normalization pairs generated for a sample of English words using hand crafted rules. 27
4.3 The figure shows a 3-step decoding process for the sentence *"Yar cn anyone tel me k twitr account bnd ksy krty hn plz" (Friend can anyone tell me how to close twitter account please)*. 29
4.4 Devanagari to Roman character mapping table 31

5.1 An illustration of how a monolingual parser fails to correctly parse the fragment '*raat ki baarish*' which does not belong to the matrix language (English in this case). Red arcs represent incorrect attachments, where as, black arcs represent correct attachments. . . 33
5.2 Resolving structural ambiguity problem using a token-level language tag. 34
5.3 *First Pass*: Parse individual fragments using their respective parsing models. *Second Pass*: Parse the root nodes of the parsed fragments by the matrix language parsing model. 35
5.4 Example case of an imperfect segmentation 35
5.5 *First Pass*: Parse subordinate language first. *Second Pass*: Parse the roots of the subordinate fragments with the fragments of matrix language using the matrix language parser. 36
5.6 Example case of an imperfect segmentation 36

6.1 POS tagging and parsing network based on stack-propagation model proposed in [82]. 44
6.2 Code-switching tweet showing grammatical fragments from Hindi and English. 45
6.3 Neural Stacking-based parsing architecture for incorporating monolingual syntactic knowledge. 46

List of Tables

Table		Page
3.1	Universal dependency relations	19
3.2	UD lables with their meaning.	20
3.3	Statistics on training, testing and development sets used in all the experiments reported in this thesis.	22
4.1	Language Identification results on CS development set and test set.	26
4.2	Normalization accuracy based on the number of noisy tokens in the evaluation set. FB = First Best, and FW = Fragment Wise	29
5.1	POS Tagging accuracies for monolingual and multilingual models. LID = Language tag, G = Gold LID, A = Auto LID.	38
5.2	Accuracy of different parsing strategies on Code-switching as well as Hindi and English evaluation sets. Multipass$_{f\|s}$ = fragment-wise and subordinate-first parsing methods.	38
5.3	Parsing accuracies with exact search and k-best search (k = 5)	40
6.1	Accuracy of different parsing models on the evaluation set. POS tags are jointly predicted with parsing. LID = Language tag, TRN = Transliteration/normalization.	49
6.2	POS and parsing results for Hindi and English monolingual test sets using pipeline and stack-prop models.	49
6.3	POS tagging accuracies of different models on CS evaluation set. SP = stack-prop.	50
6.4	Accuracy of different parsing models on the test set using predicted language tags, normalized/back-transliterated words and predicted POS tags. POS tags are predicted separately before parsing. In Neural Stacking model, only parsing knowledge from the Bilingual model is transferred.	50
6.5	Impact of normalization and back-transliteration on POS tagging and parsing models.	51
6.6	Impact of monolingual and cross-lingual embeddings on stacking model performance.	51

Chapter 1

Introduction

Code-switching (henceforth CS) or code-mixing is the juxtaposition, within the same speech utterance, of grammatical units such as words, phrases, and clauses belonging to two or more different languages [27]. Code-switching is a sociolinguistic phenomenon, where multilingual speakers switch back and forth between two or more common languages or language varieties in a single utterance. The phenomenon is mostly prevalent in spoken language and in informal settings on social media such as in news groups, blogs, chat forums etc. and is often prompted by multiple social factors [47]. Moreover, code-switching is mostly prominent in colloquial language use in daily conversations, both online and offline.

Most of the benchmark corpora used in NLP for training and evaluation are based on edited monolingual texts which strictly adhere to the norms of a language related, for example, to orthography, morphology, and syntax. Social media data in general and CS data, in particular, deviate from these norms implicitly set forth by the choice of corpora used in the community. This is the reason why the current technologies often perform miserably on social media data, be it monolingual or mixed language data [11, 24, 40, 59, 69, 73]. CS data offers additional challenges over the monolingual social media data as the phenomenon of code-switching transforms the data in many ways, for example, by creating new lexical forms and syntactic structures by mixing morphology and syntax of two languages making it much more diverse than any monolingual corpora [11]. As the current computational models fail to cater to the complexities of CS data, there is often a need for dedicated techniques tailored to its specific characteristics. In this thesis, we investigate the indispensable processes of language identification, normalization and/or back-transliteration and other problems associated with syntactic parsing of code-switching data and propose methods to mitigate their effects. In particular, we study dependency parsing of Hindi-English code-switching data of multilingual Indian speakers from Twitter.

1.1 Code Mixing vs Code Switching

Code-switching is the use of two or more language varieties in the same conversation [10]. Hymes define code-switching as the use of more than one language by communicants in the execution of a

speech act [33]. He mentions that code-switching has become a common term for alternate use of two or more language, or language varieties, or even speech style. Fischer (1972) suggests that code-switching or inter-sentential code-alternation occurs when a bilingual speaker uses more than one language in a single utterance above the clause level to appropriately convey his/her intents. Few example of Hindi-English code-switching are given below:

(1) *RCB team ka wese hi budget khatam ho chuka hai . They need to win to buy team next year .*
RCB team of likewise DAT budget over DAT done is . They need to win to buy team next year .
RCB team's budget is over . They need to win to buy team next year .

(2) *What do you expect from the youth jin ko shaam khailne ke samay pe tution bhaij diya jata tha !*
What do you expect from the youth who DAT evening play of time on tution send give go was !
What do you expect from the youth who were sent to tuition at evening play time !

Code-mixing occurs when conversants use two or more languages together to the extent that they change from one language to the other in the course of a single utterance. Code mixing takes place without a change of topic and can involve various levels of language, e.g., morphology and lexical items [75]. The concept of code-mixing is used to refer to a more general form of language contact that may include cases of code-switching and the other form of contacts which emphasizes the lexical items. This definition is found in the following excerpt. Muysken [46] define the term code-mixing to refer to all cases where lexical item and grammatical features from two languages appear in one sentence. In code-mixing pieces of one language are used while a speaker is basically using another language [27] and we can see the borrowing elements of sentence from one language to other language [72]. Few example of Hindi-English code-mixing are given below:

(1) *First time Linkedin pe job offering ka message aaya hai .*
First time Linkedin on job offering of message come is .
For the first time, received a job offering message on Linkedin .

(2) *Warm up match kon si site pe live aa raha hai ?*
Warm up match kon si site pe live aa raha hai ?
Which site is live streaming the warm up match ?

From here onwards, we will not differentiate between intra- and inter-sentential mixing of languages and use the terms code-mixing and code-switching interchangeably throughout the thesis.

1.2 Matrix Language and Embedded Language

In a code-switching scenario, the distribution and functional usage of languages are often asymmetrical. The dominant language is called the Matrix Language, which governs the grammar of the CS utterance. Embedded Language fill slots of the matrix language grammar. Elements of Embedded Language are inserted into the morphosyntactic frame of the Matrix Language [47]. The point where there is a switch in the language is called a juncture or switch-point. The base structure of the language follows the grammatical structure of the matrix language. Consider the following Hindi-English CS sentences:

(1) *5k se zyada old currency deposit karne main explanation deni padti hai ?*
5k than more old currency deposit do in explanation give to has ?
To deposit more than 5k old currency, one has to give an explanation ?

(2) *What the hell is ' Halka ka lathi charge ' by police ?*
What the hell is ' light DAT baton charge ' by police ?
What the hell is a light lathi-charge by police ?

In the first sentence, the matrix framework and the construction of the sentence follows the structure of Hindi with English words inserted into it. Thus in the first example, Hindi is the Matrix Language and English is the Embedded Language. Where as, in case of second sentence, the matrix framework and the construction of the sentence follows the structure of English with Hindi words inserted into it. Thus in the second example English is the Matrix Language.

1.3 Background Study

Given the peculiar nature of CS data, it has been widely studied in linguistics literature [27, 47, 61]. The focus had been particularly on the structural (i.e., the grammatical constraints on CS) and functional (i.e, the motivation and intension behind CS) aspects of CS in various mediums, contexts, languages and geographies [3, 4, 48]. More recently, there has been a surge in studies concerning CS data in NLP as well [5, 7, 12, 29, 36, 62, 63, 67, 68, 68, 73, and others]. Barman et al. [5] presented an initial study on automatic language identification with Indian language code mixing from social media communication. Vyas et al. [73] describe their initial efforts to POS tag Hindi-English CS data while

trying to address the challenges of code-switching, transliteration and non-standard spelling, as well as lack of annotated data. The authors conclude that while CS is a common phenomenon in all multilingual societies, transliteration still remains an issue. Besides the individual computational works, a series of shared-tasks and workshops on preprocessing and shallow syntactic analysis of CS data have also been conducted at multiple venues such as Empirical Methods in NLP (EMNLP 2014 and 2016), International Conference on NLP (ICON 2015 and 2016) and Forum for Information Retrieval Evaluation (FIRE 2015 and 2016). Most of these works have attempted to address preliminary tasks such as language identification, normalization and/or back-transliteration as these data often need to go through these additional processes for their efficient processing.

1.4 Parsing Code-Switching Hindi-English: Challenges and Issues

- **Lexical Variation:** Due to mixing of two or more languages, CS data would be lexically more diverse than the individual monolingual data. More importantly, CS data would contain additional lexical forms unknown to the monolingual lexicons due to mixing of morphologies. It would lead to a high rate of out-of-vocabulary words unseen in the annotated data. Moreover, mixed lexicons would also mask the syntactic similarities between the languages.

- **Structural Diversity:** Hindi-English code-switching presents an interesting scenario for the parsing community. Mixing among typologically diverse languages would intensify structural variations which would make parsing more challenging. For example, there would be many sentences containing:

  - both SOV and SVO word orders[1],
  - both head-initial and head-final genitives,
  - both prepositional and postpositional phrases, etc.

  More importantly, none among the Hindi and English treebanks would provide any training instance for these mixed structures within individual sentences.

- **Back-transliteration/Normalization:** Due to colloquial nature of CS data, we would additionally need normalization of non-standard word forms and back-transliteration of Romanized Hindi

[1] Order of Subject, Object and Verb in transitive sentences.

words for addressing out-of-vocabulary problem, and lexical and syntactic ambiguity introduced due to contracted word forms.

- **Language Identification:** As we would train separate normalization and back-transliteration models for Hindi and English, we need language identification for selecting which model to use for inference for each word form separately. Moreover, we also need language information for decoding best word sequences (see Chapter 4) and using the monolingual parsing models.

1.5 Contributions of the Thesis

In this thesis, we present the first code-switching treebank that provides syntactic annotations required for parsing mixed-grammar syntactic structures. Moreover, we present a parsing pipeline designed explicitly for Hindi-English CS data. The pipeline comprises of several modules such as a language identification system, a back-transliteration system, and a dependency parser. The major contributions of the thesis can be summarized as:

1. **State-of-the-art Language Identification System:** A very accurate language identification system for code-switched Hindi-English texts which has a language identification f1-score of 98.27% and 97.36% on our tuning and evaluation sets respectively. The model is trained using multilayer perceptron (MLP) stacked on top of recurrent bidirectional LSTM (Bi-LSTM) network.

2. **State-of-the-art Transliteration Model:** Back-transliteration and normalization models based on encoder-decoder frameworks with sentence decoding tailored for code-switching data.

3. **First Code-Switching Treebank:** A dependency treebank of Hindi-English code-switching tweets under Universal Dependencies scheme that provides syntactic annotations required for parsing mixed-grammar syntactic structures. The treebank contains 1,900 POS and dependency annotated sentences.

4. **Neural Stacking Models:** A neural parsing model which learns POS tagging and parsing jointly and also incorporates knowledge from the monolingual treebanks using neural stacking. The model achieves an accuracy of 90.53% for POS tagging and 80.23% UAS and 71.03% LAS for dependency parsing on our evaluation set.

1.6 Related Publications

Major part of the work described in this thesis has previously been presented as the below listed publications. The total number of citations for these publications are 11 [source: Google scholar, January 2017[2]].

Conference Papers

1. **Irshad Ahmad Bhat**, Riyaz Ahmad Bhat, Manish Shrivastava and Dipti Misra Sharma. "Joining Hands: Exploiting Monolingual Treebanks for Parsing of Code-mixing Data." In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017).
2. **Irshad Ahmad Bhat**, Riyaz Ahmad Bhat, Manish Shrivastava and Dipti Misra Sharma. "Universal Dependency Parsing for Hindi-English Code-switching." Under review as a conference paper at NAACL 2018.

Other publications during my MS which are not part of this thesis, although relevant, are as follows:

Shared Tasks

3. **Irshad Ahmad Bhat**, Vandan Mujadia, Aniruddha Tammewar, Riyaz Ahmad Bhat and Manish Shrivastava. "IIIT-H System Submission for FIRE2014 Shared Task on Transliterated Search" In Proceedings of the Forum for Information Retrieval Evaluation (FIRE 2014).
4. **Irshad Ahmad Bhat**, Manish Shrivastava and Riyaz Ahmad Bhat. "Code Mixed Entity Extraction in Indian Languages using Neural Networks." In Proceedings of the Forum for Information Retrieval Evaluation (FIRE 2016).

Conference Papers

5. Riyaz Ahmad Bhat, **Irshad Ahmad Bhat** and Dipti Misra Sharma. "Leveraging Newswire Treebanks for Parsing Conversational Data with Argument Scrambling." In Proceedings of the 15th International Conference on Parsing Technologies (IWPT 2017).

[2] https://goo.gl/FtvWF8

6. Riyaz Ahmad Bhat, **Irshad Ahmad Bhat**, Naman Jain, and Dipti Misra Sharma. "A House United: Bridging the Script and Lexical Barrier between Hindi and Urdu." In Proceedings of the 26th International Conference on Computational Linguistics (COLING 2016).

Journal Papers

7. **Riyaz Ahmad Bhat**, Irshad Ahmad Bhat, and Dipti Misra Sharma. "Improving Transition-based Dependency Parsing of Hindi and Urdu by Modeling Syntactically Relevant Phenomena." In ACM Transactions on Asian and Low-Resource Language Information Processing (TALIP), 2016.

1.7 Thesis Overview

- **Chapter 2.** In this chapter, we provide the necessary background for the thesis, particularly focusing on different approaches to parsing and our choice of parsing paradigm and its formal description.

- **Chapter 3.** This chapter is devoted to data collection of Hindi-English code-switching tweets and the Universal Dependencies (UD) Grammar formalism that underlie the dependency annotation scheme we used for building the first code-switching treebank.

- **Chapter 4.** In this chapter, we present language identification and back-transliteration/normalization models. Language identifiaction and back-transliteration/normalization are the priliminary tasks for parsing CS data. These steps are indispensable for processing CS data and without them the parsing performance drops considerably.

- **Chapter 5.** In this chapter, we propose efficient and less resource-intensive strategies for parsing of CS data. These strategies are not constrained by in-domain annotations, rather they leverage pre-existing monolingual annotated resources for training.

- **Chapter 6.** In this chapter, we propose a neural stacking model for parsing that efficiently leverages part-of-speech tag and syntactic tree annotations in the code-switching treebank and the preexisting Hindi and English treebanks.

- **Chapter 7.** In this chapter, we provide the concluding remarks and outline directions for possible future research.

Chapter 2

General Background

A significant part of the research work presented in this thesis is based on the application of deterministic transition systems to dependency parsing of unrestricted natural language text. In this chapter, we provide the necessary background of dependency parsing, particularly covering the inner workings of a transition-based parsing system. Moreover, we also discuss different oracles that underlie the learning process of such a parsing system.

2.1 Dependency Parsing

Dependency parsing is an approach to automatic syntactic analysis of a natural language text based on dependency grammar [41]. The basic assumption that underlie a dependency grammar is that sentential structure primarily consists of words linked by *binary*, *asymmetrical* relations called dependency relations. A dependency relation holds between a pair of words in which one word called the *head* syntactically dominates the other called the *dependent*. Formally these dependencies are represented as: $X \xrightarrow{l} Y$, meaning "Y depends on X"; X is the head of Y, Y is a dependent of X and l encodes the type of dependency. Essentially, the goal of dependency parsing is to elucidate these binary word-level dependencies in a labeled dependency graph. Consider an input sentence as a string of words $W = w_0,...,w_n,\ n \geq 1$, where w_0 is a dummy ROOT symbol. A dependency tree for W is a labeled directed graph $T = (V, A)$, where $V = \{w_i \mid \in [0,\ n]\}$ is a set of words, and A is a set of labeled arcs (w_i, l, w_j). Arc $(w_i,\ l,\ w_j)$ encodes a labeled dependency $w_i \xrightarrow{l} w_j$, where l is a permissible dependency label from $L = \{l_i \mid \in [0,\ m]\}$. The arc direction is defined by the sign of inequality, if $j > i$ for $(w_i,\ w_j) \in A_w$, the arc is *right* directed, while it is $left$ directed otherwise. To state it precisely, dependency parsing tries to automatically construct a well-formed labeled dependency graph T for an

input sentence W. A dependency graph T is well-formed if it is **acyclic** and **connected** as the one shown in 2.1.

(3) *This raat ki baarish always scares me .*
This night of rain always scares me .
This rain of night always scares me .

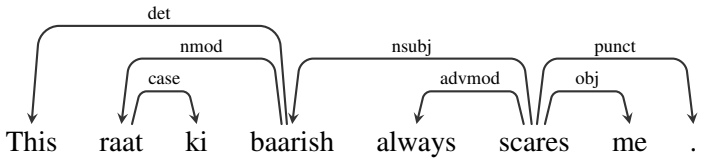


Figure 2.1: Dependency tree of Example sentence 3.

During the last two decades, a number of varied algorithms have been proposed for automatic dependency parsing of unrestricted natural language text. These algorithms are used in conjunction with machine learning techniques to learn accurate dependency parsers. Syntactically annotated corpora, called *treebanks*, are usually at the heart of these approaches, since they provide necessary information for learning accurate parsers in supervised machine learning settings. The approaches can be broadly categorized as *graph-based* and *transition-based*. Graph-based parsers use near exhaustive search over the graphical representation of a sentence to find a maximum scoring dependency graph, while transition-based parsers use a local greedy search to derive a dependency tree. Graph-based methods were first explored for dependency parsing by Eisner [23] who proposed a $O(n^3)$ parsing algorithm based on dynamic programming and a generative learning model. The transition based approach was first explored by Kudo and Matsumoto [42] for Japanese and Yamada and Matsumoto [76] for English. Both the methods have their strengths and weaknesses. While graph-based parsers are very accurate, they run at quadratic time for non-projective parsing using Chu-Liu-Edmonds algorithm and at cubic time for projective parsing with Eisner's algorithm [44]. On the other hand transition based parsers have linear time complexity but they are less accurate. However, recent advancements in transition based parsing have minimized the accuracy gap between the two approaches while compromising less on their efficiency [25, 26, 80, 81]. Next we discuss our choice of parsing paradigm and give its formal description.

2.2 Parsing framework

In this thesis we use transition-based dependency parsing paradigm [53] to experiment with parsing of Indian language texts. In the last decade, transition based parsers have gained popularity due to their efficiency. Transition-based greedy parsers allow us to carry out wide range of experiments in reasonable time with commodity hardware. Moreover an arc-eager system, which is the basis of our parser, nearly follows an incremental parsing strategy thus making it cognitively plausible as well [51].

2.2.1 Transition-based Dependency Parsing

Transition-based dependency parsing aims to predict a transition sequence from an initial configuration to some terminal configuration, which derives a target dependency parse tree for an input sentence. In data-driven settings such an optimal transition sequence is predicted using a classifier. Even though, quite advanced machine learning algorithms like neural networks and structured prediction algorithms [13, 79] have been used to train the classifier, it has been observed that even simple memory-based algorithms work well for the task [30].

In the last two decades, a number of incremental parsing algorithms have been proposed to parse natural language text. In this thesis, we restrict our choice to **arc-eager** system [50]. The **arc-eager** system is one of the most popular transition systems. It defines a set of configurations for a sentence w_1,...,w_n, where each configuration $C = (S, B, A)$ consists of a stack S, a buffer B, and a set of dependency arcs A. For each sentence, the parser starts with an initial configuration where S = [ROOT], $B = [w_1,...,w_n]$ and $A = \emptyset$ and terminates with a configuration C if the buffer is empty and the stack contains the ROOT. The parse trees derived from transition sequences are given by A. Denoting S_i and B_j as the i_{th} and j_{th} elements on the stack and buffer, the acr-eager system defines four types of transitions (t):

1. A LEFT-ARC(l) adds an arc $B_j \rightarrow S_i$ to A with label l, where S_i is the node on top of the stack and B_j is the first node in the buffer, and removes the node B_j from the stack. It has as a precondition that the token S_i is not the artificial root node 0 and does not already have a head.

2. A RIGHT-ARC(l) adds an arc $S_i \rightarrow B_j$ to A with label l, where S_i is the node on top of the stack and B_j is the first node in the buffer, and pushes the node B_j onto the stack.

3. The REDUCE transition removes the top node in the stack and is subject to the precondition that the node has a head.

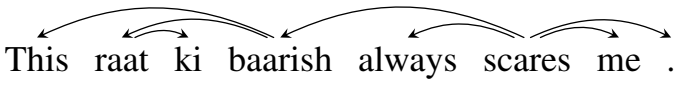
4. The SHIFT transition removes the first node in the buffer and pushes it onto the stack.

As an illustration of the arc-eager parsing algorithm, we derive the transition sequence for an example code-mixed sentence. The transition sequence is derived by the oracle presented in Algorithm 1 guided by the tree representation (G_{gold}) of the sentence shown in Figure 2.1. The algorithm derives $2n$-1 transitions for a sentence of length n[1]. The overall derivation process in shown in Figure 2.2.

if c = $(S|i,j|B, A)$ and $(j,l,i) \in A_{gold}$ **then**
    $t \leftarrow$ LEFT-ARC(l)
else if c = $(S|i,j|B, A)$ and $(i,l,j) \in A_{gold}$ **then**
    $t \leftarrow$ RIGHT-ARC(l)
else if c = $(S|i,j|B, A)$ and $\exists k[k < i \wedge \exists l[(k,l,j) \in A_{gold} \vee (j,l,k) \in A_{gold}]]$ **then**
    $t \leftarrow$ REDUCE
else
    $t \leftarrow$ SHIFT
return t

Algorithm 1: Standard Oracle for Arc-eager parsing algorithm adapted from Goldberg and Nivre [25].

[1]Including dummy ROOT node.



$\Downarrow \theta = f : D \rightarrow T$

Transition	Stack	Buffer	A
	[]	[This raat ki baarish always scares me]	$\emptyset$
SHIFT	[This]	[raat ki baarish always scares me]	
SHIFT	[This raat]	[ki baarish always scares me]	
RIGHT-ARC	[This raat ki]	[baarish always scares me]	$A \cup$ (raat, ki)
REDUCE	[This raat]	[baarish always scares me]	
LEFT-ARC	[This]	[baarish always scares me]	$A \cup$ (baarish, raat)
LEFT-ARC	[]	[baarish always scares me]	$A \cup$ (baarish, this)
SHIFT	[baarish]	[always scares me]	
SHIFT	[baarish always]	[scares me]	
LEFT-ARC	[baarish]	[scares me]	$A \cup$ (scares, always)
LEFT-ARC	[]	[scares me]	$A \cup$ (scares, baarish)
SHIFT	[scares]	[me]	
RIGHT-ARC	[scares me]	[]	$A \cup$ (scares, me)
REDUCE	[scares]	[]	

$\Uparrow \theta$

This raat ki baarish always scares me .

$\Downarrow$

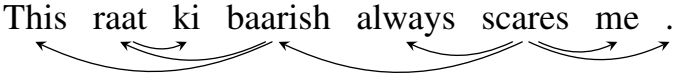


Figure 2.2: Transition sequence for Example sentence 3 based on Arc-eager algorithm.

2.2.2 Oracle

Transition-based parsers use an *oracle* to learn a sequence of actions from a gold-standard tree that they should take in order to derive it back. Until quite recently, these oracles were defined as *functions* from trees to transition sequences, mapping each gold-standard tree to a single sequence of actions, even if more than one sequence of actions can potentially derive them[2]. In the parsing literature, these oracles have been referred as *static-greedy oracles*. Goldberg and Nivre in their recent works [25, 26] have redefined these oracles as *relations* from configurations to transitions. These oracles, aptly called as *dynamic oracles*, allow the learner to choose dynamically from the transitions defined as optimal at a given parser configuration.

Algorithm 2 details the learning process of an arc-eager parser. Given a sentence s, the parser is initialized with the configuration c (line 2). Then, a feature function $\phi(c)$ represents the configuration c as a vector, which is fed to a scoring function *SCORE* assigning scores to (configuration, transition) pairs. *SCORE* scores the possible transitions t, and the highest scoring transition $\hat{t}$ is chosen (line 4). The transition $\hat{t}$ is applied to the configuration, resulting in a new parser configuration. The process ends when reaching a final configuration, from which the resulting parse tree is read and returned (line 6).

1: **Input:** sentence $s = w_1, \cdots, x_w, t_1, \cdots, t_n$, parameterized function $\text{SCORE}_\theta(.)$ with parameters θ
2: $c \leftarrow$ INITIAL(s)
3: **while not** TERMINAL(c) **do**
4: $\quad \hat{t} \leftarrow \text{argmax}_{t \in \text{LEGAL}(c)} \text{SCORE}_\theta(\phi(c), t)$
5: $\quad c \leftarrow \hat{t}(c)$
6: **return** *tree(c)*

Algorithm 2: The parsing algorithm for the transition-based parser.

We use a Multi Layer Perceptron (MLP) for training the oracle. The training objective is to set the score of correct transitions above the scores of incorrect transitions. We use a margin-based objec-

[2]This type of ambiguity is defined as spurious ambiguity.

[2]p regulates the percentage of non-optimal transitions to be explored, while k ensures that the model is in a good region of the parameter space.

tive, aiming to maximize the margin between the highest scoring correct action and the highest scoring incorrect action. The hinge loss at each parsing configuration c is defined as:

$$\max\left(0, 1 - \max_{t_o \in G} \text{MLP}\left(\phi(c)\right)[t_o] + \max_{t_p \in A \setminus G} \text{MLP}\left(\phi(c)\right)[t_p]\right)$$

where A is the set of possible transitions and G is the set of correct (gold) transitions at the current stage. At each stage of the training process the parser scores the possible transitions A, incurs a loss, selects a transition to follow, and moves to the next configuration based on it. The local losses are summed throughout the parsing process of a sentence, and the parameters are updated with respect to the sum of the losses at sentence boundaries. The gradients of the entire network with respect to the sum of the losses are calculated using the backpropagation algorithm. As usual, we perform several training iterations over the training corpus, shuffling the order of sentences in each iteration.

2.2.2.1 Dynamic Oracle Training

We follow Goldberg and Nivre [25, 26] in using error exploration training with a dynamic-oracle. At each stage in the training process, the parser assigns scores to all the possible transitions $t \in A$. It then selects a transition, applies it, and moves to the next step. Which transition should be followed? A common approach follows the highest scoring transition that can lead to the gold tree. However, when training in this way the parser sees only configurations that result from following correct actions, and as a result tends to suffer from error propagation at test time. Instead, in error-exploration training the parser follows the highest scoring action in A during training even if this action is incorrect, exposing it to configurations that result from erroneous decisions. This strategy requires defining the set G such that the correct actions to take are well-defined also for states that cannot lead to the gold tree. Such a set G is called a dynamic oracle.

2.2.3 Non-Projectivity

If we clearly observe the arc-eager oracle algorithm 1, we may note that arc-eager system is restricted to projective trees or in simple words, disallows crossing of arcs. For example, line **5** of the algorithm clearly prohibits S_0 to have dependents in the buffer. As Nivre [52] remarks, natural languages approve grammatical constructs that violate the condition of projectivity. In those languages where non-projectivity is a common scene, one may use the arc-eager system with a caveat that non-

projective arcs will never be parsed correctly. In case of Hindi, for example, we may loose $\geq 2\%$ arcs which are non-projective.

Given such huge loss of accuracy in Hindi and other Indian languages, we need a workaround to tackle non-projective structures in our arc-eager parser. As a possible solution, we use the pseudo-projective transformations of Nivre and Nilsson [55]. The fact that dependency trees are labeled, we can transform the non-projective arcs while preserving the lift information in their dependency labels. At parsing time, inverse transformation based on breadth-first search can be applied to recover the non-projective arcs efficiently. There is, however, a trade-off between the parsing accuracy and parsing time as these transformations can increase the cardinality of the label set by a factor of *n-square*[3]. Nevertheless, we will use the encoding schemes proposed by Nivre and Nilsson and explore and evaluate them for different Indian languages.

Concluding Remarks. As a concluding remark, all the parsing experiments in this thesis are carried using our implementation of arc-eager system with dynamic oracle. We use Bi-LSTM to learn the feature representations and MLP to train the parser and use pseudo-projective transformations to handle non-projectivity. Our implementation including the state-of-the-art models will be made available for download from the authors web page.

[3] At least in case of the most informative encoding scheme i.e., $head + path$. n is cardinality of the original label set.

Chapter 3

Hindi-English Treebanking: Grammar Formalism and Annotation Procedure

This chapter is devoted to Universal Dependency (UD) Grammar formalism that underlie the dependency annotation scheme used for code-switching Hindi-English treebanking. Besides, we also discuss the annotation procedure adapted to build treebanks based on the formalism.

3.1 Introduction

The need for manually annotated linguistic resources is widely acknowledged in the field of computational linguistics (CL) and natural language processing (NLP). In the NLP-CL research community, a great deal of effort has been put into the creation of these linguistic resources due to the reliance of basic as well as advanced NLP applications on manual annotations. Specifically, syntactic treebanking projects have generated a lot of interest in the community due to their manifold usage. A syntactic treebank is, by definition, a set of syntactic trees capturing the syntactic or semantic structure of sentences. Creation of these treebanks has interested both linguists and computational linguists. For the former, they provide insights about the linguistic theory they have been built upon, and the later use them for the development of data driven parsers.

Treebanking efforts for languages like English and Czech started in the last decade of 20^{th} century. The interest in Indian language treebanking started of late with the development of a pilot treebank for Hindi [6], which later culminated in a multi-layered and multi-representational treebanking project for Hindi and Urdu [8].

To the best of our knowledge, there is no available code-switching Hindi-English data set that contains dependency annotations. There are, however, a few available code-switching data sets that provide

annotations related to language of a token, its POS and chunk tags. In what follows, we discuss the UD grammar formalism, the annotation scheme based on UD grammar and the annotation procedure adapted for treebanking of Hindi-English.

3.2 Universal Dependency Grammar

Universal Dependencies provides a set of multilingual treebanks with cross-lingually consistent dependency-based lexicalist annotations, designed to aid development and evaluation for cross-lingual systems, such as multilingual parsers [57]. The current version of Universal Dependencies comprises not only major treebanks for 47 languages but also their siblings for domain-specific corpora and dialects. The annotation scheme is based on an evolution of (universal) Stanford dependencies [16, 17, 18], Google universal part-of-speech tags [60], and the Interset interlingua for morphosyntactic tagsets [77]. The general idea is to provide a universal inventory of categories and guidelines to facilitate consistent annotation of similar constructions across languages, while allowing language-specific extensions when necessary.

3.2.1 Dependency Relations and Labels

Table 3.1 lists the 37 universal syntactic relations used in UD v2. It is a revised version of the relations originally described in Universal Stanford Dependencies: A cross-linguistic typology [18]. The definitions of these relations are given in Table 3.2.

	Nominls	**Clauses**	**Modifier words**	**Function words**
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

Table 3.1: Universal dependency relations

The upper part of the table follows the main organizing principles of the UD taxonomy:

- Rows correspond to functional categories in relation to the head:
    - Core arguments of clausal predicates
    - Non-core dependents of clausal predicates
    - Dependents of nominals
- Columns correspond to structural categories of the dependent:
    - Nominals
    - Clauses
    - Modifier words

- Function words

The lower part of the table lists relations that are not dependency relations in the narrow sense:

- Relations used to analyze coordination
- Relations used to analyze multiword expressions (MWE)
- Loose joining relations
- Special relations for ellipsis, disfluencies, and orthographic errors
- Special relations for clausal heads, punctuation and other relations

Relation	**Meaning**	**Relation**	**Meaning**
advcl	adverbial clause modifier	fixed	fixed multiword expression
advmod	adverbial modifier	flat	flat multiword expression
amod	adjectival modifier	goeswith	goes with
appos	appositional modifier	iobj	indirect object
aux	auxiliary	list	list
case	case marking	mark	marker
cc	coordinating conjunction	nmod	nominal modifier
ccomp	clausal complement	nsubj	nominal subject
clf	classifier	nummod	numeric modifier
compound	compound	obj	object
conj	conjunct	obl	oblique nominal
cop	copula	orphan	orphan
csubj	clausal subject	parataxis	parataxis
dep	unspecified dependency	punct	punctuation
det	determiner	reparandum	overridden disfluency
discourse	discourse element	root	root
dislocated	dislocated elements	vocative	vocative
expl	expletive	xcomp	open clausal complement

Table 3.2: UD lables with their meaning.

3.2.2 Annotation Procedure

We manually annotated a data set of Hindi-English code-switching tweets with dependency structures. The code-switching tweets were sampled from a large set of tweets of Indian language users that we crawled from Twitter using Tweepy[1]–a Twitter API wrapper. We then used a language identification system trained on ICON dataset (see Section 4) to filter Hindi-English CS tweets from the crawled Twitter data. Only those tweets were selected that satisfied a minimum ratio of 30:70(%) code-switching. From this data set, we manually selected 1,900 tweets for annotation. The selected tweets are thoroughly checked for code-switching ratio. For POS tagging and dependency annotation, we used Version 2 of Universal dependency guidelines [18], while language tags are assigned based on the tagset defined in [34, 70].

The dataset was annotated by two expert annotators who have been associated with annotation projects involving syntactic annotations for around 10 years. Nonetheless, we also ensured the quality of the manual annotations by carrying an inter-annotator agreement analysis. We randomly selected a dataset of 150 tweets which were annotated by both annotators for both POS tagging and dependency structures. The inter-annotator agreement has a 96.20% accuracy for POS tagging and a 95.94% UAS and a 92.65% LAS for dependency parsing.

We split the treebank data with a ratio of 80:10:10 for training, testing and tuning the parsers. The statistics of the treebank are provided in Table 3.3. Figure 3.1 show some of the dependency trees from our Hindi-English dependency treebank.

[1] http://www.tweepy.org/

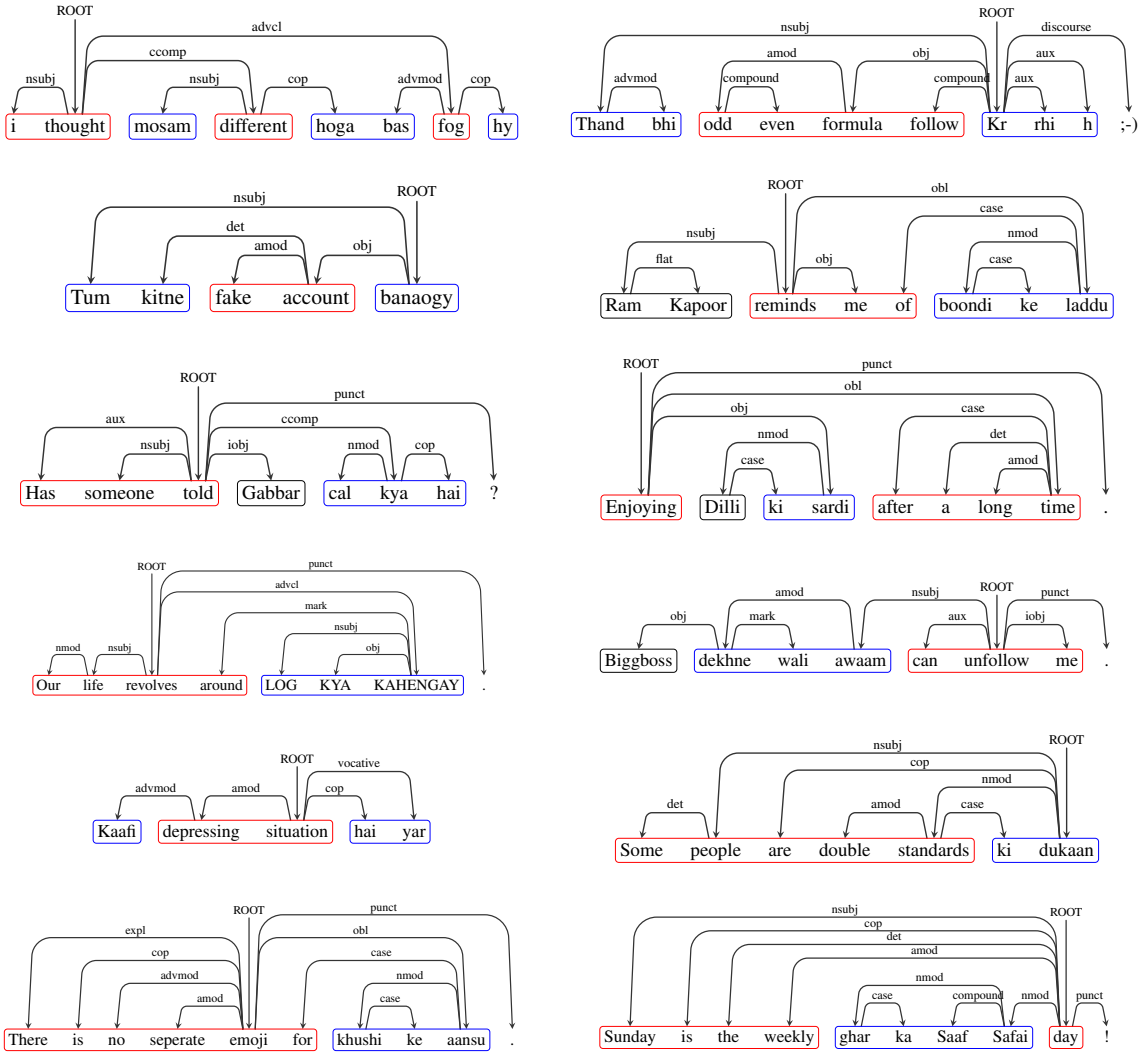


Figure 3.1: Few examples trees our Hindi-English Code-Mixed dependency treebank.

Data-set	Sentence Count	Token Count	Tag Counts				
			Hi	En	Ne	Univ	Acro
Train	1,448	20,203	8,363	8,270	698	2,730	142
Dev	225	3,411	1,549	1,300	151	379	32
Test	225	3,295	1,465	1,283	168	349	30

Table 3.3: Statistics on training, testing and development sets used in all the experiments reported in this thesis.

3.3 Summary

In this chapter, we discussed the annotation procedure and the grammar formalism used to build dependency treebank for Hindi-English code-switching texts.

Chapter 4

Priliminary Tasks: Language Identification and Back-Transliteration/Normalization

As preliminary steps before parsing of CS data, we need to identify the language of tokens and normalize and/or back-transliterate them to enhance the parsing performance. These steps are indispensable for processing CS data and without them the performance drops drastically as we would see in Results Section of Chapter 5 and 6. We need normalization of non-standard word forms and back-transliteration of Romanized Hindi words for addressing out-of-vocabulary problem, and lexical and syntactic ambiguity introduced due to contracted word forms. As we would train separate normalization and back-transliteration models for Hindi and English, we need language identification for selecting which model to use for inference for each word form separately. Moreover, we also need language information for decoding best word sequences.

4.0.1 Language Identification

In social media texts, where informal styles and code-switching are common, language identification remains a difficult problem. Code-switching texts use vocabulary from two or more languages and also contain new lexical forms by mixing their morphologies which makes it more difficult to identify the language of individual tokens. In this chapter, we address this issue and present a neural network based language identification system for Hindi-English code-switching. Our method uses pretrained word embeddings and character-RNN embeddings as features for the neural network and therefore, can easily be replicated across other language pairs.

For language identification task, we train a multilayer perceptron (MLP) stacked on top of a recurrent bidirectional LSTM (Bi-LSTM) network as shown in Figure 4.1. We represent each token by a

concatenated vector of its English embedding, back-transliterated Hindi embedding, character Bi-LSTM embedding and flag embedding (English dictionary flag and word length flag with length bins of 0-3, 4-6, 7-10, and 10-all). These concatenated vectors are passed to a Bi-LSTM network to generate a sequence of hidden representations which encode the contextual information spread across the sentence. Finally, output layer uses the feed-forward neural network with a softmax function for a probability distribution over the language tags. We train the network on our CS training set concatenated with the data set provided in ICON 2015[1] shared task (728 Facebook comments) on language identification and evaluate it on our evaluation set (see Chapter 3). We achieved the state-of-the-art performance on both the development and the test set. The results are shown in Table 4.1.

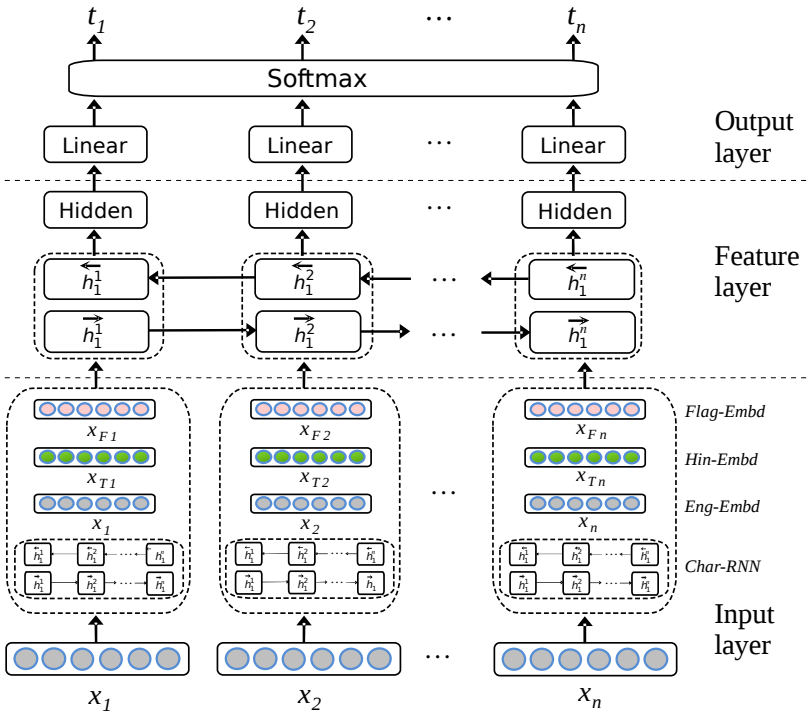


Figure 4.1: Language identification network

[1]http://ltrc.iiit.ac.in/icon2015/

Label	Development-Set				Test-Set			
	Precision	**Recall**	**F1-Score**	**Count**	**Precision**	**Recall**	**F1-Score**	**Count**
hi	98.46	99.03	98.74	1549	97.76	98.09	97.92	1465
en	98.32	98.77	98.54	1300	96.87	98.83	97.84	1283
ne	92.59	82.78	87.41	151	94.33	79.17	86.08	168
acro	93.94	96.88	95.38	32	92.00	76.67	83.64	30
univ	1.00	1.00	1.00	379	99.71	1.00	99.86	349
Average	98.27	98.30	98.27	3411	97.39	97.42	97.36	3295

Table 4.1: Language Identification results on CS development set and test set.

4.0.1.1 Hyperparameters

- **Word Representations:** We include the lexical features in the input layer of our neural networks using 64-dimension pre-trained word embeddings. For character RNNs, we use 32-dimensional character embeddings as input features. We use randomly initialized embeddings within a range of $[-0.1, +0.1]$ for non-lexical units such as character embeddings and dictionary flags.
- **Hidden dimensions:** The Bi-LSTM has 64 cells, where as, the character RNNs have 32 cells. The hidden layer of MLP has 64 nodes. We use hyperbolic tangent as the activation function.
- **Learning:** We use momentum SGD for learning with a minibatch size of 1. The LSTM weights are initialized with random orthonormal matrices as described in [66]. We set the dropout rate to 50%. The model is trained for up to 100 epochs, with early stopping based on the development set.

All the code is implemented in DyNet [49].

4.0.2 Normalization and Back-transliteration

We learn two separate but similar character-level models for normalization-cum-transliteration of noisy Romanized Hindi words and normalization of noisy English words. We treat both normalization and back-transliteration problems as a general sequence to sequence learning problem. In general, our

goal is to learn a mapping for non-standard English and Romanized Hindi word forms to standard forms in their respective scripts. In case of Hindi, we address the problem of normalization and back-transliteration of Romanized Hindi words using a single model. We use the attention-based encoder-decoder model of Luong [43] with global attention for learning. For Hindi, we train the model on the transliteration pairs (87,520) extracted from ILCI and Bojar Hindi-English parallel corpora [9, 35] which are further augmented with noisy transliteration pairs (1,75,668) for normalization. Similarly, for normalization of noisy English words, we train the model on noisy word forms (4,29,715) synthetically generated from the English vocabulary. We use simple rules such as dropping non-initial vowels and replacing consonants based on their phonological proximity to generate synthetic data for normalization.

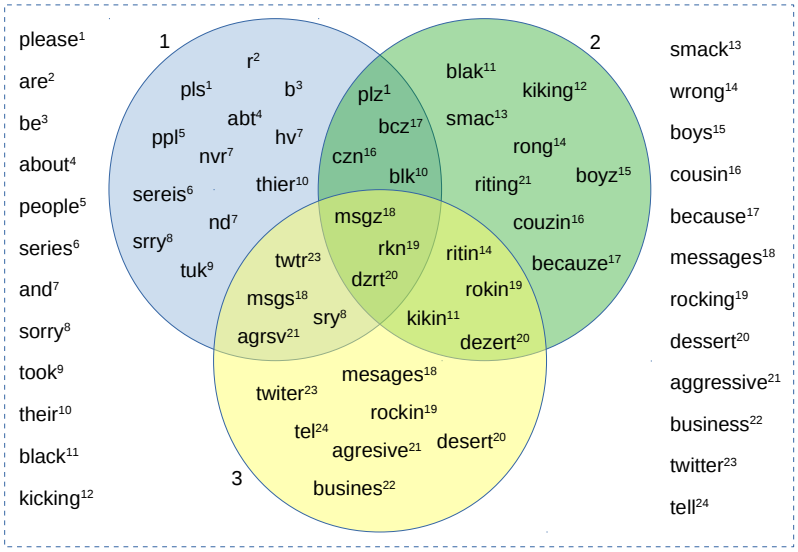


Figure 4.2: Synthetic normalization pairs generated for a sample of English words using hand crafted rules.

Figure 4.2 shows some of the noisy forms generated from standard word forms using simple and finite rules which include vowel elision (`please` → `pls`), interchanging similar consonants and vowels (`cousin` → `couzin`), replacing consonant or vowel clusters with a single letter (`Twitter` → `Twiter`), etc. From here onwards, we would refer to both normalization and back-transliteration as normalization.

At inference time, our normalization models would predict the most likely word form for each input word. However, the single-best output from the model may not always be the best option considering an overall sentential context. Contracted word forms in social media content are quite often ambiguous and can represent different standard word forms. For example, noisy form '`pt`' can expand to different

standard word forms such as 'put', 'pit', 'pat', 'pot' and 'pet'. The choice of word selection would solely depend on the sentential context. To select contextually relevant forms, we use exact search over n-best normalizations from the respective models extracted using beam-search decoding. The best word sequence is selected using the Viterbi decoding over b^n word sequences scored by a trigram language model. b is the size of beam-width and n is the sentence length. The language models are trained on the monolingual data of Hindi and English using KenLM toolkit [31]. For each word, we extract five best normalizations (b=5). Decoding the best word sequence is a non-trivial problem for CS data due to lack of normalized and back-transliterated CS data for training a language model. One obvious solution is to apply decoding on individual language fragments in a CS sentence [22]. One major problem with this approach is that the language models used for scoring are trained on complete sentences but are applied on sentence fragments. Scoring individual CS fragments would often lead to wrong word selection due to incomplete context. We solve this problem by using a 3-step decoding process that works on two separate versions of a CS sentence, one in Hindi, and one in English. In the first step, we replace first-best back-transliterated forms of Hindi words by their translation equivalents using a Hindi-English bilingual lexicon.[2] An exact search is used over the top '5' normalizations of English words, the translation equivalents of Hindi words and the actual word itself. In the second step, we decode best word sequence over Hindi version of the sentence by replacing best English word forms decoded from the first step by their translation equivalents. An exact search is used over the top '5' normalizations of Hindi words, the dictionary equivalents of decoded English words and the original words. In the final step, English and Hindi words are selected from their respective decoded sequences using the predicted language tags from the language identification system. Note that the bilingual mappings are only used to aid the decoding process by making the CS sentences lexically monolingual so that the monolingual language models could be used for scoring. They are not used in the final decoded output. The overall decoding process is shown in Figure 4.3.

Both of our normalization and back-transliteration systems are evaluated on our evaluation set (see Chapter 3). Results of our systems are reported in Table 4.2 with a comparison of accuracies based on the nature of decoding used. The results clearly show the significance of our 3-step decoding over first-best and fragment-wise decoding.

[2] An off-the-shelf MT system would have been appropriate for this task, however, we would first need to adapt it to CS data which in itself is a non-trivial task.

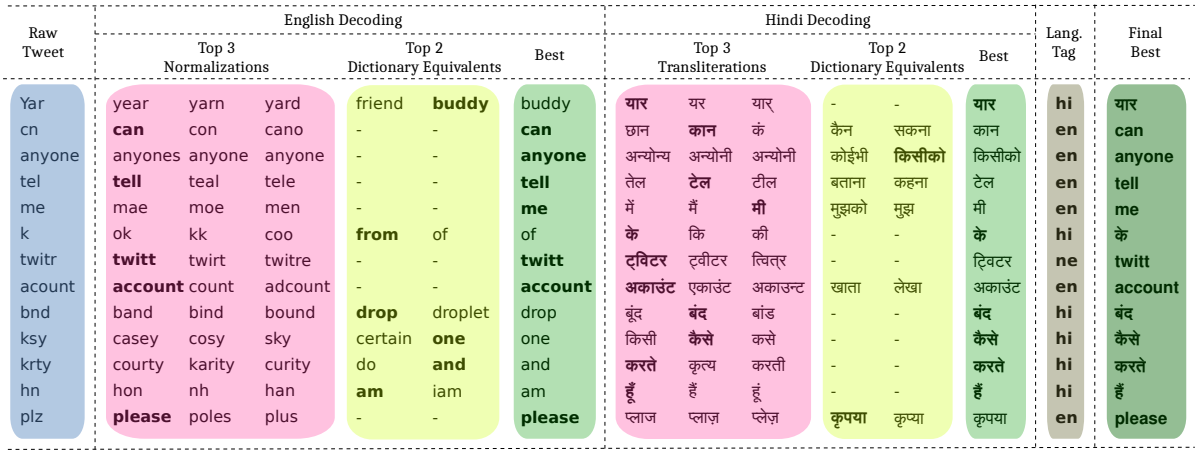


Figure 4.3: The figure shows a 3-step decoding process for the sentence *"Yar cn anyone tel me k twitr account bnd ksy krty hn plz" (Friend can anyone tell me how to close twitter account please).*

Data-set	Hindi				English			
	Tokens	FB	FW	3-step	Tokens	FB	FW	3-step
Dev	1549	82.82	87.28	**90.01**	34	82.35	**88.23**	**88.23**
Test	1465	83.54	88.19	**90.64**	28	71.42	75.21	**81.71**

Table 4.2: Normalization accuracy based on the number of noisy tokens in the evaluation set. FB = First Best, and FW = Fragment Wise

4.0.2.1 Hyperparameters

We use 32-dimensional character embeddings uniformly initialized within a range of $[-0.1, +0.1]$. We use single layered Bi-LSTMs with 512 cells for both encoding and decoding of character sequences. We train our encoder-decoder models for 25 epochs using vanilla SGD. We start with a learning rate of 1.0 and after 8 epochs reduce it to half for every epoch. We use a mini-batch size of 128, and the normalized gradient is rescaled whenever its norm exceeds 5. We use a dropout rate of 30% for the Bi-LSTM.

We used Open-NMT toolkit for training the models [38].

4.0.2.2 Extraction of Transliteration Pairs

Like any other supervised machine learning approach, supervised machine transliteration requires a strong list of transliteration pairs to learn the model parameters. However, such lists are not readily available and are expensive to create manually. Sajjad et al. [64, 65] have proposed algorithms to automatically mine transliteration pairs from parallel corpora. Sajjad et al. [64] propose an iterative algorithm based on phrase-based SMT coupled with a filtering technique, while Sajjad et al. [65] model transliteration mining as an interpolation of transliteration and non-transliteration sub-models. The model parameters are learned via EM procedure and the transliteration pairs are mined by setting an appropriate threshold. In this work, we use a simple edit distance-based approach to extract the transliteration pairs from the translation pairs.

We use the sentence aligned ILCI Hindi-English parallel corpora and Bojar Hindi-English parallel corpora [9, 35] to extract the transliteration pairs. Initially, the parallel corpus is word-aligned using GIZA++ [58], and the alignments are refined using the grow-diag-final-and heuristic [39]. We extract all the word pairs which occur as 1-to-1 alignments in the word-aligned corpus as potential transliteration equivalents. We extracted a total of 1,07,437 translation pairs from the parallel corpora. A rule-based approach with edit distance metric is used to extract the transliteration pairs from these translation pairs. To compute the edit distances, we use the Hindi-English character mappings presented in Table 4.4. We compute the levenshtein distance between the translation pairs based on insertion, deletion and replace operations. For each translation pair, we compare the letters via their mappings in the character mapping table. Finally, the distance scores are normalized by dividing them with the length of the longest string in a translation pair. Translation pairs with a normalized score of less than a small threshold of $\sim$0.3 are considered as transliteration pairs. Using this procedure, we extracted 87,520 transliteration pairs from the Hindi-English parallel corpus.

Hindi Letters	English Letters	Hindi Letters	English Letters	Hindi Letters	English Letters	Hindi Letters	English Letters
अ	a	ल	l	त	t, th	झ	jh
ब	b	म	m	द	d, dh	ख	kh
च	c, ch	न	n	य	y, i, e	ण	n
ड	d, dh	ओ	o, u	आ	a, aa	औ	o, u
ए	e, i, y, a	प	p	भ	bh	फ	ph, f
ङ	n	ऋ	r, ri	छ	ch	श	sh
ग	g, gh	र	r	ढ	dh	ष	sh
ह	h	स	s, c	ऐ	e, i, y	ठ	th
इ	e, i, y	ट	t, th	ञ	n	ऊ	o, u
ज	j, jh, z, g	उ	o, u	घ	gh	थ	th
क	k, q, c	व	v, w	ई	e, i, y	ध	dh

Figure 4.4: Devanagari to Roman character mapping table

4.1 Summary

In this chapter, we have presented a language identification system for Hindi-English CS data. We have also presented normalization and back-transliteration models with a decoding process tailored for CS data.

Chapter 5

Adapting Monolingual Parsing Models to Code-switching Data

In this chapter, we propose efficient and less resource-intensive strategies for parsing of code-switching data. These strategies are not constrained by in-domain annotations, rather they leverage pre-existing monolingual annotated resources for training. We show that these methods can produce significantly better results as compared to an informed baseline.

Due to paucity of annotated resources in code-switching genre, the performance of monolingual parsing models is yet to be evaluated on code-switching structures. This chapter serves to fill this gap by presenting a thorough evaluation of multiple models trained on monolingual treebanks. We show that direct application of monolingual parsing models is not feasible for parsing of CS data. Instead, we propose different parsing strategies that exploit nothing but the pre-existing annotated monolingual data. We show that by making trivial adaptations, monolingual parsing models can effectively parse code-switching data.

5.1 Parsing Strategies

We explore three different parsing strategies to parse code-switching data and evaluate their performance on a manually annotated evaluation set. These strategies are distinguished by the way they use pre-existing treebanks for parsing code-switching data.

5.1.1 Monolingual

The monolingual method uses two separate models trained from the respective monolingual treebanks of the languages which are present in the code-switching data. We can use the monolingual models in two different ways. Firstly, we can parse each code-switching sentence by intelligently choosing

the monolingual model based on the matrix language of the sentence. As illustrated in Figure 5.1, a clear disadvantage of this method is that the monolingual parser may not accurately parse those fragments of a sentence which belong to a language unknown to the model. Therefore, we consider this as the baseline method.

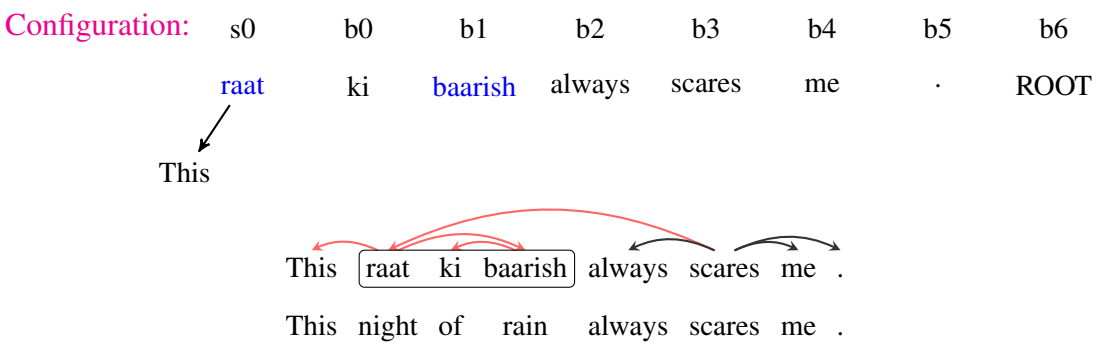


Figure 5.1: An illustration of how a monolingual parser fails to correctly parse the fragment '*raat ki baarish*' which does not belong to the matrix language (English in this case). Red arcs represent incorrect attachments, where as, black arcs represent correct attachments.

Secondly, we can linearly interpolate the predictions of both monolingual models at the inference time. The interpolation weights are chosen based on the matrix language of each parsing configuration. The interpolated oracle output is defined as:

$$y = \text{argmax}(\lambda_m * f(\phi(c_m)) + (1 - \lambda_m) * f(\phi(c_s))) \tag{5.1}$$

where $f(\cdot)$ is a *softmax* layer of our neural parsing model, $\phi(c_m)$ and $\phi(c_s)$ are the feature functions of the matrix and subordinate languages respectively and λ_m is the interpolation weight for the matrix language (see Section §6.2 for more details on the parsing model).

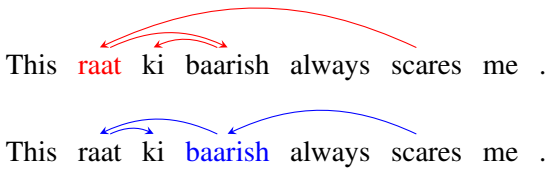
Instead of selecting the matrix language at sentence level, we define the matrix language individually for each parsing configuration. We define the matrix language of a configuration based on the language tags of top 2 nodes in the stack and buffer belonging to certain syntactic categories such as adposition, auxiliary, particle and verb.

5.1.2 Multilingual

In the second approach, we train a single model on a combined treebank of the languages represented in the code-switching data. This method has a clear advantage over the baseline Monolingual method in

that it would be aware of the grammars of both languages of the code-switching data. However, it may not be able to properly connect the fragments of two languages as the model lacks evidence for such mixed structures in the augmented data. This would particularly happen if the code-switching languages are typologically diverse.

Moreover, training a parsing model on augmented data with more diverse structures will worsen the structural ambiguity problem. But we can easily circumvent this problem by including token-level language tag as an additional feature in the parsing model [1]. Figure 5.2 illustrates the structural ambiguity problem for a genitive construction '*raat ki baarish*' (rain of night). In case of English, the head noun is towards the left of the adposition (tree 1 in Figure 5.2), where as, for Hindi the head noun is towards the right of the adposition (tree 2 in Figure 5.2).



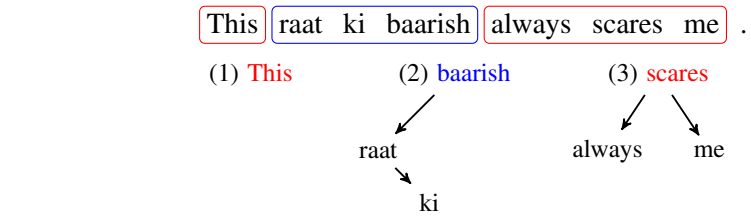
$$\mathcal{L} = \begin{cases} [1,0,0], & \text{if } s_0 \text{ or } b_0 \in (ADP, AUX, VERB, PART) \wedge \text{Lang.} = \text{English} \\ [0,1,0], & \text{if } s_0 \text{ or } b_0 \in (ADP, AUX, VERB, PART) \wedge \text{Lang.} = \text{Hindi} \\ [0,0,1], & \text{otherwise} \end{cases}$$

Figure 5.2: Resolving structural ambiguity problem using a token-level language tag.

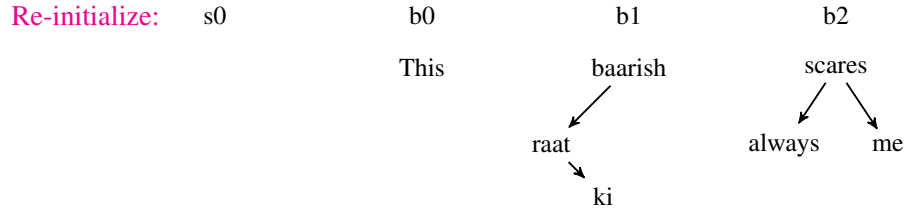
5.1.3 Multipass

In the Multipass method, we train two separate models like the Monolingual method. However, we apply these models on the code-switching data differently. Unlike Monolingual method, we use both models simultaneously for each sentence and pass the input to the models twice. There are two possible ways to accomplish this. We can first parse all the fragments of each language using their respective parsing models one by one and then the root nodes of the parsed fragments would be parsed by the matrix language parsing model. This is illustrated in Figure 5.3. Figure 5.4 illustrates a case of imperfect segmentation, where this method would fail to produce the correct parse tree no matter how appropriate the monolingual parsing models be. In this example, node '*youth*' should be the head of

the node '*bhaij*', but since node '*youth*' has already been assigned a head (*expect*), node '*youth*' can no longer appear in any configurations of the second pass and thus, no matter what, node '*youth*' cannot be the head of the node '*bhaij*'. We can easily fix these kinds of segmentation issues by parsing the subordinate language first and then parse the roots of the subordinate fragments with the fragments of matrix language using the matrix language parser. This is illustrated in Figure 5.5. However, this method has its own segmentation issues as illustrated in Figure 5.6. In this example node '*dekhte*' should be the head of the node '*TV*', but since node '*dekhte*' has already been assigned a head (*karle*), node '*dekhte*' can no longer appear in any configurations of the second pass and thus, node '*dekhte*' cannot be the head of the node '*TV*'.



(a) First Pass



(b) Second Pass

Figure 5.3: *First Pass*: Parse individual fragments using their respective parsing models. *Second Pass*: Parse the root nodes of the parsed fragments by the matrix language parsing model.

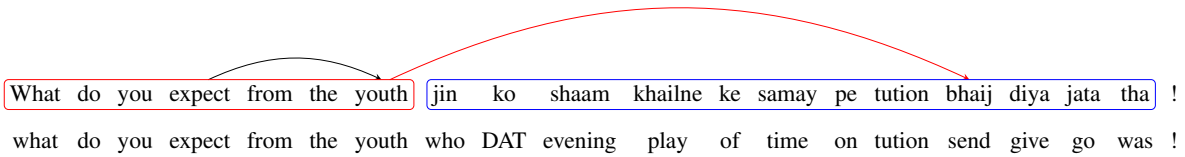
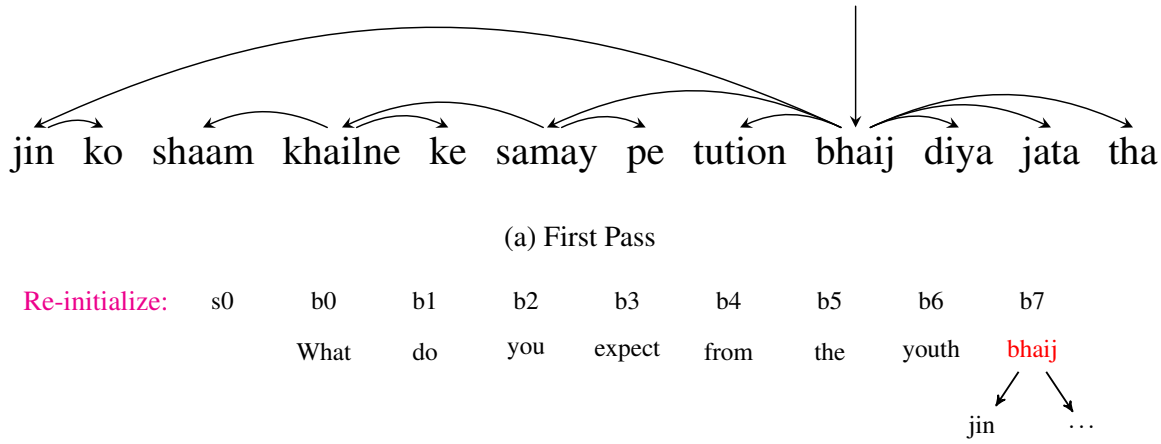


Figure 5.4: Example case of an imperfect segmentation



(a) First Pass

(b) Second Pass

Figure 5.5: *First Pass*: Parse subordinate language first. *Second Pass*: Parse the roots of the subordinate fragments with the fragments of matrix language using the matrix language parser.

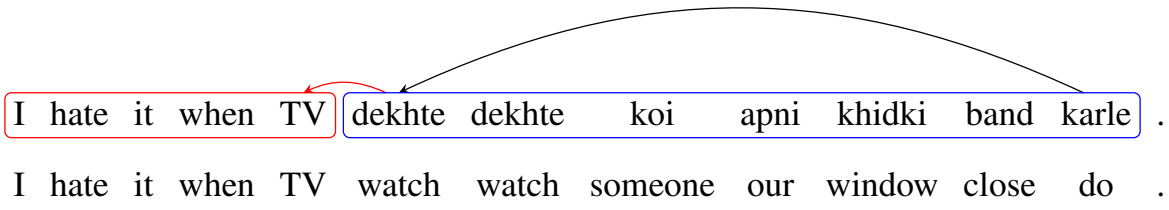


Figure 5.6: Example case of an imperfect segmentation

In both of these methods, monolingual parsers would not be affected by the cross language structures. More importantly, matrix language parser in the second pass would be unaffected by the internal structure of the subordinate language fragments. But there is a caveat, we need to identify the code-switching fragments accurately, which is a non-trivial task. In this work, we use token-level language information to segment tweets into subordinate or matrix language fragments.

5.2 Experimental Setup

The parsing experiments reported in this chapter are conducted using a non-linear neural network-based transition system which is similar to Chen and Manning (2014) [13]. The models are trained on Universal Dependency Treebanks of Hindi and English released under version 1.4 of Universal Dependencies [56].

5.2.1 Parsing Models

Our parsing model is based on transition-based dependency parsing paradigm [53]. Particularly, we use an arc-eager transition system [50]. Similar to Chen and Manning (2014) [13], we use a non-linear neural network to predict the transitions for the parser configurations. The neural network model is the standard feed-forward neural network with a single layer of hidden units. We use 200 hidden units and RelU activation function. The output layer uses softmax function for probabilistic multi-class classification. The model is trained by minimizing cross entropy loss with an $l2$-regularization over the entire training data. We also use mini-batch Adagrad for optimization [21] and apply dropout [32].

From each parser configuration, we extract features related to the top four nodes in the stack, top four nodes in the buffer and leftmost and rightmost children of the top two nodes in the stack and the leftmost child of the top node in the buffer.

5.2.2 POS Models

We train POS tagging models using a similar neural network architecture as discussed above. Unlike Collobert et al. (2011) [15], we do not learn separate transition parameters. Instead we include the structural features in the input layer of our model with other lexical and non-lexical units. We use second-order structural features, two words to either side of the current word, and last three characters of the current word.

We trained two POS tagging models: *Monolingual* and *Multilingual*. In the Monolingual approach, we divide each code-switching sentence into contiguous fragments based on the language tags assigned by the language identifier. Words with language tags other than 'Hi' and 'En' (such as univ, ne and acro) are merged with the preceding fragment. Each fragment is then individually tagged by the monolingual POS taggers trained on their respective monolingual POS data sets. In the Multilingual approach, we train a single model on combined data sets of the languages in the code-switching data. We concatenate an additional 1x2 vector[1] in the input layer of the neural network representing the language tag of the current word. Table 6.3 gives the POS tagging accuracies of the two models.

[1] In our experiments we fixed these to be {-0.25,0.25} for Hindi and {0.25,-0.25 } for English

Model	LID	Development-Set			Test-Set		
		HIN	ENG	Total	HIN	ENG	Total
Monolingual	G	0.849	0.903	0.873	0.832	0.889	0.860
	A	0.841	0.892	0.866	0.825	0.883	0.853
Multilingual	G	0.835	0.903	0.867	0.798	0.892	0.843
	A	0.830	0.900	0.862	0.790	0.888	0.836

Table 5.1: POS Tagging accuracies for monolingual and multilingual models. LID = Language tag, G = Gold LID, A = Auto LID.

Data-set	Gold (POS + language tag)									
	Monolingual		**Interpolated**		**Multilingual**		**Multipass$_f$**		**Multipass$_s$**	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
CM-Dev	60.77	49.24	74.62	64.11	**75.77**	**65.32**	69.37	58.83	70.23	59.64
CM-Test	60.05	48.52	**74.40**	63.65	74.16	**64.11**	68.54	57.87	69.12	58.64
HIN-Test	**95.29**	**92.60**	94.61	91.64	93.96	90.46	**95.29**	**92.60**	**95.29**	**92.60**
ENG-Test	88.63	**86.58**	87.93	85.43	**88.73**	86.38	88.63	**86.58**	88.63	**86.58**
	Auto (POS + language tag)									
CM-Dev	55.80	43.36	**68.24**	**56.07**	67.71	55.18	63.34	52.22	64.60	53.03
CM-Test	54.95	43.03	65.14	54.00	**66.18**	**54.40**	62.37	51.11	63.74	52.34
HIN-Test	**92.92**	**89.39**	92.82	89.34	90.52	85.83	**92.92**	**89.39**	**92.92**	**89.39**
ENG-Test	**86.28**	**83.90**	86.08	82.54	85.53	83.11	**86.28**	**83.90**	**86.28**	**83.90**

Table 5.2: Accuracy of different parsing strategies on Code-switching as well as Hindi and English evaluation sets. Multipass$_{f|s}$ = fragment-wise and subordinate-first parsing methods.

5.2.3 Word Representations

For both POS tagging and parsing models, we include the lexical features in the input layer of the Neural Network using the pre-trained word representations while for the non-lexical features, we use

randomly initialized embeddings within a range of -0.25 to $+0.25$.[2] The distributed representation of Hindi and English vocabulary are learned separately from the Hindi and English monolingual corpora. The English monolingual data contains around 280M sentences, while the Hindi data is comparatively smaller and contains around 40M sentences. The word representations are learned using Skip-gram model with negative sampling which is implemented in `word2vec` toolkit [45]. For multilingual models, we use robust projection algorithm of Guo et al. [28] to induce bilingual representations using the monolingual embedding space of English and a bilingual lexicon of Hindi and English ($\sim$63,000 entries). We extracted the bilingual lexicon from ILCI and Bojar Hi-En parallel corpora [9, 35].

5.3 Experiments and Results

We conducted multiple experiments to measure effectiveness of the proposed parsing strategies in both gold and predicted settings. In predicted settings, we use the monolingual POS taggers for all the experiments. We used the Monolingual method as the baseline for evaluating other parsing strategies. The baseline model parses each sentence in the evaluation sets by either using Hindi or English parsing model based on the matrix language of the sentence. For baseline and the Multipass methods, we use bilingual embedding space derived from matrix language embedding space (Hindi or English) to represent lexical nodes in the input layer of our parsing architecture. In the Interpolation method, we use separate monolingual embedding spaces for each model. The interpolation weights are tuned using the development set and the best results are achieved at λ_m ranging from 0.7 to 0.8 (see eq. 5.1). The results of our experiments are reported in Table 5.2. Table 5.3 shows the impact of sentential decoding for choosing the best normalized and/or back-transliterated tweets on different parsing strategies (see 4).

[2]Dimensionality of input units in POS and parsing models: 80 for words, 20 for POS tags, 2 for language tags and 20 for affixes.

Data-set	First Best				K-Best			
	Multilingual		**Interpolated**		**Multilingual**		**Interpolated**	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
CM-Dev	66.21	53.55	66.70	53.68	67.71	55.18	68.24	56.07
CM-Test	65.87	53.92	64.26	53.35	66.18	54.40	65.14	54.00

Table 5.3: Parsing accuracies with exact search and k-best search (k = 5)

All of our parsing models produce results that are at-least 10 LAS points better than our baseline parsers which otherwise provide competitive results on Hindi and English evaluation sets [71].[3] Among all the parsing strategies, the Interpolated methods perform comparatively better on both monolingual and code-switching evaluation sets. Interpolation method manipulates the parameters of both languages quite intelligently at each parsing configuration. Despite being quite accurate on code-switching evaluation sets, the Multilingual model is less accurate in single language scenario. Also the Multilingual model performs worse for Hindi since its lexical representation is derived from English embedding space. It is at-least 2 LAS points worse than the Interpolated and the Multipass methods. However, unlike the latter methods, the Multilingual models do not have a run-time and computational overhead. In comparison to Interpolated and Multilingual methods, Multipass methods are mostly affected by the errors in language identification. Quite often these errors lead to wrong segmentation of code-switching fragments which adversely alter their internal structure.

Despite higher gains over the baseline models, the performance of our models is nowhere near the performance of monolingual parsers on newswire texts. This is due to inherent complexities of code-switching social media content [11, 69, 73].

5.4 Conclusion

In this chapter, we have evaluated different strategies for parsing code-switching data that only leverage monolingual annotated data. We have shown that code-switching texts can be efficiently parsed by the monolingual parsing models if they are intelligently manipulated. Against an informed monolingual

[3]Our results are not directly comparable to [71] due to different parsing architectures. While we use a simple greedy, projective transition system, Straka et al. [71] use a search-based swap system.

baseline, our parsing strategies are at-least 10 LAS points better. Among different strategies that we proposed, Multilingual and Interpolation methods are two competitive methods for parsing code-switching data.

Chapter 6

Leveraging Monolingual and Code-switching treebanks using Neural Stacking

In this chapter, we propose a neural stacking model for parsing Hindi-English CS data that efficiently leverages part-of-speech tag and syntactic tree annotations in the code-switching treebank and the pre-existing Hindi and English treebanks. Results show that our neural stacking parser is 1.5% LAS points better than the augmented parsing model and our decoding process improves results by 3.8% LAS points over the first-best normalization and/or back-transliteration.

6.1 Dependency Parsing

We adapt Kiperwasser and Goldberg [37] transition-based parser as our base model and incorporate POS tag and monolingual parse tree information into the model using neural stacking, as shown in Figures 6.1 and 6.3. Our parsing models are based on an arc-eager transition system [50]. We use the training by exploration method of [25] for decoding a transition sequence which helps in mitigating error propagation at evaluation time. We also use pseudo-projective transformations of [54] to handle a higher percentage of non-projective arcs in the CS data ($\sim$2%). We use the most informative scheme of `head+path` to store the transformation information.

6.1.1 Base Models

Our base model is a stack of a tagger network and a parser network inspired by stack-propagation model of Zhang and Weiss (2016) [82]. The parameters of the tagger network are shared and act as a regularization on the parsing model. The model is trained by minimizing a joint negative log-likelihood loss for both tasks. Unlike Zhang and Weiss [82], we compute the gradients of the log-loss function

simultaneously for each training instance. While the parser network is updated given the parsing loss only, the tagger network is updated with respect to both tagging and parsing losses. Both tagger and parser networks comprise of an input layer, a feature layer, and an output layer as shown in Figure 6.1. Following Zhang and Weiss [82], we refer to this model as stack-prop.

6.1.2 Tagger network

The input layer of the tagger encodes each input word in a sentence by concatenating a pre-trained word embedding with its character embedding given by a character Bi-LSTM. In the feature layer, the concatenated word and character representations are passed through two stacked Bi-LSTMs to generate a sequence of hidden representations which encode the contextual information spread across the sentence. The first Bi-LSTM is shared with the parser network while the other is specific to the tagger. Finally, output layer uses the feed-forward neural network with a softmax function for a probability distribution over the Universal POS tags. We only use the forward and backward hidden representations of the focus word for classification.

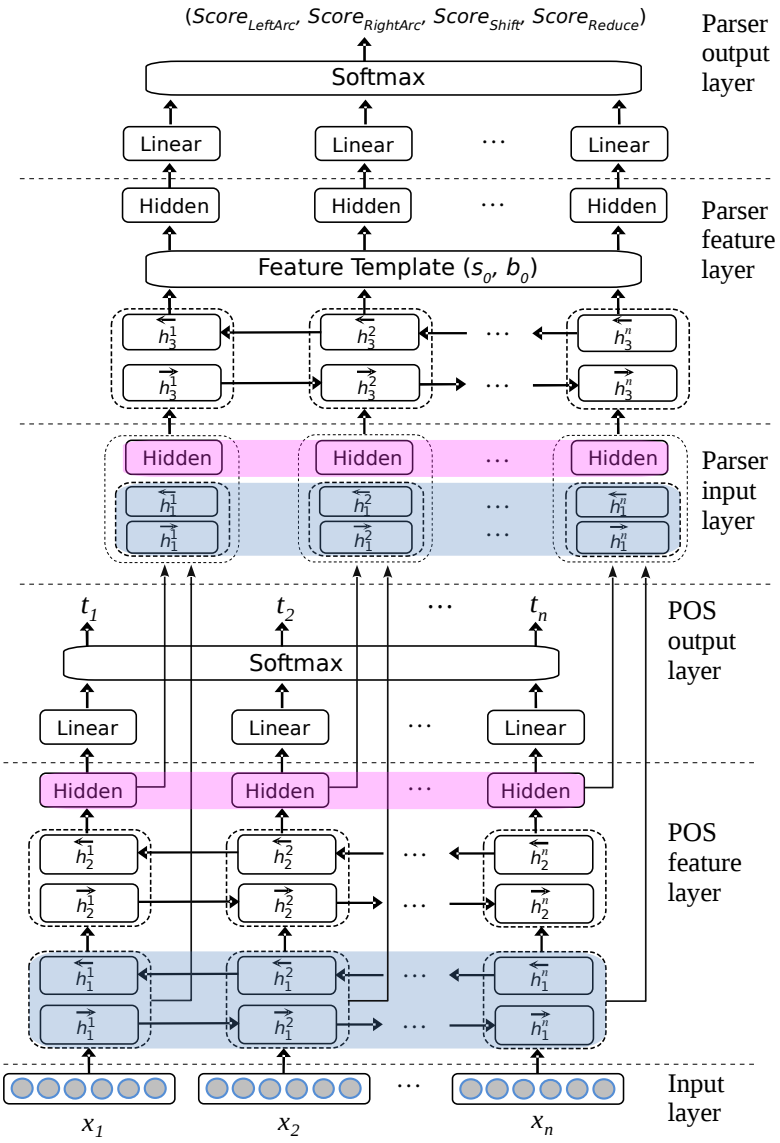


Figure 6.1: POS tagging and parsing network based on stack-propagation model proposed in [82].

6.1.3 Parser Network

Similar to the tagger network, the input layer encodes the input sentence using word and character embeddings which are then passed to the shared Bi-LSTM. The hidden representations from the shared Bi-LSTM are then concatenated with the dense representations from the feed-forward network of the tagger and passed through the Bi-LSTM specific to the parser. This ensures that the tagging network is penalized for the parsing error caused by error propagation by back-propagating the gradients to the

shared tagger parameters [82]. Finally, we use a non-linear feed-forward network to predict the labeled transitions for the parser configurations. From each parser configuration, we extract the top node in the stack and the first node in the buffer and use their hidden representations from the parser specific Bi-LSTM for classification.

6.1.4 Stacking Models

It seems reasonable that limited CS data would complement large monolingual data in parsing CS data and a parsing model which leverages both data would significantly improve parsing performance. While a parsing model trained on our limited CS data might not be enough to accurately parse the individual grammatical fragments of Hindi and English, the preexisting Hindi and English treebanks are large enough to provide sufficient annotations to capture their structure. Similarly, parsing model(s) trained on the Hindi and English data may not be able to properly connect the divergent fragments of the two languages as the model lacks evidence for such mixed structures in the monolingual data. This would happen quite often as Hindi and English are typologically very diverse (see Figure 6.2).

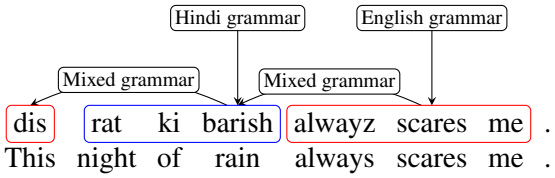


Figure 6.2: Code-switching tweet showing grammatical fragments from Hindi and English.

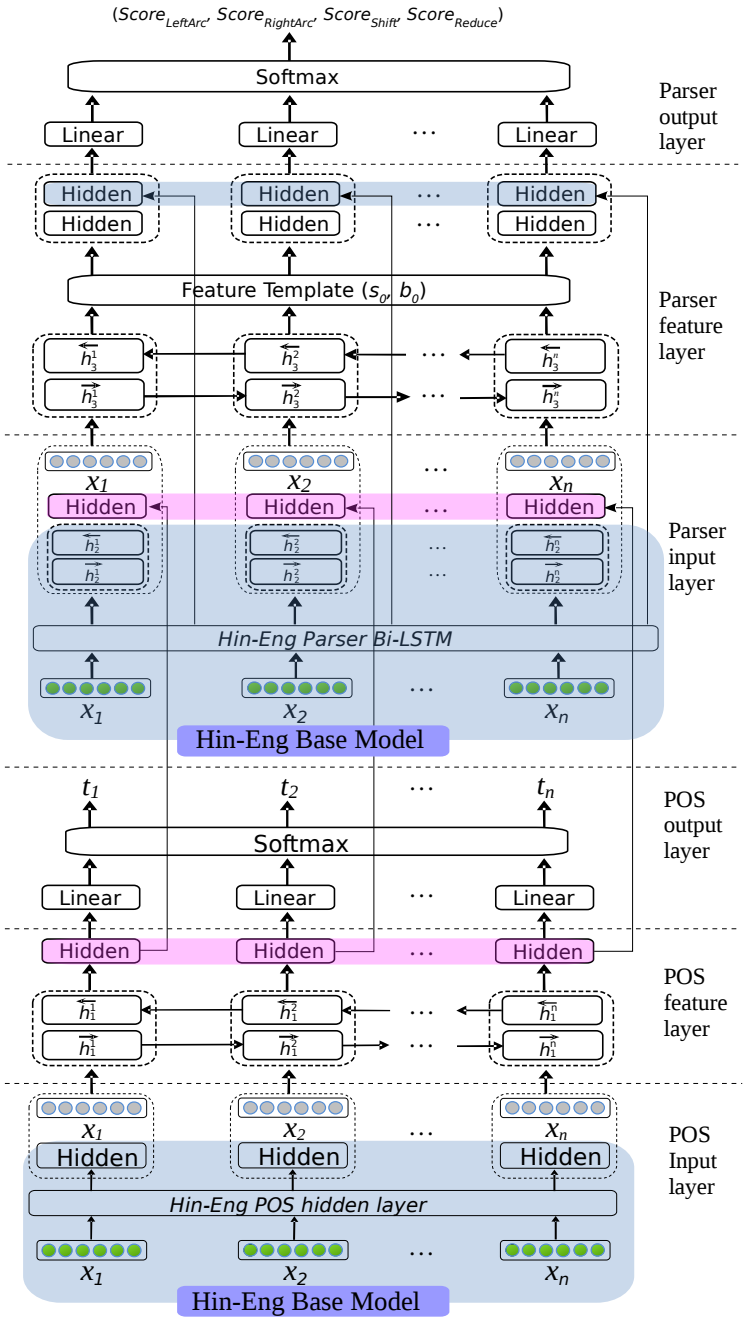


Figure 6.3: Neural Stacking-based parsing architecture for incorporating monolingual syntactic knowledge.

As we discussed above, we adapted feature-level neural stacking [14, 82] for joint learning of POS tagging and parsing. Similarly, we also adapt this stacking approach for incorporating the monolingual syntactic knowledge into the base CS model. Recently, [74] used neural stacking for injecting syntactic

knowledge of English into a graph-based Singlish parser which lead to significant improvements in parsing performance. Unlike [74], our base stacked models would allow us to transfer the POS tagging knowledge as well along the parse tree knowledge.

As shown in Figure 6.3, we transfer both POS tagging and parsing information from the source model trained on augmented Hindi and English data. For tagging, we augment the input layer of the CS tagger with the MLP layer of the source tagger. For transferring parsing knowledge, hidden representations from the parser specific Bi-LSTM of the source parser are augmented with the input layer of the CS parser which already includes the hidden layer of the CS tagger, word and character embeddings. In addition, we also add the MLP layer of the source parser to the MLP layer of the CS parser. The MLP layers of the source parser are generated using raw features from CS parser configurations. Apart from the addition of these learned representations from the source model, the overall CS model remains similar to the base model shown in Figure 6.1. The tagging and parsing losses are back-propagated by traversing back the forward paths to all trainable parameters in the entire network for training and the whole network is used collectively for inference.

6.2 Experiments

We train all of our POS tagging and parsing models on training sets of the Hindi and English UD-v2 treebanks and our Hindi-English CS treebank. We use the development and evaluation sets for tuning and evaluation of our models (see Chapter 3). We conduct multiple experiments in gold and predicted settings to measure the effectiveness of the sub-modules of our parsing pipeline. In predicted settings, we use the POS taggers separately trained on the Hindi, English and CS training sets. All of our models use word embeddings from transformed Hindi and English embedding spaces to address the problem of lexical differences prevalent in CS sentences.

6.2.1 Hyperparameters

6.2.1.1 Word Representations

For POS tagging and parsing models, we include the lexical features in the input layer of our neural networks using 64-dimension pre-trained word embeddings, while we use randomly initialized embeddings within a range of $[-0.1, +0.1]$ for non-lexical units such as POS tags and character embeddings.

We use 32-dimensional character embeddings for both the models and 32-dimensional POS tag embeddings for pipelined parsing models. We use same distributed representation of Hindi and English vocabulary as discussed in Chapter 5. We use the projection algorithm of Artetxe et al. (2016) [2] to transform the Hindi and English monolingual embeddings into same semantic space using a bilingual lexicon (∼63,000 entries). The bilingual lexicon is extracted from ILCI and Bojar Hindi-English parallel corpora [9, 35].

6.2.1.2 Hidden dimensions

The POS tagger specific Bi-LSTMs have 128 cells while the parser specific Bi-LSTMs have 256 cells. The character Bi-LSTMs have 32 cells for both the models. The hidden layer of MLP has 128 nodes for the POS tagger and 256 nodes for the parser. We use hyperbolic tangent as the activation function in all the tasks.

6.2.1.3 Learning

For POS tagging and parsing networks, we use momentum SGD for learning with a minibatch size of 1. The LSTM weights are initialized with random orthonormal matrices as described in [66]. We set the dropout rate to 30% for POS tagger and parser Bi-LSTM and MLP hidden states. All the models are trained for up to 100 epochs, with early stopping based on the development set.

All the code is implemented in DyNet [49].

6.3 Results

In Table 6.1, we present the results of our main model that uses neural stacking for learning POS tagging and parsing and also for knowledge transfer from the Bilingual model. Transferring POS tagging and syntactic knowledge using neural stacking gives 1.5% LAS[1] improvement over a naive approach of data augmentation. The Bilingual model which is trained on the union of Hindi and English data sets is least accurate of all our parsing models. However, it achieves better or near state-of-the-art results on the Hindi and English evaluation sets (see Table 6.2). As compared to the best system in CoNLL 2017 Shared Task on Universal Dependencies [20, 78], our results for English are around 3% better in LAS, while for Hindi only 0.5% LAS points worse. The CS model trained only on the CS training

[1]The improvements discussed in the running text are for the models that are evaluated in auto settings.

data is slightly more accurate than the Bilingual model. Augmenting the CS data to Hindi-English data complements their syntactic structures relevant for parsing mixed grammar structures which are otherwise missing in the individual datasets. The average improvements of around ∼5% LAS clearly show their complementary nature.

Table 6.3 summarizes the POS tagging results on the CS evaluation set. The tagger trained on the CS training data is 2.5% better than the Bilingual tagger. Adding CS training data to Hindi and English train sets further improves the accuracy by 1%. However, our stack-prop tagger achieves the highest accuracy of 90.53% by leveraging POS information from Bilingual tagger using neural stacking.

Model	Gold (LID+TRN)		Auto (LID+TRN)	
	UAS	LAS	UAS	LAS
Bilingual	75.26	65.41	73.29	63.18
CS	76.69	66.90	75.84	64.94
Augmented	80.39	71.27	78.95	69.51
Neural Stacking	**81.50**	**72.44**	**80.23**	**71.03**

Table 6.1: Accuracy of different parsing models on the evaluation set. POS tags are jointly predicted with parsing. LID = Language tag, TRN = Transliteration/normalization.

Data-set	Pipeline					Stack-prop		
	Gold POS		Auto POS					
	UAS	LAS	POS	UAS	LAS	POS	UAS	LAS
Hindi	95.66	93.08	97.52	94.08	90.69	**97.65**	**94.36**	**91.02**
English	89.95	87.96	95.75	87.71	84.59	**95.80**	**88.30**	**85.30**

Table 6.2: POS and parsing results for Hindi and English monolingual test sets using pipeline and stack-prop models.

Model	Gold (LID+TRN)		Auto (LID+TRN)	
	Pipeline	SP	Pipeline	SP
Bilingual	88.36	88.12	86.71	86.27
CS	90.32	90.38	89.12	89.19
Augmented	91.20	91.50	90.02	90.20
Neural Stacking	**91.76**	**91.90**	**90.36**	**90.53**

Table 6.3: POS tagging accuracies of different models on CS evaluation set. SP = stack-prop.

6.3.1 Pipeline vs Stack-prop

Table 6.4 summarizes the parsing results of our pipeline models which use predicted POS tags as input features. As compared to our stack-prop models (Table 6.1), pipeline models are less accurate (average 1% LAS improvement across models) which clearly emphasizes the significance of back-propagating the parsing loss to tagging parameters as well.

Model	Gold (LID+TRN+POS)		Auto (LID+TRN+POS)	
	UAS	LAS	UAS	LAS
Bilingual	82.29	73.79	72.09	61.18
CS	82.73	73.38	75.20	64.64
Augmented	85.66	77.75	77.98	69.16
Neural Stacking	**86.87**	**78.57**	**78.90**	**69.45**

Table 6.4: Accuracy of different parsing models on the test set using predicted language tags, normalized/back-transliterated words and predicted POS tags. POS tags are predicted separately before parsing. In Neural Stacking model, only parsing knowledge from the Bilingual model is transferred.

6.3.2 Significance of normalization

We also conducted experiments to evaluate the impact of normalization on both POS tagging and parsing. The results are shown in Table 6.5. As expected, tagging and parsing models that use normalization without decoding achieve an average of 1% improvement over the models that do not use

normalization at all. However, our 3-step decoding leads to higher gains in tagging as well as parsing accuracies. We achieved around 2.8% improvements in tagging and around 4.6% in parsing over the models that use first-best word forms from the normalization models. More importantly, there is a moderate drop in accuracy (1.4% LAS points) caused due to normalization errors (see results in Table 6.1 for gold vs auto normalization).

System	POS	UAS	LAS
No Normalization	86.98	76.25	66.02
First Best	87.74	78.26	67.22
3-step Decoding	**90.53**	**80.23**	**71.03**

Table 6.5: Impact of normalization and back-transliteration on POS tagging and parsing models.

6.3.3 Monolingual vs Cross-lingual Embeddings

We also conducted experiments with monolingual and cross-lingual embeddings to evaluate the need for transforming the monolingual embeddings into a same semantic space for processing of CS data. Results are shown in Table 6.6. Cross-lingual embeddings have brought around $\sim$0.5% improvements in both tagging and parsing. Cross-lingual embeddings are essential for removing lexical differences which is one of the problems encountered in CS data. Addressing the lexical differences would help in better learning by exposing syntactic similarities between languages.

Embedding	POS	UAS	LAS
Monolingual	90.07	79.46	70.53
Crosslingual	**90.53**	**80.23**	**71.03**

Table 6.6: Impact of monolingual and cross-lingual embeddings on stacking model performance.

6.4 Summary

In this chapter, we have presented a dependency parser designed explicitly for Hindi-English CS data. The parser uses neural stacking architecture of [82] and [14] for learning POS tagging and parsing

and for knowledge transfer from Bilingual models trained on Hindi and English UD treebanks. Our neural stacking parser is 1.5% LAS points better than the augmented parsing model and 3.8% LAS points better than the one which uses first-best normalizations.

Chapter 7

Summary and Future Work

In this Chapter, we summarize the main concepts and contributions of the thesis. We also briefly discuss, the possible directions for extending the research presented in the thesis.

7.1 Conclusion

This thesis targets the problems in dependency analysis of code-switching Hindi-English which is lexically and structurally more diverse than the individual monolingual data. Due to mixing of the morphologies of the two languages, Hindi-English CS data contain additional lexical forms that are unknown to the monolingual lexicons, leading to a higher rate of out-of-vocabulary words. Moreover, mixing the two typologically diverse languages also intensify the structural variations which make parsing even more challenging.

As the current computational models fail to cater to the complexities of CS data, there is often a need for dedicated techniques tailored to its specific characteristics. We have investigated the indispensable processes of language identification, normalization/back-transliteration and other problems associated with syntactic parsing of code-switching data and have proposed methods to mitigate their effects. Our primary goals in this thesis have been to tackle high lexical and structural diversity of code-switching data.

Firstly, we have evaluated different strategies for parsing code-mixed data that only leverage monolingual annotated data. We have shown that code-mixed texts can be efficiently parsed by the monolingual parsing models if they are intelligently manipulated. Against an informed monolingual baseline, our parsing strategies are at-least 10 LAS points better. Secondly, we have presented a dependency parser designed explicitly for Hindi-English CS data. We have proposed a neural stacking model for

parsing that efficiently leverages part-of-speech tag and syntactic tree annotations in the code-switching treebank and the preexisting Hindi and English treebanks. Our neural stacking parser is 1.5% LAS points better than the augmented parsing model and 3.8% LAS points better than the one which uses first-best normalizations. Further, we also present normalization and back-transliteration models with a decoding process tailored for code-switching data. Finally, we present a treebank of Hindi-English code-switching tweets under Universal Dependencies scheme. This is the first code-switching treebank that provides syntactic annotations required for parsing mixed-grammar syntactic structures.

7.2 Future Research Directions

While this thesis has presented effective strategies for tackling the issues concerning structural and lexical diversities in statistical parsing of code-switching Hindi-English, many opportunities for extending the scope of this thesis remain. Some of the future research directions include the following:

1. The methods and approaches presented in this thesis can be applied to more Indian languages code-switched with English to evaluate their efficiency.

2. The parsing results can be further improved by using more advanced learning architectures for transition-based parsers such as graph-based deep biaffine attention parsers [19, 37].

Bibliography

[1] W. Ammar, G. Mulcaire, M. Ballesteros, C. Dyer, and N. Smith. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444, 2016.

[2] M. Artetxe, G. Labaka, and E. Agirre. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2289–2294, 2016.

[3] P. Auer. The pragmatics of code-switching: A sequential approach. *One speaker, two languages: Cross-disciplinary perspectives on code-switching*, pages 115–135, 1995.

[4] P. Auer. *Code-switching in conversation: Language, interaction and identity*. Routledge, 2013.

[5] U. Barman, A. Das, J. Wagner, and J. Foster. Code mixing: A challenge for language identification in the language of social media. In *Proceedings of The First Workshop on Computational Approaches to Code Switching*, pages 13–23, 2014.

[6] R. Begum, S. Husain, A. Dhwaj, D. M. Sharma, L. Bai, and R. Sangal. Dependency annotation scheme for indian languages. In *IJCNLP*, pages 721–726. Citeseer, 2008.

[7] I. Bhat, R. A. Bhat, M. Shrivastava, and D. Sharma. Joining hands: Exploiting monolingual treebanks for parsing of code-mixing data. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 324–330, Valencia, Spain, April 2017. Association for Computational Linguistics.

[8] R. A. Bhat, R. Bhatt, A. Farudi, P. Klassen, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma, A. Vaidya, S. R. Vishnu, et al. The Hindi/Urdu treebank project. In *Handbook of Linguistic Annotation*. Springer Press, 2015.

[9] O. Bojar, V. Diatka, P. Rychlý, P. Straňák, V. Suchomel, A. Tamchyna, and D. Zeman. HindEnCorp - Hindi-English and Hindi-only Corpus for Machine Translation. In N. C. C. Chair), K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, and S. Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation*

(LREC'14), Reykjavik, Iceland, 2014. European Language Resources Association (ELRA). ISBN 978-2-9517408-8-4.

[10] M.-S. Carol. Multiple voices: an introduction to bilingualism, 2006.

[11] O. Çetinoğlu, S. Schulz, and N. T. Vu. Challenges of computational processing of code-switching. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 1–11, Austin, Texas, November 2016. Association for Computational Linguistics.

[12] K. R. Chandu, M. Chinnakotla, A. W. Black, and M. Shrivastava. Webshodh: A code mixed factoid question answering system for web. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 104–111. Springer, 2017.

[13] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750, 2014.

[14] H. Chen, Y. Zhang, and Q. Liu. Neural network for heterogeneous annotations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 731–741, Austin, Texas, November 2016. Association for Computational Linguistics.

[15] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12, 2011.

[16] M.-C. De Marneffe and C. D. Manning. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8. Association for Computational Linguistics, 2008.

[17] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa Italy, 2006.

[18] M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, volume 14, pages 4585–92, 2014.

[19] T. Dozat and C. D. Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

[20] T. Dozat, P. Qi, and C. D. Manning. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, 2017.

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

[22] S. Dutta, T. Saha, S. Banerjee, and S. K. Naskar. Text normalization in code-mixed social media text. In *Proceedings of the 2nd International Conference on Recent Trends in Information Systems (ReTIS)*, 2015.

[23] J. M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics, 1996.

[24] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics, 2011.

[25] Y. Goldberg and J. Nivre. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics*, pages 959–976, 2012.

[26] Y. Goldberg and J. Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1:403–414, 2013.

[27] J. J. Gumperz. *Discourse strategies*, volume 1. Cambridge University Press, 1982.

[28] J. Guo, W. Che, D. Yarowsky, H. Wang, and T. Liu. Cross-lingual dependency parsing based on distributed representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1, pages 1234–1244, 2015.

[29] G. Guzmán, J. Ricard, J. Serigos, B. E. Bullock, and A. J. Toribio. Metrics for modeling code-switching across corpora. *Proceedings of Interspeech 2017*, pages 67–71, 2017.

[30] J. Hall, J. Nivre, and J. Nilsson. Discriminative classifiers for deterministic dependency parsing. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 316–323. Association for Computational Linguistics, 2006.

[31] K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn. Scalable modified kneser-ney language model estimation. In *ACL (2)*, pages 690–696, 2013.

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*,

2012.

[33] D. Hymes and J. Gumperz. Direction in sociolinguistics: The ethnography of communication. *New York: Rinehart and Winston*, 1972.

[34] A. Jamatia, B. Gambäck, and A. Das. Part-of-speech tagging for code-mixed english-hindi twitter and facebook chat messages. page 239, 2015.

[35] G. N. Jha. The TDIL program and the Indian language corpora initiative (ILCI). In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010). European Language Resources Association (ELRA)*, 2010.

[36] A. Joshi, A. Prabhu, M. Shrivastava, and V. Varma. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2482–2491, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[37] E. Kiperwasser and Y. Goldberg. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4: 313–327, 2016.

[38] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017. doi: 10.18653/v1/P17-4012. URL `https://doi.org/10.18653/v1/P17-4012`.

[39] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[40] L. Kong, N. Schneider, S. Swayamdipta, A. Bhatia, and C. Dyer. A dependency parser for tweets. In *Proceedings of Conference on Empirical Methods In Natural Language Processing (EMNLP)*, volume 1001, page 1012, 2014.

[41] S. Kübler, R. McDonald, and J. Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

[42] T. Kudo and Y. Matsumoto. Japanese dependency analysis using cascaded chunking. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

[43] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[44] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[45] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[46] P. Muysken. *Bilingual speech: A typology of code-mixing*, volume 11. Cambridge University Press, 2000.

[47] C. Myers-Scotton. *Social motivations for codeswitching: Evidence from Africa*. Oxford University Press, 1995.

[48] C. Myers-Scotton. *Multiple voices: An introduction to bilingualism*. Wiley-Blackwell, 2005.

[49] G. Neubig, C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[50] J. Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, 2003.

[51] J. Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics, 2004.

[52] J. Nivre. Constraints on non-projective dependency parsing. In *Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006.

[53] J. Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

[54] J. Nivre and J. Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005.

[55] J. Nivre and J. Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics, 2005.

[56] J. Nivre, Ž. Agić, L. Ahrenberg, M. J. Aranzabe, M. Asahara, A. Atutxa, M. Ballesteros, J. Bauer, K. Bengoetxea, Y. Berzak, R. A. Bhat, E. Bick, C. Börstell, C. Bosco, G. Bouma, S. Bowman, G. Cebirolu Eryiit, G. G. A. Celano, F. Chalub, Ç. Çöltekin, M. Connor, E. Davidson, M.-C. de Marneffe, A. Diaz de Ilarraza, K. Dobrovoljc, T. Dozat, K. Droganova, P. Dwivedi, M. Eli, T. Erjavec, R. Farkas, J. Foster, C. Freitas, K. Gajdošová, D. Galbraith, M. Garcia, M. Gärdenfors, S. Garza, F. Ginter, I. Goenaga, K. Gojenola, M. Gökrmak, Y. Goldberg, X. Gómez Guinovart, B. Gonzáles Saavedra, M. Grioni, N. Grūzītis, B. Guillaume, J. Hajič, L. Hà M, D. Haug, B. Hladká, R. Ion, E. Irimia, A. Johannsen, F. Jørgensen, H. Kaşkara, H. Kanayama, J. Kanerva, B. Katz, J. Kenney, N. Kotsyba, S. Krek, V. Laippala, L. Lam, P. Lê Hng, A. Lenci, N. Ljubešić, O. Lyashevskaya, T. Lynn, A. Makazhanov, C. Manning, C. Mărănduc, D. Mareček, H. Martínez Alonso, A. Martins, J. Mašek, Y. Matsumoto, R. McDonald, A. Missilä, V. Mititelu, Y. Miyao, S. Montemagni, K. S. Mori, S. Mori, B. Moskalevskyi, K. Muischnek, N. Mustafina, K. Müürisep, L. Nguyn Th, H. Nguyn Th Minh, V. Nikolaev, H. Nurmi, P. Osenova, R. Östling, L. Øvrelid, V. Paiva, E. Pascual, M. Passarotti, C.-A. Perez, S. Petrov, J. Piitulainen, B. Plank, M. Popel, L. Pretkalnia, P. Prokopidis, T. Puolakainen, S. Pyysalo, A. Rademaker, L. Ramasamy, L. Real, L. Rituma, R. Rosa, S. Saleh, B. Saulīte, S. Schuster, W. Seeker, M. Seraji, L. Shakurova, M. Shen, N. Silveira, M. Simi, R. Simionescu, K. Simkó, M. Šimková, K. Simov, A. Smith, C. Spadine, A. Suhr, U. Sulubacak, Z. Szántó, T. Tanaka, R. Tsarfaty, F. Tyers, S. Uematsu, L. Uria, G. van Noord, V. Varga, V. Vincze, L. Wallin, J. X. Wang, J. N. Washington, M. Wirén, Z. Žabokrtský, A. Zeldes, D. Zeman, and H. Zhu. Universal dependencies 1.4, 2016. URL `http://hdl.handle.net/11234/1-1827`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague.

[57] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, et al. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, 2016.

[58] F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51, 2003.

[59] O. Owoputi, B. OConnor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL-HLT*, pages 380–390, 2013.

[60] S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. In N. Calzolari, K. Choukri, T. Declerck, M. U. Doğan, B. Maegaard, J. Mariani, J. Odijk, and S. Piperidis, editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2089–2096, Istanbul, Turkey, 2012. European Language Resources Association (ELRA).

[61] S. Poplack. Sometimes ill start a sentence in spanish y termino en español: toward a typology of code-switching1. *Linguistics*, 18(7-8):581–618, 1980.

[62] S. Rijhwani, R. Sequiera, M. Choudhury, K. Bali, and C. S. Maddila. Estimating code-switching on twitter with a novel generalized word-level language detection technique. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1971–1982, 2017.

[63] K. Rudra, S. Rijhwani, R. Begum, K. Bali, M. Choudhury, and N. Ganguly. Understanding language preference for expression of opinion and sentiment: What do hindi-english speakers do on twitter? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1131–1141, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1121`.

[64] H. Sajjad, A. Fraser, and H. Schmid. An algorithm for unsupervised transliteration mining with an application to word alignment. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 430–439. Association for Computational Linguistics, 2011.

[65] H. Sajjad, A. Fraser, and H. Schmid. A statistical model for unsupervised and semi-supervised transliteration mining. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 469–477. Association for Computational Linguistics, 2012.

[66] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[67] A. Sharma, S. Gupta, R. Motlani, P. Bansal, M. Shrivastava, R. Mamidi, and D. M. Sharma. Shallow parsing pipeline - hindi-english code-mixed social media text. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1340–1345, San Diego, California, June 2016. Association for Computational Linguistics.

[68] T. Solorio and Y. Liu. Learning to predict code-switching points. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 973–981. Association for Computational Linguistics, 2008.

[69] T. Solorio and Y. Liu. Part-of-speech tagging for english-spanish code-switched text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1051–1060. Association for Computational Linguistics, 2008.

[70] T. Solorio, E. Blair, S. Maharjan, S. Bethard, M. Diab, M. Gonheim, A. Hawwari, F. AlGhamdi, J. Hirshberg, A. Chang, and P. Fung. Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code-Switching. EMNLP 2014, Conference on Empirical Methods in Natural Language Processing, Octobe, 2014, Doha, Qatar*, 2014.

[71] M. Straka, J. Hajic, and J. Straková. Udpipe: Trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016), Portorož, Slovenia*, pages 4290–4297, 2016.

[72] D. Suwito. Sosiolinguistik teori & problema, 1983.

[73] Y. Vyas, S. Gella, J. Sharma, K. Bali, and M. Choudhury. Pos tagging of english-hindi code-mixed social media content. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 14, pages 974–979, 2014.

[74] H. Wang, Y. Zhang, G. L. Chan, J. Yang, and H. L. Chieu. Universal dependencies parsing for colloquial singaporean english. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1732–1744, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[75] R. Wardhaugh. Introduction to sociolinguistics. 1997, 1986.

[76] H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206, 2003.

[77] D. Zeman. Reusable tagset conversion using tagset drivers. In *LREC*, volume 2008, pages 28–30, 2008.

[78] D. Zeman, M. Popel, M. Straka, J. Hajic, J. Nivre, F. Ginter, J. Luotolahti, S. Pyysalo, S. Petrov, M. Potthast, F. Tyers, E. Badmaeva, M. Gokirmak, A. Nedoluzhko, S. Cinkova, J. Hajic jr., J. Hlavacova, V. Kettnerová, Z. Uresova, J. Kanerva, S. Ojala, A. Missilä, C. D. Manning,

S. Schuster, S. Reddy, D. Taji, N. Habash, H. Leung, M.-C. de Marneffe, M. Sanguinetti, M. Simi, H. Kanayama, V. dePaiva, K. Droganova, H. Martínez Alonso, c. Çöltekin, U. Sulubacak, H. Uszkoreit, V. Macketanz, A. Burchardt, K. Harris, K. Marheinecke, G. Rehm, T. Kayadelen, M. Attia, A. Elkahky, Z. Yu, E. Pitler, S. Lertpradit, M. Mandl, J. Kirchner, H. F. Alcalde, J. Strnadová, E. Banerjee, R. Manurung, A. Stella, A. Shimada, S. Kwak, G. Mendonca, T. Lando, R. Nitisaroj, and J. Li. Conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[79] Y. Zhang and S. Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2008.

[80] Y. Zhang and S. Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics, 2008.

[81] Y. Zhang and J. Nivre. Transition-based dependency parsing with rich non-local features. pages 188–193, 2011.

[82] Y. Zhang and D. Weiss. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566, Berlin, Germany, August 2016. Association for Computational Linguistics.