Universal Dependency Parsing for Hindi-English Code-switching

Irshad Ahmad Bhat
LTRC, IIIT-H,
Hyderabad, India
irshad.bhat@iiit.ac.in

Riyaz Ahmad Bhat
Interaction Labs,
Bangalore, India
rbhat@interactions.com

Manish Shrivastava
LTRC, IIIT-H,
Hyderabad, India
m.shrivastava@iiit.ac.in

Dipti Misra Sharma
LTRC, IIIT-H,
Hyderabad, India
dipti@iiit.ac.in

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 paper, 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 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. We also present normalization and back-transliteration models with a decoding process tailored for code-switching data. 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.

1 Introduction

Code-switching[1] (henceforth CS) is the juxtaposition, within the same speech utterance, of grammatical units such as words, phrases, and clauses belonging to two or more different languages (Gumperz, 1982). The phenomenon is prevalent in multilingual societies where speakers share more than one language and is often prompted by multiple social factors (Myers-Scotton, 1995). 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 (Solorio and Liu, 2008b; Vyas et al., 2014; Çetinoğlu et al., 2016; Gimpel et al., 2011; Owoputi et al., 2013; Kong et al., 2014). 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 (Çetinoğlu et al., 2016). 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.

Given the peculiar nature of CS data, it has been widely studied in linguistics literature (Poplack, 1980; Gumperz, 1982; Myers-Scotton, 1995), and more recently, there has been a surge in studies concerning CS data in NLP as well (Solorio and Liu, 2008a,a; Vyas et al., 2014; Sharma et al., 2016; Rudra et al., 2016; Joshi et al., 2016; Bhat et al., 2017; Chandu et al., 2017; Rijhwani et al.,

[1] Code-mixing is another term in the linguistics literature used interchangeably with code-switching. Both terms are often used to refer to the same or similar phenomenon of mixed language use.

Proceedings of NAACL-HLT 2018, pages 987–998
New Orleans, Louisiana, June 1 - 6, 2018.

2017; Guzmán et al., 2017, and others). 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. In this paper, 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 Hindi-English code-switching data of multilingual Indian speakers from Twitter. Hindi-English code-switching presents an interesting scenario for the parsing community. Mixing among typologically diverse languages will intensify structural variations which will make parsing more challenging. For example, there will be many sentences containing: (1) both SOV and SVO word orders[2], (2) both head-initial and head-final genitives, (3) 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. In this paper, 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 gist of these modules and our overall research contributions are listed as follows:

- back-transliteration and normalization models based on encoder-decoder frameworks with sentence decoding tailored for code-switching data;
- a dependency treebank of Hindi-English code-switching tweets under Universal Dependencies scheme; and
- a neural parsing model which learns POS tagging and parsing jointly and also incorporates knowledge from the monolingual treebanks using neural stacking.

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

2 Preliminary Tasks

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 will see in Results Section. 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 will 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.

2.1 Language Identification

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 1.

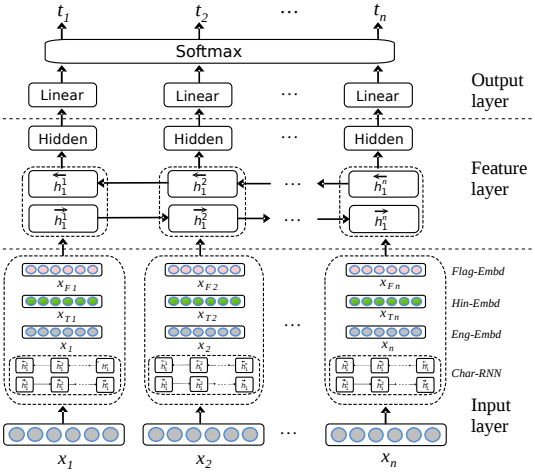

Figure 1: Language identification network

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[3] shared task (728 Facebook comments) on language identification and evaluate it on the datasets from Bhat et al. (2017). We achieved the state-of-the-art performance on both development and test sets (Bhat et al., 2017). The results are shown in Table 1.

Label	Precision	Recall	F1-Score	count
hi	97.76	98.09	97.92	1465
en	96.87	98.83	97.84	1283
ne	94.33	79.17	86.08	168
acro	92.00	76.67	83.64	30
univ	99.71	1.00	99.86	349
average	97.39	97.42	97.36	3295
(Bhat et al., 2017)	-	96.10	-	-

Table 1: Language Identification results on CS test set.

2.2 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 (Luong et al., 2015) with global attention for learning. For Hindi, we train the model on the transliteration pairs (87,520) from the Libindic transliteration project[4] and Brahmi-Net (Kunchukuttan et al., 2015) 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 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 will refer to both normalization and back-transliteration as normalization.

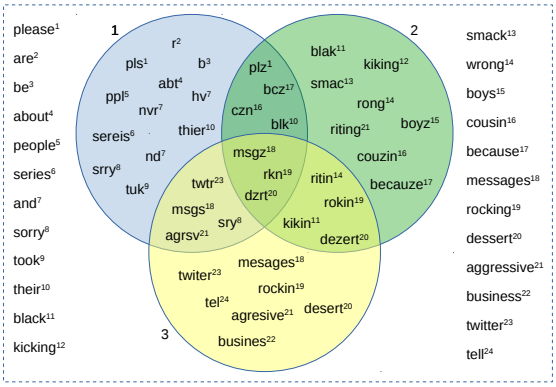

Figure 2: Synthetic normalization pairs generated for a sample of English words using hand crafted rules.

At inference time, our normalization models will 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 will 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 (Heafield et al., 2013). 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 (Dutta et al., 2015). One major prob-

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

[4] https://github.com/libindic/indic-trans

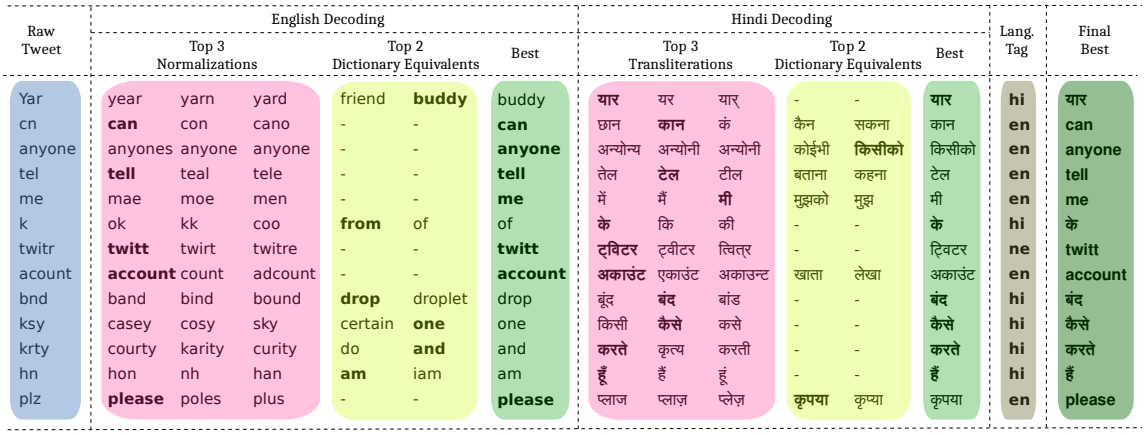

Figure 3: The figure shows a 3-step decoding process for the sentence *"Yar cn anyone tel me k twitr account bnd ksy krty hn plz" (Friend can anyone tell me how to close twitter account please).*

lem 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 might often lead to wrong word selection due to incomplete context, particularly at fragment peripheries. 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.[5] 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 3.

Both of our normalization and back-transliteration systems are evaluated on the evaluation set of Bhat et al. (2017). Results of our systems are reported in Table 3 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.

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

3 Universal Dependencies for Hindi-English

Recently Bhat et al. (2017) provided a CS dataset for the evaluation of their parsing models which they trained on the Hindi and English Universal Dependency (UD) treebanks. We extend this dataset by annotating 1,448 more sentences. Following Bhat et al. (2017) we first sampled CS data from a large set of tweets of Indian language users that we crawled from Twitter using Tweepy[6]–a Twitter API wrapper. We then used a language identification system trained on ICON dataset (see Section 2) 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 dataset, we manually selected 1,448 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 (De Marneffe et al., 2014),

[5] 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.

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

while language tags are assigned based on the tag set defined in (Solorio et al., 2014; Jamatia et al., 2015). 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 use our dataset for training while the development and evaluation sets from Bhat et al. (2017) are used for tuning and evaluation of our models. Since the annotations in these datasets follow version 1.4 of the UD guidelines, we converted them to version 2 by using carefully designed rules. The statistics about the data are given in Table 3.

Data-set	Sentences	Tokens	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: Data Statistics. Dev set is used for tuning model parameters, while Test set is used for evaluation.

4 Dependency Parsing

We adapt Kiperwasser and Goldberg (2016) 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 4 and 6.

4.1 Parsing Algorithm

Our parsing models are based on an arc-eager transition system (Nivre, 2003). The arc-eager system defines a set of configurations for a sentence $w_1, \ldots, w_n$, 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 $S = [ROOT]$, $B = [w_1, \ldots, w_n]$ 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. To derive the parse tree, the arc-eager system defines four types of transitions (t): Shift, Left-Arc, Right-Arc, and Reduce.

We use the training by exploration method of Goldberg and Nivre (2012) for decoding a transition sequence which helps in mitigating error propagation at evaluation time. We also use pseudo-projective transformations of Nivre and Nilsson (2005) to handle a higher percentage of non-projective arcs in the CS data (∼2%). We use the most informative scheme of head+path to store the transformation information.

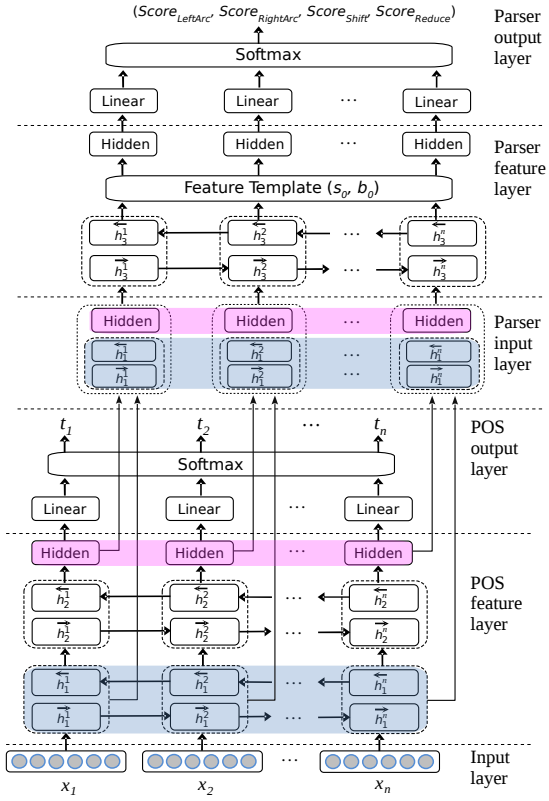

Figure 4: POS tagging and parsing network based on stack-propagation model proposed in (Zhang and Weiss, 2016).

4.2 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). 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 (2016), 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 4. Following Zhang and Weiss (2016), we refer to this model as stack-prop.

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.

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 (Zhang and Weiss, 2016). 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.

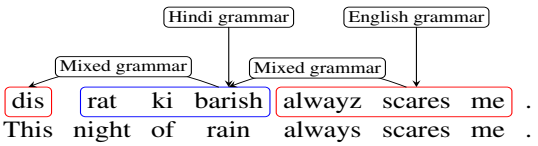

Figure 5: Code-switching tweet showing grammatical fragments from Hindi and English.

4.3 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 will happen quite often as Hindi and English are typologicalls very diverse (see Figure 5).

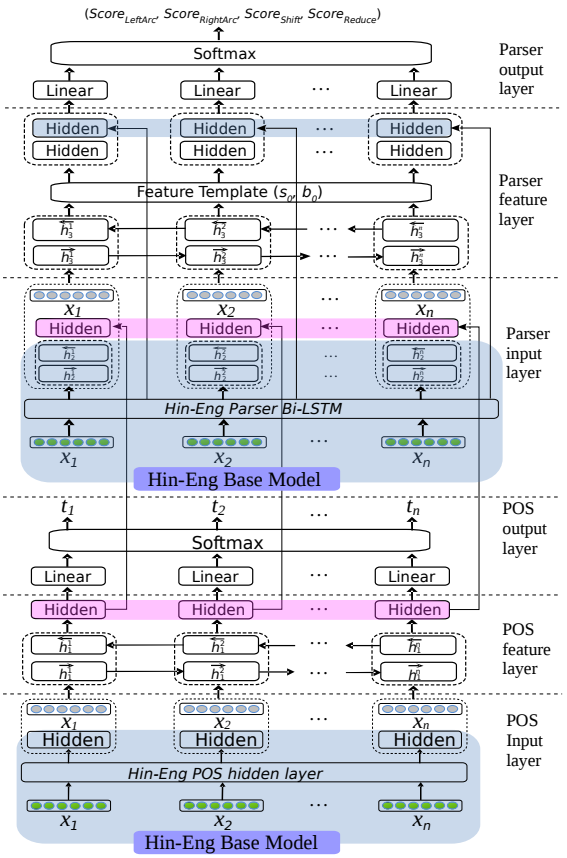

Figure 6: Neural Stacking-based parsing architecture for incorporating monolingual syntactic knowledge.

As we discussed above, we adapted feature-level neural stacking (Zhang and Weiss, 2016; Chen et al., 2016) 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, Wang et al. (2017) used neural stacking for injecting syntactic knowledge of English into a graph-based Singlish parser which lead to significant improvements in parsing performance. Unlike Wang et al. (2017), our base stacked models will allow us to transfer the POS tagging knowledge as well along the parse tree knowledge.

As shown in Figure 6, 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 4. 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.

5 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. For tuning and evaluation, we use the development and evaluation sets from Bhat et al. (2017). 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.

5.1 Hyperparameters

Word Representations For language identification, 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 dictionary flags. We use 32-dimensional character embeddings for all the three models and 32-dimensional POS tag embeddings for pipelined parsing models. 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 (Mikolov et al., 2013). We use the projection algorithm of Artetxe et al. (2016) to transform the Hindi and English monolingual embeddings into same semantic space using a bilingual lexicon ($\sim$63,000 entries). The bilingual lexicon is extracted from ILCI and Bojar Hindi-English parallel corpora (Jha, 2010; Bojar et al., 2014). For normalization models, we use 32-dimensional character embeddings uniformly initialized within a range of $[-0.1, +0.1]$.

Hidden dimensions The POS tagger specific Bi-LSTMs have 128 cells while the parser specific Bi-LSTMs have 256 cells. The Bi-LSTM in the language identification model has 64 cells. The character Bi-LSTMs have 32 cells for all three models. The hidden layer of MLP has 64 nodes for the language identification network, 128 nodes for the POS tagger and 256 nodes for the parser. We use hyperbolic tangent as an activation function in all tasks. In the normalization models, we use single layered Bi-LSTMs with 512 cells for both encoding and decoding of character sequences.

Learning For language identification, 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 (Saxe et al., 2013). We set the dropout rate to 30% for POS tagger and parser Bi-LSTM and MLP hidden states while for language identification network we set the dropout to 50%. All three models are trained for up to 100 epochs, with early stopping based on the development set.

In case of normalization, 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.

Language identification, POS tagging and parsing code is implemented in DyNet (Neubig et al., 2017) and for normalization without decoding, we use Open-NMT toolkit for neural machine translation (Klein et al., 2017). All

the code is available at https://github.com/irshadbhat/nsdp-cs and the data is available at https://github.com/CodeMixedUniversalDependencies/UD_Hindi_English.

6 Results

In Table 4, 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[7] 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 5). As compared to the best system in CoNLL 2017 Shared Task on Universal Dependencies (Zeman et al., 2017; Dozat et al., 2017), 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 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.

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**
(Bhat et al., 2017)	74.16	64.11	66.18	54.40

Table 4: Accuracy of different parsing models on the evaluation set. POS tags are jointly predicted with parsing. LID = Language tag, TRN = Transliteration/normalization.

Table 6 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.

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

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 5: 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**
(Bhat et al., 2017)	86.00		85.30	

Table 6: POS tagging accuracies of different models on CS evaluation set. SP = stack-prop.

Pipeline vs Stack-prop Table 7 summarizes the parsing results of our pipeline models which use predicted POS tags as input features. As compared to our stack-prop models (Table 4), 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 7: 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.

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 8. 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 mod-

erate drop in accuracy (1.4% LAS points) caused due to normalization errors (see results in Table 4 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 8: Impact of normalization and back-transliteration on POS tagging and parsing models.

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 9. Cross-lingual embeddings have brought around ∼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 will 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 9: Impact of monolingual and cross-lingual embeddings on stacking model performance.

7 Conclusion

In this paper, we have presented a dependency parser designed explicitly for Hindi-English CS data. The parser uses neural stacking architecture of Zhang and Weiss (2016) and Chen et al. (2016) for learning POS tagging and parsing and for knowledge transfer from Bilingual models trained on Hindi and English UD treebanks. We have also presented normalization and back-transliteration models with a decoding process tailored for CS data. 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.

References

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016. 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.

Irshad Bhat, Riyaz A. Bhat, Manish Shrivastava, and Dipti Sharma. 2017. 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*. Association for Computational Linguistics, Valencia, Spain, pages 324–330.

Ondřej Bojar, Vojtěch Diatka, Pavel Rychlý, Pavel Straňák, Vít Suchomel, Aleš Tamchyna, and Daniel Zeman. 2014. HindEnCorp - Hindi-English and Hindi-only Corpus for Machine Translation. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. European Language Resources Association (ELRA), Reykjavik, Iceland.

Özlem Çetinoğlu, Sarah Schulz, and Ngoc Thang Vu. 2016. Challenges of computational processing of code-switching. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*. Association for Computational Linguistics, Austin, Texas, pages 1–11.

Khyathi Raghavi Chandu, Manoj Chinnakotla, Alan W Black, and Manish Shrivastava. 2017. Webshodh: A code mixed factoid question answering system for web. In *International Conference of the Cross-Language Evaluation Forum for European Languages*. Springer, pages 104–111.

Hongshen Chen, Yue Zhang, and Qun Liu. 2016. Neural network for heterogeneous annotations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 731–741.

Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. volume 14, pages 4585–92.

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. 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.

Sukanya Dutta, Tista Saha, Somnath Banerjee, and Sudip Kumar Naskar. 2015. Text normalization in code-mixed social media text. In *Proceedings of the*

2nd International Conference on Recent Trends in Information Systems (ReTIS).

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. 2011. 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*. Association for Computational Linguistics, pages 42–47.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics*. pages 959–976.

John J Gumperz. 1982. *Discourse strategies*, volume 1. Cambridge University Press.

Gualberto Guzmán, Joseph Ricard, Jacqueline Serigos, Barbara E Bullock, and Almeida Jacqueline Toribio. 2017. Metrics for modeling code-switching across corpora. *Proceedings of Interspeech 2017* pages 67–71.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H Clark, and Philipp Koehn. 2013. Scalable modified kneser-ney language model estimation. In *ACL (2)*. pages 690–696.

Anupam Jamatia, Björn Gambäck, and Amitava Das. 2015. Part-of-speech tagging for code-mixed english-hindi twitter and facebook chat messages. page 239.

Girish Nath Jha. 2010. 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)*.

Aditya Joshi, Ameya Prabhu, Manish Shrivastava, and Vasudeva Varma. 2016. 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*. The COLING 2016 Organizing Committee, Osaka, Japan, pages 2482–2491.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*. https://doi.org/10.18653/v1/P17-4012.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, and Chris Dyer. 2014. A dependency parser for tweets. In *Proceedings of Conference on Empirical Methods In Natural Language Processing (EMNLP)*. volume 1001, page 1012.

Anoop Kunchukuttan, Ratish Puduppully, and Pushpak Bhattacharyya. 2015. Brahmi-net: A transliteration and script conversion system for languages of the indian subcontinent.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* .

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

Carol Myers-Scotton. 1995. *Social motivations for codeswitching: Evidence from Africa*. Oxford University Press.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* .

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

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*.

Olutobi Owoputi, Brendan OConnor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL-HLT*. pages 380–390.

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

Shruti Rijhwani, Royal Sequiera, Monojit Choudhury, Kalika Bali, and Chandra Shekhar Maddila. 2017. 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.

Koustav Rudra, Shruti Rijhwani, Rafiya Begum, Kalika Bali, Monojit Choudhury, and Niloy Ganguly.

2016. 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*. Association for Computational Linguistics, Austin, Texas, pages 1131–1141. https://aclweb.org/anthology/D16-1121.

Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120* .

Arnav Sharma, Sakshi Gupta, Raveesh Motlani, Piyush Bansal, Manish Shrivastava, Radhika Mamidi, and Dipti M. Sharma. 2016. 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*. Association for Computational Linguistics, San Diego, California, pages 1340–1345.

Thamar Solorio, Elizabeth Blair, Suraj Maharjan, Steve Bethard, Mona Diab, Mahmoud Gonheim, Abdelati Hawwari, Fahad AlGhamdi, Julia Hirshberg, Alison Chang, and Pascale Fung. 2014. 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*.

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

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

Yogarshi Vyas, Spandana Gella, Jatin Sharma, Kalika Bali, and Monojit Choudhury. 2014. 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.

Hongmin Wang, Yue Zhang, GuangYong Leonard Chan, Jie Yang, and Hai Leong Chieu. 2017. Universal dependencies parsing for colloquial singaporean english. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 1732–1744.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinkova, Jan Hajic jr., Jaroslava Hlavacova, Václava Kettnerová, Zdenka Uresova, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria dePaiva, Kira Droganova, Héctor Martínez Alonso, Çağr Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonca, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. 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*. Association for Computational Linguistics, Vancouver, Canada, pages 1–19.

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

A Supplemental Material

A.1 Example Annotations from our CS Treebank