A House United: Bridging the Script and Lexical Barrier between Hindi and Urdu

Riyaz Ahmad Bhat **Irshad Ahmad Bhat** **Naman Jain** **Dipti Misra Sharma**
LTRC, IIIT-Hyderabad, India
{riyaz.bhat,irshad.bhat,naman.jain}@research.iiit.ac.in, dipti@iiit.ac.in

Abstract

In Computational Linguistics, Hindi and Urdu are not viewed as a monolithic entity and have received separate attention with respect to their text processing. From part-of-speech tagging to machine translation, models are separately trained for both Hindi and Urdu despite the fact that they represent the same language. The reasons mainly are their divergent literary vocabularies and separate orthographies, and probably also their political status and the social perception that they are two separate languages. In this paper, we propose a simple but efficient approach to bridge the lexical and orthographic differences between Hindi and Urdu texts. With respect to text processing, addressing the differences between their texts would be beneficial in the following ways: (a) instead of training separate models, their individual resources can be augmented to train single, unified models for better generalization, and (b) their individual text processing applications can be used interchangeably under varied resource conditions.

To remove the script barrier, we learn accurate statistical transliteration models which use sentence-level decoding to resolve word ambiguity. Similarly, we learn cross-register word embeddings from the harmonized Hindi and Urdu corpora to nullify their lexical divergences. As a proof of the concept, we evaluate our approach on the Hindi and Urdu dependency parsing under two scenarios: (a) resource sharing, and (b) resource augmentation. We demonstrate that a neural network-based dependency parser trained on augmented, harmonized Hindi and Urdu resources performs significantly better than the parsing models trained separately on the individual resources. We also show that we can achieve near state-of-the-art results when the parsers are used interchangeably.

1 Introduction

Hindi and Urdu are spoken primarily in northern India and Pakistan and together constitute the third largest language spoken in the world.[1] They are two standardized registers of what has been called the Hindustani language, which belong to the Indo-Aryan language family. Masica (1993) explains that, while they are different languages officially, they are not even different dialects or sub-dialects in a linguistic sense; rather, they are different literary styles based on the same linguistically defined sub-dialect. He further explains that at the colloquial level, Hindi and Urdu are nearly identical, both in terms of core vocabulary and grammar. However, at formal and literary levels, vocabulary differences begin to loom much larger (Hindi drawing its higher lexicon from Sanskrit and Urdu from Persian and Arabic) to the point where the two styles/languages become mutually unintelligible. In written form, not only the vocabulary but the way Urdu and Hindi are written makes one believe that they are two separate languages. They are written in separate orthographies, Hindi being written in Devanagari, and Urdu in a modified Perso-Arabic script. Given these differences in script and vocabulary, Hindi and Urdu are

[1] see http://www.ethnologue.com/statistics/size and https://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 397–408, Osaka, Japan, December 11-17 2016.

socially and even officially considered two separate languages. These apparent divergences have also led to parallel efforts for resource creation and application building in computational linguistics. The Hindi-Urdu treebanking project is one such example where the influence of differences between Hindi and Urdu texts have led to the creation of separate treebanks for Hindi and Urdu (Bhatt et al., 2009; Bhat et al., 2015). However, pursuing them separately in computational linguistics makes sense. If the two texts differ in form and vocabulary they can not be processed with same models unless the differences are accounted for and addressed. In this paper, we aim to remove these differences between Hindi and Urdu texts. We learn accurate machine transliteration models for the common orthographic representation of their texts. To resolve their lexical divergences, we learn cross-register word embeddings from the harmonized Hindi and Urdu corpora. So as to evaluate our approach, we empirically demonstrate the impact of text harmonization on the dependency parsing of both Hindi and Urdu under varied supervised training conditions. We show that a neural network-based parser trained on cross-register word embeddings sets the new benchmark for dependency parsing of Hindi and Urdu. A summary of our overall contributions is provided as follows:

- **Common Representation:** To remove the orthographic differences between Hindi and Urdu texts, we evaluate Devanagari and Perso-Arabic scripts for their common representation. We propose accurate statistical transliteration models which use sentence-level decoding on n-best transliterations to resolve word ambiguity. We empirically show that the Devanagari script is better suited for automatic text processing of both Hindi and Urdu. We show significant gains in accuracy in different modules of Hindi and Urdu dependency parsing pipeline when the training and evaluation data are represented in Devanagari instead of Perso-Arabic.
- **Resource Sharing and Augmentation:** To facilitate resource sharing and augmentation, we use statistical transliteration to harmonize the orthographic differences between Hindi and Urdu texts, while their lexical divergences are resolved by learning cross-register word embeddings. We augment their harmonized treebanks and train neural network-based parsing models using different supervised domain adaptation techniques. We empirically show that our augmented models perform significantly better than the models trained separately on individual treebanks. Moreover, we also demonstrate that the individual parsing models trained on harmonized Hindi and Urdu resources can be used interchangeably to parse both Hindi and Urdu texts and give near state-of-the-art results.

2 Experimental Setup

To experiment on resource sharing and augmentation between Hindi and Urdu, we need to mitigate their orthographic and lexical differences. To that end, we propose a simple approach which uses machine transliteration and distributional similarity-based methods (see §3 and §4). After script harmonization, we learn distributed word representations to project Hindi and Urdu lexicon in the same distributional space. To make effective use of these word representations, we employ the non-linear neural network architecture for transition-based dependency parsing proposed by Chen and Manning (2014). We use a similar architecture for sequence labeling as well.

2.1 Parsing Models

Our parsing model is based on transition-based dependency parsing paradigm (Nivre, 2008). Particularly, we use an arc-eager transition system, which is one of the famous transition-based parsing systems. Arc-eager algorithm defines four types of transitions to derive a parse tree namely: 1) Shift, 2) Left-Arc, 3) Right-Arc, and 4) Reduce. To predict these transitions, a classifier is employed. We follow Chen and Manning (2014) and use a non-linear neural network to predict these transitions for any parser configuration. The neural network model is the standard feed-forward neural network with a single layer of hidden units. 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 and apply dropout (Chen and Manning, 2014).

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. For each node, we use the distributed representation of its lexical form, POS tag, chunk tag and/or dependency label. We use the Hindi and Urdu monolingual corpora to learn the distributed representation of the lexical units. The Hindi monolingual data contains around 40M raw sentences, while the Urdu data is comparatively smaller and contains around 6M raw sentences. The distributed representation of non-lexical units such as POS, chunk and dependency labels are randomly initialized within a range of -0.01 to +0.01 (Chen and Manning, 2014).

To decode a transition sequence, we use dynamic oracle recently proposed by Goldberg and Nivre (2012) instead of the vanilla static oracle. Dynamic oracle allows training by exploration that helps to mitigate the effect of error propagation. We use the same value for the exploration hyperparameter as suggested by Goldberg and Nivre (2012).

2.2 Sequence Labeling Models

For the training of POS tagging and chunking models, we use a similar neural network architecture as discussed above. Unlike Collobert et al. (2011), 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. For POS tagging, we use second-order structural features, 2 words to either side of the current word, and last 3 letters of the current word. Similarly, for chunking we use POS tags of the current word and the 2 words surrounding it on the either side, in addition to the features used for POS tagging.

2.3 Hindi and Urdu Treebanks

We use the annotations in the Hindi and Urdu treebanks (henceforth HDTB and UDTB) for conducting all the experiments. Both treebanks are multi-layered and multi-representational (Bhatt et al., 2009). They contain three layers of annotation namely **dependency structure (DS)** for annotation of modified-modifier relations, **PropBank-style annotation** for predicate-argument structure, and an independently motivated **phrase-structure annotation**. For our experiments, we only need annotations in the first layer of the treebanks i.e., annotations in the DS layer. Dependency Structure involves dependency analysis based on the Pāṇinian Grammatical framework (Bharati et al., 1995). In addition to dependency analysis, the sentence annotation also includes morphological analysis, part-of-speech (POS) tagging and chunking. POS tagging and chunking are based on Indian Language Machine Translation (ILMT) guidelines (Bharati et al., 2006). There are around 32 POS tags and 11 chunk tags used in the treebanks, while the dependency labels are around 82.

We split the treebank data with a ratio of 80:10:10 for training, testing and tuning separate models for POS tagging, chunking and parsing. For both treebanks, the internal structure of annotation files is preserved. However, we randomly distribute the files across training, testing and development sets. Each document or annotation file mainly contains newswire articles. Statistics about the data are provided in Table 1.

Count of	Hindi			Urdu		
	Training	Testing	Development	Training	Testing	Development
Tokens	3,47,744	43,556	43,556	1,53,317	19,065	19,065
Chunks	1,87,029	23,418	23,417	72,319	9,010	9,010
Sentences	16,629	2,077	2,077	5,432	677	677

Table 1: Statistics of training, testing and development sets used in all the experiments reported in this paper.

3 Common Representation

Despite their similarities, we can not use Hindi text processing tools for Urdu as such and vice versa as they are written in two different scripts. To address this problem, we need to represent both Hindi and Urdu texts in a single script. For this purpose, we can either use Devanagari or Perso-Arabic script. We can transliterate Hindi texts in Devanagari to Perso-Arabic or Urdu texts in Perso-Arabic to Devanagari. Either way, transliteration between these two scripts is a non-trivial task. There are genuine cases of character ambiguity due to one-to-many character mappings in both directions of transliteration. A detailed description of the challenges in Hindi-Urdu transliteration can be found in the works of Malik et al. (2008) and Lehal and Saini (2014). In addition to character ambiguity, Perso-Arabic to Devanagari

transliteration has to deal with missing short vowels in Urdu texts also. In Urdu writing, short vowels are hardly represented, even though the Perso-Arabic script has the provision for their representation. A major drawback of dropping short vowels in Urdu writing is that it generates homographs. For example, without an appropriate short vowel on the first letter, ہوا could mean 'air' (ہَوا) or 'become' (ہُوا) depending on the context. These homographs would lead to ambiguity in the Devanagari script. There would be more than one genuine Devanagari representation for such homographs, since Devanagari represents each phoneme uniquely and explicitly. Usually word-level transliteration models do not deal with word ambiguity and leave it unresolved. However, we need our transliteration model to pick a transliteration that best fits the sentential context.

It should be noted that both Devanagari and Perso-Arabic scripts are a natural choice for the common representation. The use of a third script (e.g. Roman script) for this purpose would be computationally expensive, as we need to transliterate both Hindi and Urdu resources. Moreover, the transliteration errors would also double. More importantly, if we choose a third script, we have to manually develop a reasonably-sized corpus of transliteration pairs for training the transliteration models. On the other hand, transliteration pairs in Devanagari and Perso-Arabic scripts can be automatically extracted from the corpora available in these scripts (see §3.1.1 for more details).

To measure the suitability of both scripts for the common representation of Hindi and Urdu texts, we perform extrinsic evaluation on the dependency parsing pipeline which involves POS tagging, chunking and dependency parsing. The script that maximizes the accuracy across the pipeline would imply its feasibility for uniformly representing the Hindi and Urdu texts for computational purposes.

3.1 Hindi-Urdu Transliteration

Hindi and Urdu transliteration has received a lot of attention from the NLP research community of South Asia (Malik et al., 2008; Lehal and Saini, 2012; Lehal and Saini, 2014). It has been seen to break the barrier that makes the two look different. Most of the existing works on Hindi-Urdu transliteration have considered basic rule-based models which use character tables coupled with a set of heuristics to resolve ambiguous mappings. Statistical approaches have hardly been explored (Sajjad et al., 2011). Unlike rule-based systems, statistical approaches are more robust and efficient. Among data-driven approaches, machine learning methods like noisy-channel model and structured prediction algorithms have been widely used for machine transliteration (Knight and Graehl, 1998; Zelenko and Aone, 2006). In this work, we model Hindi-Urdu transliteration as a structured prediction problem using a linear model. We use the structured perceptron (DHMM) of Collins (Collins, 2002) to learn the parameters of our transliteration model. Given an input training data of aligned character sequences $D = d_1 ... d_n$, a vector feature function $\vec{f}(d)$, and an initial weight vector $\vec{w}$, the algorithm performs two steps for each training example $d_i \in D$: (1) **Decode**: $\hat{t} = \underset{t_1 \cdots t_n}{\arg\max}(\vec{w} \cdot \vec{f}(d))$, and (2) **Update**: $\vec{w} = \vec{w} + \vec{f}(d) - \vec{f}(\hat{t})$. We use Viterbi-search for decoding in case of Devanagari to Perso-Arabic transliteration, while we use beam-search for Perso-Arabic to Devanagari transliteration to decode the best letter sequence in the target script. The latter is used to extract n-best transliterations for resolving word ambiguity.

In case of Perso-Arabic to Devanagari transliteration, to resolve the word ambiguity as discussed above, we perform sentence-level decoding on the n-best transliterations from the perceptron model. We use a noisy channel model and exact Viterbi search to find the most likely Hindi (Devanagari) sentences. The noisy-channel model can be formally defined as follows:

$$\mathbf{h}^* = \arg\max \; p(h) \times p(h|u) \tag{1}$$

$p(h)$ is the language model score which gives a prior distribution over the most likely sentences in Hindi and $p(h|u)$ is the perceptron score which indicates how likely the Hindi (Devanagari) sentence h is a word by word transliteration of the Urdu sentence u. Since $p(h|u)$ is not a probability score, we assign uniform probabilities to all the transliteration options. Thus redefining our model without $p(h|u)$ as:

$$\mathbf{h}^* = \arg\max \; p(h) \tag{2}$$

Thus, our model only relies on language model to find the best sentence from the n-best transliterations. We use trigram language model learned from 40M multi-domain Hindi corpus with Kneser-Ney smoothing. Here, it should be noted that it is plausible to score Urdu sentences in Devanagari using a language model trained on Hindi data, since there is a considerable overlap in the Hindi and Urdu grammar and vocabulary.

3.1.1 Transliteration Pair Extraction and Character Alignment

Like any other supervised machine learning approach, supervised machine transliteration requires a strong list of transliteration pairs to learn the model parameters. We use the sentence aligned ILCI Hindi-Urdu parallel corpora (Jha, 2010) to extract the transliteration pairs. Initially, the parallel corpus is word-aligned using GIZA++ (Och and Ney, 2003). 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 54,035 translation pairs from the parallel corpus of 50,000 sentences. To further complement the translation pairs, we also extracted 66,668 pairs from IndoWordNet (Narayan et al., 2002) synset mappings.[2] 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-Urdu character mappings presented in (Lehal and Saini, 2014). We compute the levenshtein distance between the translation pairs based on insertion, deletion and replace operations. 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$0.1 are considered as transliteration pairs. Using this procedure, we extracted 21,972 unique transliteration pairs from the Hindi-Urdu parallel corpus and 24,614 unique transliteration pairs from the Hindi-Urdu synsets. After mining the transliteration pairs, we character align them using Giza++ for training and testing the transliteration models.

3.1.2 Experiments and Results

We train two structured perceptron models on the transliteration pairs discussed above. We maintain 80:10:10 data split for training, testing and tuning both models. Additionally, we also manually transliterate 1000 Urdu sentences in Devanagari script to tune and evaluate our noisy-channel model which we use on top of the Perso-Arabic to Devanagari transliteration system to resolve word ambiguity. The parameters such as number of training iterations, order of ngram context, number of transliterations for noisy-channel model are tuned on the respective development sets. We found that the top 5 transliterations gave the best results.

To compare our results with the existing systems, we choose HUMT, Malerkotla (MAL) and SANGAM available on the Internet,[3] while choosing the SMT-based transliteration as a baseline. The baseline model is a phrase-based machine translation system (PSMT) for transliteration built using the Moses toolkit.[4] We train the system with the default settings with the distortion limit set to 0. We list the performance of all these systems in Table 2 for comparison. The performance of our noisy-channel models is reported in Table 3.

System	Devanagari → Perso-Arabic	Perso-Arabic → Devanagari
PSMT	96.23%	74.30%
HUMT	90.34%	40.75%[5]
MAL	93.78%	78.23%
SANGAM	97.38%	87.56%
DHMM	**98.03%**	**88.03%**

Table 2: Comparison of type-level accuracies of the available systems on the Internet with our system.

[2] http://www.cfilt.iitb.ac.in/~sudha/bilingual_mapping.tar.gz

[3] http://www.sanlp.org/HUMT/HUMT.aspx, translate.malerkotla.co.in/ and http://sangam.learnpunjabi.org/

[4] http://www.statmt.org/moses/

[5] HUMT system performed worst on Perso-Arabic to Devanagari transliteration because it relies on diacritical marks in Urdu texts for correct transliteration.

System	Testing Set	Development Set
DHMM	94.21%	94.63%
+Noisy-channel Model	**96.37%**	**96.72%**

Table 3: Token-level performance of noisy-channel model in resolving word ambiguity in Perso-Arabic to Devanagari transliteration.

As shown in Table 2, we have established a new best system for the transliteration of Hindi and Urdu texts bidirectionally. Our Devanagari to Perso-Arabic system outperforms SANGAM by 0.65%. There is also an improvement of 0.47% over SANGAM in case of Perso-Arabic to Devanagari transliteration. Out of the two transliteration models, Perso-Arabic to Devanagari performs worst because of the missing vowels in the Urdu texts. Our noisy-channel model improved the results of our basic perceptron model for Perso-Arabic to Devanagari transliteration. It improved the accuracy by an absolute 2% on the test set. This clearly shows how often homographs are generated due to missing vowels in Perso-Arabic script.

3.1.3 Extrinsic Evaluation

As we already mentioned, the script that fares well in an extrinsic evaluation on a dependency parsing pipeline will be chosen for the common representation of Hindi and Urdu texts. For training and evaluation in each script, we made two copies of Hindi and Urdu training and evaluation sets. For each module, we trained two models–one in each script. In Table 4, we report the results on the respective evaluation sets in both scripts. Besides using predicted POS and chunk tag features for chunking and parsing, we also conduct experiments using the gold features. This is important for capturing the impact of orthographic representation on each module independently.

Data	POS tagging	Chunking	Parsing (LAS)
		Gold Features	
Hindi Devanagari	**96.48**	**98.40**	91.70
Hindi Perso-Arabic	96.00	98.35	91.52
Urdu Perso-Arabic	93.13	96.62	88.08
Urdu Devanagari	**93.42**	**96.65**	88.38
		Predicted Features	
Hindi Devanagari	-	**97.84**	**88.32**
Hindi Perso-Arabic	-	97.58	87.95
Urdu Perso-Arabic	-	95.60	81.67
Urdu Devanagari	-	**96.03**	**82.28**

Table 4: Performance of different modules of a dependency parsing pipeline trained and evaluated on Hindi and Urdu treebank data in Devanagari and Perso-Arabic scripts.

The results presented in Table 4 clearly favor Devanagari script over Perso-Arabic for the orthographic representation of Hindi and Urdu texts. For all the modules, accuracy decreases in Perso-Arabic script, while there is a significant increase in accuracy in Devanagari script. The reason mainly lies in the fact that Devanagari represents information explicitly while Perso-Arabic script does not. Devanagari is a phonetic script which represents phonemes uniquely and explicitly. Perso-Arabic on the other hand is not a phonetic script. The better results in Devanagari script clearly would, therefore, be due to the less ambiguous representation of words in this script as can be seen in Table 5. The table represents the impact of transliteration on lexical and POS-tag merging.

Script	**Lexical Merging** (%)	**Tag Ambiguity** (%)
Hindi in Perso-Arabic	-11.87	+3.12
Urdu in Devanagari	+11.26	-2.51

Table 5: Comparison of lexical and POS-tag merging rates in Devanagari and Perso-Arabic transliterated data.

We define *lexical merging rate* as the amount of percentage drop in the size of the vocabulary after transliteration. Similarly *tag ambiguity* captures the increase in ambiguity of POS categories due to lexical merging. Both *lexical merging* and *tag ambiguity* rates are higher in the case of Devanagari to Perso-Arabic transliteration, which explains the drop in accuracies when the models are trained and evaluated on data represented in Perso-Arabic script. There is an 11% drop in type counts when we transliterate HDTB data in Perso-Arabic, while on the other hand, there is a similar percentage increase in

vocabulary when UDTB in represented in Devanagari. Interestingly, not all the lexical merging/expansion leads to/resolves syntactic ambiguity. In HDTB, Perso-Arabic script created 3% homographs which are syntactically ambiguous, while Devanagari scripts resolves ambiguity for around 3% words in UDTB. Given these statistics, it seems reasonable to conclude that Devanagari is better suited for automatic text processing of Hindi and Urdu texts. However, it is also interesting that the POS models are able to resolve most of the syntactic ambiguity created by Perso-Arabic script.

4 Resource Sharing and Augmentation

In the above section, we empirically showed that the Devanagari script can serve as a common representation for both Hindi and Urdu texts without harming the performance of text processing applications such as a POS tagger and a parser. Given the fact that Hindi and Urdu are syntactically or grammatically similar, resource sharing and augmentation should become feasible just by removing the script barrier. A common orthographic representation would, however, only affect that part of Hindi and Urdu vocabularies which is shared. It will not fill the lexical gaps. It is the lexical differences between Hindi and Urdu texts that leave them mutually unintelligible (Masica, 1993). The severity of lexical differences can be clearly seen by comparing the OOV rates of Hindi and Urdu evaluation sets. As shown in Table 6, almost half of the tokens in both Hindi and Urdu development sets are missing from the Urdu and Hindi training sets respectively. Such excessive OOV rates would worsen the problem of lexical data sparsity for any statistical model.

Training	Development	OOV (%)
Hindi	Urdu	**51.6**
Urdu	Urdu	15.40
Urdu	Hindi	**62.76**
Hindi	Hindi	22.06

Table 6: OOV rates of Hindi and Urdu development sets. Both Hindi and Urdu data sets are represented in Devanagari.

Lexical data sparseness is considered as one of the major challenges in tackling the problem of data sparsity in data-driven approaches to natural language processing. A common approach to bridge the lexical gaps between the source and the target data is to use distributional similarity-based methods. The distributional similarity methods exploit Harris' distributional hypothesis which states that words that occur in the same contexts tend to have similar meanings (Harris, 1954). To address the lexical differences between Hindi and Urdu, distributional similarity-based methods seem as an appropriate choice. Consider a case of **pratikshā** and **intizār**, both words are semantic equivalents and have the same meaning i.e., **wait**. **pratikshā** is a Sanskrit word used in Hindi texts, while **intizār** is its Perso-Arabic equivalent used in Urdu texts. In both Hindi and Urdu, **pratikshā** and **intizār** form complex predicates with similar light verbs. One such complex predicate is **pratikshā/intizār kar** 'wait do'. The complex predicate takes a genitive-marked theme argument, licenses ergative case on its agentive argument in perfective aspect and can take similar tense, aspect and modal auxiliaries. Even though, **pratikshā** and **intizār** are different word forms, they have identical syntactic distributions which could be used as an approximation of their semantic similarity.

To capture the similarity between Sanskrit and Perso-Arabic words in Hindi and Urdu vocabularies, we could apply distributional similarity methods on the union of harmonized (represented in same script) Hindi and Urdu corpora. Augmenting source domain corpus with the target domain data to learn distributional representation of words is a common practice to address lexical sparseness encountered in domain adaptation tasks (Candito et al., 2011; Plank and Moschitti, 2013). We could also use bilingual word clustering or word embedding approaches which have been used to address the loss of lexical information in delexicalized parsing (Täckström et al., 2012; Xiao and Guo, 2014). In case of Hindi and Urdu, the former approach is more simple and direct way to capture the distributional similarity. The similar grammar and partially shared vocabularies would ensure semantically similar words of Hindi and Urdu are assigned similar distributional representations. The cross-lingual approaches, on the other hand, are computationally complex, while capture the distributional similarity indirectly using a seed bilingual

lexicon.

The distributional similarity can be incorporated in a statistical model either by using word clusters or word vectors (Turian et al., 2010). Similar to Collobert et al. (2011) and Chen and Manning (2014), we represent lexical units in the input layer of our neural network model by the word embeddings instead of one-hot vectors. We augment Hindi monolingual data with the transliterated Urdu data and use word2vec toolkit[6] to learn the word embeddings. The toolkit provides an efficient implementation of the continuous bag-of-words (CBOW) and skip-gram (SG) approaches of Mikolov et al. (2013) to compute distributed representation of words. To learn distributed word representations, we considered context windows of 2 to 5 words to either side of the central element. We varied vector dimensionality within the 50 to 100 range in steps of 10. The model choice, window size and vector dimensionality were selected on the development set in a POS tagging task. The optimal parameters are: learning algorithm as skip-gram, window size as 1, embedding dimensionality as 50 and minimum word frequency as 2.

Once we have represented both Hindi and Urdu texts in same distributional space, we model sharing and augmentation of their resources as a supervised domain adaptation task. Supervised domain adaptation assumes the availability of annotated data in both source and target domains to improve model performance in the target domain. We discuss the best practices in supervised domain adaptation and evaluate their performance for Hindi and Urdu resource sharing and augmentation. An overview of the supervised domain adaptation methods can be found in (Daumé III, 2007), which we repeat here briefly.

- The SRCONLY method trains a single model on the source data while ignoring any target data.
- The TGTONLY method trains a single model only on the target data and acts as the baseline.
- In the ALL method, we simply train our model on the union of the Hindi and Urdu training sets.
- In the WEIGHTED method, we weight the instances of a data set with larger instances to train a single unbiased model. The weights are appropriately chosen by cross-validation.
- In the LININT method, we train SRCONLY and TGTONLY models and linearly interpolate their predictions at the inference time. The interpolation weights are tuned via cross-validation.
- PRIOR method relies on the use of SRCONLY model as a prior on the weights of the target model while training. In our neural network model, we simply replace the regularization term with $\lambda||w - w^s||_2^2$ where w^s is the weight vector from the SRCONLY model.

Among these supervised methods, SRCONLY will address resource sharing, while WEIGHTED, LININT and PRIOR are the methods for resource augmentation.

4.1 Experiments and Results

In any non-linear neural network model, we need to tune a number of hyperparameters for an optimal performance. Tuning these parameters is usually as cumbersome as designing appropriate feature combinations in a linear model. The hyperparameters include number of hidden units, choice of activation function, learning rate, dropout, dimensionality of input units, etc. Furthermore, we had to tune these parameters for each individual task separately. Interestingly, we found that the hyperparameters take similar optimal values for all the three tasks. This could be due to the fact that POS tagging, chunking and parsing are correlated to each other. There is, however, some variation in learning rate and dropout. The optimal parameters include: 200 hidden units, rectilinear activation function, 20 batch size, 20 dimensional non-lexical input units, 0.01 learning rate for POS tagging and chunking and 0.3 for parsing, and 15 training iterations.

After tuning the hyperparameters of our neural network models, we trained multiple models for both Hindi and Urdu to evaluate the performance of each domain adaptation method. To use uniform POS and chunk features, we used 10-fold jackknifing to assign these features to the training data instead of using gold features. For chunking, we used the auto POS features from the best performing POS tagger. Similarly for parsing, we used the best POS and chunk taggers to generate these features. The results of our experiments are reported in Table 7.

[6] https://code.google.com/p/word2vec/

Source	Target	SRCONLY	TGTONLY	ALL	WEIGHTED	LININT	PRIOR
POS tagging							
Hindi	Urdu	89.34	93.42	93.74	93.77	**93.93**	93.52
Urdu	Hindi	86.06	96.48	96.50	96.54	**96.61**	96.22
Chunking							
Hindi	Urdu	92.53	96.03	96.40	96.31	**96.52**	96.13
Urdu	Hindi	90.27	97.64	**97.77**	97.63	97.71	97.44
Dependency Parsing							
Hindi	Urdu	78.93	82.28	82.64	82.53	**82.65**	82.32
Urdu	Hindi	75.12	88.32	**88.41**	88.37	88.39	88.18

Table 7: Results of different supervised domain adaptation methods on Hindi and Urdu resource sharing and augmentation.

For resource augmentation, it is encouraging to note that all the methods led to some improvement over the TGTONLY baseline. Surprisingly, PRIOR method did not perform well in our case. It was one of the best methods reported by Daumé III (2007) when used with a linear model. On the other hand, LININT consistently performed better than other methods. Nevertheless, we achieved substantial improvements in accuracies in all the three tasks for both Hindi and Urdu. Particularly, improvements in Urdu are more prominent. Our augmentation results clearly show that Hindi and Urdu annotations can be complementary to each other. In our case, large number of annotations in HDTB proved useful for parsing of Urdu, which comparatively has a smaller-sized treebank.

The SRCONLY models, under resource sharing experiments, did not perform at par with the TGTONLY baseline models. The performance is still better for Urdu (as a target domain) than it is for Hindi which again could be attributed to the sheer size of the Hindi training data. The larger gaps in accuracies between the SRCONLY and TGTONLY models can be attributed to domain shift problem inherent in data-driven approaches. To empirically verify it, we explored the impact of domain shift on the Hindi tagger and parser trained on newswire data by applying it on Hindi texts other than news articles. For this task we used annotated data from four different domains of Hindi which include cricket, recipes, gadgets and box office. Each domain contains around 500 hundred sentences annotated with POS, chunk and dependency structures. The accuracies of the Hindi parser and tagger on these domains is shown in Table 8. If we compare the accuracies of the Hindi POS tagger and parser on Urdu and the four domains, it appears that the performance on the Urdu test set is comparable. The tagging and parsing accuracies on gadget and recipe data are lower than the accuracies on the Urdu test data. This encourages us to suggest that we can consider Hindi and Urdu as two separate domains representing the same language at least in computational sense and use their tools interchangeably instead of building tools for them separately.

Domains	Parsing			POS tagging
	UAS	LS	LAS	
Cricket	87.90	85.26	79.72	94.02
Box-office	86.64	83.43	78.98	89.55
Gadget	83.27	81.30	75.35	85.93
Recipe	81.12	79.31	71.37	88.95
Urdu	86.13	83.15	78.93	89.34

Table 8: Comparison of parsing and tagging accuracy of Hindi parser and POS tagger on Urdu test data and four different domains of Hindi.

5 Related Work

Recently, there have been several attempts at leveraging the similarity between Hindi and Urdu for sharing and interoperability of their individual resources. Most of the works suggest that Hindi and Urdu resources can be used interchangeably with some modifications (Sinha, 2009; Visweswariah et al., 2010). Sinha (2009) show that an English-Urdu machine translation system can be easily build by using Hindi as a bridge language. Urdu translations of English sentences are derived from the output of an existing English-Hindi MT system. They use lexical mappings between Hindi and Urdu words, lexical and syntactic disambiguation rules, and a transliteration module for converting the MT output to Urdu.

Adeeba and Hussain (2011) used a transliteration-based approach to create an Urdu WordNet from an existing Hindi WordNet (Narayan et al., 2002). They used a rule-based transliteration system to convert

the lexical database of the Hindi WordNet to Urdu (Perso-Arabic). They manually pruned typical Sanskrit words that are not used in Urdu texts and added additional entries specific to Urdu. Similarly Ahmed and Hautli (2010) proposed to use a simple transliteration-based approach to access Hindi WordNet for Urdu texts, instead of creating a separate Urdu WordNet.

Mukund et al. (2010) have explored the use of Hindi specific POS tagger and chunker on Urdu text. Both training and testing data are transliterated to a common form for model transfer. They show that Hindi POS tagger performs worst on Urdu text which suffered an absolute loss of 32.5% in accuracy from an Urdu specific tagger. Their observation is same for chunking, however the results are not reported due to lack of an evaluation set. Visweswariah et al. (2010) explore complementary role of linguistic resources present in Hindi and Urdu for better system performance. They show improvements in machine translation, bitext alignment and POS tagging.

Besides these empirical studies that show Hindi and Urdu can mutually benefit from sharing their individual resources, there are also arguments against any such endeavour (Riaz, 2009; Mukund et al., 2010; Prasad et al., 2012). In a theoretical study, Riaz (2009) argues that lexical divergences between Hindi and Urdu hinder the interoperability of their computational resources. On the basis of the extensive variation in their vocabularies, he argues that any method that relies on maximum likelihood estimation may not work jointly for both Hindi and Urdu.

Our work differs from these related works in multiple ways. Firstly, we showed that Urdu text processing suffers substantially by the use of an ambiguous script. For computational purposes, we argued to represent Urdu texts in Devanagari instead of Perso-Arabic. To that end, we proposed an efficient and accurate transliteration method that resolves the lexical ambiguity due to missing short vowels and ambiguous characters in Perso-Arabic writing. Secondly, in addition to resource sharing, we also showed that resource augmentation can improve the performance of individual text processing modules of Hindi and Urdu. Furthermore, to mitigate the effect of lexical sparsity, we also used distributional similarity-based method besides transliteration.

6 Conclusion

In this paper, we explored the possibility of sharing and augmenting annotation resources of Hindi and Urdu to improve the performance of their individual text processing modules. To bridge the script and lexical differences between their texts, we proposed a simple and efficient technique based on script transliteration and distributional similarity. We showed that we can easily abstract away from the orthographic differences between Hindi and Urdu texts by representing their lexicons in a same distributional space. As a proof-of-the-concept, we showed that by bridging their script and lexical differences we can enhance the performance of Hindi and Urdu dependency parsers by simply merging their training data. Moreover, our experimental results suggest that Hindi and Urdu parsers can even be used interchangeably with reasonable accuracies.

In the future, we would like to explore the possibility of merging the semantic role annotations in HDTB and UDTB for training a better semantic role labeler. It would also be interesting to see whether our observations related to resource sharing between Hindi and Urdu would hold for applications other than parsing.

References

Farah Adeeba and Sarmad Hussain. 2011. Experiences in building the Urdu WordNet. In *Proceedings of Asian Language Resources collocated with IJCNLP*.

Tafseer Ahmed and Annette Hautli. 2010. Developing a basic lexical resource for Urdu using Hindi WordNet. In *Proceedings of CLT10, Islamabad, Pakistan*.

Akshar Bharati, Vineet Chaitanya, Rajeev Sangal, and KV Ramakrishnamacharyulu. 1995. *Natural language processing: a Paninian perspective*. Prentice-Hall of India New Delhi.

Akshar Bharati, Rajeev Sangal, Dipti Misra Sharma, and Lakshmi Bai. 2006. Anncorra: Annotating corpora

guidelines for POS and Chunk annotation for Indian languages. Technical report, (TR-LTRC-31), LTRC, IIIT-Hyderabad.

Riyaz Ahmad Bhat, Rajesh Bhatt, Annahita Farudi, Prescott Klassen, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Ashwini Vaidya, Sri Ramagurumurthy Vishnu, et al. 2015. The Hindi/Urdu treebank project. In *Handbook of Linguistic Annotation*. Springer Press.

Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. A multi-representational and multi-layered treebank for Hindi/Urdu. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 186–189. Association for Computational Linguistics.

Marie Candito, Enrique Henestroza Anguiano, and Djamé Seddah. 2011. A word clustering approach to domain adaptation: Effective parsing of biomedical texts. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 37–42. Association for Computational Linguistics.

Danqi Chen and Christopher D Manning. 2014. 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.

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

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12.

Hal Daumé III. 2007. Frustratingly easy domain adaptation. In *Proceedings of Association of Computational Linguistics*.

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.

Zellig S Harris. 1954. Distributional structure. *Word*.

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)*.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599–612.

Gurpreet Singh Lehal and Tejinder Singh Saini. 2012. Development of a complete Urdu-Hindi transliteration system. In *Proceedings of the 24th International Conference on Computational Linguistics (Posters)*, pages 643–652.

Gurpreet Singh Lehal and Tejinder Singh Saini. 2014. Sangam: A Perso-Arabic to Indic script machine transliteration model. In *Proceedings of the 11th International Conference on Natural Language Processing*.

Muhammad G Malik, Christian Boitet, and Pushpak Bhattacharyya. 2008. Hindi Urdu machine transliteration using finite-state transducers. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 537–544. Association for Computational Linguistics.

Colin P Masica. 1993. *The Indo-Aryan Languages*. Cambridge University Press.

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

Smruthi Mukund, Rohini Srihari, and Erik Peterson. 2010. An information-extraction system for Urdu—a resource-poor language. *ACM Transactions on Asian Language Information Processing (TALIP)*, 9(4):15.

Dipak Narayan, Debasri Chakrabarti, Prabhakar Pande, and Pushpak Bhattacharyya. 2002. An experience in building the IndoWordNet-a WordNet for Hindi. In *Proceedings of the First International Conference on Global WordNet, Mysore, India*.

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

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

Barbara Plank and Alessandro Moschitti. 2013. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (acl)*. Association for Computational Linguistics.

KVS Prasad, Shafqat Virk, John Camilleri, Krasimir Angelov, Kaarel Kaljurand, Olga Caprotti, and Aarne Ranta. 2012. Computational evidence that Hindi and Urdu share a grammar but not the lexicon. In *Proceedings of the 3rd Workshop on South and Southeast Asian Natural Language Processing (SANLP), COLING 2012*, volume 7427.

Kashif Riaz. 2009. Urdu is not Hindi for information access. In *Proceedings of the Workshop on Multilingual Information Access, SIGIR*.

Hassan Sajjad, Nadir Durrani, Helmut Schmid, and Alexander Fraser. 2011. Comparing two techniques for learning transliteration models using a parallel corpus. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 129–137, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

R Mahesh K Sinha. 2009. Developing English-Urdu machine translation via Hindi. In *Proceedings of the Third Workshop on Computational Approaches to Arabic-Script-based Languages*.

Oscar Täckström, Ryan McDonald, and Jakob Uszkoreit. 2012. Cross-lingual word clusters for direct transfer of linguistic structure. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 477–487. Association for Computational Linguistics.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics.

Karthik Visweswariah, Vijil Chenthamarakshan, and Nandakishore Kambhatla. 2010. Urdu and Hindi: Translation and sharing of linguistic resources. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1283–1291. Association for Computational Linguistics.

Min Xiao and Yuhong Guo. 2014. Distributed word representation learning for cross-lingual dependency parsing. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, page 119.

Dmitry Zelenko and Chinatsu Aone. 2006. Discriminative methods for transliteration. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 612–617. Association for Computational Linguistics.