Can Distributed Word Embeddings be an alternative to costly linguistic features: A Study on Parsing Hindi

Aniruddha Tammewar*, Karan Singla*, Bhasha Agrawal, Riyaz Bhat and Dipti Misra Sharma
Language Technologies Research Center
IIIT-Hyderabad, India
(uttam.tammewar, karan.singla, bhasha.agrawal, riyaz.bhat)@research.iiit.ac.in
dipti@iiit.ac.in

Abstract

Word Embeddings have shown to be useful in wide range of NLP tasks. We explore the methods of using the embeddings in Dependency Parsing of Hindi, a MoR-FWO (morphologically rich, relatively freer word order) language and show that they not only help improve the quality of parsing, but can even act as a cheap alternative to the traditional features which are costly to acquire. We demonstrate that if we use distributed representation of lexical items instead of features produced by costly tools such as Morphological Analyzer, we get competitive results. This implies that only mono-lingual corpus will suffice to produce good accuracy in case of resource poor languages for which these tools are unavailable. We also explored the importance of these representations for domain adaptation.

1 Introduction

Hindi is a MoR-FWO language. It exerts a relatively free word order with SOV being the default configuration. Due to the flexible word order, dependency representations are preferred over constituency for its syntactic analysis (Bharati and Sangal, 1993). The dependency representations do not constrain the order of words in a sentence and thus are better suited for flexible ordering of words(Hudson, 1984)(Sheiber, 1985)(Bharati et al., 1995). The dependency grammar formalism used for Hindi Treebank annotation is *Computational Paninian Framework*(CPG) (Begum et al., 2008; Bharati et al., 2009). The dependency relations in CPG formalism are closer to semantics and hence are also referred as *syntactico-semantic relations*.

Last decade has witnessed several efforts towards developing robust data driven dependency parsing techniques (Kübler et al., 2009). The efforts, in turn, initiated a parallel drive for building dependency annotated treebanks (Tsarfaty et al., 2013) which serve as a data source for training data driven dependency parsers. The annotations are often multi-layered and furnish information on part of speech category of word forms, their morphological features, chunking related words and syntactic relations. But error analysis show that the annotated information is not enough for good quality parsing. The efforts towards adding richer information to the treebank started to help parser disambiguate syntactic relations and suggested that semantic information could help parsing (Jain et al., 2013).

Annotating treebank with rich information improves the results but annotation in itself is a very costly task in terms of both, time and manual work. Hence, manually annotated data can not be made available for all the languages. Even if treebanks with rich information are available, it is difficult to accurately generate these features automatically in real time parsing. The tools used to generate these features such as morph analyzer, WordNet (in Case of semantic information), etc. are also costly to build. Resource poor languages do not have these tools readily available. The only thing very easily available for any language is the mono-lingual text corpus. Efforts are being made to exploit the use of mono-lingual corpus to capture all this information in word-embeddings using continuous vector space models. Using word-embeddings as features have shown to be useful and an easy replacement to the costly features in different NLP tasks. We try to exploit the same for the task of Hindi Dependency Parsing.

We show that the word embeddings learned

* Authors have equal contribution to the paper

from vector space models can replace almost all the features and hence costly tools and using it as a stand-alone feature can produce better results. We also show that these features when used as complementary features instead of replacement, improve the results further. Another outcome of our experiments is the improvement in the parsing quality when the training and testing data are from different domains.

2 Related Work

Chen and Manning (2014) have recently proposed a way of learning a neural network classifier for use in a greedy, transition-based dependency parser. They used small number of dense features, unlike most of the current parsers which use millions of sparse indicator features. The small number of features makes it work very fast. Use of low dimensional dense word embeddings as features in place of sparse features has recently been used in many NLP tasks and successfully shown improvements in terms of both, time and accuracies. The recent works include POS tagging (Collobert et al., 2011), Machine Translation (Devlin et al., 2014) and Constituency Parsing (Socher et al., 2013). These dense, continuous word-embeddings give strength to the words to be used more effectively in statistical approaches. The problem of data sparsity is reduced and also similarity between words can now be calculated using vectors.

3 Data & Tools

3.1 Hindi TreeBank

Here, we give an overview of Hindi Treebank (Hindi DTB). Pre-release version of Hindi Dependency Treebank data has been made available for download for researchers [1]. It is a multi-layered dependency treebank with morphological, part of speech and dependency annotations based on the CPG. CPG provides an essentially syntactico-semantic dependency annotation, incorporating karaka (e.g., agent, theme, etc.), non-karaka (e.g. possession, purpose) and other (part of) relations. A complete tag-set of dependency relations based on CPG can be found in (Bharati et al., 2009).

[1]http://ltrc.iiit.ac.in/treebank_H2014

	Training	Testing
# sentences	14321	1799
# Token-Count	299690	41514
#Chunk-Count	167910	20992

Table 1: Hindi TreeBank Statistics

3.2 Hindi Mono-Lingual Data

We used news corpus of Hindi(distributed as a part of WMT'14[2] translation task) for vector space modeling. The data consists of about 4.4 million sentences, which also includes the sentences from the treebank we are using for training and testing. The data has around 79.7 million tokens, with unique vocabulary of 682,362 words.

3.3 Tools Used

We used MaltParser(version 1.8)[3] (Nivre et al., 2007) for training the parser and Word2Vec[4] for vector space modeling.
Word2Vec provides an efficient implementation of continuous bag-of-words and skip-gram architectures for computing vector representations of words. We give more information about the working of vector space modeling in the next section.

4 Background and Experimental setup

In this section, we will explain the setup required for our experiments.

4.1 Inter-Chunk Parsing

In our experiments, we focus on establishing dependency relations between the chunk[5] heads which we henceforth denoted as inter-chunk parsing. The relations between the tokens of a chunk (intra-chunk dependencies) are not considered for experimentation. The decision is motivated by the fact that intra-chunk dependencies can easily be predicted automatically using a finite set of rules (Kosaraju et al., 2012). Moreover, we also observed high learnability of intra-chunk relations from an initial experiment. We found the accuracies of intra-chunk dependencies to be more than 99.00% for both Labeled Attachment and Unlabeled Attachment.

[2]http://www.statmt.org/wmt14/translation-task.html

[3]http://www.maltparser.org/

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

[5]A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single arc

4.2 Available Features

Table 2 describes all the features and their properties annotated for each token, in Hindi treebank. The *Col* column tells the column number where the corresponding feature can be found in the CoNLL file. Following list describes different kinds of features provided in the treebank:

- ***Word-form:*** The word itself.
- ***POS Information:*** POS tags are annotated for each node in the sentence following the POS and chunk annotation guidelines (Bharati, 2006). List of POS tags used for annotation can be found in Appendix 13.1.
- ***Morph Information:*** The morphological features have eight mandatory feature attributes for each node. These features are classified as root(lemma), category(CPOS), gender, number, person, case, post position/Vibhakti (for a noun) or tense, aspect, modality (TAM) (for a verb).
- ***Chunk Information:*** After annotation of POS tags, chunk boundaries are marked with appropriate assignment of chunk labels (Bharati, 2006). List of Chunk tags used for annotation can be found in Appendix 13.2.
- ***Other Features:*** In the dependency treebank, apart from POS, morph, chunk and dependency annotations, some special features for some nodes are marked. For example, for the main verb of a sentential clause, information about whether the clause is declarative, interrogative or imperative is marked (stype). Similarly, whether the sentence is in active or passive voice is also marked (voice-type).

In the treebank, all the features are manually annotated and thus have good quality. Almost all of the features provided contain rich information, useful for parsing. But, at parsing time, it is unrealistic to work with gold features. Instead these features should be derived automatically using the available tools. The corresponding tools required to obtain each feature are mentioned in Column 2. Note that even the TAM and Vibhakti features are considered as morph features, they are extracted using a different tool '*Vibhakti Computation Tool*'. We divide the features used in parsing into two sets based on how efficiently they can be learned and produced. A feature is trivial if the tool used to obtain it can be made easily available and provides good quality. In case of Hindi, the widely used Paradigm Based Analyzer morph analyzer (Bharati et al., 1995) is very costly to build and inefficient in handling OOV words. So we consider features generated by it like G,N,P,C to be "Non-Trivial". Similarly, we consider POS tag as a trivial feature. Note that there are no tools available to predict the stype and voice-type features therefore these fall into non-trivial category.

Feature	**Tool used**	**Trivial or Non-Trivial**	**Col**
word-form	none	Trivial	2
Lemma	Morph Analyzer	Non-Trivial	3,6
CPOS	Morph Analyzer	Non-Trivial	4,6
POS	POS Tagger	Trivial	5
G,N,P,C	Morph Analyzer	Non-Trivial	6
TAM,Vibhakti	Vibhakti Computation	Trivial	6
Chunk ID, Chunk Type	Chunker	Trivial	6
stype, voice-type	Manual	Non-Trivial	6

Table 2: All the features available in Hindi DTB. G,N,P,C refers to gender, number, person and case. TAM: Tense, Aspect, Modality

We try out different combinations of features in gold tagged data as well as on the auto-data (In auto-data, features are generated using tools). We combine these strategies along with the different strategies of incorporating word embeddings to see the effect and find out most suitable combination for the auto-data.
Here we show the different combinations of features we tried.

1. All features available in treebank for gold data & all features derivable from tools for auto-data (all features except stype, voice-type).
2. Only trivial features which includes word-form + POS (part of speech tag)+ TVC (TAM, Vib, ChunkType, ChunkID)
3. Only word-form and POS

Note: To make evaluation feasible for output of auto-data, we have used gold features for chunking to handle the problem of chunk-heads mismatch between gold test-data and output of auto test data.

As the accuracy of the chunker[6] we have used is very high, this does not affect much to the system.

4.3 Learning Vector Space model

In recent years, there has been a trend in the NLP research community of learning distributed representations for different natural language units, from morphemes, words and phrases, to sentences and documents. Using distributed representations, these symbolic units are embedded into a low dimensional and continuous space, thus it is often referred to as *embeddings*.

Here, we give a brief idea about the working of Word2Vec tool. The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words.

Various approaches have been studied for learning word embeddings from large-scale plain texts. Most commonly used approaches are, Continuous bag-of-words model (CBOW) and Skip Gram NNLM. CBOW being faster than Skip Gram approach, in this study, we consider the Continuous Bag-of-Words (CBOW)(Mikolov et al., 2013) model.

The basic principle of the CBOW model is to predict each individual word in a sequence given the bag of its context words within a fixed window size as input, using a log-linear classifier. This model avoids the non-linear transformation in hidden layers, and hence can be trained with high efficiency.

With large window size, grouped words using the resulting word embeddings are more topically similar; whereas with small window size, the grouped words will be more syntactically similar. So we set the window size to 5 for our parsing task.

Figure 1 represents a typical CBOW model. The model learns compressed, continuous representations of words. We call the vectors in the matrix between the input and hidden layer, *word vectors*. Each word is associated with a real valued vector in N-dimensional space (usually N = 50 - 1000). These word vectors can be subsequently used as features in many NLP tasks. As word vectors can be trained on huge text datasets, they provide generalization for systems trained with limited amount of supervised data. Word vectors capture many linguistic properties (gender, tense, plurality, even complex semantic concepts)

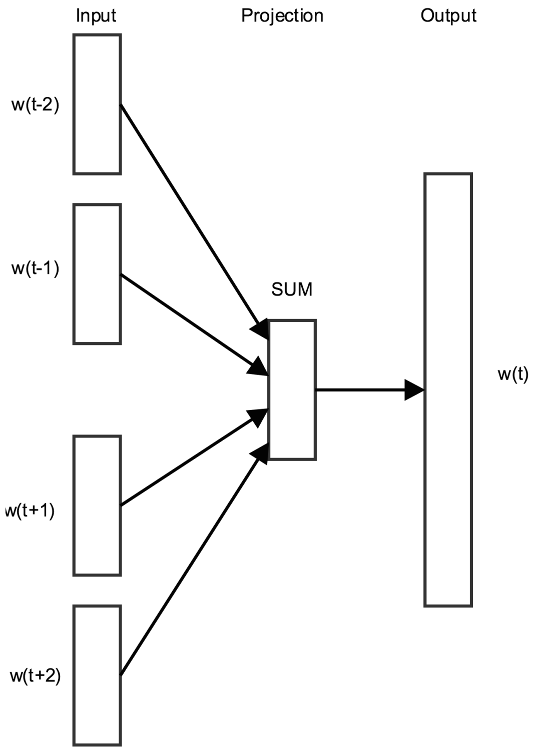

Figure 1: CBOW model

Training: First of all, we train a vector space model using the raw data mentioned in section 3.2. We keep the size of word vectors to be 100. So now we have word-embeddings ready to be used.

We perform K-means clustering over the vectors formed to group similar words into one cluster. We keep models ready with different values of 'K'.

4.4 Incorporating Embeddings in Conll Data

In the FEATS column (in a CoNLL file) of commonly used template of Malt Parser, the features are generally discrete and are separated by pipes ("—"). Whereas the features generated by Word2Vec are Numeric (Continuous). So, we created new columns for each numeric-feature and added them in the template/Conll X dataformat file to consider them as numeric features. Similarly in the 'feature model specification file' where, the features to be considered are defined, we added top of stack and current input into consideration for every new dimension/feature from the vector. So, in the present state, the FEATS column is not affected and new columns are added to the feature templates.

[6]`http://ltrc.iiit.ac.in/showfile.php?filename=downloads/shallow_parser.php` we are using shallow parser for automatic features

4.5 Different Strategies To Gather More Information In Word Embeddings

Just like different combinations of features as described in section 4.2, we also try out different strategies to produce different word embeddings. Every strategy used, produces vectors containing different types and amount of information.To understand different strategies, we take a dummy example and apply all the strategies to it.
Let's say "a b c" is a chunk consisting of 3 tokens a, b and c, where 'b' is the chunk head. Let's say our word vectors consist of 3 dimensions.
word vectors for a, b and c:
a: 1, 2, 1
b: 2, 1, 2
c: 0, 3, 3

1. **Chunk-Head Word Vector:** In this strategy we directly add word embeddings for chunk heads as features in the inter-chunk data. So in our dummy example, the word vector features of chunk-head will look like:
   b: 2, 1, 2

2. **Concatenate Window-1:** To capture the context information which we miss-out during inter-chunk parsing, we concatenate vectors of previous, current and next word (from full data) to get a combined vector of size 300. We add this new vector to the chunk head as features. In our dummy example, size of word vector would become 9.
   b: 1, 2, 1, 2, 1, 2, 0, 3, 3

3. **Average-out vectors in chunk:** When we are dealing with inter-chunk parsing, we generally focus only on the information provided by the chunk-heads and ignore other information from other words in the chunks. To include this information, we take vectors of all the words present in the chunk and take average of each feature, to get a new vector of the same length. We use this vector as a feature for the chunk-head. Now, the features in chunk head also capture entire information provided by the words in that chunk. In our example,
   b: (1+2+0)/3, (2+1+3)/3, (1+2+3)/3
   b: 1, 2, 2
   We know that concatenation of vectors works better than averaging them, but we can not concatenate the vectors in a chunk, as the number of tokens in a chunk vary from chunk to chunk. If we do concatenation in a chunk, we would get vectors of different dimensions for different chunk-heads, which would not be consistent with the feature template.

4. **Cluster ID:** In this strategy we add Cluster ID of word (or lemma in case of word not found) from the clusters formed as described in section 4.3 in the FEATS column as an additional feature.This strategy can also be combined with the other three strategies.

5 Experiments and Results

5.1 Baseline

We set up two baselines, one for gold-data and another for auto-data. For learning and parsing of baseline systems, we use MaltParser with all the default settings. In the baseline systems, we use all the available features in the gold and auto data. The baseline for gold data gives LAS of 82.23% & for auto data the baseline LAS comes out to be 76.55%. As expected, the baseline for auto data is much lower (about 6% LAS) than gold data because the features in auto data are obtained using automatic tools and are worse in quality than gold features.

5.2 Combinations of different features and word embedding strategies

In our first experiment, we try to find out optimal number of clusters to be used while clustering the vectors to get best results. We try out different values of 'K' ranging from 25 to 700 and come to conclusion that K=200 gives the best results when we use cluster ID as a feature in the parsing. So, we fix the value of 'K' to be 200 for further experiments.

5.2.1 Gold Data

In experiments with gold data, we first try out different strategies of incorporating word embeddings, and taking all features from FEATS column. The results can be seen in table 3. We observe that the strategy of concatenation of context vectors works best for gold data, giving an increment of 0.6% in LAS Score. We can see that all the strategies of word embeddings produced some increment in accuracy. In further experiments on gold data, we show results for the technique of concatenation only as this strategy performed best in all the experiments. From these experiments we can say that, word-embeddings capture some *additional real world aspects* of lexical items, which are quite distinct and unlikely to be deduced from the morpho-syntactic information like morph, POS-tag and chunk. This complementing semantic information is helping improve parse quality. We also performed combination of both the strategies (cluster ID along with concatenation) and did not find it helpful.

Experiment (Features Used)	**LAS**	**UAS**	**LS**
Baseline / All feats	82.23	90.33	84.54
All feats+vectors(1)	82.51	90.43	84.84
All feats+context(2)	**82.83**	**90.73**	**85.18**
All feats+chunk avg.(3)	82.61	90.46	84.95
All feats+cluster ID(4)	82.57	90.58	84.84
All feats+context(2) +clusterID(4)	82.76	90.66	85.14

Table 3: Results on Gold Data, All features, different Word Embedding strategies

Then we apply all the strategies of word embeddings to the data where we take only trivial features. The baseline accuracy (where no word vectors are added) is decreased in this case. This shows that not only trivial but all the features (if provided correctly) are important for parsing. This time also, we observe similar pattern as seen in above experiments. Once again the concatenation performs better than chunk average and it in turn performs better than normal adding of word vectors. Here we observe that combination of concatenation with Cluster ID improves the results further. In Table 4, we show the improvements for the strategy of concatenation. We observe that cluster ID in itself does not improve results much but when used with other strategies, it helps improve the results further.

Experiment (Features Used)	**LAS**	**UAS**	**LS**
trivial features only	81.95	90.19	84.18
trivial+vectors(1)	82.39	90.31	84.62
trivial+context(2)	82.73	90.76	84.95
trivial+context(2) +clusterID(4)	**82.74**	**90.76**	**85.00**

Table 4: Trivial features with different word embedding strategies

So, to conclude, we get a maximum LAS score of 82.83% using the context concatenation strategy. And we have shown that using chunk ID as a feature may sometimes help improve the scores.

5.2.2 Auto Data

Our main focus is to improve the results of auto-data as real time systems only have access to automatically extracted features. We do the same experiments with auto-data to see the results. In our first set of experiments, we use all the features in FEATS column and see effect of different embedding strategies.

Experiment (Features Used)	**LAS**	**UAS**	**LS**
Baseline	76.55	87.45	79.02
All + vectors(1)	77.09	87.62	79.62
All + context(2)	77.34	87.67	80.00
All+chunk avg.(3)	**78.07**	**87.98**	**80.63**
All + Chunk ID(4)	77.05	87.63	79.57

Table 5: Results on Auto Data, All features, different Word Embedding strategies

Here, we see maximum improvement of 1.52% over the baseline. In case of auto-data we observe a little change in pattern. We see that here, the averaging of all the chunk vectors performs better than the concatenation of context vectors, unlike the gold-data. One of the reasons behind this might be that, "vibhakti" feature captures somewhat information in the chunk. Gold data being manually annotated, this information is accurate in it but in auto data, it may not be accurate. So, to gather this information more accurately, need of help from other words in chunk arises and hence helps in improving results for auto-data. We observed that averaging of chunk performs better in all further set of experiments. Similar to the gold-data experiments, we will now show results only for this strategy in the further experiments.

Experiment (Features Used)	**LAS**	**UAS**	**LS**
trivial features only	77.64	88.26	79.87
trivial + vectors(1)	78.04	88.34	80.36
trivial + chunk avg.(3)	79.24	88.79	81.66
trivial+chunk avg.(3) + cluster ID(4)	**79.30**	**88.83**	**81.67**

Table 6: Trivial features with different word embeddings

The second set of experiments (table 6) is performed similar to gold-data, taking only trivial features. Unlike gold data, we find that the results improve over that of the experiments with all features. This shows that the non-trivial features, we skipped in these experiments are not accurate and hence produce sub-optimal results.

The maximum improvement of 2.52% is observed over baseline in the combination of two strategies Cluster ID and Chunk avg.

In third set of experiments on the auto-data we try to use only the most basic feature "POS", which is very easy to obtain for any language.(table 7)

Experiment (Features Used)	**LAS**	**UAS**	**LS**
POS	67.99	82.16	69.75
POS + vectors(1)	69.14	82.76	70.84
POS + chunk avg.(3)	**79.16**	**88.45**	**81.77**
POS+chunk avg.(3) + cluster ID(4)	79.10	88.43	80.97
POS + cluster ID(4)	47.18	57.73	55.22

Table 7: Only POS feature in feats column

Here we can see that using just one additional feature (POS) along with word embeddings provides very good results.

On the conrtary, Cluster ID as a feature had a negative effect on these experiments which is based on the fact that there is no feature which now captures the information about the context which was earlier captured by *V*ibhakti feature.

5.3 Domain Adaptation

Another problem we try to tackle is of domain difference between training and testing data. When we have parser trained on data of certain domain (in our case news articles and heritage) and we want to use it for parsing data from another domain, we need to bridge the gap between diverse vocabulary of different domains, which makes parsing difficult. For this task, we group words with similar semantics into clusters.(as mentioned in section 4.3). These clusters help in handling new words by treating them similar to other words from the same cluster.

We experiment different approaches on the data of four domains: box-office, cricket, gadget and recipe. The data statistics are shown in table 8. The data we have, for different domains, is manually annotated for dependency relations, but the features provided in the FEATS column are automatically obtained using tools. So, we also tried out using different combination of features, as we did in previous experiments.

Counts Domain	**sentences**	**chunks**	**word vector not found**
Box Office	509	3986	20
Cricket	508	4487	36
Gadget	527	4340	141
Recipe	544	4082	47

Table 8: Domain Data statistics

Table 9 shows LAS for different experiments we tried.

As expected, we can see that removing the non trivial and less accurate features, improves the results of parsing by large amount as compared to the baseline. For each domain we can see that using cluster ID as a feature improves the quality significantly.

One strange thing we observed is, using word embeddings as features has adverse effects on parsing. One reason behind this might be, by using **100** dense features while training, parser might have learned more accurately for the domain of training data and got biased toward the specific domain. We can see that the same pattern is followed by each domain.

Domain features	**Box -office**	**Cricket**	**Gadget**	**Recipe**
all	69.84	65.73	63.00	60.31
all + vect	60.59	51.74	53.25	49.56
all+cluster	**70.92**	**67.13**	**64.40**	**61.02**
triv	**77.32**	**72.17**	**71.36**	**66.88**
triv + vect	67.89	58.58	62.67	52.72
triv+cluster	**77.47**	**74.11**	**72.88**	**68.57**

Table 9: Domain data LAS (all: all features in FEATS, triv: only trivial features).

6 Observations and Discussion

From the above experiments on gold-data and auto data, we observe that using trivial features (which can be made easily available for any language) along with the word embeddings, the gap between auto-data accuracy and gold data accuracy is reduced to almost half compared to original. We also observed that using merely one feature "POS", which is very trivial to obtain, we can reach close to the maximum auto-data accuracy.

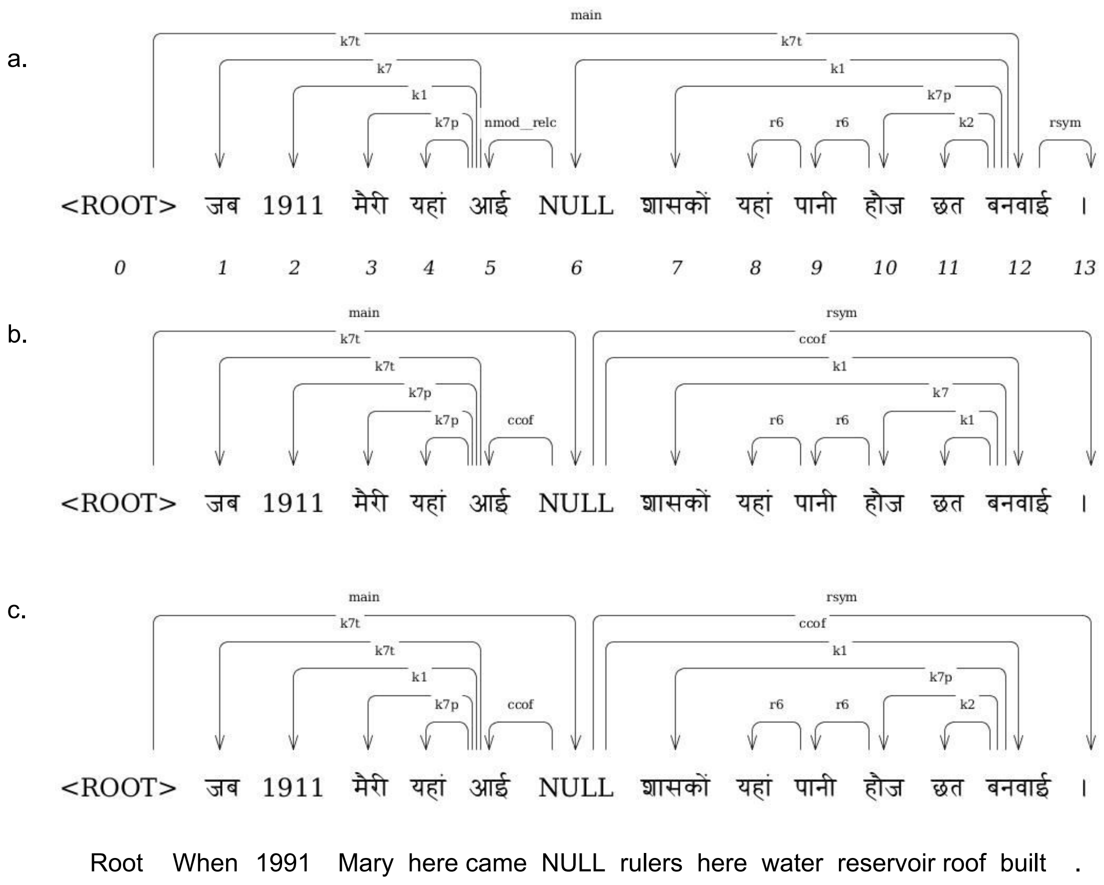
a.
b.
c.
main
k7t
k7
k1
k7p
nmod__relc
r6
k2
rsym
ccof
<ROOT> जब 1911 मेरी यहां आई NULL शासकों यहां पानी हौज छत बनवाई ।
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Root When 1991 Mary here came NULL rulers here water reservoir roof built .

Figure 2: Gold tree(a), Auto tree(b), Auto+chunk Avg.+clusterID tree(c)

Let us look at an example.
(jab)(when) (1911 mein)(1911 in) (queen Mary)(queen Mary) (yahaan)(here) (aai thi ,)(came aux ,) (NULL)(then) (Brtish shasako ne)(British Rulers-Erg) (yaha ke)(here's) (paani ke)(water of) (hauj ke oopar)(reservoir above) (ek chhat)(one roof) (banwai thi)(built aux) (.)(.)

Translation: In 1911, when Queen Mary came here, British rulers had built here a roof above the water reservoir.
Figure 2(a) shows the gold tree, 2(b) the trees produced by auto trivial features and 2(c) the tree produced by auto trivial + chunk avg + clusterID (the trees are interChunk, i.e., between chunk-heads). Improvement can be clearly seen in the tree by using the chunk avg. technique (strategy 3 under section 4.5). 'Mary' should be marked as 'k1' (doer) of 'came' but it is wrongly marked as 'k7p' (place of action) as the information provided by trivial features is not sufficient to identify that the word 'Mary' is name of a person (the POS tag 'NNP' (proper noun) is assigned). The problem is resolved when we use the technique of chunk avg. and clusterID.

We found that most of the words in the cluster containing word 'Mary' are British names as the treebank data is from news and heritage domain. Data from Indian heritage domain contains description of British people at various places. The information that Mary is a person and not a place is captured by the clusterID and the word-vector. In similar way the case of 'k7p' for 'reservoir' and 'k2' for 'roof' is handled.

From above observation we can say that some semantic properties of words are getting captured in the word vectors which also include some morphological characteristics.

7 Conclusions

Experiments on gold data have shown that use of word embeddings and cluster ID as features, is helpful for the resource rich languages for which accurate tools for obtaining features are available.

The main outcome of our experiments is that, we can achieve good results for parsing of resource-poor languages (for which tools and treebanks are not available) by using simple features or even just the POS tags and having a large monolingual corpus in hand.

The strategy of clustering is most useful in the case of domain adaptation. It helps in reducing difficulty faced in parsing because of vocabulary mismatch across different domains.

8 Future Work

In our experiments, we have neglected very low frequency words to efficiently learn a vector space model, which tends to loss of important vocabulary words. In Morphological rich languages like Dravidian Languages, where same root-words inflect to have many word forms with different suffixes and prefixes, it becomes extremely difficult to model all words efficiently while learning word-embeddings for them. Therefore it might be a good idea to treat suffixes as separate words and learn embedding for them too. According to various works on compositional semantics (Krishnamurthy and Mitchell, 2013; Kalchbrenner et al., 2014; dos Santos and Gatti, 2014), it has been shown that we can efficiently learn embedding of group of words by knowing embedding of individual words. In this case, a word can have multiple suffixes and prefixes along with one root-word. So this word will be formed with compositional semantics of all those word-embedding.

References

Rafiya Begum, Samar Husain, Arun Dhwaj, Dipti Misra Sharma, Lakshmi Bai, and Rajeev Sangal. 2008. Dependency annotation scheme for indian languages. In *IJCNLP*, pages 721–726. Citeseer.

Akshar Bharati and Rajeev Sangal. 1993. Parsing free word order languages in the paninian framework. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 105–111. Association for Computational Linguistics.

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

Akshara Bharati, Dipti Misra Sharma, Samar Husain, Lakshmi Bai, Rafiya Begam, and Rajeev Sangal. 2009. Anncorra: Treebanks for indian languages, guidelines for annotating hindi treebank.

Akshar Bharati. 2006. Anncorra: Annotating corpora guidelines for pos and chunk annotation for indian languages.

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*, 12:2493–2537.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, June*.

Cıcero Nogueira dos Santos and Maıra Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING), Dublin, Ireland*.

RA Hudson. 1984. Word grammar: Blackwell oxford.

Sambhav Jain, Naman Jain, Aniruddha Tammewar, Riyaz Ahmad Bhat, and Dipti Misra Sharma. 2013. Exploring semantic information in hindi wordnet for hindi dependency parsing. In *The sixth international joint conference on natural language processing (IJCNLP2013)*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Prudhvi Kosaraju, Samar Husain, Bharat Ram Ambati, Dipti Misra Sharma, and Rajeev Sangal. 2012. Intra-chunk dependency annotation: expanding hindi inter-chunk annotated treebank. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 49–56. Association for Computational Linguistics.

Jayant Krishnamurthy and Tom M Mitchell. 2013. Vector space semantic parsing: A framework for compositional vector space models. *ACL 2013*, page 1.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

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

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

S Sheiber. 1985. Evidence against the context-freeness of natural languages. *Linguistics and Philosophy*, 8:333–343.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*. Citeseer.

Reut Tsarfaty, Djamé Seddah, Sandra Kübler, and Joakim Nivre. 2013. Parsing morphologically rich languages: Introduction to the special issue. *Computational Linguistics*, 39(1):15–22.