Reducing Lexical Features in Parsing by Word Embeddings

Hiroya Komatsu, Ran Tian, Naoaki Okazaki and **Kentaro Inui**
Tohoku University, Japan
{h-komatsu, tianran, okazaki, inui}@ecei.tohoku.ac.jp

Abstract

The high-dimensionality of lexical features in parsing can be memory consuming and cause over-fitting problems. We propose a general framework to replace all lexical feature templates by low-dimensional features induced from word embeddings. Applied to a near state-of-the-art dependency parser (Huang et al., 2012), our method improves the baseline, performs better than using cluster bit string features, and outperforms a recent neural network based parser. A further analysis shows that our framework has the effect hypothesized by Andreas and Klein (2014), namely (i) connecting unseen words to known ones, and (ii) encouraging common behaviors among in-vocabulary words.

1 Introduction

Lexical features are powerful machine learning ingredients for many NLP tasks, but the very high-dimensional feature space brought by these features can be memory consuming and cause over-fitting problems. Is it possible to use low-dimensional word embeddings to reduce the high-dimensionality of lexical features? In this paper, we propose a general framework for this purpose. As a proof of concept, we apply the framework to dependency parsing, since this is a task where lexical features are essential.

Our approach is illustrated in Figure 1. Consider a transition-based dependency parser (Yamada and Matsumoto, 2003; Nivre et al., 2006; Zhang and Clark, 2008; Huang and Sagae, 2010; Zhang and Nivre, 2011), in which the words on top of the stack and the queue (denoted by $s_0\mathtt{w}$ and $q_0\mathtt{w}$, respectively) are typically used as features to calculate scores of transitions. When $s_0\mathtt{w}$ is used as a feature template, the features in this template (e.g. $s_0\mathtt{w}_{saw}$ and $s_0\mathtt{w}_{look}$) can be viewed as one-hot vectors of a dimension of the lexicon size (Figure 1). Corresponding to $s_0\mathtt{w}$, a weight is assigned to *each word* (e.g. $W(s_0\mathtt{w}_{saw})$ and $W(s_0\mathtt{w}_{look})$) for calculating a transition score. Instead, we propose to utilize a d-dimensional word embedding, and replace the feature template $s_0\mathtt{w}$ by d features, namely $s_0e_1, \ldots, s_0e_d$. Given the vector representation of a word (e.g., $\mathtt{e}_{saw} = (0.6, \ldots, 0.2)$), we replace the lexical feature (e.g. $s_0\mathtt{w}_{saw}$) by a linear combination of the d features (e.g., $s_0\mathtt{e}_{saw} := 0.6s_0e_1 + \ldots + 0.2s_0e_d$). Then, instead of the weights in a number of lexicon size assigned to $s_0\mathtt{w}$, now we use d

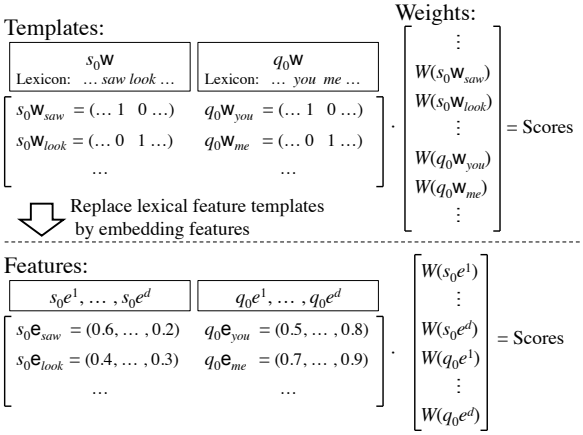


Figure 1: Each lexical feature template is replaced by a small number of embedding features.

29th Pacific Asia Conference on Language, Information and Computation pages 106 - 113
Shanghai, China, October 30 - November 1, 2015

weights (i.e., $W(s_0e_1), \ldots, W(s_0e_d)$) to calculate a transition score. In this work, we reduce feature space dimensionality by *replacing all lexical features*, including combined features such as $s_0\texttt{w}q_0\texttt{w}$, by the word embedding features.

In experiments, we applied the framework to a near state-of-the-art dependency parser (Huang et al., 2012), evaluated different vector operations for replacing combined lexical features, and explored different word embeddings trained from unlabeled or automatically labeled corpora. We expect word embeddings to augment parsing accuracy, by the mechanism hypothesized in Andreas and Klein (2014), namely (i) to connect unseen words to known ones, and (ii) to encourage common behaviors among in-vocabulary words. In contrast to the negative results reported in Andreas and Klein (2014), we find that our framework indeed has these effects, and significantly improves the baseline. As a comparison, our method performs better than the technique of replacing words by cluster bit strings (Koo et al., 2008; Bansal et al., 2014), and the results outperform a neural network based parser (Chen and Manning, 2014).

2 Related Work

A lot of recent work has been done on training word vectors (Mnih and Hinton, 2009; Mikolov et al., 2013; Lebret and Collobert, 2014; Pennington et al., 2014), and utilizing word vectors in various NLP tasks (Turian et al., 2010; Andreas and Klein, 2014; Bansal et al., 2014). The common approach (Turian et al., 2010; Koo et al., 2008; Bansal et al., 2014) is to use vector representations in *new features*, added to (near) state-of-the-art systems, and make improvement. As a result, the feature space gets even larger. We instead propose to *reduce lexical features* by word embeddings. To our own surprise, though the feature space gets much smaller, the resulted system performs better.

Another stream of research is to use word embeddings in whole neural network architectures (Collobert et al., 2011; Socher et al., 2013; Chen and Manning, 2014; Weiss et al., 2015; Dyer et al., 2015; Watanabe and Sumita, 2015). Though this is a promising direction and has brought breakthroughs in the field, the question is left open on what exactly has contributed to the power of neural based approaches. In this work, we conjecture that the power may partly come from the low-dimensionality of word embeddings, and this advantage can be transferred to traditional feature based systems. Our experiments support this conjecture, and we expect the proposed method to help more mature, proven-to-work existing systems.

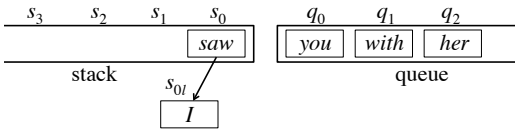


Figure 2: An internal state of a dependency parser.

Machine learning techniques have been proposed for reducing model size and imposing feature sparsity (Suzuki et al., 2011; Yogatama and Smith, 2014). Compared to these methods, our approach is simple, without extra twists of objective functions or learning algorithms. More importantly, by using word embeddings to reduce lexical features, we explicitly exploit the inherited syntactic and semantic similarities between words.

Another technique to reduce features is dimension reduction by matrix or tensor factorization (Argyriou et al., 2007; Lei et al., 2014), but typically applied to supervised learning. In contrast, we use word embeddings trained from unlabeled or automatically labeled corpora, bringing the aspects of semi-supervised learning or self-training.

3 Formalization

In this section, we formalize the framework of reducing lexical features. We take transition-based parsing as an example, but the framework can be applied to other systems using lexical features.

3.1 Transition-based Parsing

In typical transition-based parsing, input words are put into a queue and partially built parse trees are cached in a stack (Figure 2). At each step, a shift-reduce action is selected, which consumes words from the queue and/or build new structures in the stack. For the set of actions, we adopt the arc-standard system (Yamada and Matsumoto, 2003; Nivre, 2008; Huang and Sagae, 2010), in which the actions are:

1. `Shift`, which pops the top of the queue and pushes it to the stack;

2. `Reduce-Left`, which replaces the top two trees in the stack by their consolidated tree, left as child;

3. `Reduce-Right`, which replaces the top two trees in the stack by their consolidated tree, right as child.

Following Huang et al. (2012), we use the max-violation perceptron for global learning and beam-search for decoding.

In order to select the appropriate action, a set of features are used for calculating transition scores of each action. The features are typically extracted from internal states of the queue and the stack. For example, if we denote the elements in the stack by $s_0, s_1, \ldots$ from the top, and elements in the queue by $q_0, q_1, \ldots$ from the front; then, the words such as $s_0\mathtt{w}$ and $q_0\mathtt{w}$, the POS-tags such as $s_0\mathtt{t}$, and the combined word and POS-tags such as $s_0\mathtt{wt}$ are used as features. Other features include the POS-tag $s_{0l}\mathtt{t}$ (where s_{0l} denotes the leftmost child of s_0, and s_{0r} denotes the rightmost child of s_0), and the combined feature $s_0\mathtt{w}q_0\mathtt{w}$, *etc*.

If the corresponding words and POS-tags are specified in a concrete state, we use subscripts of `w` and `t` to denote the concrete feature. For example, from the state illustrated in Figure 2, we can extract features such as $s_0\mathtt{w}_{saw}$, $q_0\mathtt{w}_{you}$, $s_0\mathtt{t}_{\mathrm{VBD}}$, $s_0\mathtt{w}_{saw}\mathtt{t}_{\mathrm{VBD}}$, $s_{0l}\mathtt{t}_{\mathrm{PRP}}$, and $s_0\mathtt{w}_{saw}q_0\mathtt{w}_{you}$, *etc*.

For the purpose of this work, we mainly focus on the words (e.g., $\mathtt{w}_{saw}$, $\mathtt{w}_{you}$) in the above features. Other parts, including positions such as s_0 and q_0, and POS-tags such as $\mathtt{t}_{\mathrm{VBD}}$, are regarded as formal symbols.

3.2 Reducing Lexical Features

Formally, we define **lexial features** as the features comprising one or more words, possibly in combination with other symbols. We propose to replace lexical features as follows, and leaving other features (e.g. $s_0\mathtt{t}_{\mathrm{VBD}}$) unchanged in the system.

Lexical Feature of One Word Let $s\mathtt{w}$ be a one word lexical feature, where `w` is the word and s is an arbitrary symbol. Let $\mathtt{e} = (v_i)_{1\le i\le d}$ be a d-dimensional vector representation of the word `w`, where v_i is the i-th entry. Then, we replace $s\mathtt{w}$ by $s\mathtt{e}$, a linear combination of $se_1, \ldots, se_d$:

$$s\mathtt{e} := \sum_{i=1}^{d} v_i \cdot (se_i).$$

For example, assume that the word "*saw*" has a vector representation $\mathtt{e}_{saw} = (0.6, \ldots, 0.2)$. Then, the feature $s_0\mathtt{w}_{saw}$ is replaced by

$$s_0\mathtt{e}_{saw} := 0.6s_0e_1 + \ldots + 0.2s_0e_d.$$

In the above, $s_0e_1, \ldots, s_0e_d$ are introduced to replace the feature template $s_0\mathtt{w}$. Note that, instead of using a different feature $s_0\mathtt{w}_x$ for each different word x, now we only have d features, $s_0e_1, \ldots, s_0e_d$, commonly used by all words, across the feature template $s_0\mathtt{w}$.

As another example, in the case of features combining a word and its POS tag, such as $s_0\mathtt{t}_{\mathrm{VBD}}\mathtt{w}_{saw}$, we treat $s_0\mathtt{t}_{\mathrm{VBD}}$ as a formal symbol and replace the feature as the following:

$$s_0\mathtt{t}_{\mathrm{VBD}}\mathtt{e}_{saw} := 0.6s_0\mathtt{t}_{\mathrm{VBD}}e_1 + \ldots + 0.2s_0\mathtt{t}_{\mathrm{VBD}}e_d.$$

Lexical Feature of Two or More Words For lexical features of two or more words, such as $s_0\mathtt{w}q_0\mathtt{w}$, we replace the words by a combination of the two or more corresponding word vectors. More precisely, for a two-word lexical feature $s\mathtt{w}_1\mathtt{w}_2$, assume that the vectors $\mathtt{e}_1 = (u_i)_{1\le i\le d}$ and $\mathtt{e}_2 = (v_i)_{1\le i\le d}$ represent $\mathtt{w}_1$ and $\mathtt{w}_2$, respectively. Then, we propose the following operations[1] to replace $s\mathtt{w}_1\mathtt{w}_2$:

- OUTER PRODUCT ($\otimes$):

$$s(\mathtt{e}_1 \otimes \mathtt{e}_2) := \sum_{i=1}^{d}\sum_{j=1}^{d} u_iv_j \cdot (se_i\tilde{e}_j),$$

For example, if $\mathtt{e}_{saw} = (0.6, \ldots, 0.2)$ and $\mathtt{e}_{you} = (0.5, \ldots, 0.8)$, then $s_0\mathtt{w}_{saw}q_0\mathtt{w}_{you}$ is replaced by:

$$\begin{aligned} s_0q_0(\mathtt{e}_{saw} \otimes \mathtt{e}_{you}) := {} & (0.6 \times 0.5)s_0q_0e_1\tilde{e}_1 + \ldots \\ & + (0.6 \times 0.8)s_0q_0e_1\tilde{e}_d + \ldots \\ & + (0.2 \times 0.8)s_0q_0e_d\tilde{e}_d. \end{aligned}$$

Here, $\tilde{e}_1, \ldots, \tilde{e}_d$ are copies of $e_1, \ldots, e_d$.

[1] Operations for more than three word vectors are similar.

- SUM ($+$):

$$s(\mathtt{e}_1 + \mathtt{e}_2) := \sum_{i=1}^{d} (u_i + v_i) \cdot (se_i).$$

Following the previous example, $s_0\mathtt{w}_{saw}q_0\mathtt{w}_{you}$ is replaced by:

$$s_0q_0(\mathtt{e}_{saw} + \mathtt{e}_{you}) := (0.6 + 0.5)s_0q_0e_1 + \ldots + (0.2 + 0.8)s_0q_0e_d.$$

- CONCATENATION ($\oplus$):

$$s(\mathtt{e}_1 \oplus \mathtt{e}_2) := \sum_{i=1}^{d} u_i \cdot (se_i) + \sum_{j=1}^{d} v_j \cdot (s\tilde{e}_j).$$

Following the example, replace $s_0\mathtt{w}_{saw}q_0\mathtt{w}_{you}$ by

$$s_0q_0(\mathtt{e}_{saw} \oplus \mathtt{e}_{you}) := 0.6s_0q_0e_1 + \ldots + 0.2s_0q_0e_d + 0.5s_0q_0\tilde{e}_1 + \ldots + 0.8s_0q_0\tilde{e}_d.$$

Theoretically, OUTER PRODUCT is the natural operation, because if $s_0\mathtt{w}_x$ and $q_0\mathtt{w}_y$ are regarded as high-dimensional one-hot vectors (Figure 1), the feature combination $s_0\mathtt{w}_xq_0\mathtt{w}_y$ corresponds to the outer product of $s_0\mathtt{w}_x$ and $q_0\mathtt{w}_y$ (i.e., $s_0\mathtt{w}_xq_0\mathtt{w}_y$ fires when $s_0\mathtt{w}_x$ *and* $q_0\mathtt{w}_y$ fire). Empirically, we find that OUTER indeed performs the best among the three operations; however, the outer product also introduces d^2 embedding features, many more than the d features in SUM or $2d$ features in CONCATENATION. We also find that SUM performs better than CONCATENATION, being both effective and low-dimensional (Section 4.1).

4 Experiments

We reimplemented the parser of Huang et al. (2012) and replaced all lexical feature templates by embedding features, according to our framework. We set beam size to 8, and report unlabeled attachment scores (UAS) on the standard Penn Treebank (PTB) split, using the data attached to Huang et al. (2012)'s system[2]. POS-tags are assigned by Stanford Tagger[3]. To highlight the effect of word embeddings on unseen words, we also report UAS on 148 sentences in the Dev. set which contain words in vocabulary of the embeddings but unseen in PTB training data (Unseen).

	Dev	Test	Unseen
Huang et al. (2012)	91.93	91.68	89.01
Different Operations, using STATE embedding:			
OUTER	**92.57***	**92.20***	**90.27***
SUM	92.25*	91.85	90.10*
CONCATENATION	92.18	91.86	89.96
Different Embeddings, using OUTER operation:			
PLAIN	92.33*	91.78	90.08*
TREE	92.37*	92.09*	89.82
STATE	**92.57***	**92.20***	**90.27***
Cluster Bit String:			
PLAIN	91.71	91.20	89.18
TREE	90.38	90.07	88.00
STATE	91.31	90.96	89.04
Bansal et al. (2014)	92.06	91.75	90.13
Neural Network (Chen and Manning, 2014):			
Random	86.37	86.19	81.06
PLAIN	90.68	90.48	87.02
TREE	91.06	90.82	87.38
STATE	91.03	90.57	87.88

Table 1: Parsing Results (UAS). Numbers marked by asterisk (*) are statistically significant ($p < 0.05$), compared to the baseline (Huang et al., 2012) under a paired bootstrap test.

We built 300 dimensional word embeddings from 6 months articles in New York Times Corpus[4] (01/2007-06/2007, 1.5M sentences), for words of frequencies greater than 50. Word vectors are obtained from singular value decomposition (SVD) of the PPMI matrices (Levy and Goldberg, 2014b), for co-occurrence matrices of target words with various types of contexts (Levy and Goldberg, 2014a), to be specified later. We choose SVD for training word vectors because it is fast; and recent research suggests that SVD can perform as well as other embedding methods (Levy et al., 2015).

We investigated the following types of contexts for training word vectors: PLAIN, which uses words within a window of 3 to each side of the target word as contexts; TREE, which uses words within 3 steps of the target in the dependency trees, obtained from applying Huang et al. (2012)'s parser to the corpus; and STATE, which records the internal states of

[2]http://acl.cs.qc.edu/~lhuang/

[3]http://nlp.stanford.edu/software/corenlp.shtml

[4]https://catalog.ldc.upenn.edu/LDC2008T19

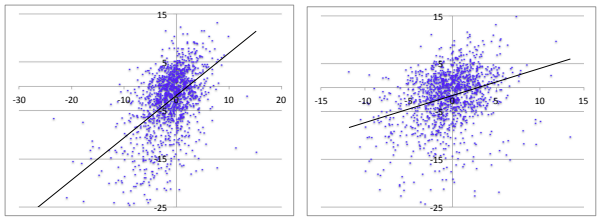


Figure 3: We plot X by the weight of the feature $s_0\mathtt{w}_x$, and Y by the weight of $s_0\mathtt{e}_x$, for x of high (Left) and middle (Right) frequency words.

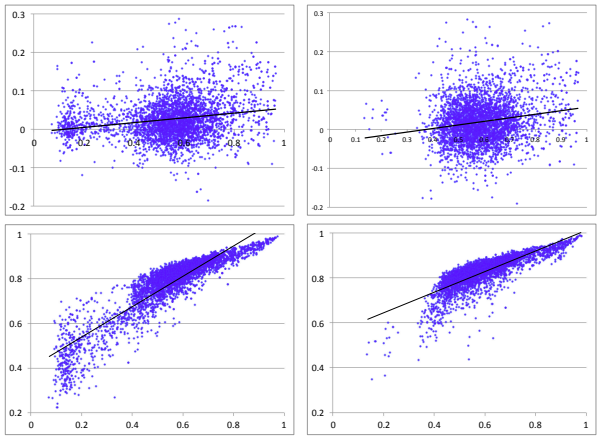


Figure 4: We plot X by cosine similarities between words, and Y by cosine similarities of weights, learned for lexical features (Upper) and embedding features (Lower). Words are of high (Left) and middle (Right) frequencies.

Huang et al. (2012)'s parser, and uses words at positions $\{s_1, s_2, s_3, s_{0l}, s_{0r}, s_{1l}, s_{1r}, q_0, q_1, q_2\}$ as contexts for a target s_0. These positions are where parsing features are extracted from. We expect TREE and STATE to encode more syntactic related information.

4.1 Parsing Results

The parsing results are shown in Table 1. We find that, the OUTER operation used for combined features and the STATE contexts for training word vectors perform the best for transition-based parsing, but other settings also improve the baseline (Huang et al., 2012), especially for sentences containing unseen words. We conducted paired bootstrap test to compare our proposed method with the baseline, and find out that most improvements are statistically significant.

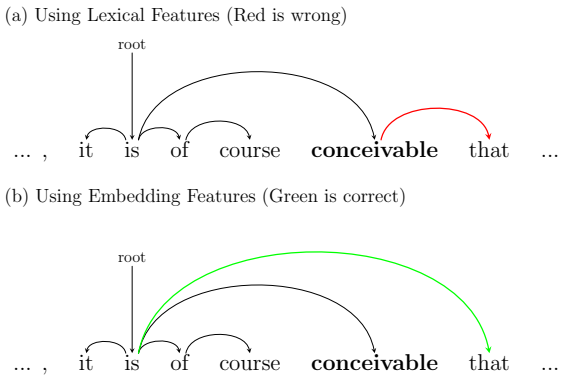
We also compared with the method of replacing words in lexical features by cluster bit strings (Koo et al., 2008; Bansal et al., 2014). We use bit strings constructed from hierarchical clusters induced from the previous word embeddings; as well as the the bit strings constructed in Bansal et al. (2014)[5]. Lengths of the bit strings are set to 4, 6, 8, 12, 16, and 20. It turns out that the performance gains are not as significant as our proposed method.

For reference, we report results by a neural network based parser (Chen and Manning, 2014), since our method shares a similar motivation with Chen and Manning's work, i.e. to use low-dimensional dense features instead of high-dimensional sparse features in parsing, aiming to obtain better generalization. For initializing word embeddings in the neural network, we tried 300 dimensional random vectors and the PLAIN, TREE, STATE vectors as described previously. We find that pre-trained word embeddings can improve performance, with TREE and STATE slightly better than PLAIN, suggesting that TREE and STATE may contain more information useful to parsing. However, the STATE vector is not as powerful as used with Huang et al. (2012)'s parser, suggesting that for a given baseline, it may be more helpful to train word vectors from contexts specific to that baseline. Chen and Manning's parser generally performs worse than Huang et al. (2012)'s baseline, suggesting that we cannot immediately obtain a better parser by switching to neural networks; other factors, such as global optimization and carefully selected features may still have merits, which makes our method useful for improving existing mature parsers.

4.2 Analysis

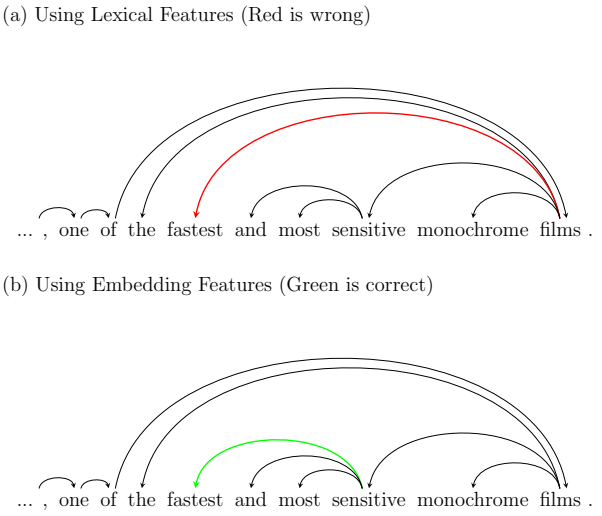
Is our modified parser really a feature reduction of the baseline system, i.e. is the parsing model trained for embedding features actually correlated to the baseline parsing model using lexical features? In Figure 3, we plot weights learned for the feature $s_0\mathtt{w}_x$ as X, and weights for $s_0\mathtt{e}_x$ as Y, where x ranges over high or middle frequency words. The weight for $s_0\mathtt{e}_x$ is calculated by taking inner product of the vector $s_0\mathtt{e}_x$ and the weight vector $(W(s_0e_1), \ldots, W(s_0e_d))$. As the direction of

[5]http://ttic.uchicago.edu/~mbansal/



"*While it is possible that the Big Green initiative will be ruled unconstitutional, it is of course conceivable that in modern California it could slide through.*"

Figure 5: Improved parsing results with unseen (bold) words.



"*The Rochester, N.Y., photographic giant recently began marketing T-Max 3200, one of the fastest and most sensitive monochrome films.*"

Figure 6: Improved parsing results on parallel structure of adjectives.

the regression lines show, weights learned for $s_0\texttt{e}_x$ are positively correlated to weights learned for $s_0\texttt{w}_x$. It suggests that the parsing model trained for embedding features is indeed correlated to the parsing model of the baseline, which implies that the baseline parser and our modified parser would have similar behaviors. This may explain the significance results reported in Table 1: though our improvements against the baseline is fairly moderate, they are still statistically significant because our modified parser behaves similarly as the baseline parser, but would correct the mistakes made by the baseline while preserving most originally correct labels. Such improvements are easier to achieve statistical significance (Berg-Kirkpatrick et al., 2012), and are arguably indicating better generalization.

So how does our modified parser improve from the baseline? In Figure 4, we plot cosine similarities between word vectors as X, and cosine similarities between weight vectors of all one-word lexical features as Y, compared to the similarities of weights of the corresponding embedding features. The plots show that, for similar words, the learned weights for the corresponding lexical features are only slightly similar; but after the lexical features are reduced to low-dimensional embedding features, the learned weights for the corresponding features are more strongly correlated. In other words, weights for embedding features encourage similar behaviors between similar words, due to a much lower dimensionality. This property may have two favorable effects on parsing, as hypothesized in Andreas and Klein (2014): (i) to connect unseen words to known ones, and (ii) to encourage common behaviors among in-vocabulary words.

The effects on unseen words have been observed in the Unseen column in Table 1, and we present a concrete example in Figure 5. In this example, "*conceivable*" is unseen in the training data, thus cannot be recognized by the baseline parser; however, its word vector is similar to "*subjective*" and "*undeniably*", whose behaviors are learned and generalized to "*conceivable*", by our modified parser using embedding features.

To illustrate the effects on in-vocabulary words, we take a specific parallel structure of adjectives. More precisely, we consider an internal state of the parser such that: $s_1\texttt{t}$ and $s_0\texttt{t}$ have POS-tags JJ, JJS or JJR; and $s_{0l}\texttt{t}$ has a POS-tag CC or Comma. Then, in 98.8% instances of such a state in the training data, the golden label action is `Reduce-Left`, suggesting a strong tendency of the state to become a parallel structure of adjectives, such as "*black and white*". However, when we parse New York Times data using the baseline parser, the proportion of `Reduce-Left` action when facing the state decreases to 96.7%, suggesting that this tendency is

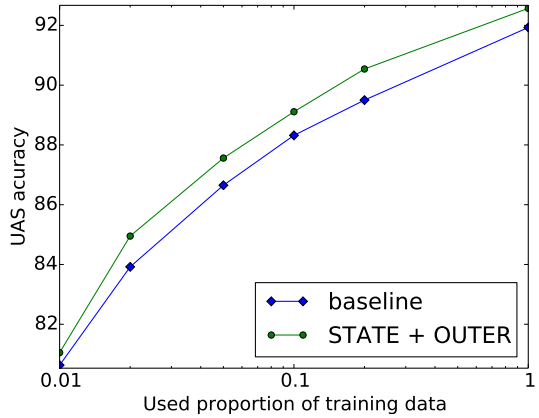


Figure 7: UAS on Dev. set, of models trained on less data.

not fully generalized as a rule for parallel structure of adjectives. This is not astonishing, because POS-tags and surface forms of lexical features are diverse in the training data. However, when we use our modified parser, the proportion of `Reduce-Left` action turns out to be 99.4%, significantly higher than using the baseline parser according to a permutation test. It suggests that our modified parser generalizes and strengthens the rule of parallel structure, by enforcing similar behaviors among similar adjectives. A concrete example of improvement is presented in Figure 6.

In Figure 7, we vary the size of training data and plot UAS of the obtained parsing models. As the figure shows, our modified parser using embedding features constantly outperforms the baseline. However, the performance of both settings decrease as the training data size decreases, suggesting that there may not be much syntactic information encoded in the word embeddings, even though the word embeddings are trained on internal states of the baseline parser, which is trained on full training data. We believe this graph indicates that, word embeddings can help parsing, but not because they encode extra syntactic information; rather, it is because word embeddings bring better generalization.

5 Conclusion

We have proposed a framework for reducing lexical features by word embeddings, and applied the framework to transition-based dependency parsing. A near state-of-the-art parser is improved, even though the features are reduced. This work is still preliminary, as we have only tested on one parser; however, our results are promising and our analysis suggests that the proposed method may indeed bring better generalization. We believe our framework can help more systems to reduce lexical features and alleviate the risk of overfitting, thanks to its generality.

References

Jacob Andreas and Dan Klein. 2014. How much do word embeddings encode about syntax? In *Proceedings of ACL*.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. 2007. Multi-task feature learning. In *Advances in NIPS*.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of ACL*.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in nlp. In *Proceedings of EMNLP-CoNLL*.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL-IJCNLP*.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of NAACL-HLT*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL*.

Rémi Lebret and Ronan Collobert. 2014. Word embeddings through Hellinger PCA. In *Proceedings of EACL*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of ACL*.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *Proceedings of ACL*.

Omer Levy and Yoav Goldberg. 2014b. Neural word embedding as implicit matrix factorization. In *Advances in NIPS*.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Trans. ACL*, 3.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in NIPS*.

Andriy Mnih and Geoffrey E. Hinton. 2009. A scalable hierarchical distributed language model. In *Advances in NIPS*.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*.

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

Jun Suzuki, Hideki Isozaki, and Masaaki Nagata. 2011. Learning condensed feature representations from large unsupervised data sets for supervised learning. In *Proceedings of ACL-HLT*.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of ACL-IJCNLP*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL-IJCNLP*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *In Proceedings of IWPT*.

Dani Yogatama and Noah A. Smith. 2014. Linguistic structured sparsity in text categorization. In *Proceedings of ACL*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL*.