Embedding Lexical Features via Low-Rank Tensors

Mo Yu*
Harbin Institute of Technology
IBM Watson
yum@us.ibm.com

Mark Dredze
HLTCOE
Johns Hopkins University
mdredze@cs.jhu.edu

Raman Arora
Johns Hopkins University
arora@cs.jhu.edu

Matthew R. Gormley
Carnegie Mellon University
mgormley@cs.cmu.edu

Abstract

Modern NLP models rely heavily on engineered features, which often combine word and contextual information into complex lexical features. Such combination results in large numbers of features, which can lead to overfitting. We present a new model that represents complex lexical features—comprised of parts for words, contextual information and labels—in a tensor that captures conjunction information among these parts. We apply low-rank tensor approximations to the corresponding parameter tensors to reduce the parameter space and improve prediction speed. Furthermore, we investigate two methods for handling features that include n-grams of mixed lengths. Our model achieves state-of-the-art results on tasks in relation extraction, PP-attachment, and preposition disambiguation.

1 Introduction

Statistical NLP models usually rely on hand-designed features, customized for each task. These features typically combine lexical and contextual information with the label to be scored. In relation extraction, for example, there is a parameter for the presence of a specific relation occurring with a feature conjoining a word type (lexical) with dependency path information (contextual). In measuring phrase semantic similarity, a word type is conjoined with its position in the phrase to signal its role. Figure 1b shows an example in dependency parsing, where multiple types (words) are conjoined with POS tags or distance information.

To avoid model over-fitting that often results from features with lexical components, several smoothed lexical representations have been proposed and shown to improve performance on various NLP tasks; for instance, word embeddings (Bengio et al., 2006) help improve NER, dependency parsing and semantic role labeling (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010; Sun et al., 2011; Roth and Woodsend, 2014; Hermann et al., 2014).

However, using only word embeddings is not sufficient to represent complex lexical features (e.g. ϕ in Figure 1c). In these features, the same word embedding conjoined with different non-lexical properties may result in features indicating different labels; the corresponding lexical feature representations should take the above interactions into consideration. Such important interactions also increase the risk of over-fitting as feature space grows exponentially, yet how to capture these interactions in representation learning remains an open question.

To address the above problems,[1] we propose a general and unified approach to reduce the feature space by constructing low-dimensional feature representations, which provides a new way of combining word embeddings, traditional non-lexical properties, and label information. Our model exploits the inner structure of features by breaking the feature into multiple parts: lexical, non-lexical and (optional) label. We demonstrate that the full feature is an outer product among these parts. Thus, a parameter tensor scores each feature to produce a prediction. Our model then reduces the number of param-

*Paper submitted during Mo Yu's PhD study at HIT.

[1] Our paper only focuses on lexical features, as non-lexical features usually suffer less from over-fitting.

Proceedings of NAACL-HLT 2016, pages 1019–1029,
San Diego, California, June 12-17, 2016.

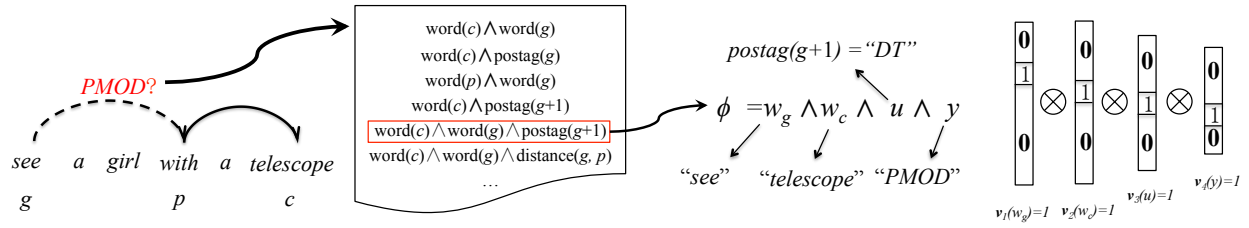

Figure 1: An example of lexical features used in dependency parsing. To predict the “PMOD” arc (the dashed one) between “see” and “with” in (a), we may rely on lexical features in (b). Here p, c, g are indices of the word “with”, its child (“telescope”) and a candidate head. Figure (c) shows what the fifth feature (ϕ) is like, when the candidate is “see”. As is common in multi-class classification tasks, each template generates a different feature for each label y. Thus a feature $\phi = \mathrm{w}_g \wedge \mathrm{w}_c \wedge \mathrm{u} \wedge \mathrm{y}$ is the conjunction of the four parts. Figure (d) is the one-hot representation of ϕ, which is equivalent to the outer product (i.e. a 4-way tensor) among the four one-hot vectors. $\mathbf{v}(x) = 1$ means the vector $\mathbf{v}$ has a single non-zero element in the x position.

eters by approximating the parameter tensor with a low-rank tensor: the Tucker approximation of Yu et al. (2015) but applied to each embedding type (view), or the **C**anonical/**P**arallel-Factors Decomposition (CP). Our models use fewer parameters than previous work that learns a separate representation for each feature (Ando and Zhang, 2005; Yang and Eisenstein, 2015). CP approximation also allows for much faster prediction, going from a method that is cubic in rank and exponential in the number of lexical parts, to a method linear in both. Furthermore, we consider two methods for handling features that rely on n-grams of mixed lengths.

Our model makes the following contributions when contrasted with prior work:

Lei et al. (2014) applied CP to combine different views of features. Compared to their work, our usage of CP-decomposition is different in the application to feature learning: (1) We focus on dimensionality reduction of existing, well-verified features, while Lei et al. (2014) generates new features (usually different from ours) by combining some “atom” features. Thus their work may ignore some useful features; it relies on binary features as supplementary but our model needs not. (2) Lei et al. (2014)’s factorization relies on views with explicit meanings, e.g. head/modifier/arc in dependency parsing, making it less general. Therefore its applications to tasks like relation extraction are less obvious.

Compared to our previous work (Gormley et al., 2015; Yu et al., 2015), this work allows for higher-order interactions, mixed-length n-gram features, lower-rank representations. We also demonstrate the strength of our new model via applications to new tasks.

The resulting method learns smoothed feature representations combining lexical, non-lexical and label information, achieving state-of-the-art performance on several tasks: relation extraction, preposition semantics and PP-attachment.

2 Notation and Definitions

We begin with some background on notation and definitions. Let $\mathcal{T} \in \mathbb{R}^{d_1 \times \cdots \times d_K}$ be a K-way tensor (i.e., a tensor with K views). In this paper, we consider the tensor k-mode product, i.e. multiplying a tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times \cdots \times d_K}$ by a matrix $\mathbf{x} \in \mathbb{R}^{d_k \times J}$ (or a vector if $J = 1$) in mode (view) k. The product is denoted by $\mathcal{T} \times_k \mathbf{x}$ and is of size $d_1 \times \cdots \times d_{k-1} \times J \times d_{k+1} \times \cdots \times d_K$. Elementwise, we have

$$(\mathcal{T} \times_k \mathbf{x})_{i_1 \ldots i_{k-1}\, j\, i_{k+1} \ldots i_K} = \sum_{i_k=1}^{d_k} \mathcal{T}_{i_1 \ldots i_k \ldots i_K} \mathbf{x}_{i_k j},$$

for $j = 1, \ldots, J$. A mode-k fiber $\mathcal{T}_{i_1 \ldots i_{k-1} \bullet i_{k+1} \ldots i_K}$ of $\mathcal{T}$ is the d_k dimensional vector obtained by fixing all but the kth index. The mode-k unfolding $\mathcal{T}_{(k)}$ of $\mathcal{T}$ is the $d_k \times \prod_{i \neq k} d_i$ matrix obtained by concatenating all the $\prod_{i \neq k} d_i$ mode-k fibers along columns.

Given two matrices $\mathrm{W}_1 \in \mathbb{R}^{d_1 \times r_1}, \mathrm{W}_2 \in \mathbb{R}^{d_2 \times r_2}$, we write $\mathrm{W}_1 \otimes \mathrm{W}_2$ to denote the Kronecker product between W_1 and W_2 (outer product for vectors). We define the Frobenius product (matrix dot product) $\mathbf{A} \odot \mathbf{B} = \sum_{i,j} A_{ij} B_{ij}$ between two matrices with

the same sizes; and define element-wise (Hadamard) multiplication $\mathbf{a} \circ \mathbf{b}$ between vectors with the same sizes.

Tucker Decomposition: Tucker Decomposition represents a $d_1 \times d_2 \times \ldots \times d_K$ tensor $\mathcal{T}$ as:

$$\mathcal{T} = g \times_1 \mathrm{W}_1 \times_2 \mathrm{W}_2 \ldots \times_K \mathrm{W}_K \quad (1)$$

where each $\times_i$ is the tensor i-mode product and each W_i is a $r_i \times d_i$ matrix. Tensor g with size $r_1 \times r_2 \times \ldots \times r_K$ is called the *core tensor*. We say that $\mathcal{T}$ has a Tucker rank $(r_{(1)}, r_{(2)}, \ldots, r_{(K)})$, where $r_{(i)} = \mathrm{rank}(\mathcal{T}_{(i)})$ is the rank of mode-i unfolding. To simplify learning, we define the Tucker rank as $r_{(i)}$=rank($\mathbf{g}_{(i)}$), which can be bounded simply by the dimensions of g, i.e. $r_{(i)} \leq r_i$; this allows us to enforce a rank constraint on $\mathcal{T}$ simply by restricting the dimensions r_i of g, as described in §6.

CP Decomposition: CP decomposition represents a $d_1 \times d_2 \times \ldots \times d_K$ tensor $\mathcal{T}$ as a sum of rank-one tensors (i.e. a sum of outer products of K vectors):

$$\mathcal{T} = \sum_{j=1}^{r} \mathrm{W}_1[j,:] \otimes \mathrm{W}_2[j,:] \otimes \ldots \otimes \mathrm{W}_K[j,:] \quad (2)$$

where each W_i is an $r \times d_i$ matrix and $\mathrm{W}_i[j,:]$ is the vector of its j-th row. For CP decomposition, the rank r of a tensor $\mathcal{T}$ is defined to be the number of rank-one tensors in the decomposition. CP decomposition can be viewed as a special case of Tucker decomposition in which $r_1 = r_2 = \ldots = r_K = r$ and g is a superdiagonal tensor.

3 Factorization of Lexical Features

Suppose we have feature ϕ that includes information from a label y, multiple lexical items $\mathrm{w}_1, \ldots, \mathrm{w}_n$ and non-lexical property u. This feature can be factorized as a conjunction of each part: $\phi = \mathrm{y} \wedge \mathrm{u} \wedge \mathrm{w}_1 \wedge \ldots \wedge \mathrm{w}_n$. The feature fires when all $(n+2)$ parts fire in the instance (reflected by the $\wedge$ symbol in ϕ). The one-hot representation of ϕ can then be viewed as a tensor $e_\phi = \mathrm{y} \otimes \mathrm{u} \otimes \mathrm{w}_1 \otimes \cdots \otimes \mathrm{w}_n$, where each feature part is also represented as a one-hot vector.[2] Figure 1d illustrates this case with two lexical parts.

Given an input instance $\mathbf{x}$ and its associated label y, we can extract a set of features $S(\mathbf{x}, \mathrm{y})$. In a traditional log-linear model, we view the instance $\mathbf{x}$ as a bag-of-features, i.e. a feature vector $F(\mathbf{x}, \mathrm{y})$. Each dimension corresponds to a feature ϕ, and has value 1 if $\phi \in S(\mathbf{x}, \mathrm{y})$. Then the log-linear model scores the instance as $s(\mathbf{x}, \mathrm{y}; \mathbf{w}) = \mathbf{w}^T F(\mathbf{x}, \mathrm{y}) = \sum_{\phi \in S(\mathbf{x}, \mathrm{y})} s(\phi; \mathbf{w})$, where $\mathbf{w}$ is the parameter vector. We can re-write $s(\mathbf{x}, \mathrm{y}; \mathbf{w})$ based on the factorization of the features using tensor multiplication; in which $\mathbf{w}$ becomes a parameter tensor $\mathcal{T}$:

$$s(\mathbf{x}, \mathrm{y}; \mathbf{w}) = s(\mathbf{x}, \mathrm{y}; \mathcal{T}) = \sum_{\phi \in S(\mathbf{x}, \mathrm{y})} s(\phi; \mathcal{T}) \quad (3)$$

Here each ϕ has the form $(\mathrm{y}, \mathrm{u}, \mathrm{w}_1, \ldots, \mathrm{w}_n)$, and

$$s(\phi; \mathcal{T}) = \mathcal{T} \times_l \mathrm{y} \times_f \mathrm{u} \times_{\mathrm{w}_1} \mathrm{w}_1 \ldots \times_{\mathrm{w}_n} \mathrm{w}_n. \quad (4)$$

Note that one-hot vectors w_i of words themselves are large ($|\mathrm{w}_i| > 500$k), thus the above formulation with parameter tensor $\mathcal{T}$ can be very large, making parameter estimation difficult. Instead of estimating only the values of the dimensions which appear in training data as in traditional methods, we will reduce the size of tensor $\mathcal{T}$ via a low-rank approximation. With different approximation methods, (4) will have different equivalent forms, e.g. (6), (7) in §4.1.

Optimization objective: The loss function ℓ for training the log-linear model uses (3) for scores, e.g., the log-loss $\ell(\mathbf{x}, \mathrm{y}; \mathcal{T}) = -\log \frac{\exp\{s(\mathbf{x}, \mathrm{y}; \mathcal{T})\}}{\sum_{\mathrm{y}' \in L} \exp\{s(\mathbf{x}, \mathrm{y}'; \mathcal{T})\}}$. Learning can be formulated as the following optimization problem:

$$\begin{aligned} \underset{\mathcal{T}}{\text{minimize:}} \quad & \sum_{(\mathbf{x}, y) \in \mathcal{D}} \ell(\mathbf{x}, y; \mathcal{T}) \\ \text{subject to:} \quad & \begin{cases} \mathrm{rank}(\mathcal{T}) \leq (r_1, r_2, ..., r_{n+2}) \\ \qquad\qquad \text{(Tucker-form)} \\ \mathrm{rank}(\mathcal{T}) \leq r \quad \text{(CP-form)} \end{cases} \end{aligned} \quad (5)$$

where the constraints on rank($\mathcal{T}$) depend on the chosen tensor approximation method (§2).

The above framework has some advantages: First, as discussed in §1 and here, we hope the representations capture rich interactions between different parts of the lexical features; the low-rank tensor approximation methods keep the most important interaction information of the original tensor, while significantly reducing its size. Second, the low-rank structure will encourage weight-sharing among lexical features with similar decomposed parts, leading

[2] $\mathrm{u}, \mathrm{y}, \mathrm{w}_i$ denote one-hot vectors instead of symbols.

to better model generalization. Note that there are examples where features have different numbers of multiple lexical parts, such as both unigram and bigram features in PP-attachment. We will use two different methods to handle these features (§5).

Remarks (advantages of our factorization) Compared to prior work, e.g. (Lei et al., 2014; Lei et al., 2015), the proposed factorization has the following advantages:

1. **Parameter explosion** when mapping a view with lexical properties to its representation vector (as will be discussed in 4.3): Our factorization allows the model to treat word embeddings as inputs to the views of lexical parts, dramatically reducing the parameters. Prior work cannot do this since its views are mixtures of lexical and non-lexical properties. Note that Lei et al. (2014) uses embeddings by *concatenating* them to specific views, which increases dimensionality, but the improvement is limited.

2. **No weight-sharing** among conjunctions with same lexical property, like the child-word "word(c)" and its conjunction with head-postag "word(c) $\wedge$ word(g)" in Figure 1(b). The factorization in prior work treats them as *independent* features, greatly increasing the dimensionality. Our factorization builds representations of both features based on the embedding of "word(c)", thus utilizing their connections and reducing the dimensionality.

The above advantages are also key to overcome the problems of prior work mentioned at the end of §1.

4 Feature Representations via Low-rank Tensor Approximations

Using one-hot encodings for each of the parts of feature ϕ results in a very large tensor. This section shows how to compute the score in (4) without constructing the full feature tensor using two tensor approximation methods (§4.1 and §4.2).

We begin with some intuition. To score the original (full rank) tensor representation of ϕ, we need a parameter tensor $\mathcal{T}$ of size $d_1 \times d_2 \times \ldots \times d_{n+2}$, where $d_3 = \cdots = d_{n+2} = |V|$ is the vocabulary size, n is the number of lexical parts in the feature and $d_1 = |L|$ and $d_2 = |F|$ are the number of different labels and non-lexical properties, respectively. (§5 will handle n varying across features.) Our methods reduce the tensor size by embedding each part of ϕ into a lower dimensional space, where we represent each label, non-lexical property and words with an $r_1, r_2, r_3, \ldots, r_{n+2}$ dimensional vector respectively ($r_i \ll d_i$, $\forall i$). These embedded features can then be scored by much smaller tensors. We denote the above transformations as matrices $\mathrm{W}_l \in \mathbb{R}^{r_1 \times d_1}$, $\mathrm{W}_f \in \mathbb{R}^{r_2 \times d_2}$, $\mathrm{W}_i \in \mathbb{R}^{r_{i+2} \times d_{i+2}}$ for $i = 1, \ldots, n$, and write corresponding low-dimensional hidden representations as $\mathbf{h}_{\mathrm{y}}^{(l)} = \mathrm{W}_l \mathrm{y}$, $\mathbf{h}_{\mathrm{u}}^{(f)} = \mathrm{W}_f \mathrm{u}$ and $\mathbf{h}_{\mathrm{w}}^{(i)} = \mathrm{W}_i \mathrm{w}$.

In our methods, the above transformations of embeddings are *parts of low-rank tensors* as in (5), so the embeddings of non-lexical properties and labels can be trained simultaneously with the low-rank tensors. Note that for one-hot input encodings the transformation matrices are essentially lookup tables, making the computation of these transformations sufficiently fast.

4.1 Tucker Form

For our first approximation, we assume that tensor $\mathcal{T}$ has a low-rank Tucker decomposition: $\mathcal{T} = g \times_l \mathrm{W}_l \times_f \mathrm{W}_f \times_{\mathrm{w}_1} \mathrm{W}_1 \times_{\mathrm{w}_2} \cdots \times_{\mathrm{w}_n} \mathrm{W}_n$. We can then express the scoring function (4) for a feature $\phi = (\mathrm{y}, \mathrm{u}, \mathrm{w}_1, \ldots \mathrm{w}_n)$ with n-lexical parts, as:

$$s(\mathrm{y}, \mathrm{u}, \mathrm{w}_1, \cdots, \mathrm{w}_n; g, \mathrm{W}_l, \mathrm{W}_f, \{\mathrm{W}_i\}_{i=1}^n) = g \times_l \mathbf{h}_{\mathrm{y}}^{(l)} \times_f \mathbf{h}_{\mathrm{u}}^{(f)} \times_{\mathrm{w}_1} \mathbf{h}_{\mathrm{w}_1}^{(1)} \cdots \times_{\mathrm{w}_n} \mathbf{h}_{\mathrm{w}_n}^{(n)}, \quad (6)$$

which amounts to first projecting u, y, and w_i (for all i) to lower dimensional vectors $\mathbf{h}_{\mathrm{u}}^{(f)}, \mathbf{h}_{\mathrm{y}}^{(l)}, \mathbf{h}_{\mathrm{w}_i}^{(i)}$, and then weighting these hidden representations using the flattened core tensor g. The low-dimensional representations and the corresponding weights are learned jointly using a discriminative (supervised) criterion. We call the model based on this representation the *Low-Rank Feature Representation with Tucker form*, or LRFR$_n$-TUCKER.

4.2 CP Form

For the Tucker approximation the number of parameters in (6) scale exponentially with the number of lexical parts. For instance, suppose each $\mathbf{h}_{\mathrm{w}_i}^{(i)}$ has di-

mensionality r, then $|g| \propto r^n$. To address scalability and further control the complexity of our tensor based model, we approximate the parameter tensor using CP decomposition as in (2), resulting in the following scoring function:

$$s(\mathrm{y}, \mathrm{u}, \mathrm{w}_1, \cdots, \mathrm{w}_n; \mathrm{W}_l, \mathrm{W}_f, \{\mathrm{W}_i\}_{i=1}^n) = \sum_{j=1}^{r} \left(\mathbf{h}_{\mathrm{y}}^{(l)} \circ \mathbf{h}_{\mathrm{u}}^{(\mathbf{f})} \circ \mathbf{h}_{\mathrm{w}_1}^{(1)} \circ \cdots \circ \mathbf{h}_{\mathrm{w}_n}^{(n)} \right)_j . \quad (7)$$

We call this model *Low-Rank Feature Representation with CP form* (LRFR$_n$-CP).

4.3 Pre-trained Word Embeddings

One of the computational and statistical bottlenecks in learning these LRFR$_n$ models is the vocabulary size; the number of parameters to learn in each matrix W_i scales linearly with $|V|$ and would require very large sets of labeled training data. To alleviate this problem, we use pre-trained continuous word embeddings (Mikolov et al., 2013) as input embeddings rather than the one-hot word encodings. We denote the m-dimensional word embeddings by e_{w}; so the transformation matrices W_i for the lexical parts are of size $r_i \times m$ where $m \ll |V|$.

We note that when sufficiently large labeled data is available, our model allows for fine-tuning the pre-trained word embeddings to improve the expressive strength of the model, as is common with deep network models.

Remarks Our LRFRs introduce embeddings for non-lexical properties and labels, making them better suit the common setting in NLP: rich linguistic properties; and large label sets such as open-domain tasks (Hoffmann et al., 2010). The LRFR-CP better suits n-gram features, since when n increases 1, the only new parameters are the corresponding W_i. It is also very efficient during prediction ($O(nr)$), since the cost of transformations can be ignored with the help of look-up tables and pre-computing.

5 Learning Representations for n-gram Lexical Features of Mixed Lengths

For features with n lexical parts, we can train an LRFR$_n$ model to obtain their representations. However, we often have features of varying n (e.g. both unigrams (n=1) and bigrams (n=2) as in Figure 1). We require representations for features with arbitrary different n simultaneously.

We propose two solutions. The first is a straightforward solution based on our framework, which handles each n with a $(n+2)$-way tensor. This strategy is commonly used in NLP, e.g. Taub-Tabib et al. (2015) have different kernel functions for different order of dependency features. The second is an approximation method which aims to use a single tensor to handle all ns.

Multiple Low-Rank Tensors Suppose that we can divide the feature set $S(\mathbf{x}, \mathrm{y})$ into subsets $S_1(\mathbf{x}, \mathrm{y}), S_2(\mathbf{x}, \mathrm{y}), \ldots, S_n(\mathbf{x}, \mathrm{y})$ which correspond to features with one lexical part (unigram features), two lexical parts (bigram features), ... and n lexical parts (n-gram features), respectively. To handle these types of features, we modify the training objective as follows:

$$\underset{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_n}{\text{minimize}} \sum_{(\mathbf{x}, \mathrm{y}) \in D} \ell(\mathbf{x}, \mathrm{y}; \mathcal{T}_1, \mathcal{T}_2, \ldots, ...\mathcal{T}_n), \quad (8)$$

where the score of a training instance $(\mathbf{x}, \mathrm{y})$ is defined as $s(\mathbf{x}, \mathrm{y}; \mathcal{T}) = \sum_{i=1}^{n} \sum_{\phi \in S_i(\mathbf{x}, \mathrm{y})} s(\phi; \mathcal{T}_i)$. We use the Tucker form low-rank tensor for $\mathcal{T}_1$, and the CP form for $\mathcal{T}_i$ $(\forall i > 1)$. We refer to this method as LRFR$_1$-TUCKER & LRFR$_2$-CP.

Word Clusters Alternatively, to handle different numbers of lexical parts, we replace some lexical parts with discrete word clusters. Let $c(\mathrm{w})$ denote the word cluster (e.g. from Brown clustering) for word w. For bigram features we have:

$$\begin{aligned} & s(\mathrm{y}, \mathrm{u}, \mathrm{w}_1, \mathrm{w}_2; \mathcal{T}) \\ &= s(\mathrm{y}, \mathrm{u} \wedge c(\mathrm{w}_1), \mathrm{w}_2; \mathcal{T}) + s(\mathrm{y}, \mathrm{u} \wedge c(\mathrm{w}_2), \mathrm{w}_1; \mathcal{T}) \\ &= \mathcal{T} \times_l \mathrm{y} \times_f (\mathrm{u} \wedge c(\mathrm{w}_1)) \times_{\mathrm{w}} e_{\mathrm{w}_2} \\ &\quad + \mathcal{T} \times_l \mathrm{y} \times_f (\mathrm{u} \wedge c(\mathrm{w}_2)) \times_{\mathrm{w}} e_{\mathrm{w}_1} \end{aligned} \quad (9)$$

where for each word we have introduced an additional set of non-lexical properties that are conjunctions of word clusters and the original non-lexical properties. This allows us to reduce an n-gram feature representation to a unigram representation. The advantage of this method is that it uses a single low-rank tensor to score features with different numbers of lexical parts. This is particularly helpful when we have very limited labeled data. We denote this method as LRFR$_1$-BROWN, since we use Brown clusters in practice. In the experiments we use the

Tucker form for LRFR$_1$-BROWN.

6 Parameter Estimation

The goal of learning is to find a tensor $\mathcal{T}$ that solves problem (5). Note that this is a non-convex objective, so compared to the convex objective in a traditional log-linear model, we are trading better feature representations with the cost of a harder optimization problem. While stochastic gradient descent (SGD) is a natural choice for learning representations in large data settings, problem (5) involves rank constraints, which require an expensive proximal operation to enforce the constraints at each iteration of SGD. We seek a more efficient learning algorithm. Note that we fixed the size of each transformation matrix $\mathrm{W}_i \in \mathbb{R}^{r_i \times d_i}$ so that the smaller dimension ($r_i < d_i$) matches the upper bound on the rank. Therefore, the rank constants are always satisfied through a run of SGD and we in essence have an unconstrained optimization problem. Note that in this way we do not guarantee orthogonality and full-rank of the learned transformation matrices. These properties are assumed in general, but are not necessary according to (Kolda and Bader, 2009).

The gradients are computed via the chain-rule. We use AdaGrad (Duchi et al., 2011) and apply L2 regularization on all W_is and g, except for the case of r_i=d_i, where we will start with $\mathrm{W}_i = \mathbf{I}$ and regularize with $\|\mathrm{W}_i - \mathbf{I}\|_2$. We use early-stopping on a development set.

7 Experimental Settings

We evaluate LRFR on three tasks: relation extraction, PP attachment and preposition disambiguation (see Table 1 for a task summary). We include detailed feature templates in Table 2.

PP-attachment and relation extraction are two fundamental NLP tasks, and we test our models on the largest English data sets. The preposition disambiguation task was designed for compositional semantics, which is an important application of deep learning and distributed representations. On all these tasks, we compare to the state-of-the-art.

We use the same word embeddings in Belinkov et al. (2014) on PP-attachment for a fair comparison. For the other experiments, we use the same 200-d word embeddings in Yu et al. (2015).

Relation Extraction We use the English portion of the ACE 2005 relation extraction dataset (Walker et al., 2006). Following Yu et al. (2015), we use both gold entity spans and types, train the model on the news domain and test on the broadcast conversation domain. To highlight the impact of training data size we evaluate with all 43,518 relations (entity mention pairs) and a reduced training set of the first 10,000 relations. We report precision, recall, and F1.

We compare to two baseline methods: 1) a log-linear model with a rich binary feature set from Sun et al. (2011) and Zhou et al. (2005) as described in Yu et al. (2015) (BASELINE); 2) the embedding model (FCM) of Gormley et al. (2015), which uses rich linguistic features for relation extraction. We use the same feature templates and evaluate on fine-grained relations (sub-types, 32 labels) (Yu et al., 2015). This will evaluate how LRFR can utilize non-lexical linguistic features.

PP-attachment We consider the prepositional phrase (PP) attachment task of Belinkov et al. (2014),[3] where for each PP the correct head (verbs or nouns) must be selected from content words before the PP (within a 10-word window). We formulate the task as a ranking problem, where we optimize the score of the correct head from a list of candidates with varying sizes.

PP-attachment suffers from data sparsity because of bi-lexical features, which we will model with methods in §5. Belikov et al. show that rich features – POS, WordNet and VerbNet – help this task. The combination of these features give a large number of non-lexical properties, for which embeddings of non-lexical properties in LRFR should be useful.

We extract a dev set from section 22 of the PTB following the description in Belinkov et al. (2014).

Preposition Disambiguation We consider the preposition disambiguation task proposed by Ritter et al. (2014). The task is to determine the spatial relationship a preposition indicates based on the two objects connected by the preposition. For example, "the apple on the refrigerator" indicates the "support by Horizontal Surface" relation, while "the apple on the branch" indicates the "Support from Above" relation. Since the meaning of a preposition depends

[3] http://groups.csail.mit.edu/rbg/code/pp

Task	Benchmark	Dataset	Numbers on Each View	
			#Labels (d_1)	#Non-lexical Features (d_2)
Relation Extraction	Yu et al. (2015)	ACE 2005	32	264
PP-attachment	Belinkov et al. (2014)	WSJ	-	1,213 / 607
Preposition Disambiguation	Ritter et al. (2014)	Ritter et al. (2014)	6	9/3

Table 1: Statistics of each task. PP-attachment and preposition disambiguation have both unigram and bigram features. Therefore we list the numbers of non-lexical properties for both types.

Set	Template
HeadEmb	$\{I[i=h_1], I[i=h_2]\}$ (head of M_1/M_2) $\&\{\phi, t_{h_1}, t_{h_2}, t_{h_1}\&t_{h_2}\}$
Context	$I[i = h_1/h_2 \pm 1]$ (left/right token of w_{h_1/h_2})
In-between	$I[i > h_1]\&I[i < h_2]\&\{\phi, t_{h_1}, t_{h_2}, t_{h_1}\&t_{h_2}\}$
On-path	$I[w_i \in P]\ \&\{\phi, t_{h_1}, t_{h_2}, t_{h_1}\&t_{h_2}\}$

Set	Template
Bag of Words	w, p & w (w is w_m or w_h)
Word-Position	w_m, w_h, w_m & w_h
Preposition	p, p & w_m, p & w_h, p & w_m & w_h

Set	Template
Bag of Words	w (w is w_m or w_h), $w_m\&w_h$
Distance	$\text{Dis}(w_h, w_m)$ & $\{w_m, w_h, w_m\&w_h\}$
Prep	w_p & $\{w_m, w_h, w_m\&w_h\}$
POS	$t(w_h)$ & $\{w_m, w_h, w_m\&w_h\}$
NextPOS	$t(w_{h+1})$ & $\{w_m, w_h, w_m\&w_h\}$
VerbNet	$P = \{p(w_h)\}$ & $\{w_m, w_h, w_m\&w_h\}$
	$I[w_p \in P]$ & $\{w_m, w_h, w_m\&w_h\}$
WordNet	$R_h = \{r(w_h)\}$ & $\{w_m, w_h, w_m\&w_h\}$
	$R_m = \{r(w_m)\}$ & $\{w_m, w_h, w_m\&w_h\}$

Table 2: **Up-left**: Unigram lexical features (only showing non-lexical parts) for **relation extraction** (from Yu et al. (2014)). We denote the two target entities as M_1, M_2 (with head indices h_1, h_2, NE types t_{h_1}, t_{h_2}), and their dependency path as P. **Right**: Uni/bi-gram feature for **PP-attachment**: Each feature is defined on tuple (w_m, w_p, w_h), where w_p is the preposition word, w_m is the child of the preposition, and w_h is a candidate head of w_p. $t(w)$: POS tag of word w; $p(w)$: a preposition collocation of verb w from VerbNet; $r(w)$: the root hypernym of word w in WordNet. $\text{Dis}(\cdot,\cdot)$: the number of candidate heads between two words. **Down-left**: Uni/bi-gram feature for **preposition disambiguation** (for each preposition word p, its modifier noun w_m and head noun w_h). Since the sentences are different from each other on only p, w_m and w_h, we ignore the words on the other positions.

on the combination of both its head and child word, we expect conjunctions between these word embeddings to help, i.e. features with two lexical parts.

We include three baselines: point-wise addition (SUM) (Mitchell and Lapata, 2010), concatenation (Ritter et al., 2014), and an SVM based on handcrafted features in Table 2. Ritter et al. show that the first two methods beat other compositional models.

Hyperparameters are all tuned on the dev set. The chosen values are learning rate $\eta = 0.05$ and the weight of L2 regularizer $\lambda = 0.005$ for LRFR, except for the third LRFR in Table 3 which has $\lambda = 0.05$. We select the rank of LRFR-TUCKER with a grid search from the following values: $r_1 = \{10, 20, d_1\}$, $r_2 = \{20, 50, d_2\}$ and $r_3 = \{50, 100, 200\}$. For LRFR-CP, we select $r = \{50, 100, 200\}$. For the *PP-attachement* task there is no r_1 since it uses a ranking model. For the *Preposition Disambiguation* we do not choose r_1 since the number of labels is small.

8 Results

Relation Extraction All LRFR-TUCKER models improve over BASELINE and FCM (Table 3), making these the best reported numbers for this task. However, LRFR-CP does not work as well on the features with only one lexical part. The Tucker-form does a better job of capturing interactions between different views. In the limited training setting, we find that LRFR-CP does best.

Additionally, the primary advantage of the CP approximation is its reduction in the number of model parameters and running time. We report each model's running time for a single pass on the development set. The LRFR-CP is by far the fastest. The first three LRFR-TUCKER models are slightly slower than FCM, because they work on dense non-lexical property embeddings while FCM benefits from sparse vectors.

PP-attachment Table 4 shows that LRFR (89.6 and 90.3) improves over the previous best standalone system HPCD (88.7) by a large margin, with exactly the same resources. Belinkov et al. (2014) also reported results of parsers and parser re-rankers, which can access to additional resources (complete parses for training and complete sentences as inputs) so it is unfair to compare them with the standalone systems like HPCD and our LRFR. Nonethe-

Method	Parameters			Full Set ($\|D\|$=43,518)			Reduced Set ($\|D\|$=10,000)			Prediction
	$\mathbf{r_1}$	$\mathbf{r_2}$	$\mathbf{r_3}$	**P**	**R**	**F1**	**P**	**R**	**F1**	**Time (ms)**
BASELINE	-	-	-	60.2	**51.2**	55.3	-	-	-	-
FCM	32/N	264/N	200/N	62.9	49.6	55.4	**61.6**	37.1	46.3	2,242
LRFR$_1$-TUCKER	32/N	20/Y	200/Y	62.1	**52.7**	**57.0**	51.5	40.8	45.5	3,076
LRFR$_1$-TUCKER	32/N	20/Y	200/N	**63.5**	51.1	56.6	52.8	40.1	45.6	2,972
LRFR$_1$-TUCKER	20/Y	20/Y	200/Y	62.4	51.0	56.1	52.1	41.2	46.0	2,538
LRFR$_1$-TUCKER	32/Y	20/Y	50/Y	57.4	52.4	54.8	49.7	**46.1**	47.8	1,198
LRFR$_1$-CP	200/Y			61.3	50.7	55.5	58.3	41.6	**48.6**	**502**

Table 3: Results on test for relation extraction. Y(es)/N(o) indicates whether embeddings are updated during training.

System	Resources Used	Acc
SVM (Belinkov et al., 2014)	distance, word, embedding, clusters, POS, WordNet, VerbNet	86.0
HPCD (Belinkov et al., 2014)	distance, embedding, POS, WordNet, VerbNet	88.7
LRFR$_1$-TUCKER & LRFR$_2$-CP	distance, embedding, POS, WordNet, VerbNet	**90.3**
LRFR$_1$-BROWN	distance, embedding, clusters, POS, WordNet, VerbNet	89.6
RBG (Lei et al., 2014)	dependency parser	88.4
Charniak-RS (McClosky et al., 2006)	dependency parser + re-ranker	88.6
RBG + HPCD (combined model)	dependency parser + distance, embedding, POS, WordNet, VerbNet	90.1

Table 4: PP-attachment test accuracy. The baseline results are from Belinkov et al. (2014).

less LRFR$_1$-TUCKER & LRFR$_2$-CP (90.3) still outperforms the state-of-the-art parser RBG (88.4), reranker Charniak-RS (88.6), and the combination of the state-of-the-art parser and compositional model RBG + HPCD (90.1). Thus, even with fewer resources, LRFR becomes the new best system.

Not shown in the table: we also tried LRFR$_1$-TUCKER & LRFR$_2$-CP with *postag features only* (89.7), and with grand-head-modifier conjunctions removed (89.3) . Note that compared to LRFR, RBG benefits from binary features, which also exploit grand-head-modifier structures. Yet the above reduced models still work better than RBG (88.4) without using additional resources.[4] Moreover, the results of LRFR can still be potentially improved by combining with binary features. The above results show the advantage of our factorization method, which allows for utilizing pre-trained word embeddings, and thus can benefit from semi-supervised learning.

Preposition Disambiguation LRFR improves (Table 5) over the best methods (SUM and Concatenation) in Ritter et al. (2014) as well as the SVM based on the original lexical features (85.1). In this task LRFR$_1$-BROWN better represents the unigram and bigram lexical features, compared to the usage of two low-rank tensors (LRFR$_1$-TUCKER & LRFR$_2$-CP). This may be because LRFR$_1$-BROWN has fewer parameters, which is better for smaller training sets.

Method	Accuracy
SVM - Lexical Features	85.09
SUM	80.55
Concatenation	86.73
LRFR$_1$-TUCKER & LRFR$_2$-CP	87.82
LRFR$_1$-BROWN	**88.18**
LRFR$_1$-BROWN - Control	84.18

Table 5: Accuracy for spatial classification of PPs.

We also include a control setting (LRFR$_1$-BROWN - Control), which has a full rank parameter tensor with the same inputs on each view as LRFR$_1$-BROWN, but represented as one hot vectors without transforming to the hidden representations $\mathbf{h}$s. This is equivalent to an SVM with the compound cluster features as in Koo et al. (2008). It performs much worse than LRFR$_1$-BROWN, showing the advantage of using word embeddings and low-rank tensors.

Summary For unigram lexical features, LRFR$_n$-TUCKER achieves better results than LRFR$_n$-CP. However, in settings with fewer training examples,

[4] Still this is not a fair comparison since we have different training objectives. Using RBG's factorization and training with our objective will give a fair comparison and we leave it to future work.

features with more lexical parts (n-grams), or when faster predictions are advantageous, LRFR$_n$-CP does best as it has fewer parameters to estimate. For n-grams of variable length, LRFR$_1$-TUCKER & LRFR$_2$-CP does best. In settings with fewer training examples, LRFR$_1$-BROWN does best as it has only one parameter tensor to estimate.

9 Related Work

Dimensionality Reduction for Complex Features is a standard technique to address high-dimensional features, including PCA, alternating structural optimization (Ando and Zhang, 2005), denoising autoencoders (Vincent et al., 2008), and feature embeddings (Yang and Eisenstein, 2015). These methods treat features as atomic elements and ignore the inner structure of features, so they learn separate embedding for each feature without shared parameters. As a result, they still suffer from large parameter spaces when the feature space is very huge.[5]

Another line of research studies the inner structures of lexical features: e.g. Koo et al. (2008), Turian et al. (2010), Sun et al. (2011), Nguyen and Grishman (2014), Roth and Woodsend (2014), and Hermann et al. (2014) used pre-trained word embeddings to replace the lexical parts of features ; Srikumar and Manning (2014), Gormley et al. (2015) and Yu et al. (2015) propose splitting lexical features into different parts and employing tensors to perform classification. The above can therefore be seen as special cases of our model that only embed a certain part (view) of the complex features. This restriction also makes their model parameters form a full rank tensor, resulting in data sparsity and high computational costs when the tensors are large.

Composition Models (Deep Learning) build representations for structures based on their component word embeddings (Collobert et al., 2011; Bordes et al., 2012; Socher et al., 2012; Socher et al., 2013b). When using only word embeddings, these models achieved successes on several NLP tasks, but sometimes fail to learn useful syntactic or semantic patterns beyond the strength of combinations of word embeddings, such as the dependency relation in Figure 1(a). To tackle this problem, some work designed their model structures according to a specific kind of linguistic patterns, e.g. dependency paths (Ma et al., 2015; Liu et al., 2015), while a recent trend enhances compositional models with linguistic features. For example, Belinkov et al. (2014) concatenate embeddings with linguistic features before feeding them to a neural network; Socher et al. (2013a) and Hermann and Blunsom (2013) enhanced Recursive Neural Networks by refining the transformation matrices with linguistic features (e.g. phrase types). These models are similar to ours in the sense of learning representations based on linguistic features and embeddings.

Low-rank Tensor Models for NLP aim to handle the conjunction among different views of features (Cao and Khudanpur, 2014; Lei et al., 2014; Chen and Manning, 2014). Yu and Dredze (2015) proposed a model to compose phrase embeddings from words, which has an equivalent form of our CP-based method under certain restrictions. Our work applies a similar idea to exploiting the inner structure of complex features, and can handle n-gram features with different ns. Our factorization (§3) is general and easy to adapt to new tasks. More importantly, it makes the model benefit from pre-trained word embeddings as shown by the PP-attachment results.

10 Conclusion

We have presented LRFR, a feature representation model that exploits the inner structure of complex lexical features and applies a low-rank tensor to efficiently score features with this representation. LRFR attains the state-of-the-art on several tasks, including relation extraction, PP-attachment, and preposition disambiguation. We make our implementation available for general use.[6]

Acknowledgements

A major portion of this work was done when MY was visiting MD and RA at JHU. This research was supported in part by NSF grant IIS-1546482.

[5] For example, a state-of-the-art dependency parser (Zhang and McDonald, 2014) extracts about 10 million features; in this case, learning 100-dimensional feature embeddings involves estimating approximately a billion parameters.

[6] https://github.com/Gorov/LowRankFCM

References

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *The Journal of Machine Learning Research*, 6.

Yonatan Belinkov, Tao Lei, Regina Barzilay, and Amir Globerson. 2014. Exploring compositional architectures and word vector representations for prepositional phrase attachment. *Transactions of the Association for Computational Linguistics*, 2.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*. Springer.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2012. A semantic matching energy function for learning with multi-relational data. *Machine Learning*.

Yuan Cao and Sanjeev Khudanpur. 2014. Online learning in tensor space. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Danqi Chen and Christopher 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. *JMLR*, 12.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12.

Matthew R. Gormley, Mo Yu, and Mark Dredze. 2015. Improved relation extraction with feature-rich compositional embedding models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Karl Moritz Hermann and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Association for Computational Linguistics*.

Karl Moritz Hermann, Dipanjan Das, Jason Weston, and Kuzman Ganchev. 2014. Semantic frame identification with distributed word representations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. 2010. Learning 5000 relational extractors. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review*, 51(3).

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

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Tao Lei, Yuan Zhang, Lluís Màrquez, Alessandro Moschitti, and Regina Barzilay. 2015. High-order low-rank tensors for semantic role labeling. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Yang Liu, Furu Wei, Sujian Li, Heng Ji, Ming Zhou, and Houfeng WANG. 2015. A dependency-based neural network for relation classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*.

Mingbo Ma, Liang Huang, Bowen Zhou, and Bing Xiang. 2015. Dependency-based convolutional neural networks for sentence embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In *Proceedings of HLT-NAACL*.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8).

Thien Huu Nguyen and Ralph Grishman. 2014. Employing word representations and regularization for domain adaptation of relation extraction. In *Association for Computational Linguistics (ACL)*.

Samuel Ritter, Cotie Long, Denis Paperno, Marco Baroni, Matthew Botvinick, and Adele Goldberg. 2014.

Leveraging preposition ambiguity to assess representation of semantic interaction in cdsm. In *NIPS Workshop on Learning Semantics*.

Michael Roth and Kristian Woodsend. 2014. Composition of word representations improves semantic role labelling. In *Proceedings of EMNLP*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of EMNLP-CoNLL 2012*.

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*.

Vivek Srikumar and Christopher D Manning. 2014. Learning distributed representations for structured output prediction. In *Advances in Neural Information Processing Systems*.

Ang Sun, Ralph Grishman, and Satoshi Sekine. 2011. Semi-supervised relation extraction with large-scale word clustering. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Hillel Taub-Tabib, Yoav Goldberg, and Amir Globerson. 2015. Template kernels for dependency parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Association for Computational Linguistics*.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. ACE 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*.

Yi Yang and Jacob Eisenstein. 2015. Unsupervised multi-domain adaptation with feature embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 672–682, Denver, Colorado, May–June. Association for Computational Linguistics.

Mo Yu and Mark Dredze. 2015. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3.

Mo Yu, Matthew R. Gormley, and Mark Dredze. 2015. Combining word embeddings and feature embeddings for fine-grained relation extraction. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of ACL*.

GuoDong Zhou, Jian Su, Jie Zhang, and Min Zhang. 2005. Exploring various knowledge in relation extraction. In *Proceedings of ACL*.