Low-Rank Tensors for Scoring Dependency Structures

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{taolei, yuxin, yuanzh, regina, tommi}@csail.mit.edu

Abstract

Accurate scoring of syntactic structures such as head-modifier arcs in dependency parsing typically requires rich, high-dimensional feature representations. A small subset of such features is often selected manually. This is problematic when features lack clear linguistic meaning as in embeddings or when the information is blended across features. In this paper, we use tensors to map high-dimensional feature vectors into low dimensional representations. We explicitly maintain the parameters as a low-rank tensor to obtain low dimensional representations of words in their syntactic roles, and to leverage modularity in the tensor for easy training with online algorithms. Our parser consistently outperforms the Turbo and MST parsers across 14 different languages. We also obtain the best published UAS results on 5 languages.[1]

1 Introduction

Finding an expressive representation of input sentences is crucial for accurate parsing. Syntactic relations manifest themselves in a broad range of surface indicators, ranging from morphological to lexical, including positional and part-of-speech (POS) tagging features. Traditionally, parsing research has focused on modeling the direct connection between the features and the predicted syntactic relations such as head-modifier (arc) relations in dependency parsing. Even in the case of first-order parsers, this results in a high-dimensional vector representation of each arc. Discrete features, and their cross products, can be further complemented with auxiliary information about words participating in an arc, such as continuous vector representations of words. The exploding dimensionality of rich feature vectors must then be balanced with the difficulty of effectively learning the associated parameters from limited training data.

A predominant way to counter the high dimensionality of features is to manually design or select a meaningful set of feature templates, which are used to generate different types of features (McDonald et al., 2005a; Koo and Collins, 2010; Martins et al., 2013). Direct manual selection may be problematic for two reasons. First, features may lack clear linguistic interpretation as in distributional features or continuous vector embeddings of words. Second, designing a small subset of templates (and features) is challenging when the relevant linguistic information is distributed across the features. For instance, morphological properties are closely tied to part-of-speech tags, which in turn relate to positional features. These features are not redundant. Therefore, we may suffer a performance loss if we select only a small subset of the features. On the other hand, by including all the rich features, we face over-fitting problems.

We depart from this view and leverage high-dimensional feature vectors by mapping them into low dimensional representations. We begin by representing high-dimensional feature vectors as multi-way cross-products of smaller feature vectors that represent words and their syntactic relations (arcs). The associated parameters are viewed as a tensor (multi-way array) of low rank, and optimized for parsing performance. By explicitly representing the tensor in a low-rank form, we have direct control over the effective dimensionality of the set of parameters. We obtain role-dependent low-dimensional representations for words (head, modifier) that are specifically tailored for parsing accuracy, and use standard online algorithms for optimizing the low-rank tensor components.

The overall approach has clear linguistic and

[1] Our code is available at https://github.com/taolei87/RBGParser.

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 1381–1391,
Baltimore, Maryland, USA, June 23-25 2014.

computational advantages:

- Our low dimensional embeddings are tailored to the syntactic context of words (head, modifier). This low dimensional syntactic abstraction can be thought of as a proxy to manually constructed POS tags.
- By automatically selecting a small number of dimensions useful for parsing, we can leverage a wide array of (correlated) features. Unlike parsers such as MST, we can easily benefit from auxiliary information (e.g., word vectors) appended as features.

We implement the low-rank factorization model in the context of first- and third-order dependency parsing. The model was evaluated on 14 languages, using dependency data from CoNLL 2008 and CoNLL 2006. We compare our results against the MST (McDonald et al., 2005a) and Turbo (Martins et al., 2013) parsers. The low-rank parser achieves average performance of 89.08% across 14 languages, compared to 88.73% for the Turbo parser, and 87.19% for MST. The power of the low-rank model becomes evident in the absence of any part-of-speech tags. For instance, on the English dataset, the low-rank model trained without POS tags achieves 90.49% on first-order parsing, while the baseline gets 86.70% if trained under the same conditions, and 90.58% if trained with 12 core POS tags. Finally, we demonstrate that the model can successfully leverage word vector representations, in contrast to the baselines.

2 Related Work

Selecting Features for Dependency Parsing A great deal of parsing research has been dedicated to feature engineering (Lazaridou et al., 2013; Marton et al., 2010; Marton et al., 2011). While in most state-of-the-art parsers, features are selected manually (McDonald et al., 2005a; McDonald et al., 2005b; Koo and Collins, 2010; Martins et al., 2013; Zhang and McDonald, 2012a; Rush and Petrov, 2012a), automatic feature selection methods are gaining popularity (Martins et al., 2011b; Ballesteros and Nivre, 2012; Nilsson and Nugues, 2010; Ballesteros, 2013). Following standard machine learning practices, these algorithms iteratively select a subset of features by optimizing parsing performance on a development set. These feature selection methods are particularly promising in parsing scenarios where the optimal feature set is likely to be a small subset of the original set of candidate features. Our technique, in contrast, is suitable for cases where the relevant information is distributed across a larger set of related features.

Embedding for Dependency Parsing A lot of recent work has been done on mapping words into vector spaces (Collobert and Weston, 2008; Turian et al., 2010; Dhillon et al., 2011; Mikolov et al., 2013). Traditionally, these vector representations have been derived primarily from co-occurrences of words within sentences, ignoring syntactic roles of the co-occurring words. Nevertheless, any such word-level representation can be used to offset inherent sparsity problems associated with full lexicalization (Cirik and Şensoy, 2013). In this sense they perform a role similar to POS tags.

Word-level vector space embeddings have so far had limited impact on parsing performance. From a computational perspective, adding non-sparse vectors directly as features, including their combinations, can significantly increase the number of active features for scoring syntactic structures (e.g., dependency arc). Because of this issue, Cirik and Şensoy (2013) used word vectors only as unigram features (without combinations) as part of a shift reduce parser (Nivre et al., 2007). The improvement on the overall parsing performance was marginal. Another application of word vectors is compositional vector grammar (Socher et al., 2013). While this method learns to map word combinations into vectors, it builds on existing word-level vector representations. In contrast, we represent words as vectors in a manner that is directly optimized for parsing. This framework enables us to learn new syntactically guided embeddings while also leveraging separately estimated word vectors as starting features, leading to improved parsing performance.

Dimensionality Reduction Many machine learning problems can be cast as matrix problems where the matrix represents a set of co-varying parameters. Such problems include, for example, multi-task learning and collaborative filtering. Rather than assuming that each parameter can be set independently of others, it is helpful to assume that the parameters vary in a low dimensional subspace that has to be estimated together with the parameters. In terms of the parameter matrix, this corresponds to a low-rank assumption. Low-rank constraints are commonly used for improving

generalization (Lee and Seung, 1999; Srebro et al., 2003; Srebro et al., 2004; Evgeniou and Pontil, 2007)

A strict low-rank assumption can be restrictive. Indeed, recent approaches to matrix problems decompose the parameter matrix as a sum of low-rank and sparse matrices (Tao and Yuan, 2011; Zhou and Tao, 2011). The sparse matrix is used to highlight a small number of parameters that should vary independently even if most of them lie on a low-dimensional subspace (Waters et al., 2011; Chandrasekaran et al., 2011). We follow this decomposition while extending the parameter matrix into a tensor.

Tensors are multi-way generalizations of matrices and possess an analogous notion of rank. Tensors are increasingly used as tools in spectral estimation (Hsu and Kakade, 2013), including in parsing (Cohen et al., 2012) and other NLP problems (de Cruys et al., 2013), where the goal is to avoid local optima in maximum likelihood estimation. In contrast, we expand features for parsing into a multi-way tensor, and operate with an explicit low-rank representation of the associated parameter tensor. The explicit representation sidesteps inherent complexity problems associated with the tensor rank (Hillar and Lim, 2009). Our parameters are divided into a sparse set corresponding to manually chosen MST or Turbo parser features and a larger set governed by a low-rank tensor.

3 Problem Formulation

We will commence here by casting first-order dependency parsing as a tensor estimation problem. We will start by introducing the notation used in the paper, followed by a more formal description of our dependency parsing task.

3.1 Basic Notations

Let $A \in \mathbb{R}^{n \times n \times d}$ be a 3-dimensional tensor (a 3-way array). We denote each element of the tensor as $A_{i,j,k}$ where $i \in [n], j \in [n], k \in [d]$ and $[n]$ is a shorthand for the set of integers $\{1, 2, \cdots, n\}$. Similarly, we use $M_{i,j}$ and u_i to represent the elements of matrix M and vector u, respectively.

We define the *inner product* of two tensors (or matrices) as $\langle A, B \rangle = \text{vec}(A)^T \text{vec}(B)$, where $\text{vec}(\cdot)$ concatenates the tensor (or matrix) elements into a column vector. The *squared norm* of a tensor/matrix is denoted by $\|A\|^2 = \langle A, A \rangle$.

The *Kronecker product* of three vectors is denoted by $u \otimes v \otimes w$ and forms a rank-1 tensor such that

$$(u \otimes v \otimes w)_{i,j,k} = u_i v_j w_k.$$

Note that the vectors u, v, and w may be column or row vectors. Their orientation is defined based on usage. For example, $u \otimes v$ is a rank-1 matrix uv^T when u and v are column vectors ($u^T v$ if they are row vectors).

We say that tensor A is in Kruskal form if

$$A = \sum_{i=1}^{r} U(i,:) \otimes V(i,:) \otimes W(i,:) \qquad (1)$$

where $U, V \in \mathbb{R}^{r \times n}$, $W \in \mathbb{R}^{r \times d}$ and $U(i,:)$ is the i^{th} row of matrix U. We will directly learn a low-rank tensor A (because r is small) in this form as one of our model parameters.

3.2 Dependency Parsing

Let x be a sentence and $\mathcal{Y}(x)$ the set of possible dependency trees over the words in x. We assume that the score $S(x, y)$ of each candidate dependency tree $y \in \mathcal{Y}(x)$ decomposes into a sum of "local" scores for arcs. Specifically:

$$S(x, y) = \sum_{h \to m \in y} s(h \to m) \qquad \forall y \in \mathcal{Y}(x)$$

where $h \to m$ is the head-modifier dependency arc in the tree y. Each y is understood as a collection of arcs $h \to m$ where h and m index words in x.[2] For example, $x(h)$ is the word corresponding to h. We suppress the dependence on x whenever it is clear from context. For example, $s(h \to m)$ can depend on x in complicated ways as discussed below. The predicted parse is obtained as $\hat{y} = \arg\max_{y \in \mathcal{Y}(x)} S(x, y)$.

A key problem is how we parameterize the arc scores $s(h \to m)$. Following the MST parser (McDonald et al., 2005a) we can define rich features characterizing each head-modifier arc, compiled into a sparse binary vector $\phi_{h \to m} \in \mathbb{R}^L$ that depends on the sentence x as well as the chosen arc $h \to m$ (again, we suppress the dependence on x). Based on this feature representation, we define the score of each arc as $s_\theta(h \to m) =$

[2] Note that in the case of high-order parsing, the sum $S(x, y)$ may also include local scores for other syntactic structures, such as grandhead-head-modifier score $s(g \to h \to m)$. See (Martins et al., 2013) for a complete list of these structures.

Unigram features:		
form	form-p	form-n
lemma	lemma-p	lemma-n
pos	pos-p	pos-n
morph	bias	
Bigram features:		
pos-p, pos		
pos, pos-n		
pos, lemma		
morph, lemma		
Trigram features:		
pos-p, pos, pos-n		

Table 1: Word feature templates used by our model. pos, form, lemma and morph stand for the fine POS tag, word form, word lemma and the morphology feature (provided in CoNLL format file) of the current word. There is a bias term that is always active for any word. The suffixes -p and -n refer to the left and right of the current word respectively. For example, pos-p means the POS tag to the left of the current word in the sentence.

$\langle \theta, \phi_{h \to m} \rangle$ where $\theta \in \mathbb{R}^L$ represent adjustable parameters to be learned, and L is the number of parameters (and possible features in $\phi_{h \to m}$).

We can alternatively specify arc features in terms of rank-1 tensors by taking the Kronecker product of simpler feature vectors associated with the head (vector $\phi_h \in \mathbb{R}^n$), and modifier (vector $\phi_m \in \mathbb{R}^n$), as well as the arc itself (vector $\phi_{h,m} \in \mathbb{R}^d$). Here $\phi_{h,m}$ is much lower dimensional than the MST arc feature vector $\phi_{h \to m}$ discussed earlier. For example, $\phi_{h,m}$ may be composed of only indicators for binned arc lengths[3]. ϕ_h and ϕ_m, on the other hand, are built from features shown in Table 1. By taking the cross-product of all these component feature vectors, we obtain the full feature representation for arc $h \to m$ as a rank-1 tensor

$$\phi_h \otimes \phi_m \otimes \phi_{h,m} \in \mathbb{R}^{n \times n \times d}$$

Note that elements of this rank-1 tensor include feature combinations that are not part of the feature crossings in $\phi_{h \to m}$. In this sense, the rank-1 tensor represents a substantial feature expansion. The arc score $s_{tensor}(h \to m)$ associated with the tensor representation is defined analogously as

$$s_{tensor}(h \to m) = \langle A, \phi_h \otimes \phi_m \otimes \phi_{h,m} \rangle$$

where the adjustable parameters A also form a tensor. Given the typical dimensions of the component feature vectors, ϕ_h, ϕ_m, $\phi_{h,m}$, it is not even possible to store all the parameters in A. Indeed, in the full English training set of CoNLL-2008, the tensor involves around 8×10^{11} entries while the MST feature vector has approximately 1.5×10^7 features. To counter this feature explosion, we restrict the parameters A to have low rank.

Low-Rank Dependency Scoring We can represent a rank-r tensor A explicitly in terms of parameter matrices U, V, and W as shown in Eq. 1. As a result, the arc score for the tensor reduces to evaluating $U\phi_h$, $V\phi_m$, and $W\phi_{h,m}$ which are all r dimensional vectors and can be computed efficiently based on any sparse vectors ϕ_h, ϕ_m, and $\phi_{h,m}$. The resulting arc score $s_{tensor}(h \to m)$ is then

$$\sum_{i=1}^{r} [U\phi_h]_i [V\phi_m]_i [W\phi_{h,m}]_i \qquad (2)$$

By learning parameters U, V, and W that function well in dependency parsing, we also learn context-dependent embeddings for words and arcs. Specifically, $U\phi_h$ (for a given sentence, suppressed) is an r dimensional vector representation of the word corresponding to h as a head word. Similarly, $V\phi_m$ provides an analogous representation for a modifier m. Finally, $W\phi_{h,m}$ is a vector embedding of the supplemental arc-dependent information. The resulting embedding is therefore tied to the syntactic roles of the words (and arcs), and learned in order to perform well in parsing.

We expect a dependency parsing model to benefit from several aspects of the low-rank tensor scoring. For example, we can easily incorporate additional useful features in the feature vectors ϕ_h, ϕ_m and $\phi_{h,m}$, since the low-rank assumption (for small enough r) effectively counters the otherwise uncontrolled feature expansion. Moreover, by controlling the amount of information we can extract from each of the component feature vectors (via rank r), the statistical estimation problem does not scale dramatically with the dimensions of ϕ_h, ϕ_m and $\phi_{h,m}$. In particular, the low-rank constraint can help generalize to unseen arcs. Consider a feature $\delta(x(h) = a) \cdot \delta(x(m) =$

[3] In our current version, $\phi_{h,m}$ only contains the binned arc length. Other possible features include, for example, the label of the arc $h \to m$, the POS tags between the head and the modifier, boolean flags which indicate the occurence of in-between punctutations or conjunctions, etc.

$b) \cdot \delta(dis(x,h,m) = c)$ which is non-zero only for an arc $a \rightarrow b$ with distance c in sentence x. If the arc has not been seen in the available training data, it does not contribute to the traditional arc score $s_\theta(\cdot)$. In contrast, with the low-rank constraint, the arc score in Eq. 2 would typically be non-zero.

Combined Scoring Our parsing model aims to combine the strengths of both traditional features from the MST/Turbo parser as well as the new low-rank tensor features. In this way, our model is able to capture a wide range of information including the auxiliary features without having uncontrolled feature explosion, while still having the full accessibility to the manually engineered features that are proven useful. Specifically, we define the arc score $s_\gamma(h \rightarrow m)$ as the combination

$$\begin{aligned}
&(1-\gamma) s_{tensor}(h \rightarrow m) + \gamma s_\theta(h \rightarrow m) \\
&= (1-\gamma) \sum_{i=1}^{r} [U\phi_h]_i [V\phi_m]_i [W\phi_{h,m}]_i \\
&+ \gamma \langle \theta, \phi_{h \rightarrow m} \rangle \qquad (3)
\end{aligned}$$

where $\theta \in \mathbb{R}^L$, $U \in \mathbb{R}^{r \times n}$, $V \in \mathbb{R}^{r \times n}$, and $W \in \mathbb{R}^{r \times d}$ are the model parameters to be learned. The rank r and $\gamma \in [0,1]$ (balancing the two scores) represent hyper-parameters in our model.

4 Learning

The training set $D = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^N$ consists of N pairs, where each pair consists of a sentence x_i and the corresponding gold (target) parse y_i. The goal is to learn values for the parameters θ, U, V and W that optimize the combined scoring function $S_\gamma(x,y) = \sum_{h \rightarrow m \in y} s_\gamma(h \rightarrow m)$, defined in Eq. 3, for parsing performance. We adopt a maximum soft-margin framework for this learning problem. Specifically, we find parameters θ, U, V, W, and $\{\xi_i\}$ that minimize

$$\begin{aligned}
&C \sum_i \xi_i + \|\theta\|^2 + \|U\|^2 + \|V\|^2 + \|W\|^2 \\
&\text{s.t.} \quad S_\gamma(\hat{x}_i, \hat{y}_i) \geq S_\gamma(\hat{x}_i, y_i) + \|\hat{y}_i - y_i\|_1 - \xi_i \\
&\qquad \forall y_i \in \mathcal{Y}(\hat{x}_i),\ \forall i. \qquad (4)
\end{aligned}$$

where $\|\hat{y}_i - y_i\|_1$ is the number of mismatched arcs between the two trees, and ξ_i is a non-negative slack variable. The constraints serve to separate the gold tree from other alternatives in $\mathcal{Y}(\hat{x}_i)$ with a margin that increases with distance.

The objective as stated is not jointly convex with respect to U, V and W due to our explicit representation of the low-rank tensor. However, if we fix any two sets of parameters, for example, if we fix V and W, then the combined score $S_\gamma(x,y)$ will be a linear function of both θ and U. As a result, the objective will be jointly convex with respect to θ and U and could be optimized using standard tools. However, to accelerate learning, we adopt an online learning setup. Specifically, we use the passive-aggressive learning algorithm (Crammer et al., 2006) tailored to our setting, updating pairs of parameter sets, (θ, U), (θ, V) and (θ, W) in an alternating manner. This method is described below.

Online Learning In an online learning setup, we update parameters successively based on each sentence. In order to apply the passive-aggressive algorithm, we fix two of U, V and W (say, for example, V and W) in an alternating manner, and apply a closed-form update to the remaining parameters (here U and θ). This is possible since the objective function with respect to (θ, U) has a similar form as in the original passive-aggressive algorithm. To illustrate this, consider a training sentence x_i. The update involves finding first the best competing tree,

$$\tilde{y}_i = \arg\max_{y_i \in \mathcal{Y}(\hat{x}_i)} S_\gamma(\hat{x}_i, y_i) + \|\hat{y}_i - y_i\|_1 \qquad (5)$$

which is the tree that violates the constraint in Eq. 4 most (i.e. maximizes the loss ξ_i). We then obtain parameter increments $\Delta\theta$ and ΔU by solving

$$\begin{aligned}
&\min_{\Delta\theta,\ \Delta U,\ \xi \geq 0} \quad \frac{1}{2}\|\Delta\theta\|^2 + \frac{1}{2}\|\Delta U\|^2 + C\xi \\
&\text{s.t.} \quad S_\gamma(\hat{x}_i, \hat{y}_i) \geq S_\gamma(\hat{x}_i, \tilde{y}_i) + \|\hat{y}_i - \tilde{y}_i\|_1 - \xi
\end{aligned}$$

In this way, the optimization problem attempts to keep the parameter change as small as possible, while forcing it to achieve mostly zero loss on this single instance. This problem has a closed form solution

$$\begin{aligned}
\Delta\theta &= \min\left\{C, \frac{\text{loss}}{\gamma^2\|d\theta\|^2 + (1-\gamma)^2\|du\|^2}\right\} \gamma d\theta \\
\Delta U &= \min\left\{C, \frac{\text{loss}}{\gamma^2\|d\theta\|^2 + (1-\gamma)^2\|du\|^2}\right\} (1-\gamma) du
\end{aligned}$$

where

$$\begin{aligned}
\text{loss} &= S_\gamma(\hat{x}_i, \tilde{y}_i) + \|\hat{y}_i - \tilde{y}_i\|_1 - S_\gamma(\hat{x}_i, \hat{y}_i) \\
d\theta &= \sum_{h \to m \in \hat{y}_i} \phi_{h \to m} - \sum_{h \to m \in \tilde{y}_i} \phi_{h \to m} \\
du &= \sum_{h \to m \in \hat{y}_i} [(V\phi_m) \odot (W\phi_{h,m})] \otimes \phi_h \\
&- \sum_{h \to m \in \tilde{y}_i} [(V\phi_m) \odot (W\phi_{h,m})] \otimes \phi_h
\end{aligned}$$

where $(u \odot v)_i = u_i v_i$ is the Hadamard (element-wise) product. The magnitude of change of θ and U is controlled by the parameter C. By varying C, we can determine an appropriate step size for the online updates. The updates also illustrate how γ balances the effect of the MST component of the score relative to the low-rank tensor score. When $\gamma = 0$, the arc scores are entirely based on the low-rank tensor and $\Delta\theta = 0$. Note that ϕ_h, ϕ_m, $\phi_{h,m}$, and $\phi_{h \to m}$ are typically very sparse for each word or arc. Therefore du and $d\theta$ are also sparse and can be computed efficiently.

Initialization The alternating online algorithm relies on how we initialize U, V, and W since each update is carried out in the context of the other two. A random initialization of these parameters is unlikely to work well, both due to the dimensions involved, and the nature of the alternating updates. We consider here instead a reasonable deterministic "guess" as the initialization method.

We begin by training our model without any low-rank parameters, and obtain parameters θ. The majority of features in this MST component can be expressed as elements of the feature tensor, i.e., as $[\phi_h \otimes \phi_m \otimes \phi_{h,m}]_{i,j,k}$. We can therefore create a tensor representation of θ such that $B_{i,j,k}$ equals the corresponding parameter value in θ. We use a low-rank version of B as the initialization. Specifically, we unfold the tensor B into a matrix $B^{(h)}$ of dimensions n and nd, where $n = dim(\phi_h) = dim(\phi_m)$ and $d = dim(\phi_{h,m})$. For instance, a rank-1 tensor can be unfolded as $u \otimes v \otimes w = u \otimes \text{vec}(v \otimes w)$. We compute the top-r SVD of the resulting unfolded matrix such that $B^{(h)} = P^T S Q$. U is initialized as P. Each right singular vector $S_i Q(i,:)$ is also a matrix in $\mathbb{R}^{n \times d}$. The leading left and right singular vectors of this matrix are assigned to $V(i,:)$ and $W(i,:)$ respectively. In our implementation, we run one epoch of our model without low-rank parameters and initialize the tensor A.

Parameter Averaging The passive-aggressive algorithm regularizes the increments (e.g. $\Delta\theta$ and ΔU) during each update but does not include any overall regularization. In other words, keeping updating the model may lead to large parameter values and over-fitting. To counter this effect, we use parameter averaging as used in the MST and Turbo parsers. The final parameters are those averaged across all the iterations (cf. (Collins, 2002)). For simplicity, in our algorithm we average U, V, W and θ separately, which works well empirically.

5 Experimental Setup

Datasets We test our dependency model on 14 languages, including the English dataset from CoNLL 2008 shared tasks and all 13 datasets from CoNLL 2006 shared tasks (Buchholz and Marsi, 2006; Surdeanu et al., 2008). These datasets include manually annotated dependency trees, POS tags and morphological information. Following standard practices, we encode this information as features.

Methods We compare our model to MST and Turbo parsers on non-projective dependency parsing. For our parser, we train both a first-order parsing model (as described in Section 3 and 4) as well as a third-order model. The third order parser simply adds high-order features, those typically used in MST and Turbo parsers, into our $s_\theta(x, y) = \langle \theta, \phi(x, y) \rangle$ scoring component. The decoding algorithm for the third-order parsing is based on (Zhang et al., 2014). For the Turbo parser, we directly compare with the recent published results in (Martins et al., 2013). For the MST parser, we train and test using the most recent version of the code.[4] In addition, we implemented two additional baselines, NT-1st (first order) and NT-3rd (third order), corresponding to our model without the tensor component.

Features For the arc feature vector $\phi_{h \to m}$, we use the same set of feature templates as MST v0.5.1. For head/modifier vector ϕ_h and ϕ_m, we show the complete set of feature templates used by our model in Table 1. Finally, we use a similar set of feature templates as Turbo v2.1 for 3rd order parsing.

To add auxiliary word vector representations, we use the publicly available word vectors (Cirik

[4] http://sourceforge.net/projects/mstparser/

	First-order only				High-order				
	Ours	NT-1st	MST	Turbo	Ours-3rd	NT-3rd	MST-2nd	Turbo-3rd	Best Published
Arabic	79.60	78.71	78.3	77.23	79.95	79.53	78.75	79.64	81.12 (Ma11)
Bulgarian	92.30	91.14	90.98	91.76	93.50	92.79	91.56	93.1	94.02 (Zh13)
Chinese	91.43	90.85	90.40	88.49	**92.68**	92.39	91.77	89.98	91.89 (Ma10)
Czech	87.90	86.62	86.18	87.66	**90.50**	89.43	87.3	90.32	90.32 (Ma13)
Danish	90.64	89.80	89.84	89.42	91.39	90.82	90.5	91.48	92.00 (Zh13)
Dutch	84.81	83.77	82.89	83.61	**86.41**	86.08	84.11	86.19	86.19 (Ma13)
English	91.84	91.40	90.59	91.21	93.02	92.82	91.54	93.22	93.22 (Ma13)
German	90.24	89.70	89.54	90.52	91.97	92.26	90.14	92.41	92.41 (Ma13)
Japanese	**93.74**	93.36	93.38	92.78	93.71	93.23	92.92	93.52	93.72 (Ma11)
Portuguese	90.94	90.67	89.92	91.14	91.92	91.63	91.08	92.69	93.03 (Ko10)
Slovene	84.25	83.15	82.09	82.81	86.24	86.07	83.25	86.01	86.95 (Ma11)
Spanish	85.27	84.95	83.79	83.61	**88.00**	87.47	84.33	85.59	87.96 (Zh13)
Swedish	89.86	89.66	88.27	89.36	91.00	90.83	89.05	91.14	91.62 (Zh13)
Turkish	75.84	74.89	74.81	75.98	76.84	75.83	74.39	76.9	77.55 (Ko10)
Average	87.76	87.05	86.5	86.83	89.08	88.66	87.19	88.73	89.43

Table 2: First-order parsing (left) and high-order parsing (right) results on CoNLL-2006 datasets and the English dataset of CoNLL-2008. For our model, the experiments are ran with rank $r = 50$ and hyper-parameter $\gamma = 0.3$. To remove the tensor in our model, we ran experiments with $\gamma = 1$, corresponding to columns NT-1st and NT-3rd. The last column shows results of most accurate parsers among Nivre et al. (2006), McDonald et al. (2006), Martins et al. (2010), Martins et al. (2011a), Martins et al. (2013), Koo et al. (2010), Rush and Petrov (2012b), Zhang and McDonald (2012b) and Zhang et al. (2013).

and Şensoy, 2013), learned from raw data (Globerson et al., 2007; Maron et al., 2010). Three languages in our dataset – English, German and Swedish – have corresponding word vectors in this collection.[5] The dimensionality of this representation varies by language: English has 50 dimensional word vectors, while German and Swedish have 25 dimensional word vectors. Each entry of the word vector is added as a feature value into feature vectors ϕ_h and ϕ_m. For each word in the sentence, we add its own word vector as well as the vectors of its left and right words.

We should note that since our model parameter A is represented and learned in the low-rank form, we only have to store and maintain the low-rank projections $U\phi_h$, $V\phi_m$ and $W\phi_{h,m}$ rather than explicitly calculate the feature tensor $\phi_h \otimes \phi_m \otimes \phi_{h,m}$. Therefore updating parameters and decoding a sentence is still efficient, i.e., linear in the number of values of the feature vector. In contrast, assume we take the cross-product of the auxiliary word vector values, POS tags and lexical items of a word and its context, and add the crossed values into a normal model (in $\phi_{h \to m}$). The number of features for each arc would be at least quadratic, growing into thousands, and would be a significant impediment to parsing efficiency.

Evaluation Following standard practices, we train our full model and the baselines for 10 epochs. As the evaluation measure, we use unlabeled attachment scores (UAS) excluding punctuation. In all the reported experiments, the hyper-parameters are set as follows: $r = 50$ (rank of the tensor), $C = 1$ for first-order model and $C = 0.01$ for third-order model.

[5] https://github.com/wolet/sprml13-word-embeddings

6 Results

Overall Performance Table 2 shows the performance of our model and the baselines on 14 CoNLL datasets. Our model outperforms Turbo parser, MST parser, as well as its own variants without the tensor component. The improvements of our low-rank model are consistent across languages: results for the first order parser are better on 11 out of 14 languages. By comparing NT-1st and NT-3rd (models without low-rank) with our full model (with low-rank), we obtain 0.7% absolute improvement on first-order parsing, and 0.3% improvement on third-order parsing. Our model also achieves the best UAS on 5 languages.

We next focus on the first-order model and gauge the impact of the tensor component. First, we test our model by varying the hyper-parameter γ which balances the tensor score and the traditional MST/Turbo score components. Figure 1 shows the average UAS on CoNLL test datasets after each training epoch. We can see that the improvement of adding the low-rank tensor is consistent across various choices of hyper parame-

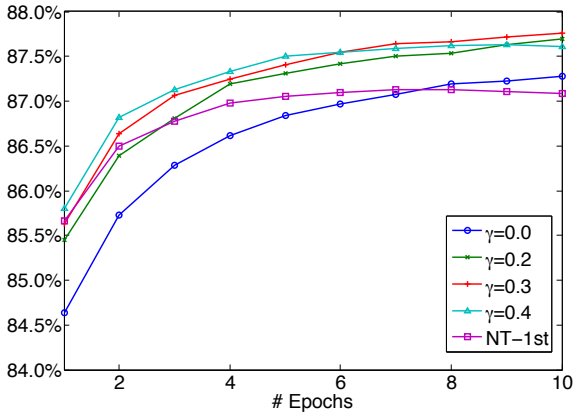

Figure 1: Average UAS on CoNLL testsets after different epochs. Our full model consistently performs better than NT-1st (its variation without tensor component) under different choices of the hyper-parameter γ.

	no word vector	with word vector
English	91.84	92.07
German	90.24	90.48
Swedish	89.86	90.38

Table 3: Results of adding unsupervised word vectors to the tensor. Adding this information yields consistent improvement for all languages.

ter γ. When training with the tensor component alone ($\gamma = 0$), the model converges more slowly. Learning of the tensor is harder because the scoring function is not linear (nor convex) with respect to parameters U, V and W. However, the tensor scoring component achieves better generalization on the test data, resulting in better UAS than NT-1st after 8 training epochs.

To assess the ability of our model to incorporate a range of features, we add unsupervised word vectors to our model. As described in previous section, we do so by appending the values of different coordinates in the word vector into ϕ_h and ϕ_m. As Table 3 shows, adding this information increases the parsing performance for all the three languages. For instance, we obtain more than 0.5% absolute improvement on Swedish.

Syntactic Abstraction without POS Since our model learns a compressed representation of feature vectors, we are interested to measure its performance when part-of-speech tags are not provided (See Table 4). The rationale is that given all other features, the model would induce representations that play a similar role to POS tags. Note that

	Our model		NT-1st	
	-POS	+wv.	-POS	+POS
English	88.89	90.49	86.70	90.58
German	82.63	85.80	78.71	88.50
Swedish	81.84	85.90	79.65	88.75

Table 4: The first three columns show parsing results when models are trained without POS tags. The last column gives the upper-bound, i.e. the performance of a parser trained with **12 Core POS tags**. The low-rank model outperforms NT-1st by a large margin. Adding word vector features further improves performance.

the performance of traditional parsers drops when tags are not provided. For example, the performance gap is 10% on German. Our experiments show that low-rank parser operates effectively in the absence of tags. In fact, it nearly reaches the performance of the original parser that used the tags on English.

Examples of Derived Projections We manually analyze low-dimensional projections to assess whether they capture syntactic abstraction. For this purpose, we train a model with only a tensor component (such that it has to learn an accurate tensor) on the English dataset and obtain low dimensional embeddings $U\phi_w$ and $V\phi_w$ for each word. The two r-dimension vectors are concatenated as an "averaged" vector. We use this vector to calculate the cosine similarity between words. Table 5 shows examples of five closest neighbors of queried words. While these lists include some noise, we can clearly see that the neighbors exhibit similar syntactic behavior. For example, "on" is close to other prepositions. More interestingly, we can consider the impact of syntactic context on the derived projections. The bottom part of Table 5 shows that the neighbors change substantially depending on the syntactic role of the word. For example, the closest words to the word "increase" are verbs in the context phrase "will increase again", while the closest words become nouns given a different phrase "an increase of".

Running Time Table 6 illustrates the impact of estimating low-rank tensor parameters on the running time of the algorithm. For comparison, we also show the NT-1st times across three typical languages. The Arabic dataset has the longest average sentence length, while the Chinese dataset

greatly	**profit**	**says**	**on**	**when**
actively	earnings	adds	with	where
openly	franchisees	predicts	into	what
significantly	shares	noted	at	why
outright	revenue	wrote	during	which
substantially	members	contends	over	who

increase	will **increase** again	an **increase** of
rise	arguing	gain
advance	be	prices
contest	charging	payment
halt	gone	members
Exchequer	making	subsidiary

hit	attacks **hit** the	hardest **hit** is
shed	distributes	monopolies
rallied	stayed	pills
triggered	sang	sophistication
appeared	removed	ventures
understate	eased	factors

Table 5: Five closest neighbors of the queried words (shown in bold). The upper part shows our learned embeddings group words with similar syntactic behavior. The two bottom parts of the table demonstrate that how the projections change depending on the syntactic context of the word.

	#Tok.	Len.	Train. Time (hour)	
			NT-1st	Ours
Arabic	42K	32	0.13	0.22
Chinese	337K	6	0.37	0.65
English	958K	24	1.88	2.83

Table 6: Comparison of training times across three typical datasets. The second column is the number of tokens in each data set. The third column shows the average sentence length. Both first-order models are implemented in Java and run as a single process.

has the shortest sentence length in CoNLL 2006. Based on these results, estimating a rank-50 tensor together with MST parameters only increases the running time by a factor of 1.7.

7 Conclusions

Accurate scoring of syntactic structures such as head-modifier arcs in dependency parsing typically requires rich, high-dimensional feature representations. We introduce a low-rank factorization method that enables to map high dimensional feature vectors into low dimensional representations. Our method maintains the parameters as a low-rank tensor to obtain low dimensional representations of words in their syntactic roles, and to leverage modularity in the tensor for easy training with online algorithms. We implement the approach on first-order to third-order dependency parsing. Our parser outperforms the Turbo and MST parsers across 14 languages.

Future work involves extending the tensor component to capture higher-order structures. In particular, we would consider second-order structures such as grandparent-head-modifier by increasing the dimensionality of the tensor. This tensor will accordingly be a four or five-way array. The online update algorithm remains applicable since each dimension is optimized in an alternating fashion.

8 Acknowledgements

The authors acknowledge the support of the MURI program (W911NF-10-1-0533) and the DARPA BOLT program. This research is developed in collaboration with the Arabic Language Technoligies (ALT) group at Qatar Computing Research Institute (QCRI) within the LYAS project. We thank Volkan Cirik for sharing the unsupervised word vector data. Thanks to Amir Globerson, Andreea Gane, the members of the MIT NLP group and the ACL reviewers for their suggestions and comments. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organizations.

References

Miguel Ballesteros and Joakim Nivre. 2012. MaltOptimizer: An optimization tool for MaltParser. In *EACL*. The Association for Computer Linguistics.

Miguel Ballesteros. 2013. Effective morphological feature selection with MaltOptimizer at the SPMRL 2013 shared task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*. Association for Computational Linguistics.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06. Association for Computational Linguistics.

Venkat Chandrasekaran, Sujay Sanghavi, Pablo A Parrilo, and Alan S Willsky. 2011. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*.

Volkan Cirik and Hüsnü Şensoy. 2013. The AI-KU system at the SPMRL 2013 shared task : Unsupervised features for dependency parsing. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*. Association for Computational Linguistics.

Shay B Cohen, Karl Stratos, Michael Collins, Dean P Foster, and Lyle Ungar. 2012. Spectral learning of latent-variable PCFGs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02. Association for Computational Linguistics.

R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*.

Tim Van de Cruys, Thierry Poibeau, and Anna Korhonen. 2013. A tensor-based factorization model of semantic compositionality. In *HLT-NAACL*. The Association for Computational Linguistics.

Paramveer S. Dhillon, Dean Foster, and Lyle Ungar. 2011. Multiview learning of word embeddings via CCA. In *Advances in Neural Information Processing Systems*.

A Evgeniou and Massimiliano Pontil. 2007. Multi-task feature learning. In *Advances in neural information processing systems: Proceedings of the 2006 conference*. The MIT Press.

Amir Globerson, Gal Chechik, Fernando Pereira, and Naftali Tishby. 2007. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*.

Christopher Hillar and Lek-Heng Lim. 2009. Most tensor problems are NP-hard. *arXiv preprint arXiv:0911.1393*.

Daniel Hsu and Sham M Kakade. 2013. Learning mixtures of spherical gaussians: moment methods and spectral decompositions. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ACM.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10. Association for Computational Linguistics.

Terry Koo, Alexander M Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Angeliki Lazaridou, Eva Maria Vecchi, and Marco Baroni. 2013. Fish transporters and miracle homes: How compositional distributional semantics can help NP parsing. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Daniel D Lee and H Sebastian Seung. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature*.

Yariv Maron, Michael Lamar, and Elie Bienenstock. 2010. Sphere embedding: An application to part-of-speech induction. In *Advances in Neural Information Processing Systems*.

André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2011a. Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11. Association for Computational Linguistics.

André FT Martins, Noah A Smith, Pedro MQ Aguiar, and Mário AT Figueiredo. 2011b. Structured sparsity in structured prediction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

André FT Martins, Miguel B Almeida, and Noah A Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Yuval Marton, Nizar Habash, and Owen Rambow. 2010. Improving arabic dependency parsing with lexical and inflectional morphological features. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, SPMRL '10. Association for Computational Linguistics.

Yuval Marton, Nizar Habash, and Owen Rambow. 2011. Improving arabic dependency parsing with form-based and functional morphological features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics.

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

Peter Nilsson and Pierre Nugues. 2010. Automatic discovery of feature sets for dependency parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Coling 2010 Organizing Committee.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics.

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*.

Alexander Rush and Slav Petrov. 2012a. Vine pruning for efficient multi-pass dependency parsing. In *The 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL '12)*.

Alexander M Rush and Slav Petrov. 2012b. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51th Annual Meeting of the Association for Computational Linguistics*.

Nathan Srebro, Tommi Jaakkola, et al. 2003. Weighted low-rank approximations. In *ICML*.

Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. 2004. Maximum-margin matrix factorization. In *Advances in neural information processing systems*.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08. Association for Computational Linguistics.

Min Tao and Xiaoming Yuan. 2011. Recovering low-rank and sparse components of matrices from incomplete and noisy observations. *SIAM Journal on Optimization*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10. Association for Computational Linguistics.

Andrew E Waters, Aswin C Sankaranarayanan, and Richard Baraniuk. 2011. SpaRCS: Recovering low-rank and sparse matrices from compressive measurements. In *Advances in Neural Information Processing Systems*.

Hao Zhang and Ryan McDonald. 2012a. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2012b. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics.

Hao Zhang, Liang Huang Kai Zhao, and Ryan McDonald. 2013. Online learning for inexact hypergraph search. In *Proceedings of EMNLP*.

Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. 2014. Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Tianyi Zhou and Dacheng Tao. 2011. Godec: Randomized low-rank & sparse matrix decomposition in noisy case. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*.