Transducing Sentences to Syntactic Feature Vectors: an Alternative Way to “*Parse*”?

Fabio Massimo Zanzotto
University of Rome “Tor Vergata”
Via del Politecnico 1
00133 Roma, Italy
fabio.massimo.zanzotto@uniroma2.it

Lorenzo Dell’Arciprete
University of Rome “Tor Vergata”
Via del Politecnico 1
00133 Roma, Italy
lorenzo.dellarciprete@gmail.com

Abstract

Classification and learning algorithms use syntactic structures as *proxies* between source sentences and feature vectors. In this paper, we explore an alternative path to use syntax in feature spaces: the *Distributed Representation “Parsers”* (DRP). The core of the idea is straightforward: DRPs directly obtain syntactic feature vectors from sentences without explicitly producing symbolic syntactic interpretations. Results show that DRPs produce feature spaces significantly better than those obtained by existing methods in the same conditions and competitive with those obtained by existing methods with lexical information.

1 Introduction

Syntactic processing is widely considered an important activity in natural language understanding (Chomsky, 1957). Research in natural language processing (NLP) exploits this hypothesis in models and systems. Syntactic features improve performance in high level tasks such as question answering (Zhang and Lee, 2003), semantic role labeling (Gildea and Jurafsky, 2002; Pradhan et al., 2005; Moschitti et al., 2008; Collobert et al., 2011), paraphrase detection (Socher et al., 2011), and textual entailment recognition (MacCartney et al., 2006; Wang and Neumann, 2007; Zanzotto et al., 2009).

Classification and learning algorithms are key components in the above models and in current NLP systems, but these algorithms cannot directly use syntactic structures. The relevant parts of phrase structure trees or dependency graphs are explicitly or implicitly stored in feature vectors.

To fully exploit syntax in learning classifiers, kernel machines (Cristianini and Shawe-Taylor, 2000) use graph similarity algorithms (e.g., (Collins and Duffy, 2002) for trees) as structural kernels (Gärtner, 2003). These structural kernels allow to exploit high-dimensional spaces of syntactic tree fragments by concealing their complexity. These feature spaces, although hidden, still exist. Then, even in kernel machines, symbolic syntactic structures act only as *proxies* between the source sentences and the syntactic feature vectors.

In this paper, we explore an alternative way to use syntax in feature spaces: the *Distributed Representation Parsers* (DRP). The core of the idea is straightforward: DRPs directly bridge the gap between sentences and syntactic feature spaces. DRPs act as syntactic parsers and feature extractors at the same time. We leverage on the *distributed trees* recently introduced by Zanzotto&Dell’Arciprete (2012) and on multiple linear regression models. *Distributed trees* are small vectors that encode the large vectors of the syntactic tree fragments underlying the tree kernels (Collins and Duffy, 2002). These vectors effectively represent the original vectors and lead to performances in NLP tasks similar to tree kernels. Multiple linear regression allows to learn linear DRPs from training data. We experiment with the Penn Treebank data set (Marcus et al., 1993). Results show that DRPs produce distributed trees significantly better than those obtained by existing methods, in the same non-lexicalized conditions, and competitive with those obtained by existing methods with lexical information. Finally, DRPs are extremely faster than existing methods.

The rest of the paper is organized as follows. First, we present the background of our idea (Sec. 2). Second, we fully describe our model (Sec. 3). Then, we report on the experiments (Sec. 4). Finally, we draw some conclusions and outline future work (Sec. 5)

2 Background

Classification and learning algorithms for NLP tasks treat syntactic structures t as vectors in feature spaces $\vec{t} \in \mathbb{R}^m$. Each feature generally represents a substructure τ_i. In simple weighting schemes, feature values are 1 if τ_i is a substructure of t and 0 otherwise. Different weighting schemes are used and possible. Then, learning algorithms exploit these feature vectors in different ways. Decision tree learners (Quinlan, 1993) elect the most representative feature at each iteration, whereas kernel machines (Cristianini and Shawe-Taylor, 2000) exploit similarity between pairs of instances, $s(t_1, t_2)$. This similarity is generally measured as the dot product between the two vectors, i.e. $s(t_1, t_2) = \vec{t_1} \cdot \vec{t_2}$.

The use of syntactic features changed when tree kernels (Collins and Duffy, 2002) appeared. Tree kernels gave the possibility to fully exploit feature spaces of tree fragments. Until then, learning algorithms could not treat these huge spaces. It is infeasible to explicitly represent that kind of feature vectors and to directly compute similarities through dot products. Tree kernels (Collins and Duffy, 2002), by computing similarities between two trees with tree comparison algorithms, exactly determine dot products of vectors in these target spaces. After their introduction, different tree kernels have been proposed (e.g., (Vishwanathan and Smola, 2002; Culotta and Sorensen, 2004; Moschitti, 2006)). Their use spread in many NLP tasks (e.g., (Zhou et al., 2007; Wang and Neumann, 2007; Moschitti et al., 2008; Zanzotto et al., 2009; Zhang and Li, 2009)) and in other areas like biology (Vert, 2002; Hashimoto et al., 2008) and computer security (Düssel et al., 2008; Rieck and Laskov, 2007; Bockermann et al., 2009).

Tree kernels have played a very important role in promoting the use of syntactic information in learning classifiers, but this method obfuscated the fact that syntactic trees are ultimately used as vectors in learning algorithms. To work with the idea of directly obtaining rich syntactic feature vectors from sentences, we need some techniques to make these high-dimensional vectors again explicit, through smaller but expressive vectors.

A solution to the above problem stems from the recently revitalized research in Distributed Representations (DR) (Hinton et al., 1986; Bengio, 2009; Collobert et al., 2011; Socher et al., 2011; Zanzotto and Dell'Arciprete, 2012). Distributed Representations, studied in opposition to symbolic representations (Rumelhart and Mcclelland, 1986), are methods for encoding data structures such as trees into vectors, matrices, or high-order tensors. The targets of these representations are generally propositions, i.e., flat tree structures. The Holographic Reduced Representations (HRR), proposed by Plate (1994), produce nearly orthogonal vectors for different structures by combining circular convolution and randomly generated vectors for basic components (as in (Anderson, 1973; Murdock, 1983)).

Building on HRRs, Distributed Trees (DT) have been proposed to encode deeper trees in low dimensional vectors (Zanzotto and Dell'Arciprete, 2012). DTs approximate the feature space of tree fragments defined for the tree kernels (Collins and Duffy, 2002) and guarantee similar performances of classifiers in NLP tasks such as question classification and textual entailment recognition. Thus, Distributed Trees are good representations of syntactic trees, that we can use in our definition of distributed representation parsers (DRPs).

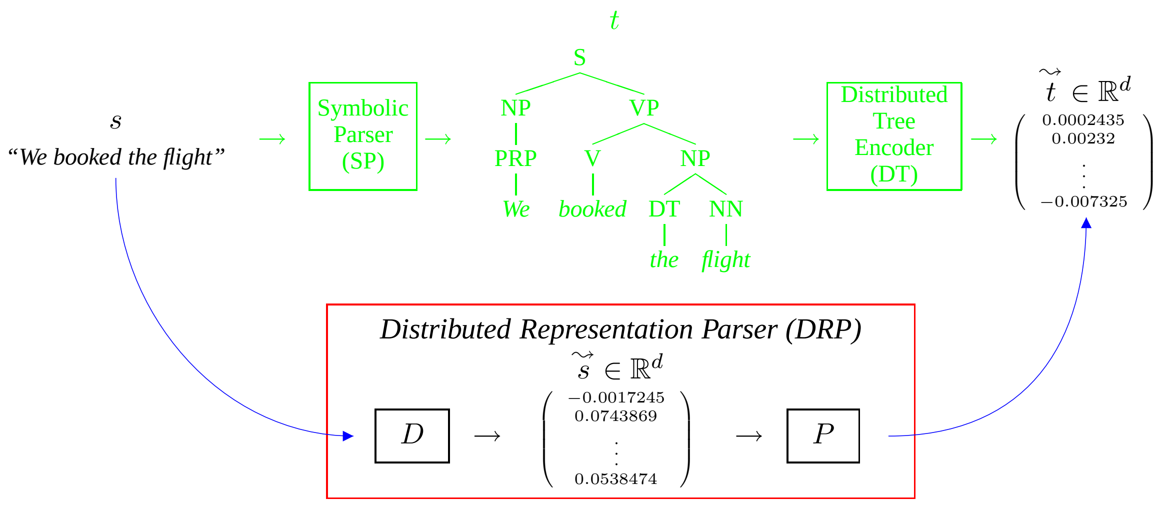
3 Distributed Representation Parsers

In this section, first, we sketch the idea of Distributed Representation "Parsers" (DRPs). Then, we review the *distributed trees* as a way to represent trees in low dimensional vectors. Finally, we describe how to build DRPs by mixing a function that encodes sentences in vectors and a linear regressor that can be induced from training data.

3.1 The Idea

The approach to using syntax in learning algorithms generally follows two steps: first, parse sentences s with a symbolic parser (e.g., (Collins, 2003; Charniak, 2000; Nivre et al., 2007)) and produce symbolic trees t; second, use an encoder to build syntactic feature vectors. Figure 1 sketches this idea when the final vectors are the *distributed trees* $\overset{\rightsquigarrow}{t} \in R^d$ (Zanzotto and Dell'Arciprete, 2012)[1]. In this case, the last step

[1]To represent a distributed tree for a tree t, we use the notation $\overset{\rightsquigarrow}{t}$ to stress that this small vector is an approximation of the original high-dimensional vector $\vec{t}$ in the space of tree



t
S
NP
VP
PRP
V
NP
DT
NN
We
booked
the
flight
s
"We booked the flight"
Symbolic Parser (SP)
Distributed Tree Encoder (DT)
$\overset{\leadsto}{t} \in \mathbb{R}^d$
0.0002435
0.00232
−0.007325
Distributed Representation Parser (DRP)
$\overset{\leadsto}{s} \in \mathbb{R}^d$
D
−0.0017245
0.0743869
0.0538474
P

Figure 1: "Parsing" with distributed structures in perspective

is the Distributed Tree Encoder (DT).

Our proposal is to build a Distributed Representation "Parser" (DRP) that directly maps sentences s into the final vectors. We choose the distributed trees $\overset{\leadsto}{t}$ as these reduced vectors fully represent the syntactic trees. A DRP acts as follows (see Figure 1): first, a function D encodes sentence s into a distributed vector $\overset{\leadsto}{s} \in R^d$; second, a function P transforms the input vector $\overset{\leadsto}{s}$ into a distributed tree $\overset{\leadsto}{t}$. This second step is a vector to vector transformation and, in a wide sense, "parses" the input sentence.

Given an input sentence s, a DRP is then a function defined as follows:

$$\overset{\leadsto}{t} = DRP(s) = P(D(s)) \tag{1}$$

In this paper, we design some functions D and we propose a linear function P, designed to be a regressor that can be induced from training data. In this study, we use a space with d dimensions for both sentences $\overset{\leadsto}{s}$ and *distributed trees* $\overset{\leadsto}{t}$, but, in general, these spaces can be of different size.

3.2 Syntactic Trees as Distributed Vectors

We here report on the *distributed trees*[2] (Zanzotto and Dell'Arciprete, 2012) to describe how these vectors represent syntactic trees and how the dot product between two distributed trees approximates the tree kernel defined by Collins and Duffy (2002).

fragments.

[2]For the experiments, we used the implementation of the distributed tree encoder available at http://code.google.com/p/distributed-tree-kernels/

Given a tree t, the corresponding distributed tree $\overset{\leadsto}{t}$ is defined as follows:

$$DT(t) = \sum_{\tau_i \in S(t)} \omega_i \overset{\leadsto}{\tau}_i \tag{2}$$

where $S(t)$ is the set of the subtrees τ_i of t, $\overset{\leadsto}{\tau}_i$ is the small vector corresponding to tree fragment τ_i and ω_i is the weight of subtree τ_i in the final feature space. As in (Collins and Duffy, 2002), the set $S(t)$ contains tree fragments τ such that the root of τ is any non-terminal node in t and, if τ contains node n, it must contain all the siblings of n in t (see, for example, $S_{lex}(t)$ in Figure 2). The weight ω_i is defined as:

$$\omega_i = \begin{cases} \sqrt{\lambda^{|\tau_i|-1}} & \text{if } |\tau_i| > 1 \text{ and } \lambda \neq 0 \\ 1 & \text{if } |\tau_i| = 1 \\ 0 & \text{if } \lambda = 0 \end{cases} \tag{3}$$

where $|\tau_i|$ is the number of non-terminal nodes of tree fragment τ_i and λ is the traditional parameter used to penalize large subtrees. For $\lambda = 0$, ω_i has a value 1 for productions and 0 otherwise. If different tree fragments are associated to nearly orthonormal vectors, the dot product $\overset{\leadsto}{t_1} \cdot \overset{\leadsto}{t_2}$ approximates the tree kernel (Zanzotto and Dell'Arciprete, 2012).

A key feature of the distributed tree fragments $\overset{\leadsto}{\tau}$ is that these vectors are built compositionally from a set $\mathcal{N}$ of *nearly orthonormal random vectors* $\overset{\leadsto}{n}$, associated to node labels n. Given a subtree τ, the related vector is obtained as:

$$\overset{\leadsto}{\tau} = \overset{\leadsto}{n}_1 \boxtimes \overset{\leadsto}{n}_2 \boxtimes \ldots \boxtimes \overset{\leadsto}{n}_k$$

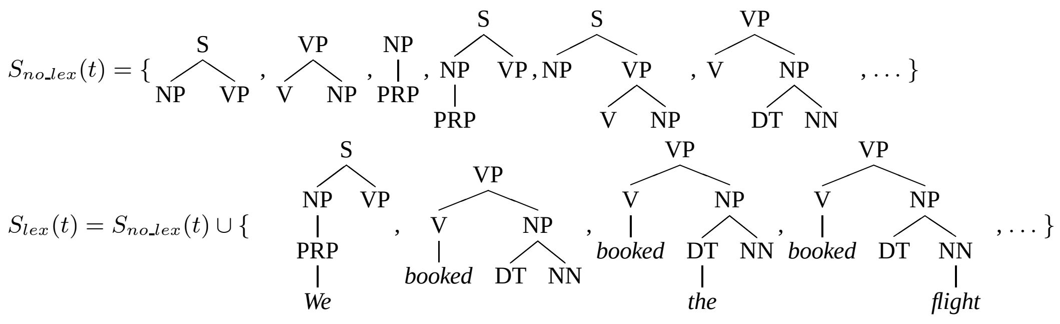


Figure 2: Subtrees of the tree t in Figure 1

where node vectors $\overset{\rightsquigarrow}{n}_i$ are ordered according to a depth-first visit of subtree τ and $\boxtimes$ is a vector composition operation, specifically the *shuffled circular convolution*[3]. This function guarantees that two different subtrees have nearly orthonormal vectors (see (Zanzotto and Dell'Arciprete, 2012) for more details). For example, the fifth tree τ_5 of set $S_{no\_lex}(t)$ in Figure 2 is:

$$\overset{\rightsquigarrow}{\tau}_5 = \overset{\rightsquigarrow}{S} \boxtimes (\overset{\rightsquigarrow}{NP} \boxtimes (\overset{\rightsquigarrow}{VP} \boxtimes (\overset{\rightsquigarrow}{V} \boxtimes \overset{\rightsquigarrow}{NP})))$$

We experiment with two tree fragment sets: the non-lexicalized set $S_{no\_lex}(t)$, where tree fragments do not contain words, and the lexicalized set $S_{lex}(t)$, including all the tree fragments. An example is given in Figure 2.

3.3 The Model

To build a DRP, we need to define the encoder D and the transformer P. In the following, we present a non-lexicalized and a lexicalized model for the encoder D and we describe how we can learn the transformer P by means of a linear regression model.

3.3.1 Sentence Encoders

Establishing good models to encode input sentences into vectors is the most difficult challenge. The models should consider the kind of information that can lead to a correct syntactic interpretation. Only in this way, the distributed representation parser can act as a vector transforming module. Unlike in models such as (Socher et al., 2011), we want our encoder to represent the whole sentence as a fixed size vector. We propose a non-lexicalized model and a lexicalized model.

[3]The *shuffled circular convolution* $\boxtimes$ is defined as $\vec{a} \boxtimes \vec{b} = s_1(\vec{a}) \otimes s_2(\vec{b})$ where $\otimes$ is the circular convolution and s_1 and s_2 are two different random permutations of vector elements.

Non-lexicalized model The non-lexicalized model relies only on the pos-tags of the sentences s: $s = p_1 \dots p_n$ where p_i is the pos-tag associated with the i-th token of the sentence. In the following we discuss how to encode this information in a $\mathbb{R}^d$ space. The basic model $D_1(s)$ is the one that considers the bag-of-postags, that is:

$$D_1(s) = \sum_i \overset{\rightsquigarrow}{p}_i \tag{4}$$

where $\overset{\rightsquigarrow}{p}_i \in \mathcal{N}$ is the vector for label p_i, taken from the set of nearly orthonomal random vectors $\mathcal{N}$. It is basically in line with the bag-of-word model used in random indexing (Sahlgren, 2005). Due to the commutative property of the sum and since vectors in $\mathcal{N}$ are nearly orthonormal: (1) two sentences with the same set of pos-tags have the same vector; and, (2) the dot product between two vectors, $D_1(s_1)$ and $D_1(s_2)$, representing sentences s_1 and s_2, approximately counts how many pos-tags the two sentences have in common. The vector for the sentence in Figure 1 is then:

$$D_1(s) = \overset{\rightsquigarrow}{PRP} + \overset{\rightsquigarrow}{V} + \overset{\rightsquigarrow}{DT} + \overset{\rightsquigarrow}{NN}$$

The general non-lexicalized model that takes into account all n-grams of pos-tags, up to length j, is then the following:

$$D_j(s) = D_{j-1}(s) + \sum_i \overset{\rightsquigarrow}{p}_i \boxtimes \dots \boxtimes \overset{\rightsquigarrow}{p}_{i+j-1}$$

where $\boxtimes$ is again the *shuffled circular convolution*. An n-gram $p_i \dots p_{i+j-1}$ of pos-tags is represented as $\overset{\rightsquigarrow}{p}_i \boxtimes \dots \boxtimes \overset{\rightsquigarrow}{p}_{i+j-1}$. Given the properties of the *shuffled circular convolution*, an n-gram of pos-tags is associated to a versor, as it composes j versors, and two different n-grams have nearly orthogonal vectors. For example, vector $D_3(s)$ for the sentence in Figure 1 is:

$$\begin{aligned} D_3(s) \;=\; & \overset{\rightsquigarrow}{PRP} + \overset{\rightsquigarrow}{V} + \overset{\rightsquigarrow}{DT} + \overset{\rightsquigarrow}{NN} + \\ & \overset{\rightsquigarrow}{PRP} \boxtimes \overset{\rightsquigarrow}{V} + \overset{\rightsquigarrow}{V} \boxtimes \overset{\rightsquigarrow}{DT} + \overset{\rightsquigarrow}{DT} \boxtimes \overset{\rightsquigarrow}{NN} + \\ & \overset{\rightsquigarrow}{PRP} \boxtimes \overset{\rightsquigarrow}{V} \boxtimes \overset{\rightsquigarrow}{DT} + \overset{\rightsquigarrow}{V} \boxtimes \overset{\rightsquigarrow}{DT} \boxtimes \overset{\rightsquigarrow}{NN} \end{aligned}$$

Lexicalized model Including lexical information is the hardest part of the overall model, as it makes vectors denser in information. Here we propose an initial model that is basically as the non-lexicalized model, but includes a vector representing the words in the unigrams. The equation representing sentences as unigrams is:

$$D_1^{lex}(s) = \sum_i \overset{\rightsquigarrow}{p}_i \boxtimes \overset{\rightsquigarrow}{w}_i$$

Vector $\overset{\rightsquigarrow}{w}_i$ represents word w_i and is taken from the set $\mathcal{N}$ of nearly orthonormal random vectors. This guarantees that $D_1^{lex}(s)$ is not lossy. Given a pair word-postag (w, p), it is possible to know if the sentence contains this pair, as $D_1^{lex}(s) \times \overset{\rightsquigarrow}{p} \boxtimes \overset{\rightsquigarrow}{w} \approx 1$ if (w, p) is in sentence s and $D_1^{lex}(s) \times \overset{\rightsquigarrow}{p} \boxtimes \overset{\rightsquigarrow}{w} \approx 0$ otherwise. Other vectors for representing words, e.g., distributional vectors or those obtained as look-up tables in deep learning architectures (Collobert and Weston, 2008), do not guarantee this possibility.

The general equation for the lexicalized version of the sentence encoder follows:

$$D_j^{lex}(s) = D_{j-1}^{lex}(s) + \sum_i \overset{\rightsquigarrow}{p}_i \boxtimes \ldots \boxtimes \overset{\rightsquigarrow}{p}_{i+j-1}$$

This model is only an initial proposal in order to take into account lexical information.

3.3.2 Learning Transformers with Linear Regression

The transformer P of the DRP (see Equation 1) can be seen as a linear regressor:

$$\overset{\rightsquigarrow}{t} = \mathbf{P}\overset{\rightsquigarrow}{s} \tag{5}$$

where $\mathbf{P}$ is a square matrix. This latter can be estimated having training sets $(\mathbf{T}, \mathbf{S})$ of *oracle vectors* and sentence input vectors $(\overset{\rightsquigarrow}{t}_i, \overset{\rightsquigarrow}{s}_i)$ for sentences s_i. Interpreting these sets as matrices, we need to solve a linear set of equations, i.e.: $\mathbf{T} = \mathbf{PS}$.

An approximate solution can be computed using Principal Component Analysis and Partial Least Square Regression[4]. This method relies on *Moore-Penrose pseudo-inversion* (Penrose, 1955). Pseudo-inverse matrices $\mathbf{S}^+$ are obtained using singular value decomposition (SVD). Matrices have the property $\mathbf{SS}^+ = \mathbf{I}$. Using the iterative method for computing SVD (Golub and Kahan, 1965), we can obtain different approximations $\mathbf{S}^+_{(k)}$ of $\mathbf{S}^+$ considering k singular values. Final approximations of DRPs are then: $\mathbf{P}_{(k)} = \mathbf{TS}^+_{(k)}$.

Matrices $\mathbf{P}$ are estimated by pseudo-inverting matrices representing input vectors for sentences $\mathbf{S}$. Given the different input representations for sentences, we can then estimate different DRPs: $DRP_1 = \mathbf{TS}_1^+$, $DRP_2 = \mathbf{TS}_2^+$, and so on. We need to estimate the best k in a separate parameter estimation set.

[4]An implementation of this method is available within the R statistical package (Mevik and Wehrens, 2007).

4 Experiments

We evaluated three issues for assessing DRP models: the performance of DRPs in reproducing oracle distributed trees (Sec. 4.2); the quality of the topology of the vector spaces of distributed trees induced by DRPs (Sec. 4.3); and the computation run time of DRPs (Sec. 4.4). Section 4.1 describes the experimental set-up.

4.1 Experimental Set-up

Data We derived the data sets from the Wall Street Journal (WSJ) portion of the English Penn Treebank data set (Marcus et al., 1993), using a standard data split for training (sections 2-21 PT_{train} with 39,832 trees) and for testing (section 23 PT_{23} with 2,416 trees). We used section 24 PT_{24} with 1,346 trees for parameter estimation.

We produced the final data sets of *distributed trees* with three different λ values: λ=0, λ=0.2, and λ=0.4. For each λ, we have two versions of the data sets: a non-lexicalized version (no_lex), where syntactic trees are considered without words, and a lexicalized version (lex), where words are considered. Oracle trees t are transformed into oracle distributed trees $\overset{\rightsquigarrow}{o}$ using the Distributed Tree Encoder DT (see Figure 1). We experimented with two sizes of the distributed trees space $\mathbb{R}^d$: 4096 and 8192.

We have designed the data sets to determine how DRPs behave with λ values relevant for syntax-sensitive NLP tasks. Both tree kernels and distributed tree kernels have the best performances in tasks such as question classification, semantic role labeling, or textual entailment recognition with λ values in the range 0–0.4.

System Comparison We compared the DRPs against the *existing way* of producing distributed trees (based on the recent paper described in (Zanzotto and Dell'Arciprete, 2012)): distributed trees are obtained using the output of a symbolic parser (SP) that is then transformed into a distributed tree using the DT with the appropriate λ. We refer to this chain as the Distributed Symbolic Parser (DSP). The DSP is then the chain $DSP(s) = DT(SP(s))$ (see Figure 1). As for the symbolic parser, we used Bikel's version (Bikel, 2004) of Collins' head-driven statistical parser (Collins, 2003). For a correct comparison, we used the Bikel's parser with oracle part-of-speech tags. We experimented with two versions: (1) a lexicalized method DSP_{lex}, i.e., the natural setting of the Collins/Bikel parser, and (2) a fully non-lexicalized version DSP_{no_lex} that exploits only part-of-speech tags. We obtained this last version by removing words in input sentences and leaving only part-of-speech tags. We trained these DSPs on PT_{train}.

Parameter estimation DRPs have two basic parameters: (1) parameter k of the pseudo-inverse, that is, the number of considered eigenvectors (see Section 3.3.2) and (2) the maximum length j of the n-grams considered by the encoder D_j (see Section 3.3.1). We performed the parameter estimation on the datasets derived from section PT_{24} by maximizing a pseudo f-measure. Section 4.2 reports both the definition of the measure and the results of the parameter estimation.

4.2 Parsing Performance

The first issue to explore is whether DRPs are actually good "distributed syntactic parsers". We compare DRPs against the distributed symbolic parsers by evaluating how well these "distributed syntactic parsers" reproduce oracle distributed trees.

Method A good DRP should produce distributed trees that are similar to oracle distributed trees. To capture this, we use the cosine similarity between the system and the oracle vectors:

$$cos(\overset{\leadsto}{t}, \overset{\leadsto}{o}) = \frac{\overset{\leadsto}{t} \cdot \overset{\leadsto}{o}}{||\overset{\leadsto}{t}|| ||\overset{\leadsto}{o}||}$$

where $\overset{\leadsto}{t}$ is the system's distributed tree and $\overset{\leadsto}{o}$ is the oracle distributed tree. We compute these the cosine similarity at the sentence-based (i.e., vector-based) granularity. Results report average values.

dim	*Model*	$\lambda = 0$	$\lambda = 0.2$	$\lambda = 0.4$
4096	DRP_1	0.6285	0.5697	0.542
	DRP_2	0.8011	0.7311	0.631
	DRP_3	0.8276‡	0.7552 ‡	0.6506‡
	DRP_4	0.8171	0.744	0.6419
	DRP_5	0.8045	0.7342	0.631
	DSP_{no_lex}	0.654	0.5884	0.4835
	DSP_{lex}	0.815	0.7813	0.7121
8192	DRP_3	0.8335‡	0.7605‡	0.6558‡
	DSP_{no_lex}	0.6584	0.5924	0.4873
	DSP_{lex}	0.8157	0.7815	0.7123

Table 1: Average *similarity* on PT_{23} of the DRPs (with different j) and the DSP on the *non-lexicalized data sets* with different λs and with the two dimensions of the distributed tree space (4096 and 8192). ‡ indicates significant difference wrt. DSP_{no_lex} ($p << .005$ computed with the Student's t test)

Model	$\lambda = 0$	$\lambda = 0.2$	$\lambda = 0.4$
DRP_3	0.7192	0.6406	0.0646
DSP_{lex}	0.9073	0.8564	0.6459

Table 2: Average *similarity* on PT_{23} of the DRP_3 and the DSP_{lex} on the *lexicalized data sets* with different λs on the distributed tree space with 4096 dimensions

Estimated parameters We estimated parameters k and j by training the different DRPs on the PT_{train} set and by maximizing the *similarity* of the DRPs on PT_{24}. The best pair of parameters is j=3 and k=3000. For completeness, we report also the best k values for the five different j we experimented with: $k = 47$ for j=1 (the linearly independent vectors representing pos-tags), $k = 1300$ for j=2, $k = 3000$ for j=3, $k = 4000$ for j=4, and $k = 4000$ for j=5. For comparison, some resulting tables report results for the different values of j.

Results Table 1 reports the results of the first set of experiments on the *non-lexicalized* data sets. The first block of rows (seven rows) reports the *average cosine similarity* of the different methods on the distributed tree spaces with 4096 dimensions. The second block (the last three rows) reports the performance on the space with 8192 dimensions. The *average cosine similarity* is computed on the PT_{23} set. Although we already selected j=3 as the best parameterization (i.e. DRP_3), the first

Output	*Model*	$\lambda = 0$	$\lambda = 0.2$	$\lambda = 0.4$
No lex	DRP_3	0.9490	0.9465	0.9408
	DSP_{no_lex}	0.9033	0.9001	0.8932
	DSP_{lex}	0.9627	0.9610	0.9566
Lex	DRP_3	0.9642	0.9599	0.0025
	DSP_{lex}	0.9845	0.9817	0.9451

Table 3: Average Spearman's Correlation: dim 4096 between the oracle's vector space and the systems' vector spaces (100 trials on lists of 1000 sentence pairs).

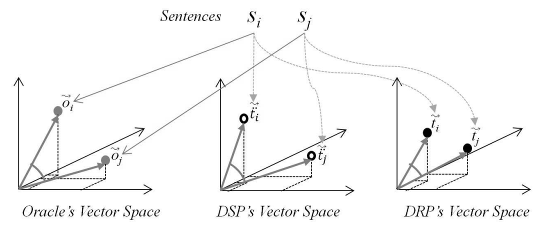

Sentences
s_i
s_j
$\overset{\rightsquigarrow}{o_i}$
$\overset{\rightsquigarrow}{o_j}$
$\overset{\rightsquigarrow}{\ddot{t}_i}$
$\overset{\rightsquigarrow}{\ddot{t}_j}$
$\overset{\rightsquigarrow}{t_i}$
$\overset{\rightsquigarrow}{t_j}$
Oracle's Vector Space
DSP's Vector Space
DRP's Vector Space

Figure 3: Topology of the resulting spaces derived with the three different methods: similarities between sentences

five rows of the first block report the results of the DRPs for five values of j. This gives an idea of how the different DRPs behave. The last two rows of this block report the results of the two DSPs.

We can observe some important facts. First, DRPs exploiting 2-grams, 3-grams, 4-grams, and 5-grams of part-of-speech tags behave significantly better than the 1-grams for all the values of λ. Distributed representation parsers need inputs that keep trace of sequences of pos-tags of sentences. But these sequences tend to confuse the model when too long. As expected, DRP_3 behaves better than all the other DRPs. Second, DRP_3 behaves significantly better than the comparable traditional parsing chain DSP_{no_lex} that uses only part-of-speech tags and no lexical information. This happens for all the values of λ. Third, DRP_3 behaves similarly to DSP_{lex} for λ=0. Both parsers use oracle pos tags to emit sentence interpretations but DSP_{lex} also exploits lexical information that DRP_3 does not access. For λ=0.2 and λ=0.4, the more informed DSP_{lex} behaves significantly better than DRP_3. But DRP_3 still behaves significantly better than the comparable DSP_{no_lex}. All these observations are valid also for the results obtained for 8192 dimensions.

Table 2 reports the results of the second set of experiments on the *lexicalized* data sets performed on a 4192-dimension space. The first row reports the *average cosine similarity* of DRP_3 trained on the lexicalized model and the second row reports the results of DSP_{lex}. In this case, DRP_3 is not behaving well with respect to DSP_{lex}. The additional problem DRP_3 has is that it has to reproduce input words in the output. This greatly complicates the work of the distributed representation parser. But, as we report in the next section, this preliminary result may be still satisfactory for λ=0 and λ=0.2.

4.3 Kernel-based Performance

This experiment investigates how DRPs preserve the topology of the oracle vector space. This correlation is an important quality factor of a distributed tree space. When using distributed tree vectors in learning classifiers, whether $\overset{\rightsquigarrow}{o_i} \cdot \overset{\rightsquigarrow}{o_j}$ in the oracle's vector space is similar to $\overset{\rightsquigarrow}{t_i} \cdot \overset{\rightsquigarrow}{t_j}$ in the DRP's vector space is more important than whether $\overset{\rightsquigarrow}{o_i}$ is similar to $\overset{\rightsquigarrow}{t_i}$ (see Figure 3). Sentences that are close using the oracle syntactic interpretations should also be close using DRP vectors. The topology of the vector space is more relevant than the actual quality of the vectors. The experiment on the parsing quality in the previous section does not properly investigate this property, as the performance of DRPs could be not sufficient to preserve distances among sentences.

Method We evaluate the coherence of the topology of two distributed tree spaces by measuring the Spearman's correlation between two lists of pairs of sentences (s_i, s_j), ranked according to the similarity between the two sentences. If the two lists of pairs are highly correlated, the topology of the two spaces is similar. The different methods and, thus, the different distributed tree spaces are compared against the oracle vector space (see Figure 3). Then, the first list always represents the oracle vector space and ranks pairs (s_i, s_j) according to $\overset{\rightsquigarrow}{o}_i \cdot \overset{\rightsquigarrow}{o}_j$. The second list instead represents the space obtained with a DSP or a DRP. Thus, it is respectively ranked with $\overset{\rightsquigarrow}{\ddot{t}_i} \cdot \overset{\rightsquigarrow}{\ddot{t}_j}$ or $\overset{\rightsquigarrow}{t_i} \cdot \overset{\rightsquigarrow}{t_j}$. In this way, we can comparatively evaluate the quality of the distributed tree vectors of our DRPs with respect to the other methods. We report average and standard deviation of the Spearman's correlation on 100 runs over lists of 1000 pairs. We used the testing set PT_{23} for extracting vectors.

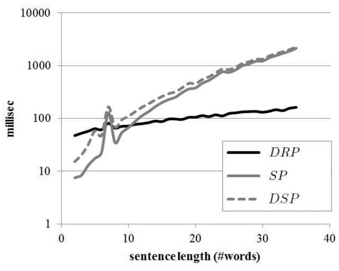


Figure 4: Running time with respect to the sentence length (dimension = 4092)

Results Table 3 reports results both on the non-lexicalized and on the lexicalized data set. For the non-lexicalized data set we report three methods (DRP_3, DSP_{no_lex}, and DSP_{lex}) and for the lexicalized dataset we report two methods (DRP_3 and DSP_{lex}). Columns represent different values of λ. Experiments are carried out on the 4096-dimension space. For the non-lexicalized data set, distributed representation parsers behave significantly better than DSP_{no_lex} for all the values of λ. The upper-bound of DSP_{lex} is not so far. For the harder lexicalized data set, the difference between DRP_3 and DSP_{lex} is smaller than the one based on the parsing performance. Thus, we have more evidence of the fact that we are in a good track. DRPs can substitute the *DSP* in generating vector spaces of distributed trees that adequately approximate the space defined by an oracle.

4.4 Running Time

In this last experiment, we compared the running time of the DRP with respect to the DSP. The analysis has been done on a dual-core processor and both systems are implemented in the same programming language, i.e. Java. Figure 4 plots the running time of the DRP, the SP, and the full $DSP = DT \circ SP$. The x-axis represents the sentence length in words and the y-axis represents the running time in milliseconds. The distance between SP and DSP shrinks as the plot is in a logarithmic scale. Figure 5 reports the average cosine similarity of DRP, DSP_{lex}, and DSP_{no_lex}, with respect to the sentence length, on the non-lexicalized data set with λ=0.4.

We observe that DRP becomes extremely convenient for sentences larger than 10 words (see Fig. 4) and the average cosine similarity difference between the different methods is nearly constant for the different sentence lengths (see Fig. 5). This test already makes DRPs very appealing methods for real time applications. But, if we consider that

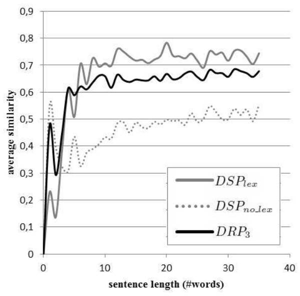


Figure 5: Average similarity with λ=0.4 with respect to the sentence length (dimension = 4092)

DRPs can run completely on Graphical Processing Units (GPUs), as dealing only with matrix products, fast-Fourier transforms, and random generators, we can better appreciate the potentials of the proposed methods.

5 Conclusions and Future Work

We presented Distributed Representation Parsers (DRP) as a novel path to use syntactic structures in feature spaces. We have shown that these ”parsers” can be learnt using training data and that DRPs are competitive with respect to traditional methods of using syntax in feature spaces.

This novel path to use syntactic structures in feature spaces opens interesting and unexplored possibilities. First, DRPs tackle the issue of computational efficiency of structural kernel methods (Rieck et al., 2010; Shin et al., 2011) from another perspective. DRPs could reduce structural kernel computations to extremely efficient dot products. Second, the tight integration of parsing and feature vector generation lowers the computational cost of producing distributed representations from trees, as circular convolution is not applied on-line.

Finally, DRPs can contribute to treat syntax in deep learning models in a uniform way. Deep learning models (Bengio, 2009) are completely based on distributed representations. But when applied to natural language processing tasks (e.g., (Collobert et al., 2011; Socher et al., 2011)), syntactic structures are not represented in the neural networks in a distributed way. Syntactic information is generally used by exploiting symbolic parse trees, and this information positively impacts performances on final applications, e.g., in paraphrase detection (Socher et al., 2011) and in semantic role labeling (Collobert et al., 2011). Building on the results presented here, an interesting line of research is then the integration of distributed representation parsers and deep learning models.

References

James A. Anderson. 1973. A theory for the recognition of items from short memorized lists. *Psychological Review*, 80(6):417 – 438.

Yoshua Bengio. 2009. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Daniel M. Bikel. 2004. Intricacies of collins' parsing model. *Comput. Linguist.*, 30:479–511, December.

Christian Bockermann, Martin Apel, and Michael Meier. 2009. Learning sql for database intrusion detection using context-sensitive modelling. In *Detection of Intrusions andMalware & Vulnerability Assessment (DIMVA)*, pages 196–205.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. of the 1st NAACL*, pages 132–139, Seattle, Washington.

Naom Chomsky. 1957. *Aspect of Syntax Theory*. MIT Press, Cambridge, Massachussetts.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL02*.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637.

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*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November.

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Patrick Düssel, Christian Gehl, Pavel Laskov, and Konrad Rieck. 2008. Incorporation of application layer protocol syntax into anomaly detection. In *Proceedings of the 4th International Conference on Information Systems Security*, ICISS '08, pages 188–202, Berlin, Heidelberg. Springer-Verlag.

Thomas Gärtner. 2003. A survey of kernels for structured data. *SIGKDD Explorations*.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics*, 28(3):245–288.

Gene Golub and William Kahan. 1965. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224.

Kosuke Hashimoto, Ichigaku Takigawa, Motoki Shiga, Minoru Kanehisa, and Hiroshi Mamitsuka. 2008. Mining significant tree patterns in carbohydrate sugar chains. *Bioinformatics*, 24:i167–i173, August.

G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. 1986. Distributed representations. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA.

Bill MacCartney, Trond Grenager, Marie-Catherine de Marneffe, Daniel Cer, and Christopher D. Manning. 2006. Learning to recognize features of valid textual entailments. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 41–48, New York City, USA, June. Association for Computational Linguistics.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330.

Bjrn-Helge Mevik and Ron Wehrens. 2007. The pls package: Principal component and partial least squares regression in r. *Journal of Statistical Software*, 18(2):1–24, 1.

Alessandro Moschitti, Daniele Pighin, and Roberto Basili. 2008. Tree kernels for semantic role labeling. *Computational Linguistics*, 34(2):193–224.

Alessandro Moschitti. 2006. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of The 17th European Conference on Machine Learning*, Berlin, Germany.

Bennet B. Murdock. 1983. A distributed memory model for serial-order information. *Psychological Review*, 90(4):316 – 338.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Glsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Roger Penrose. 1955. A generalized inverse for matrices. In *Proc. Cambridge Philosophical Society*.

T. A. Plate. 1994. *Distributed Representations and Nested Compositional Structure*. Ph.D. thesis.

Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James H. Martin, and Daniel Jurafsky. 2005. Semantic role labeling using different syntactic views. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 581–588. Association for Computational Linguistics, Morristown, NJ, USA.

J. Quinlan. 1993. *C4:5:programs for Machine Learning*. Morgan Kaufmann, San Mateo.

Konrad Rieck and Pavel Laskov. 2007. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2:243–256. 10.1007/s11416-006-0030-0.

Konrad Rieck, Tammo Krueger, Ulf Brefeld, and Klaus-Robert Müller. 2010. Approximate tree kernels. *J. Mach. Learn. Res.*, 11:555–580, March.

David E. Rumelhart and James L. Mcclelland. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition : Foundations (Parallel Distributed Processing)*. MIT Press, August.

Magnus Sahlgren. 2005. An introduction to random indexing. In *Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering TKE*, Copenhagen, Denmark.

Kilho Shin, Marco Cuturi, and Tetsuji Kuboyama. 2011. Mapping kernels for trees. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 961–968, New York, NY, USA, June. ACM.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems 24*.

Jean-Philippe Vert. 2002. A tree kernel to analyse phylogenetic profiles. *Bioinformatics*, 18(suppl 1):S276–S284, July.

S. V. N. Vishwanathan and Alexander J. Smola. 2002. Fast kernels for string and tree matching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 569–576. MIT Press.

Rui Wang and Günter Neumann. 2007. Recognizing textual entailment using sentence similarity based on dependency tree skeletons. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 36–41, Prague, June. Association for Computational Linguistics.

F.M. Zanzotto and L. Dell'Arciprete. 2012. Distributed tree kernels. In *Proceedings of International Conference on Machine Learning*, pages 193–200.

Fabio Massimo Zanzotto, Marco Pennacchiotti, and Alessandro Moschitti. 2009. A machine learning approach to textual entailment recognition. *NATURAL LANGUAGE ENGINEERING*, 15-04:551–582.

Dell Zhang and Wee Sun Lee. 2003. Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 26–32, New York, NY, USA. ACM.

Min Zhang and Haizhou Li. 2009. Tree kernel-based SVM with structured syntactic knowledge for BTG-based phrase reordering. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 698–707, Singapore, August. Association for Computational Linguistics.

GuoDong Zhou, Min Zhang, DongHong Ji, and QiaoMing Zhu. 2007. Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 728–736.