Exploring Compositional Architectures and Word Vector Representations for Prepositional Phrase Attachment

Yonatan Belinkov, Tao Lei, Regina Barzilay
Massachusetts Institute of Technology
{belinkov, taolei, regina}@csail.mit.edu

Amir Globerson
The Hebrew University
gamir@cs.huji.ac.il

Abstract

Prepositional phrase (PP) attachment disambiguation is a known challenge in syntactic parsing. The lexical sparsity associated with PP attachments motivates research in word representations that can capture pertinent syntactic and semantic features of the word. One promising solution is to use word vectors induced from large amounts of raw text. However, state-of-the-art systems that employ such representations yield modest gains in PP attachment accuracy.

In this paper, we show that word vector representations can yield significant PP attachment performance gains. This is achieved via a non-linear architecture that is discriminatively trained to maximize PP attachment accuracy. The architecture is initialized with word vectors trained from unlabeled data, and relearns those to maximize attachment accuracy. We obtain additional performance gains with alternative representations such as dependency-based word vectors. When tested on both English and Arabic datasets, our method outperforms both a strong SVM classifier and state-of-the-art parsers. For instance, we achieve 82.6% PP attachment accuracy on Arabic, while the Turbo and Charniak self-trained parsers obtain 76.7% and 80.8% respectively.[1]

1 Introduction

The problem of prepositional phrase (PP) attachment disambiguation has been under investigation for a long time. However, despite at least two decades of research (Brill and Resnik, 1994; Ratnaparkhi et al., 1994; Collins and Brooks, 1995), it remains a major source of errors for state-of-the-art parsers. For instance, in a comparative evaluation of parser performance on the Wall Street Journal corpus, Kummerfeld et al. (2012) report that PP attachment is the largest source of errors across all parsers. Moreover, the extent of improvement over time has been rather limited, amounting to about 32% error reduction since the work of (Collins, 1997).

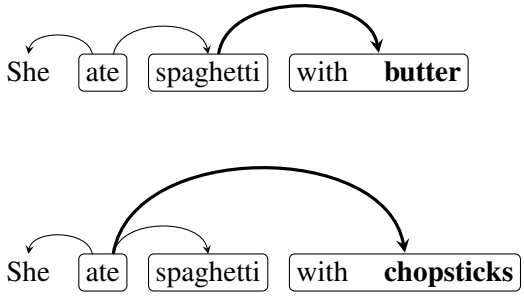


Figure 1: Two sentences illustrating the importance of lexicalization in PP attachment decisions. In the top sentence, the PP *with butter* attaches to the noun *spaghetti*. In the bottom sentence, the PP *with chopsticks* attaches to the verb *ate*.

PP attachments are inherently lexicalized and part-of-speech (POS) tags are not sufficient for their correct disambiguation. For example, the two sentences in Figure 1 vary by a single noun — *butter* vs *chopsticks*. However, this word determines the structure of the whole PP attachment. If the corre-

[1]The code and data for this work are available at http://groups.csail.mit.edu/rbg/code/pp.

Transactions of the Association for Computational Linguistics, vol. 2, pp. 561–572, 2014. Action Editor: Ryan McDonald.
Submission batch: 10/2014; Revision batch 11/2014; Published 12/2014.

sponding word is not observed in the training data, a standard lexicalized parser does not have sufficient information to distinguish between these two cases. In fact, 72% of head-child pairs (e.g. *spaghetti-butter*) from the Wall Street Journal test set are unseen in training. Not surprisingly, resolving these ambiguities is challenging for parsers that have restricted access to word semantics.

These considerations have motivated recent explorations in using distributed word representations for syntactic parsing (Cirik and Şensoy, 2013; Socher et al., 2013; Lei et al., 2014). Low-dimensional word embeddings help unveil semantic similarity between words, thereby alleviating the data sparsity problem associated with PP attachment. In this context, large amounts of raw data used to construct embeddings effectively enrich limited syntactic annotations. While these approaches show initial promise, they still lag behind self-trained parsers (McClosky et al., 2006). These parsers also utilize raw data but in a different way: self-trained parsers use it to get additional (noisy) annotations, without computing new word representations. These results suggest that embedding-based representations have not yet been utilized to their full potential.

We show that embedding-based representations can indeed significantly improve PP attachment accuracy. We achieve this by using such representations within a compositional neural network architecture. The representations are initially learned from an unlabeled corpus, but are then further discriminatively trained to maximize PP attachment accuracy. We also explore alternative representations such as dependency-based word vectors that are trained from parsed texts using the syntactic context in a dependency tree.

We test our approach for PP attachment disambiguation on English and Arabic datasets, comparing it to full-scale parsers and a support vector machine (SVM) ranker. Our model outperforms all baselines, including a self-trained parser. The difference is particularly apparent on Arabic. For instance, our model achieves PP attachment accuracy of 82.6% while the Turbo (Martins et al., 2013), RBG (Lei et al., 2014), and Charniak self-trained (McClosky et al., 2006) parsers obtain 76.7%, 80.3%, and 80.8% respectively. Our results demonstrate that relearning the embeddings contributes to the model performance, across a range of configurations. We also notice that representations based on syntactic context are more powerful than those based on linear context. This may explain the improved performance of self-trained parsers over parsers that rely on linear context embeddings.

2 Related Work

Problem formulation Typically, PP attachment disambiguation is modeled as a binary classification decision between a preceding noun or verb (Brill and Resnik, 1994; Ratnaparkhi et al., 1994; Collins and Brooks, 1995; Olteanu and Moldovan, 2005; Šuster, 2012). In addition, the problem of PP attachment has also been addressed in the context of full parsing (Atterer and Schütze, 2007; Agirre et al., 2008). For instance, Green (2009) engineered state-split features for the Stanford parser to improve Arabic PP attachment.

In this work, we do isolate PP attachments from other parsing decisions. At the same time, we consider a more realistic scenario where multiple candidate heads are allowed. We also compare against full-scale parsers and show that our model predictions improve a state-of-the-art dependency parser.

Information sources Lexical sparsity associated with disambiguating PP attachments (Figure 1) has spurred researchers to exploit a wide range of information sources. On the one hand, researchers have explored using manually crafted resources (Stetina and Nagao, 1997; Gamallo et al., 2003; Olteanu and Moldovan, 2005; Medimi and Bhattacharyya, 2007). For instance, Agirre et al. (2008) demonstrate that using WordNet semantic classes benefits PP attachment performance. On the other hand, researchers have looked into using co-occurrence statistics from raw text (Volk, 2002; Olteanu and Moldovan, 2005; Gala and Lafourcade, 2007). Such statistics can be translated into word vectors from which a cosine similarity score is calculated (Šuster, 2012). We also rely on word vectors, but our model captures more complex relations among them.

Algorithmic approach Our work is most similar to recursive neural network parsers (Costa et al., 2003; Menchetti et al., 2005; Socher et al., 2010). In

particular, Socher et al. (2013) obtain good parsing performance by building compositional representations from word vectors. However, to combat the computational complexity of the full parsing scenario, they rely on a probabilistic context-free grammar to prune search space. In contrast, focusing on PP attachment allows us to consider various neural network architectures that are more appropriate for this task, including ternary, binary, and distance-dependent compositions. Furthermore, we investigate modifications to the original word vectors in several important directions: enriching word vectors with semantic and syntactic knowledge resources, relearning them by backpropagating errors from supervised data, and using dependency-based vectors. We show that such modifications lead to better word vectors and significant performance gains.

3 Model

We begin by introducing some notation. All vectors $\mathbf{v} \in \mathbb{R}^n$ are assumed to be column vectors. We denote a given sentence by x and the set of prepositions in x by $PR(x)$. In other words, $PR(x)$ is the set of words whose POS tags are *prep*. The PP attachment label of the preposition $z \in PR(x)$ is denoted by $y(z) \in x$. Namely, $y(z) = h$ indicates that the head of the preposition z is h.

Our classification approach is to construct a scoring function $s(x, z, h; \theta)$ for a preposition $z \in PR(x)$ and its candidate head h in the sentence x. We then choose the head by maximizing $s(x, z, h; \theta)$ over h. The set of possible candidates $\{h\}$ can be of arbitrary size, thus departing from the binary classification scenario considered in much of the previous work (Section 2). The set of parameters is θ.

3.1 Compositional framework

Our approach to constructing the score function is as follows. First, we assume that all words in the sentence are represented as vectors in $\mathbb{R}^n$. Next, we compose vectors corresponding to the relevant preposition, its candidate head, and other words in the sentence to obtain a new vector $\mathbf{p} \in \mathbb{R}^n$. The final score is a linear function of this vector.

The basic composition operation is defined as a single layer in a neural network (Socher et al., 2010). Given vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, representing two words, we form a new vector via a function:

$$g(\mathbf{W}[\mathbf{u}; \mathbf{v}] + \mathbf{b}) \in \mathbb{R}^n \quad (1)$$

where $\mathbf{b} \in \mathbb{R}^n$ is a vector of bias terms, $[\mathbf{u}; \mathbf{v}] \in \mathbb{R}^{2n}$ is a concatenation of $\mathbf{u}$ and $\mathbf{v}$ into a column vector, $\mathbf{W} \in \mathbb{R}^{n \times 2n}$ is a composition matrix, and g is a non-linear activation function.[2]

Given a candidate head h for preposition z, we apply such compositions to a set of words, resulting in a vector $\mathbf{p}$. The final score $s(x, z, h; \theta)$ is given by $\mathbf{w} \cdot \mathbf{p}$, where $\mathbf{w} \in \mathbb{R}^n$ is a weight vector. The parameters to be learned are $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{w})$.

3.2 Composition architectures

There are various possible ways to compose and obtain the vector $\mathbf{p}$. Table 1 shows three basic composition architectures that are used in our model. In all cases, elements like the head of the PP, the preposition, and the first child of the preposition are composed using Eq. 1 to derive a parent vector that is then scored by the score vector $\mathbf{w}$. The architectures differ in the number of compositions and their type. For instance, the Head-Child model uses only the head and child in a single composition, ignoring the preposition. The Head-Prep-Child-Ternary composes all three elements simultenuously, reflecting ternary interactions. The Head-Prep-Child model, on the other hand, first composes the preposition and child to form a parent $\mathbf{p}_1$ representing the PP, then composes $\mathbf{p}_1$ with the head into another parent $\mathbf{p}_2$ ($= \mathbf{p}$) that is scored by $\mathbf{w}$. This two-step process facilitates capturing different syntactic relations with different composition matrices. We turn to this next.

Granularity The basic composition architectures (Table 1) assume a global matrix $\mathbf{W}$ for all composition operations. In the case of the Head-Prep-Child model, we also consider a local variant with different matrices for the two compositions: $\mathbf{W}^{bottom}$ for composing the preposition $\mathbf{z}$ with its child $\mathbf{c}$ into a parent $\mathbf{p}_1$ representing the PP, and $\mathbf{W}^{top}$ for composing the head $\mathbf{h}$ with $\mathbf{p}_1$ into a parent $\mathbf{p}_2$. The composition equations are then:

$$\mathbf{p}_1 = g(\mathbf{W}^{bottom}[\mathbf{z}; \mathbf{c}] + \mathbf{b}^{bottom})$$
$$\mathbf{p}_2 = g(\mathbf{W}^{top}[\mathbf{h}; \mathbf{p}_1] + \mathbf{b}^{top})$$

[2] We use $\tanh$ which performed slightly better than $\mathrm{sigmoid}$ in preliminary experiments.

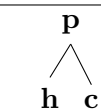
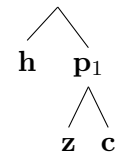
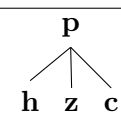
Model	Equations	Structure
Head-Child (HC)	$\mathbf{p} = g(\mathbf{W}[\mathbf{h};\mathbf{c}] + \mathbf{b})$	
Head-Prep-Child (HPC)	$\mathbf{p}_1 = g(\mathbf{W}[\mathbf{z};\mathbf{c}] + \mathbf{b})$ $\mathbf{p}_2 = g(\mathbf{W}[\mathbf{h};\mathbf{p}_1] + \mathbf{b})$	
Head-Prep-Child-Ternary (HPCT)	$\mathbf{p} = g(\mathbf{W}^{Tern}[\mathbf{h};\mathbf{z};\mathbf{c}] + \mathbf{b})$	

Table 1: Basic composition architectures. $\mathbf{h}, \mathbf{z}, \mathbf{c} \in \mathbb{R}^n$ are vectors for the head, the preposition, and its child respectively; $\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^n$ are parent vectors created during composition operations; $\mathbf{W} \in \mathbb{R}^{n \times 2n}, \mathbf{W}^{Tern} \in \mathbb{R}^{n \times 3n}$ are binary and ternary composition matrices respectively; $\mathbf{b} \in \mathbb{R}^n$ is a bias term; and g is a non-linear function.

In this case, the set of parameters is $\theta = (\mathbf{W}^{top}; \mathbf{b}^{top}; \mathbf{W}^{bottom}; \mathbf{b}^{bottom}; \mathbf{w})$. We call this variant the Head-Prep-Child-Local (HPCL) model.

The composition architectures described thus far only considered the composed words but not their relative position in the sentence. Such position information may be useful, since candidates closer to the preposition are typically more likely to attach. To model this difference, we introduce distance-dependent parameters and modify the Head-Prep-Child model (Table 1, middle row) as follows: for a head $\mathbf{h}$ at distance d from the preposition, we let:

$$\mathbf{p}_2 = g(\mathbf{W}^d[\mathbf{h};\mathbf{p}_1] + \mathbf{b}^d)$$

where $\mathbf{W}^d \in \mathbb{R}^{n \times 2n}$ and $\mathbf{b}^d \in \mathbb{R}^n$ are the matrix and bias for composing with heads at distance d from the preposition. $\mathbf{p}_1$ is defined as in Table 1. The set of parameters is then $\theta = (\{\mathbf{W}^d; \mathbf{b}^d\}_d; \mathbf{W}; \mathbf{b}; \mathbf{w})$. To reduce the number of parameters we use only $d = 1, \ldots, 5$, and clip distances greater than 5. We name this model Head-Prep-Child-Dist (HPCD).

Context It may also be useful to exploit words surrounding the candidate head such as the following word. This can be integrated in the composition architectures in the following way: for each candidate head, represented by a vector $\mathbf{h} \in \mathbb{R}^n$, concatenate a vector representing the word following the candidate. If such a vector is not available, append a zero vector. This results in a new vector $\mathbf{h}' \in \mathbb{R}^{2n}$ representing the head. To compose it with a vector $\mathbf{p}_1 \in \mathbb{R}^n$ representing the PP, we use a composition matrix of size $n \times 3n$, similar to the ternary composition described above. We refer to this model as Head-Prep-Child-Next (HPCN).

3.3 Training

For training, we adopt a max-margin framework. Given a training corpus of pairs of sentences and attachments, $\{x^{(i)}, y^{(i)}\}$, we seek to minimize the following objective function:

$$J(\theta) = \sum_{i=1}^{T} \sum_{z \in PR(x^{(i)})} \max_h \Big[s(x^{(i)}, z, h; \theta) - s(x^{(i)}, z, y^{(i)}(z); \theta) + \Delta(h, y^{(i)}(z)) \Big] \quad (2)$$

where Δ is the zero-one loss.

For optimization we use minibatch AdaGrad (Duchi et al., 2011). Note that the objective is non-differentiable so AdaGrad is used with the subgradient of $J(\theta)$, calculated with backpropagation.

For regularization we use Dropout (Hinton et al., 2012), a recent method for preventing co-adaptation of features, where input units to the neural network

are randomly dropped. Random dropping occurs independently for each training example and has the effect of creating multiple thinned networks that are trained with shared parameters. In our implementation we dropout input units before each non-linear layer, including the initial word vectors. We do not dropout units after the final non-linear layer. Note that Dropout is known to be especially useful when combined with AdaGrad (Wager et al., 2013).

Hyperparameters and initialization We use the following default hyperparameters without further tuning unless noted otherwise: Dropout parameter $\rho = 0.5$ (Hinton et al., 2012), AdaGrad initial learning rate $\eta = 1.0$ (Dyer, n.d.), and minibatch size of 500. Learned parameters are initialized similarly to previous work (Bengio and Glorot, 2010; Socher et al., 2013): composition matrices are set to $\mathbf{W} = 0.5[\mathbf{I}\ \mathbf{I}] + \epsilon$, where $\epsilon \sim U(-\frac{1}{n}, \frac{1}{n})$; bias terms $\mathbf{b}$ are set to zero; and the weight vector is set to $\mathbf{w} \sim U(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$.

4 Word vector representations

Our approach assumes a vector representation for each word. Such representations have gained popularity in recent years, due to the ability to train them from large unlabeled datasets, and their ease of use in a wide variety of tasks (Turian et al., 2010).

There are various approaches to training vector representations (Collobert and Weston, 2008; Bengio et al., 2009). Here we chose to focus on the Skip-gram method recently proposed by Mikolov et al. (2013a). The Skip-gram model maximizes the average log-probability of every word generating its context, which is modeled via a neural net architecture, but without the non-linearity. To improve efficiency, this probability is approximated by a hierarchical softmax (Mikolov et al., 2013b) with vocabulary words represented in a binary Huffman tree.[3]

In the simplest variant of our method, we train the Skip-gram representation on unlabeled text, and use it as a fixed representation when training the PP attachment model (see Section 3.3). Below we consider several variations on this approach.

[3]Preliminary experiments with other model variations (e.g. negative sampling) have not led to notable performance gains.

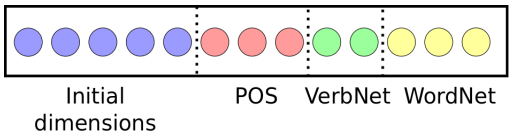


Figure 2: Illustration of an enriched word vector. Initial dimensions learned from raw texts are enriched with binary vectors indicating part-of-speech tags, VerbNet frames, and WordNet hypernyms.

4.1 Relearning word vectors

The Skip-gram word vectors are originally learned from raw text, with the objective of maximizing the likelihood of co-occurring words. Here our goal is to maximize PP attachment accuracy, and it is possible that a different representation is optimal for this task. We may thus take a discriminative approach and update the vectors to maximize PP attachment accuracy. Technically this just requires taking the subgradient of our objective (Eq. 2) with respect to the word vectors, and updating them accordingly.

Adding the word vectors as parameters significantly increases the number of free parameters in the model, and may lead to overfitting. To reduce this effect, we use Dropout regularization (Section 3.3). We also employ a smaller initial learning rate for the word vectors compared to other model parameters.[4]

Finally, note that since the objective is non-convex, the vectors obtained after this procedure will typically depend on the initial value used. The relearning procedure may thus be viewed as fine-tuning the word vectors to improve PP attachment accuracy.

4.2 Enriching word vectors

The word vectors we use are trained from raw text. However, it is easy to enrich them using structured knowledge resources such as VerbNet or WordNet, as well as morpho-syntactic information available in treebanks.

Our approach to enriching word vectors is to extend them with binary vectors. For example, given a vector $\mathbf{h}$ for the candidate head, we add binary-valued dimensions for its part-of-speech and that of the following word. Next we add a binary dimension for VerbNet indicating whether the candidate head

[4]We tuned $\eta = [0.001, 0.01, 0.1, 1]$ on the English dev set and chose the best value ($\eta = 0.001$) for all other experiments.

appears with the preposition in a verb frame. Finally, for each top hypernym in WordNet, we add a binary dimension indicating whether it is a hypernym of the candidate head, aiming for semantic clustering information. Note that we do not perform sense disambiguation so this information may be noisy.

Figure 2 illustrates the resulting enriched vector. Similar dimensions are appended to vectors representing other words participating in the compositions. Our experiments show that such an extension significantly improves performance.

4.3 Syntactic word vectors

In the standard Skip-gram model word vectors are trained from raw text using the linear context of neighboring words. We also consider an alternative method for creating word vectors by using the syntactic context of words. Such syntactic context is expected to be relevant for resolving PP attachments. Given a dependency-parsed text, we follow Bansal et al. (2014) and create a new corpus of tuples (l, g, p, c, l), for every word c, its parent p with dependency label l, and its grandparent g. Then we train an ordinary Skip-gram model on this corpus, but with a small window size of 2. Note that the label l appears on both ends so it contributes to the context of the word as well as its grandparent. We find that syntactic vectors yield significant performance gains compared to standard vectors.[5]

5 Experimental setup

5.1 Extracting PP attachments

Instances of PP attachment decisions are extracted from standard treebanks. We use the CATiB dependency treebank (Habash and Roth, 2009) for Arabic and a conversion of the Penn treebank (PTB) to dependency format for English.[6] Standard train/dev/test splits are used: sections 2-21/22/23 of the PTB for English, and the split from the SPRML shared-task for Arabic (Seddah et al., 2013). As Table 2 shows, the datasets of the two languages are fairly similar in size, except for the much larger set of prepositions in the English data.

Extracting instances of PP attachments from the treebanks is done in the following way. For each preposition, we look for all possible candidate heads in a fixed preceding window. Typically, these will be nouns or verbs. Only prepositions with a noun child are considered, leaving out some rare exceptions. Empirically, limiting candidate heads to appear close enough before the preposition is not an unrealistic assumption: we choose a 10-word window and find that it covers about 94/99% of Arabic/English PP attachments. Unambiguous attachments with a single possible candidate are discarded.

	Arabic		**English**	
	Train	Test	Train	Test
Total	42,387	3,917	35,359	1,951
Candidates	4.5	4.3	3.7	3.6
Vocab sizes				
All	8,230	2,944	11,429	2,440
Heads	8,225	2,936	10,395	2,133
Preps	13	10	72	46
Children	4,222	1,424	5,504	983

Table 2: Statistics of extracted PP attachments, showing total sizes, average number of candidate heads, and vocabulary sizes.

	Arabic	**English**	
Corpus	arTenTen	Wikipedia	BLLIP
Tokens	130M	120M	43M
Types	43K	218K	317K

Table 3: Statistics of Arabic and English corpora used for creating word vectors.

5.2 Creating word vectors

The initial word vectors are created from raw texts using the Skip-gram model with hierarchical softmax, as described in Section 4.[7] We use a portion of Wikipedia for English[8] and the arTenTen corpus for Arabic, containing web texts crawled in 2012 (Belinkov et al., 2013; Arts et al., 2014). Table 3

[5] We also experimented with another method for creating syntactic vectors by Levy and Goldberg (2014) and observed similar performance gains.

[6] We used the Pennconverter tool: `http://nlp.cs.lth.se/software/treebank-converter`.

[7] We used the word2vec tool: `https://code.google.com/p/word2vec`, with default settings. We experimented with word vectors of 25, 50, 100, and 200 dimensions, and found 100 to work best in most cases.

[8] `http://mattmahoney.net/dc/textdata`.

shows the comparable sizes of the datasets. The Arabic corpus has been tokenized and lemmatized with MADA (Habash and Rambow, 2005; Habash et al., 2005), a necessary procedure in order to separate some prepositions from their child words. In addition, lemmatization reduces vocabulary size and facilitates sharing information between different morphological variants that have the same meaning.

For syntactic word vectors, we use the English vectors in (Bansal et al., 2014), which were trained from a parsed BLLIP corpus (minus PTB). For Arabic, we first convert the morphologically-processed arTenTen corpus to CoNLL format with the SPMRL shared-task scripts (Seddah et al., 2013). Then we parse the corpus with a baseline MST parser (Section 5.3) and create syntactic word vectors as described in Section 4.3. The Arabic syntactic vectors will be made available to the research community.

For enriching word vectors, we use part-of-speech information[9] from the treebanks as well as the Arabic and English VerbNets (Kipper et al., 2008; Mousser, 2010) and WordNets (Rodríquez et al., 2008; Princeton University, 2010). In total, these resources add to each word vector 46/67 extended dimensions in Arabic/English, representing syntactic and semantic information about the word.

5.3 Baselines

We compare against full-scale parsers, an SVM ranker, and a simple but strong baseline of always choosing the closest candidate head.

Parsers We mostly compare with dependency parsers, including the state-of-the-art Turbo (Martins et al., 2010; Martins et al., 2013) and RBG parsers (Lei et al., 2014), in addition to a second-order MST parser (McDonald et al., 2005) and the Malt parser (Nivre et al., 2006). We also compare with two constituency parsers: an RNN parser (Socher et al., 2013), which also uses word vectors and a neural network approach, and the Charniak self-trained reranking parser (McClosky et al., 2006). We train all parsers on the train/dev sets and report their PP attachment accuracy on the test sets.[10] For the self-trained parser we followed the procedure in (McClosky et al., 2006) with the same unsupervised datasets that are used in our PP model.

Source	Feature Template
Treebank	`hw, pw, cw, hw-cw, ht, nt, hpd`
WordNet	`hh, ch`
VerbNet	`hpvf`
Word Vectors	`sim(hv,cv)`
Brown Clusters	`hc*, pc*, cc*, hc4, pc4, cc4, hc*-cc*, hc4-cc4`

Table 4: Feature templates for the SVM baseline. Bi-lexical templates appear with a "-". Abbreviations: `hw`/`pw`/`cw` = head/prep/child word, `ht` = head tag, `nt` = next word tag, `hpd` = head-prep distance; `hh`/`ch` = head/child hypernym; `hpvf` = head-prep found in verb frame; `hv`/`cv` = head/child vector; `hc*`/`pc*`/`cc*` = head/prep/child full bit string, `hc4`/`pc4`/`cc4` = head/prep/child 4-bit prefix.

SVM We consider a learning-to-rank formulation for our problem, where each example provides a correct candidate head and several incorrect candidates. We order these in a simple list where the correct candidate has the highest rank and all other candidates have a single lower rank. We then rank these with an SVM ranker[11] and select the top candidate. This formulation is necessary because we depart from the binary classification scenario that was used in previous work (Section 2).

The SVM ranker uses the following features: the candidate head, preposition, and child; bi-lexical conjunctions of head-child; part-of-speech tags of the head and the following word; and the candidate head's distance from the preposition. We also add top WordNet hypernyms for head and child, and an indicator of whether the preposition appears in the head's sub-categorization frame in VerbNet. This configuration parallels the information used in our model but fails to exploit raw data. Therefore, we consider two more types of features. First, we use word vectors by computing cosine similarity between vectors of the candidate head and the child

[9] We use gold POS tags in all systems and experiments.

[10] The only exception is the RNN parser, for which we use the built-in English model in Stanford parser's (version 3.4); for Arabic we do train a new RNN model.

[11] We use SVMRank: `http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`.

of the preposition. This feature was found useful in previous work on PP attachment (Šuster, 2012). While this limits the contribution of the word vectors to the learned model to one dimension, attempts to use more dimensions in the SVM were unsuccessful.[12] In contrast, the compositional models better capture the full dimensionality of the word vectors.

A second type of features induced from raw data that we consider are Brown clusters, which were found to be useful in dependency parsing (Koo et al., 2008). Compared to distributed vectors, Brown clusters provide a more discrete representation that is easier to incorporate in the SVM. We create clusters from our unsupervised corpora using the Liang (2005) implementation of Brown's algorithm, and add features in the spirit of (Koo et al., 2008). Specifically, we add full and prefixed bit strings for the head, preposition, and child, as well as bi-lexical versions for head-child pairs.[13] Table 4 shows a summary of the SVM features.

6 Results

Table 5 summarizes the results of our model and other systems. Our best results are obtained with the Head-Prep-Child-Dist (HPCD) model using syntactic vectors, enriching, and relearning. The full model outperforms both full-scale parsers and a dedicated SVM model. More advanced parsers do demonstrate higher accuracy on the PP attachment task, but our method outperforms them as well. Note that the self-trained reranking parser (Charniak-RS) performs especially well and quite better than the RNN parser. This trend is consistent with the results in (Kummerfeld et al., 2012; Socher et al., 2013).

Our compositional architecture is effective in exploiting raw data: using only standard word vectors with no enriching, our HPCD (basic) model performs comparably to an SVM with access to all enriching features. Once we improve the representation, we outperform both the SVM and full parsers. In comparison, the contribution of raw data to the SVM, as either word vectors or Brown clusters, is rather limited.

[12] For example, we tried adding all word vector dimensions as features, as well as element-wise products of the vectors representing the head and the child.

[13] As in (Koo et al., 2008), we limit the number of unique bit strings to 1,000 so full strings are not equivalent to word forms.

System	Arabic	English
Closest	62.7	81.7
SVM	77.0	86.0
w/ word vectors	77.5	85.9
w/ Brown clusters	78.0	85.7
w/ Brown clusters+prefixes	77.0	85.7
Malt	75.4	79.7
MST	76.7	86.8
Turbo	76.7	88.3
RBG	80.3	88.4
RNN	68.9	85.1
Charniak-RS	80.8	88.6
HPCD (basic)	77.1	85.4
w/ enriching	80.4	87.7
w/ syntactic	79.1	87.1
w/ relearning	80.0	86.6
HPCD (full)	82.6	88.7
RBG + HPCD (full)	82.7	90.1

Table 5: PP attachment accuracy of our HPCD model compared to other systems. HPCD (full) uses syntactic vectors with enriching and relearning. The last row is a modified RBG parser with a feature for the PP predictions of our model.

The relative performance is consistent across both English and Arabic. The table also demonstrates that the Arabic dataset is more challenging for all models. This can be explained by a larger average candidate set (Table 2), a freer word order that manifests in longer attachments (average head and PP distance is 3.3 in Arabic vs 1.5 in English), and the lexical sparsity induced by the richer morphology.

Effect on parsing To investigate how our PP attachment model contributes to the general parsing task, we incorporated the predictions of our model in an existing dependency parser. We modified the RBG parser (Lei et al., 2014) such that a binary arc feature fires for every PP attachment predicted by our model. For both test sets, we find that the parsing performance, measured as the unlabeled attachment score (UAS), increases by adding the predictions in this way (Table 6). The modified parser also achieves the best PP attachment numbers (Table 5).

Interestingly, incorporating the PP predictions in a parser leads to a gain in parsing performance that

System	Arabic	English
RBG	87.70	93.96
RBG + predicted PP	87.95	94.05
RBG + oracle PP	89.09	94.42

Table 6: Parsing performance (UAS) of the RBG parser, with predicted and oracle PPs.

is relatively larger than the gain in PP accuracy. For example, relative to an oracle upper bound of forcing gold PP arcs in the parser output (Table 6), the reduction in English parsing errors is 20%, whereas the reduction in PP errors is only 15%. This affirms the importance of PP attachment disambiguation for predicting other attachments in the sentence.

RRR dataset Much of the previous work on PP attachment focused on a binary classification scenario (Section 2) and has been evaluated on the RRR dataset (Ratnaparkhi et al., 1994). Such systems cannot be easily evaluated in our setting which allows multiple candidate heads. On the other hand, our full model exploits contextual information that is not available in the RRR dataset. Nevertheless, using a simpler version of our model we obtain an accuracy of 85.6% on the RRR test set.[14] This is comparable to much of the previous work (Olteanu and Moldovan, 2005), but still lags behind the 88.1% of Stetina and Nagao (1997), who also used WordNet information. However, our use of WordNet is rather limited compared to theirs, indicating that our enriching method can be improved with other types of information.

6.1 Alternative composition architectures

In this section we analyze how different composition architectures (Section 3.2) contribute to the overall performance. To isolate the contribution of the architecture, we focus on standard (linear) word vectors, with no relearning or enriching. As Figure 3 shows, simpler models tend to perform worse than more complex ones. The best variants use different composition matrices based on the distance of the candidate head from the PP (HPCD, HPCDN). While the results shown are for 100-dimensional vectors, similar trends are observed with lower dimensions, although the gaps between simple and complex models are then more substantial.

[14]Here we applied basic preprocessing similarly to (Collins and Brooks, 1995), converting 4-digit numbers to YEAR and other numbers to NUMBER; other tokens were lower-cased.

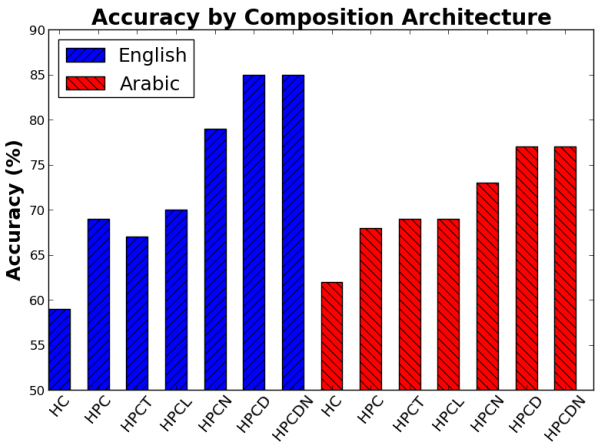


Figure 3: PP attachment accuracy of different architectures. (HC) uses only the candidate head and the child of the preposition; (HPC*) models use head, preposition, and child, with the following variants: (HPCT) ternary composition; (HPCL) local matrices for top and bottom compositions; (HPCN) context words; (HPCD) distance-dependent matrices; (HPCDN) combines HPCD+HPCN.

We have also experimented with compositions through the entire PP subtree. However, this resulted in a performance drop (to about 50%), implying that adding more words to the composite representation of the PP does not lead to a distinguishing representation with regards to the possible candidate heads.

6.2 Alternative representations

In this section, we analyze how different word vector representations (Section 4) contribute to our model. We focus on the HPCD model, which builds a two-step composite structure with distance-dependent composition matrices. We take the basic representation to be standard (linear) word vectors, without enriching or relearning. In each paragraph below, we investigate how a different aspect of the representation affects PP attachment performance.

Relearning word vectors In traditional architectures, the process of word vector induction is independent of the way the vector is used in the parsing algorithm. We hypothesize that by connecting

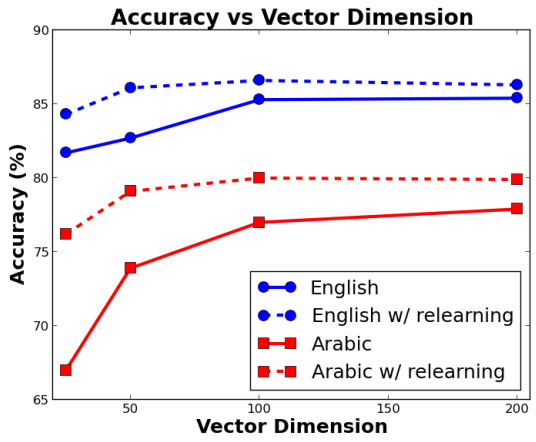


Figure 4: Effects of relearning standard word vectors in English and Arabic.

these two processes and tailoring the word vectors to the task at hand, we can further improve the accuracy of the PP attachments. We thus relearn the word vectors during training as described in Section 4.1. Indeed, as Figure 4 shows, doing so consistently improves performance, especially with low dimensional vectors. Interestingly, syntactic word vectors also benefit from the update (Table 8). This indicates that the supervised PP attachments provide complementary signal to noisy dependencies used to construct syntactic vectors.

Enriching word vectors A substantial body of work has demonstrated that multiple features can help in disambiguating PP attachments (Section 2). To this end, we enrich word vectors with additional knowledge resources (Section 4.2). As Table 7 shows, this enrichment yields sizable performance gains. Most of the gain comes from part-of-speech information, while WordNet and VerbNet have a smaller contribution. Updating the word vectors during training has an additional positive effect.

Note that even with no enrichment, our model performs comparably to an SVM with access to all enriching features (Table 5). When enriched, our model outperforms the SVM by a margin of 2-3%. With relearning, the gaps are even larger.

Syntactic word vectors While most of the work in parsing relies on linear word vectors (Socher et al., 2013; Lei et al., 2014), we consider an alternative vector representation that captures syntactic

Representation	Arabic	English
w/o enriching	77.1	85.4
w/ enriching		
+POS	78.5	86.4
+NextPOS	79.7	87.5
+WordNet+VerbNet	80.4	87.7
w/ enriching+relearning	81.7	88.1
w/ enriching+relearn.+syn.	82.6	88.7

Table 7: PP attachment accuracy when enriching word vectors with part-of-speech tags of the candidate head (POS) and the following word (NextPOS), and with WordNet and VerbNet features.

Representation	Arabic	English
Linear	77.1	85.4
Syntactic	79.1	87.1
Syntactic w/ relearning	80.7	87.7

Table 8: PP attachment accuracy of linear (standard) and syntactic (dependency-based) word vectors.

context. As described in Section 4.3, such vectors are induced from a large corpus processed by an automatic dependency parser. While the corpus is most likely fraught with parsing mistakes, it still contains sufficient dependency information for learning high-quality word vectors. Table 8 confirms our assumptions: using syntactically-informed vectors yields significant performance gains.

7 Conclusion

This work explores word representations for PP attachment disambiguation, a key problem in syntactic parsing. We show that word vectors, induced from large volumes of raw data, yield significant PP attachment performance gains. This is achieved via a non-linear architecture that is discriminatively trained to maximize PP attachment accuracy. We demonstrate performance gains by using alternative representations such as syntactic word vectors and by enriching vectors with semantic and syntactic information. We also find that the predictions of our model improve the parsing performance of a state-of-the-art dependency parser.

Acknowledgments

This research is developed in collaboration with the Arabic Language Technologies (ALT) group at Qatar Computing Research Institute (QCRI) within the IYAS project. The authors acknowledge the support of the U.S. Army Research Office under grant number W911NF-10-1-0533, the DARPA BOLT program and the US-Israel Binational Science Foundation (BSF, Grant No 2012330). We thank the MIT NLP group and the TACL reviewers for their comments, and Djamé Seddah and Mohit Bansal for helping with scripts and data. 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

Eneko Agirre, Timothy Baldwin, and David Martinez. 2008. Improving Parsing and PP Attachment Performance with Sense Information. In *Proceedings of ACL-HLT*.

Tressy Arts, Yonatan Belinkov, Nizar Habash, Adam Kilgarriff, and Vit Suchomel. 2014. arTenTen: Arabic Corpus and Word Sketches. *Journal of King Saud University - Computer and Information Sciences*.

Michaela Atterer and Hinrich Schütze. 2007. Prepositional Phrase Attachment Without Oracles. *Computational Linguistics*, 33(4).

Mohit Bansal, Keving Gimpel, and Karen Livescu. 2014. Tailoring Continuous Word Representations for Dependency Parsing. In *Proceedings of ACL*.

Yonatan Belinkov, Nizar Habash, Adam Kilgarriff, Noam Ordan, Ryan Roth, and Vít Suchomel. 2013. arTenTen: a new, vast corpus for Arabic. In *Proceedings of WACL*.

Yoshua Bengio and Xavier Glorot. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS*, volume 9, May.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of ICML*.

Eric Brill and Philip Resnik. 1994. A Rule-Based Approach to Prepositional Phrase Attachment Disambiguation. In *Proceedings of COLING*, volume 2.

Volkan Cirik and Hüsnü Şensoy. 2013. The AI-KU System at the SPMRL 2013 Shared Task : Unsupervised Features for Dependency Parsing. In *Proceedings of SPMRL*.

Michael Collins and James Brooks. 1995. Prepositional Phrase Attachment through a Backed-Off Model. *CoRR*.

Michael Collins. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of ACL*.

Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of ICML*.

Fabrizio Costa, Paolo Frasconi, Vincenzo Lombardo, and Giovanni Soda. 2003. Towards Incremental Parsing of Natural Language Using Recursive Neural Networks. *Applied Intelligence*, 19(1-2).

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *JMLR*, 12.

Chris Dyer. n.d. Notes on AdaGrad. Unpublished manuscript, available at `http://www.ark.cs.cmu.edu/cdyer/adagrad.pdf`.

Nuria Gala and Mathieu Lafourcade. 2007. PP attachment ambiguity resolution with corpus-based pattern distributions and lexical signatures. *ECTI-CIT Transactions on Computer and Information Technology*, 2.

Pablo Gamallo, Alexandre Agustini, and Gabriel P. Lopes. 2003. Acquiring Semantic Classes to Elaborate Attachment Heuristics. In *Progress in Artificial Intelligence*, volume 2902 of *LNCS*. Springer Berlin Heidelberg.

Spence Green. 2009. Improving Parsing Performance for Arabic PP Attachment Ambiguity. Unpublished manuscript, available at `http://www-nlp.stanford.edu/courses/cs224n/2009/fp/30-tempremove.pdf`.

Nizar Habash and Owen Rambow. 2005. Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop. In *Proceedings of ACL*.

Nizar Habash and Ryan Roth. 2009. CATiB: The Columbia Arabic Treebank. In *Proceedings of the ACL-IJCNLP*.

Nizar Habash, Owen Rambow, and Ryan Roth. 2005. MADA+TOKAN: A Toolkit for Arabic Tokenization, Diacritization, Morphological Disambiguation, POS Tagging, Stemming and Lemmatization. In *Proceedings of the Second International Conference on Arabic Language Resources and Tools*.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*.

Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. 2008. A large-scale classification of English verbs. *Language Resources and Evaluation*, 42(1).

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple Semi-supervised Dependency Parsing. In *Proceedings of ACL-HLT*.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser Showdown at the Wall Street Corral: An Empirical Investigation of Error Types in Parser Output. In *Proceedings of EMNLP-CoNLL*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-Rank Tensors for Scoring Dependency Structures. In *Proceedings of ACL*.

Omer Levy and Yoav Goldberg. 2014. Dependency-Based Word Embeddings. In *Proceedings of ACL*.

Percy Liang. 2005. Semi-Supervised Learning for Natural Language. Master's thesis, Massachusetts Institute of Technology.

Andre Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mario Figueiredo. 2010. Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In *Proceedings of EMNLP*.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of ACL*.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective Self-Training for Parsing. In *Proceedings of HLT-NAACL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online Large-Margin Training of Dependency Parsers. In *Proceedings of ACL*.

Srinivas Medimi and Pushpak Bhattacharyya. 2007. A Flexible Unsupervised PP-attachment Method Using Semantic Information. In *Proceedings of IJCAI*.

Sauro Menchetti, Fabrizio Costa, Paolo Frasconi, and Massimiliano Pontil. 2005. Wide coverage natural language processing using kernel methods and neural networks for structured data. *Pattern Recognition Letters*, 26(12).

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of Workshop at ICLR*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of NIPS*.

Jaouad Mousser. 2010. A Large Coverage Verb Taxonomy for Arabic. In *Proceedings of LREC*.

J. Nivre, J. Hall, and J. Nilsson. 2006. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In *Proceedings of LREC*.

Marian Olteanu and Dan Moldovan. 2005. PP-attachment Disambiguation using Large Context. In *Proceedings of HLT-EMNLP*.

Princeton University. 2010. WordNet. `http://wordnet.princeton.edu`.

Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. 1994. A Maximum Entropy Model for Prepositional Phrase Attachment. In *Proceedings of HLT*.

Horacio Rodríquez, David Farwell, Javi Ferreres, Manuel Bertran, Musa Alkhalifa, and M. Antonia Martí. 2008. Arabic WordNet: Semi-automatic Extensions using Bayesian Inference. In *Proceedings of LREC*.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, et al. 2013. Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of SPMRL*.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2010. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. In *Proceedings of NIPS Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013. Parsing with Compositional Vector Grammars. In *Proceedings of ACL*.

Jiri Stetina and Makoto Nagao. 1997. Corpus Based PP Attachment Ambiguity Resolution with a Semantic Dictionary. In *Fifth Workshop on Very Large Corpora*.

Simon Šuster. 2012. Resolving PP-attachment ambiguity in French with distributional methods. Master's thesis, Université de Lorraine & Rijksuniversiteit Groningen.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of ACL*.

Martin Volk. 2002. Combining Unsupervised and Supervised Methods for PP Attachment Disambiguation. In *Proceedings of COLING*.

Stefan Wager, Sida Wang, and Percy Liang. 2013. Dropout Training as Adaptive Regularization. In *Proceedings of NIPS*.