Neural Network Architectures for Prepositional Phrase Attachment Disambiguation

by

Yonatan Belinkov

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2014

Certified by .....
Regina Barzilay
Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor
Chairman, Department Committee on Graduate Students

Neural Network Architectures for Prepositional Phrase Attachment Disambiguation

by

Yonatan Belinkov

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

This thesis addresses the problem of Prepositional Phrase (PP) attachment disambiguation, a key challenge in syntactic parsing. In natural language sentences, a PP may often be attached to several possible candidates. While humans can usually identify the correct candidate successfully, syntactic parsers are known to have high error rated on this kind of construction. This work explores the use of compositional models of meaning in choosing the correct attachment location.

The compositional model is defined using a recursive neural network. Word vector representations are obtained from large amounts of raw text and fed into the neural network. The vectors are first forward propagated up the network in order to create a composite representation, which is used to score all possible candidates. In training, errors are backpropagated down the network such that the composition matrix is updated from the supervised data. Several possible neural architectures are designed and experimentally tested in both English and Arabic data sets.

As a comparative system, we offer a learning-to-rank algorithm based on an SVM classifier which has access to a wide range of features. The performance of this system is compared to the compositional models.

Thesis Supervisor: Regina Barzilay
Title: Professor

Acknowledgments

I am grateful to my advisor Regina Barzilay for her support and advice throughout my research, without which I could not have completed this work. I would like to thank members of the MIT NLP group and other friends and colleagues at MIT for helpful ideas and discussions: Yuan Zhang, Tao Lei, Nate Kushman, Karthik Rajagopal Narasimhan, Yoong Keok Lee, Zach Hynes, Tahira Naseem, and Yevgeni Berzak. I also thank Amir Globerson from the Hebrew University for useful suggestions. I would also like to acknowledge the support of the Qatar Foundation.

My sincere thanks to all my friends at MIT and Cambridge, and in particular the large Israeli community in the Boston area, as well as to friends from home who support me from afar. Throughout my research work I found tremendous comfort and encouragement in old and new friendships that were formed here.

Finally, I dedicate this thesis to my parents, who always stand behind me, and to my beloved wife Niva, who is my ultimate source of power and my best friend. Thank you for always believing in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the major challenges in syntactic parsing is resolving ambiguities in prepositional phrase (PP) attachment — determining the head of a PP in the tree. Traditional research has formulated this problem as a binary decision: deciding whether a PP should attach to a preceding noun or verb (Figure 1-1). Earlier work has used rule-based [5] and statistical methods [20, 6], achieving results of 80-85%. These numbers are surprisingly close to an upper bound of human judgment, when restricted to quadruples of verb-noun-preposition-noun. However, when humans have access to the full sentence, they are able perform as high as 93% [20].

The above line of work has been criticized as assuming an unrealistic parsing scenario: an oracle parser is used to provide two possible attachments, ignoring all other theoretically possible heads [1]. Indeed, recent work has shown that PP attachment remains a major source of errors in parser evaluation [13]. Furthermore, while incorporating semantic knowledge has proven useful in the constrained scenario [24], most state-of-the-art parsers do not exploit such resources. A notable exception is the work in [1], which achieved gains in parsing and specifically in PP attachment by substituting words with semantic classes as a preprocessing step before running a statistical parser.

Arabic exhibits several syntactic phenomena that make parsing, and specifically PP attachment, a challenging task. Its free word order allows for PPs to move quite far from their

heads; and its construct state construction (similar to compound nouns) causes ambiguity between several possible head nouns (Figure 1-2). In addition, its rich morphology increases out-of-vocabulary rate and sparsity challenges for lexicalized models. A comparison between Arabic and English treebanks shows that out-of-vocabulary words are more common in the Arabic data set, in particular for verbs and nouns, which are potential heads of a PP. For further statistics regarding Arabic and English data sets, see Section 4.2.

Previous work on Arabic PP attachment has attempted to address such problems by using the web as a corpus for collecting collocation scores [2] or by engineering state-split features for the Stanford parser [9]. This thesis present a different approach, based on learning compositional representation of words and phrases for disambiguating the PP attachment decision. This model exploits word vector representations and recursively builds composite representations through different architectures of neural networks. As a comparative system, a linear classifier which has access to a wide range of features is also implemented and tested.

The use of word vector representations created from raw texts is discussed in Chapter 2, while Chapter 3 defines the models explored in this work. The data sets that are used are reviewed in Chapter 4 and experiments and results are presented in Chapter 5.

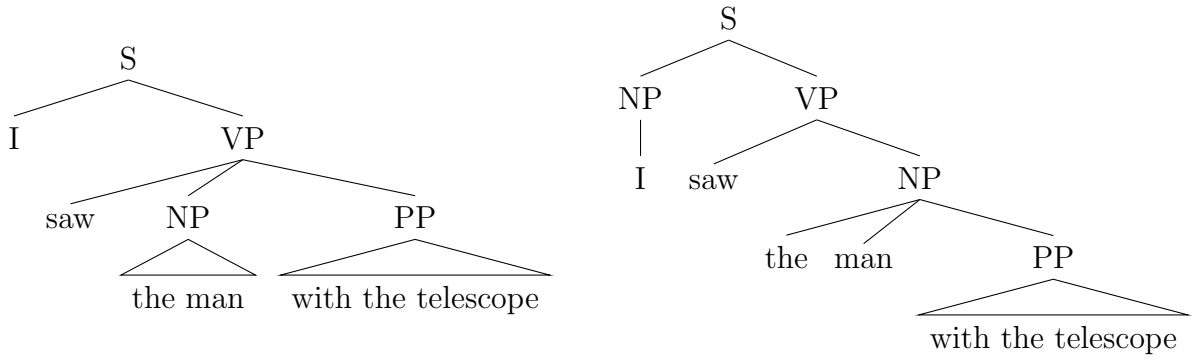


Figure 1-1: Two possible attachments for the PP "with the telescope". In the correct attachment (Left), the PP attaches at the VP level and the head is the verb "saw". In the wrong attachment (Right), the PP attaches at the NP level and the head is the noun "man".

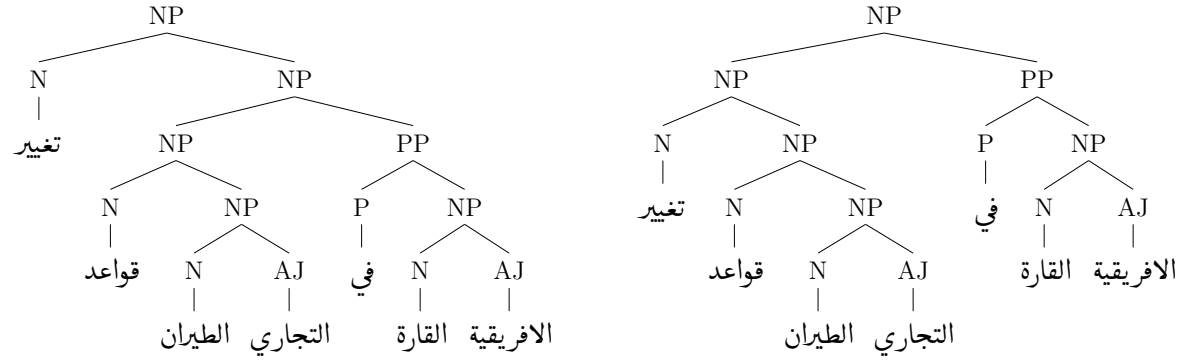


Figure 1-2: An Arabic construct NP with two possible attachments for the PP في القارة الافريقية ("in the African continent"). In the correct attachment (Left) the PP attaches to the second level, meaning "changing the rules of commercial flights in the African continent". In the wrong predicted attachment (Right), the PP attaches TO the top level, meaning the the change, rather than the rules, is in the African continent. Notice that there are three possible attachment levels in this construction.

Chapter 2

Word Vector Representations

2.1 Background

Word representations have been used in Natural Language Processing at least since the 1980s. However, the gained renewed popularity in recent years due to advances in developing efficient methods for inducing representations from large amounts of raw text. A survey of different representations is given in [25], where three types of word representations are discussed. Distributional representations are based on co-occurrences statistics of words in some context. Since they have dimensionality the size of the vocabulary, different dimensionality reduction techniques can later be applied. For example, using Singular Value Decomposition leads to Latent Semantic Analysis [8]. Another type of representation is based on word clustering, where Brown clustering is a notable example. Finally, distributed representations, also known as word embeddings, are low dimensional, real values vectors, where each dimension is a latent feature of the word.

Traditionally, distributed representations have been created by using neural network language models. A major obstacle in using such representations is that the neural language models are typically slow to train. Thus much work has focused on efficient methods for training such models, for example in [7, 4]. More recently, two successful algorithms for training distributed word representations have been suggested in [16]: the continuous bag-

of-words and Skip-gram model. Both models essentially dispense with the non-linearity that is the heavy factor in other models. Further modifications to the Skip-gram models improved efficiency [17]. The following section presents this model is some details.

2.2 The Skip-gram model

The Skip-gram model has been suggested in [16, 17] as an efficient method for learning word vector representations from large raw text data. The model optimizes the following objective function:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2.1}$$

where the training corpus is raw text with T words $w_1, ...w_T$. For each word w_t we define a context window of size c on each side of w_t, and try to maximize the average log probability of every word generating the words in its context. This probability is defined as the softmax:

$$p(w_{t+j}|w_t) = \frac{\exp({v'_{w_{t+j}}}^T v_{w_t})}{\sum_{w=1}^{W}\exp({v'_w}^T v_{w_t})} \tag{2.2}$$

where v_w and v'_w are input and output vector representations of word w that are obtained from a neural net-like architecture, but without the non-linearity. Since calculating this probability requires summing over the entire size W vocabulary, it is inefficient to calculate in practice. In [17] several modifications are suggested in order to improve efficiency, including hierarchical softmax. In this approximation, the W vocabulary words are represented in a binary Huffman tree, where more frequent words are assigned shorter binary codes. Calculating the probability $p(w_{t+j}|w_t)$ is then carried by following the tree path to w_{t+j} and taking the product of all the terms of the form $\sigma({v'_{w'}}^T v_{w_t})$, where w' is an internal node on the path to w_{t+j}.

The Skip-gram model is optimized with stochastic gradient descent, by going over the entire data set in one pass, The learning rate is decreased after each mini-batch in a linear fashion until approaching a learning rate close to zero at the end of the raw text. Another

peculiarity of the algorithm is that, for each word w_t, instead of looking at all the neighbors in a window of fixed size c, a random size $< c$ is sampled and only the words in that smaller window are considered as the current context. This has the effect of given higher weight to closer words in the gradient updating.

In the experimental results reported in Chapter 5, the word vectors are obtained using the word2vec tool.[1] The Skip-gram model is trained with hierarchical softmax and default parameter settings. This also includes removing word types of small count (< 5). The only variant considered is the dimensionality of the word vectors, which are varied from 25-200. Preliminary experiments with other model variations (e.g. negative sampling), have not resulted in notable performance gains.

[1] `https://code.google.com/p/word2vec`.

Chapter 3

Models

3.1 Compositional models

Since PP attachment disambiguation requires both semantic and syntactic knowledge, we introduce a notion of compositionality. Given vectors $u, v \in \mathbb{R}^n$, representing two words with relation R, and given some knowledge source K, let their composition vector be defined as

$$p = f(u, v, R, K) \tag{3.1}$$

There are many possible realizations of the function f, including additive, multiplicative and non-linear functions [18]. Importantly, the resulting vector is of the same dimension as its constituents, i.e., $p \in \mathbb{R}^n$, which allows for recursive application of the compositionality operator (similarly to the ideas in [23]). Let $s(p) = wp$ denote the score of the composition.

Given a sentence $x = x_1, ..., x_n$ with part-of-speech tags $t = t_1, ..., t_n$, let $PREP(x) = i_1, ..., i_k$ index prepositions in x, i.e., $t_{i_j} = prep\ \forall j$. Let $y_{i_1}, ..., y_{i_k}$ denote the corresponding PP attachments, where attachment $y_j = (h, b)$ means that the head of the PP is x_h and its (right) boundary is x_b. The score of preposition x_j and attachment y_j is $s(x_j, y_j)$. Below we discuss possible definitions for this score. During testing, we are seeking the maximizing attachment: $\arg\max_{y \in \mathcal{Y}(x_j)} s(x_j, y)$, where $\mathcal{Y}(x_j)$ is the set of all possible attachments for x_j. Since we focus on the limited problem of PP attachment, this set is expected to be tractable

in practice.

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

$$\sum_i \sum_{j \in PREP(x^{(i)})} s(x_j^{(i)}, y_j^{(i)}) - \max_{y \in \mathcal{Y}(x_j^{(i)})} (s(x_j^{(i)}, y) + \Delta(y, y_j^{(i)})) - \frac{\lambda}{2} \sum_\theta \theta^2 \tag{3.2}$$

where the loss is defined as $\Delta(y, y_j^{(i)}) = 1 - \delta(h, h_j^{(i)}) + |b - b_j^{(i)}|$, where $y = (h, b)$ and $y_j^{(i)} = (h_j^{(i)}, b_j^{(i)})$. If we know the span of the PP (b is given), then the loss reduces to $\Delta(y, y_j^{(i)}) = 1 - \delta(h, h_j^{(i)})$. Such loss functions make the objective non-differentiable, so we cannot compute a gradient. Instead, we can use the subgradient:

$$\sum_i \sum_{j \in PREP(x^{(i)})} \frac{\partial s(x_j^{(i)}, y_j^{(i)})}{\partial \theta} - \frac{\partial s(x_j^{(i)}, y_{max})}{\partial \theta} - \lambda\theta \tag{3.3}$$

The score of a single attachment $s(x_j, y_j)$, where $y_j = (h, b)$, is defined as:

$$s(x_j, y_j) = s(f(x_h, p_{j,b})) + s(f(x_j, p_{j+1,b})) + s(p_{j+1,b}) \tag{3.4}$$

where f is the composition operation from Eq. 3.1 and $p_{j,b}$ is the combined representation of words $x_j, ..., x_b$. Here the first term corresponds to the attachment of the PP $p_{j,b}$ to the head x_j and the second to the composition of the preposition x_j with the NP $p_{j+1,b}$. The third term represents the NP, and is constructed recursively according to the internal structure of the NP. Again, if we assume we know the span of the PP, as well as its internal structure, then the score reduces to $s(x_j, y_j) = s(f(x_h, p_{j,b}))$.

The function f can be defined using a neural net as follows:

$$f(u, v) = g(W[u\ v] + b) \tag{3.5}$$

where b is the bias term, $[u\ v] \in \mathbb{R}^{2n}$ is a concatenation of u and v, and g is a non-linear function such as tanh or sigmoid. Using this scoring function, the objective can be maximized

by backpropagation through structure and subgradient methods [23]. For optimization, we use L-BFGS. [1]

An example of the backpropagation algorithm is shown in Algorithm 4. If desired, errors can be backpropagated all the way down to the word vectors. This updating is explored in the experiments and discussed in Section 5.1.

The scoring and composition functions defined above are general functions used in parsing. We would like to specialize them to the case of PP attachment. The general form of the composition function (Eq. 3.1) suggests a dependency on the type of relation R and existing knowledge K. To incorporate the type of relation into the composition function, we can have different matrices W^R for the different compositions in each PP attachment. In this case, we have at least three syntactic relations involved in each PP attachment: attaching the PP to the candidate head ($R_{h,pp}$), attaching the preposition to the NP ($R_{p,np}$), and building the NP (R_{np}). We then define the corresponding matrices: $W^{R_{h,pp}}$, $W^{R_{p,np}}$, and $W^{R_{np}}$; if we assume we know the internal structure of the NP, we can ignore the last of these. Various such neural network architectures are discussed in Section 3.2.1.

The advantage of introducing multiple matrices W^R is in capturing different aspects of the data by different parameters. In theory, there's no good reason to assume one parameter matrix would perform well across all compositions. However, the downside in increasing the number of estimated parameters is more sparsity. The different possibilities are experimentally tested and discussed in Section 5.1.

3.2 Neural Net Architecture

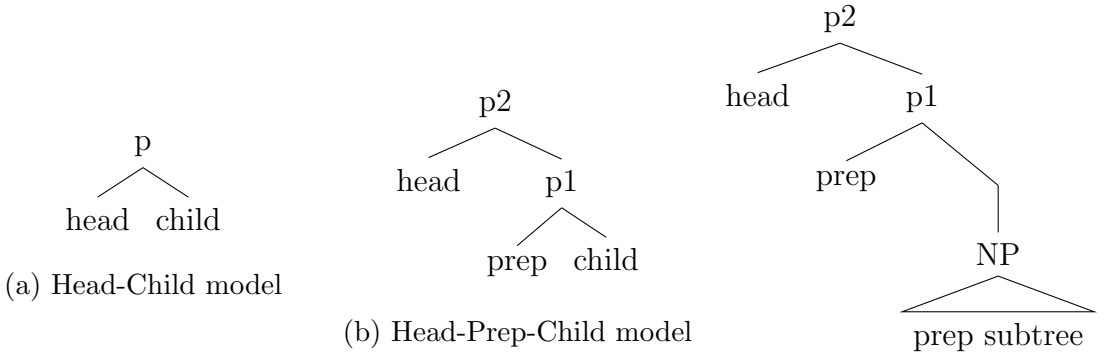
3.2.1 Tree Structure

There are different ways to exploit the elements participating in each PP attachment: the candidate head, the preposition, and any descendants of the preposition. Perhaps the sim-

[1] We use the Matlab implementation by Mark Schmidt, available at `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`.

plest is to consider only the candidate head and the first child of the preposition. These two words are then fed into a neural network and the resulting parent is scored by the scoring vector (Figure 3-1a). A more elaborate process first composes the preposition with its first child, then composes their parent with the candidate head (Figure 3-1b). Finally, we can consider the entire NP subtree of the preposition, first composing all words according to the tree structure, then composing the resulting NP parent with the preposition, and then with the head (Figure 3-1c). In all cases the top parent is then scored by the scoring vector w.

The trees in Figure 3-1 are all binary; however, a PP attachment essentially involves three elements: the candidate head, the preposition, and the child or descendants of the preposition. These may be combined at one step using a ternary composition (Figure 3-2).

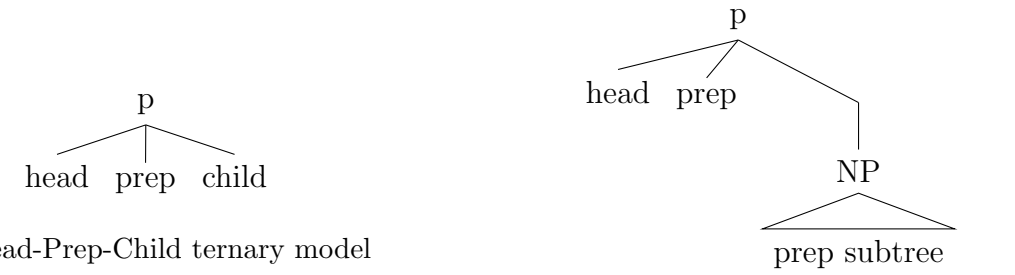


(a) Head-Child model

(b) Head-Prep-Child model

(c) Head-Prep-Subtree model

Figure 3-1: Neural network architectures using binary compositions.



(a) Head-Prep-Child ternary model

(b) Head-Prep-Subtree ternary model

Figure 3-2: Neural network architectures using ternary compositions.

The NP subtree below the preposition (as in Figure 3-1c) is given as a dependency tree in the used data sets. However, the binary composition operations assumes a binary tree. In the case where a certain word has more than one child, it is not straightforward to apply the composition operations. To deal with this issue, a binary tree is built during the forward propagation step. Suppose a word w has children c_1 and c_2. We can compose them with their head word in two steps: first compose one child with the head word w, yielding a parent p_1, then compose the second child with p_1 to get the final parent p_2. The order in which the children are composed can be, for example, starting from the closest child and moving towards the farthest one, or vice versa. Algorithm 3 shows the pseudo-code when the farthest child is the first to be composed with the head. The intuition for this is that closer children should contribute more to the meaning of the phrase, so they should be composed at higher levels of the tree. Conversely, it could be argued that composing less important words at the bottom of the tree blurs the meaning, so we experiment with both options. Yet another method for composing words in a dependency tree is suggested in [22].

Once the tree has been built, gradients are calculated with back-propagation through structure (Algorithm 4. The experimental results of these algorithms are discussed in Section 5.1.

3.2.2 Granularity of Composition Matrices

The simplest model assumes one composition matrix W for all binary composition operations. In the case of the Head-Child model (Figure 3-1a), we only have one such composition. However, the Head-Prep-Child model (Figure 3-1b), for example, contains two compositions, so we might want to use different matrices W^{top} and W^{bottom} for the two compositions. Furthermore, if we consider the entire subtree of the preposition (Figure 3-1c), we may want to use different matrices for the different compositions in the subtree. For example, we could use a different matrix based on the depth of the composition in the subtree. Such a formulation would allow the model to learn different parameters based on the depth of the node in the tree, reflecting the intuition that words that are composed farther down the tree should

have a different (arguably, smaller) contribution to the overall representation of the subtree.

An orthogonal direction to increase granularity of composition matrices is to assign different matrices when composing with different candidate heads. Each preposition may have several candidate heads where it could attach (see Table 4.2 for relevant statistics). We might have some information about these heads that could be relevant to the composition. For example, we might know their part-of-speech or some morphological features. At the least, we have access to their position in the sentence with respect to the preposition. It turns out that the distance of the candidate head from the preposition is an extremely informative feature (see also the experiments in Section 5.2), which calls for using different composition matrices when composing with heads in different distances. However, to avoid explosion of parameters, it is useful to bin the distances based on the distribution in the training data. The allowed distances are 1-5, where all cases of distance ≥ 5 are put in the same bin. The experiments in Section 5.1 show the benefit of this approach.

The experimental results reveal a complicated relation between granularity of matrices and prediction quality. To some extent, it is useful to go beyond the global matrix formulation and consider multiple matrices based on various considerations. On the other hand, some of the more complicated models (e.g. the Head-Prep-Subtree model of Figure 3-1c) do not benefit from multiple composition matrices. For further discussion, see Section 5.1.

3.2.3 Exploiting Context

The models described so far consider only three elements: the candidate head, the preposition, and its descendants. While the descendants of the preposition capture one aspect of contextual information, there are other words in the sentence which might tell us something about the attachment decision. Indeed, the experiments described in Section 5.1 suggest that it is useful to exploit the words surrounding the candidate head, especially the following word. This can be integrated in the neural net architecture in the following way: for each candidate head, represented by a vector of size n, concatenate the vector representing the following word. If the following word is not in the word vector vocabulary, or if the head is

immediately followed by the prepositions, append a zero vector. This results in a $2n$ vector representation for each head. To compose it with another vector of size n (representing the PP), we need a composition matrix of size $n \times 3n$, similar to the ternary composition described above. The utility of using context in such a way is corroborated by the results in Section 5.1.

3.3 Learning to Rank with Linear Classification

As a comparative system, we consider treating the problem as a learning-to-rank problem. Each instance provides us with a correct candidate head and several incorrect candidates. We can rank these as as a simple list where the correct candidate has the highest rank and all other candidates have a single lower rank. Then several learning-to-rank methods can be used to train a model [14]. For example, given a feature vector x, a weight vector w and a linear scoring function $f(x) = \langle w, x \rangle$, we know that:

$$f(x_i) > f(x_j) \Leftrightarrow \langle w, x_i - x_j \rangle > 0 \tag{3.6}$$

We then say that x_i is ranked before x_j if $\langle w, x_i - x_j \rangle > 0$. Therefore we can train a linear classifier to determine the pairwise decision and translate this to a ranking. In the experiments in Section 5.2, the linear classifier used is Support Vector Machines.[2]

One advantage of using an SVM is that many features can be easily incorporated. This includes syntactic and morphological features, contextual features, as well as external knowledge resources. The knowledge resources are discussed in Section 4.3; the specific features are discussed in Section 5.2. It is less obvious how to incorporate word vectors learned from raw data. Using the dot product between word vectors is shown to be empirically fruitful, but this limits their contribution to the learned model to one dimension. Attempts to use more dimensions in the SVM classification were unsuccessful. In contrast, the compositional models are able to capture the full dimensionality of the word vectors.

[2]I use the SVMRank tool: `http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`.

Chapter 4

Data

4.1 Raw Text Corpora

The majority of the experiments are conducted on Modern Standard Arabic data, which are described below, but some of the experiments are repeated on English data for comparison. The initial word vectors are obtained from raw text corpora using the Skip-gram model (Section 2.2). Two Arabic corpora are considered for creating word vectors: arTenTen and the Arabic Gigaword.

arTenTen. This is a large scale corpus of automatically crawled web texts comprising 5.8 billion words [3]. The texts have been gatherd in 2012 and a sub-corpus has been tokenized and lemmatized with MADA [10, 11]. Due to the morphological nature of the Arabic language, it is important to tokenize the text in order to separate some of the prepositions from their child words. In addition, working on the lemma level allows sharing of information between different morphological variants which share the same meaning. After pre-processing the sub-corpus we have 130 million tokens from which the word vectors are trained.

Arabic Gigaword. This corpus contains newswire texts from several news agencies, beginning from as early as 1994 up to 2011.[1] It contains 1.2 billion words. The entire corpus has been tokenized and lemmatized using MADAMIRA, a new faster version of MADA.[2] Again, prepositions are separated and lemmas are considered for subsequent experiments.

The whole Arabic Gigaword corpus is much larger than the morphologically processed sub-corpus of arTenTen. However, in preliminary experiments with word vectors obtained from the two corpora, there was no notable performance difference between them. In fact, arTenTen word vectors tend to perform slightly better for the PP attachment task, despite the smaller size of the corpus. This might be explained by its relatively more up-to-date texts. Thus, the results reported in Chapter 5 are all with word vectors created from arTenTen.

English. For the English experiments I use Wikipedia texts in order to create word vectors. In particular, the first 1 billion characters from a 2006 dump of Wikipedia are downloaded and pre-processed to produce raw text.[3] This results in 120 million words which are used to create the word vectors as is, without further pre-processing. In particular, no lemmatization has been performed, since the data set is assumed to be large enough for a morphologically poor language such as English.

4.2 Syntactically Annotated Corpora

In order to train the supervised PP attachment model we need an annotated corpus of PP attachment decisions. These can be extracted from standard treebanks. For Arabic, the CATiB dependency treebank is used [15]. For English, the standard Penn treebank is converted to a dependency format with the Pennconverter tool.[4]

[1] LDC Catalog No. LDC2011T11.

[2] The MADAMIRA version used is 1.0 beta, available at `http://innovation.columbia.edu/technologies/cu14012_arabic-language-disambiguation-for-natural-language-processing-applications`.

[3] The data and pre-processing script are available at `http://mattmahoney.net/dc/textdata.html`.

[4] `http://nlp.cs.lth.se/software/treebank_converter/`.

As explained in the introduction, the rich morphology of the Arabic language increases out-of-vocabulary rate and sparsity challenges for lexicalized models. A comparison between Arabic and English treebanks (Table 4.1) shows that out-of-vocabulary words are more common in the Arabic data set, in particular for verbs and nouns, which are potential heads of a PP. Noun heads are more difficult to detect in Arabic than in English, as evidenced by their greater distance from their child PP. However, verb heads have roughly the same distance from their child PP in both languages.

	Arabic	English
OOV rate	16.7%	12.7%
Verb OOV rate	18.0%	8.9%
Noun OOV rate	17.2%	10.9%
Verb-prep arc length	4.4	4.2
Noun-prep arc length	2.5	1.5
Sentence length	37.4	23.9

Table 4.1: A comparison of statistics from the Arabic and English treebanks. Out-of-vocabulary (OOV) rates are the fraction of test words not seen in training data; lengths are averaged over the entire corpus.

Extracting instances of PP attachment from the treebanks is done in the following manner. For each preposition, we look for all possible candidate heads (attachment locations) in a fixed window. Typically, these are preceding nouns or verbs. Only prepositions which have a noun child are considered, which leaves out some rare exceptions. Empirically, limiting the candidate heads to appear close enough before the preposition is not an unrealistic assumption: more than 90% of the PP attachments are covered when considering a maximum distance of 10 words. Unambiguous attachments with only one possible candidate head are discarded. Table 4.2 shows statistics of the extracted PP attachment data sets. The data sets of the two languages are fairly similar in numbers, except for the much larger set of prepositions in the English data.

	Arabic		English	
	Train	Test	Train	Test
Total	42387	3917	35359	1951
Candidate heads mean (std)	4.5 (1.4)	4.3 (1.4)	3.7 (1.2)	3.6 (1.2)
Vocab size	13127	4380	11429	2440
Head vocab size	8225	2936	10395	2133
Prep vocab size	13	10	72	46
First child vocab size	4222	1424	5504	983

Table 4.2: Statistics of extracted PP attachments from the Arabic and English treebanks.

4.3 Knowledge Resources

Extrnal knowledge resources can be used in both the compositional model (K in Eq. 3.1) and and the linear classifier. The contribution of such resources to PP attachment, and parsing in general, is questionable as previous works have shown mixed results (Section 1). Therefore, it is interesting to see how such resources contribute in our case in adding semantic and syntactic information.

On the semantic level, Arabic WordNet [21] contains 11,000 synsets which can abstract away from the lexical items. In particular, the top and second level hypernym for each word that is covered in the resource are extracted. On the syntactic level, Arabic VerbNet [19] provides verb classes and sub-categorization frames for 7748 verbs. These frames contain, for example, information of which prepositions tend to modify which verbs, which could be useful for determining the correct attachment. Dividing verbs into classes and nouns into sub-types (e.g. verbal nouns, participles) can help share information between training instances. As for the lexical level, it is mostly captured by using word vectors.

Chapter 5

Experimental Results

5.1 Compositional models

Table 5.1 shows the results of the compositional models on the Arabic and English test sets. In all shown models, only word vectors are used without additional features. An analysis of the Arabic results reveals some interesting patterns. As a simple but quite successful baseline, always choosing the closest candidate head gives a 63% accuracy, which is useful to keep in mind. The simplest compositional model is Head-child (Figure 3-1a), which considers only the candidate head and the first child of the preposition, with one composition operation. Is able to perform close to the baseline (59% compared to 63%), but only when high dimensional vectors (100-200 dimensions) are used. Lower dimensional vectors give relatively poor results (40-47%). Including the preposition and building a two-step composite structure (Figure 3-1b), as in model Head-prep-child, improves the results to 68% with 100-200 dimensions. Further small improvements are achieved by the local variant, which allows different composition matrices for the top (head+PP) and bottom (preposition+child) compositions, and by the ternary model (Figure 3-2a).

All of the previous models require using vectors of 100-200 for getting reasonable results. Things are much different once we consider not only the candidate head, but also its context (Section 3.2.3). When the word following the head is appended to it, the results jump to 70%

with just 25 dimensional vectors and up to 73% with higher dimensions. The best results are achieved by distinguishing the candidate heads based on their distance from the PP (Section 3.2.2), ranging from 72-73% with lower dimensional vectors to 76-77% with higher dimensional ones. Combining the contextual with the distance based model (dist+next) does not yield additional improvements, except for the 50 dimensional case (+3%).

	Arabic				English			
	25	50	100	200	25	50	100	200
Head-child	40	47	62	62	39	58	59	59
Head-prep-child	40	48	68	68	40	68	69	67
Head-prep-child-local	42	49	69	69	41	69	70	70
Head-prep-child-ternary	42	50	69	70	41	66	67	70
Head-prep-child-next	70	72	73	73	77	78	79	79
Head-prep-child-dist	72	73	77	76	82	83	82	83
Head-prep-child-dist+next	72	76	77	77	84	85	85	85

Table 5.1: Results of compositional models in Arabic and English. **Head-child** considers only candidate head and first child of the preposition (Figure 3-1a), while **Head-prep-child** considers also the preposition (Figure 3-1b). The **local** model allows different composition matrices W for the top (head + PP) and bottom (prep + child) compositions. The **ternary** model uses one ternary composition operation (Figure 3-2a). The **next** version includes the word following the candidate head as context, while the **dist** version assigns different composition matrices based on the binned distance of the candidate head from the PP.

The English results generally follow the same trends as the Arabic ones. Simpler models do not perform as well as more complex ones and require higher dimensional vectors. The more complex model achieve good results even with low dimensional word vectors. One difference is in the combined model that considers both context and distance (dist+next): in the English case it exhibits a significant improvement from the distance based model (+2%), reaching 85%, whereas in the Arabic case both give similar results. This puts the

best English results (94-95% with the Head-prep-child-dist+next model) about 8% higher than the best Arabic results (76-77% with the same model). This gap may be explained by the rich Arabic morphology which requires of lemmatization and tokenization (Section 4.1). Such pre-processing may introduce errors which affect the trained word vectors. In addition, lemmatization may collapse different forms that could have different syntactic behavior, although this is difficult to verify. Note that the Arabic and English data sets are similar in size for both the raw texts used for creating word vectors and the annotated treebanks used to extract PP attachment cases (Chapter 4).

The results discussed so far are for models which do not consider the entire PP structure, only the preposition and its first child. In practice, other words in the PP might contribute to its meaning and to disambiguating the attachment decision. Section 3.2.1 discusses how to build a compositional representation from the PP structure. See algorithms 3 and 4 for forward and backward propagation in such trees, respectively. The results of this model on the Arabic test set, with a global composition matrix, are 34%, 42%, 46%, and 50%, for vector dimensions 25, 50, 100, and 200. These results are clearly worse than most of the simpler models which only consider the first child of the preposition. Experimenting with different variations on this model has not led to any improvements. Attempted variations include: building the tree from left to right, limiting the span of the PP to 5-10 words, and allowing different composition matrices at different depths in the tree. It seems 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. The first child of the PP turns out to carry enough meaning to effectively disambiguate the attachment decisions, and including other words in the PP has no positive effect on the performance.

A natural question to explore is whether the initial word vectors can be updated based on the supervised PP attachment decisions. This can be achieved by backpropagating the errors all the way down to the word vectors. Then, the updated word vectors can be used in test time to better capture the PP attachment decisions. Unfortunately, doing this has not resulted in better accuracy on the test set. While updating the word vectors leads to slightly

better training accuracy, in test time the accuracy is similar or worse than not updating the word vectors, even when the regularization parameters is varied. A possible explanation could be that the initial word vectors are already good enough in capturing word similarities that are meaningful to PP attachment, such that further updating them is not helpful.

5.2 Learning to rank model

The results of the learning to rank model with the SVM classifier are shown in Table 5.2. The SVM classifier (Section 3.3) has access to a large number of features, which are described below. They can be divided into feature groups based on their source: Treebank, Wordnet, Verbnet, and raw texts. In this section only results for Arabic are reported.

The lexical treebank features achieve only 55.8% accuracy, with a slight improvement from POS features. The biggest jump (+17%) comes from including the normalized distance of the candidate head from the preposition. Further including contextual and morpho-syntactic features achieves almost 77%. It should be noted that not all morphological features were found beneficial. When doing ablation testing, determiner features were found useful but gender, number, and person features were redundant. Including semantic and syntactic knowledge resources brings another small improvement (+0.7%), reaching 77.6%, and using cosine similarity of word vectors from raw data has a similar effect. Combining all features, this model achieves an accuracy of 78.4%.

Treebank features. These features are extracted from the annotated treebank. They include *lexical* features like the lemma of the candidate head, the preposition, and the first child of the preposition; *morpho-syntactic* features as the part-of-speech (verb/noun) and morphological features (definiteness, number, gender, person) of the candidate head; *contextual* features which are lexical and syntactic features of neighboring words; and *distance* of the candidate head from the PP.

Verbnet features. The Arabic Verbnet (AVN) contains sub-categorization frames, from which *verb-preposition* pairs have been extracted. If the preposition appears with the candidate verb in such a frame, a boolean feature is hit. A similar feature is used if a preposition appears with a candidate verbal noun. Other features include whether the candidate head and the child are verbal nouns or participles. Furthermore, if a verb appears in an AVN verb *class*, it fires a unique feature.

Wordnet features. The Arabic Wordnet (AWN) includes synset information from which we extract the topmost and second top *hypernym*. Separate such features are given to the candidate head and the child.

Raw text features. The word vectors which were created from raw text are used by taking the *cosine similarity* between the candidate head and the child of the preposition. In order to include information about the preposition in the word vectors, the raw text has been modified such that prepositions are concatenated to their following nouns,[1] after which word vectors have been trained. This has the effect of viewing preposition-child pairs as single words, so if the same word appears with a different preposition it will have a different vector. The feature is then the cosine similarity between that *concatenated* vector and the candidate head. In practice, this heuristic gives better results than not including the preposition. Both arTenTen and Gigaword vectors are considered.

[1] In Arabic the preposition is always followed by its child, which makes this processing possible.

	Accuracy	Top-2 Accuracy
Treebank Lexical	55.8	79.9
+POS	57	80.8
+Distance	74.1	92
+Context+morph-syntax	76.9	92.4
Treebank+AVN	77.5	92.3
Treebank+AWN	77.1	92.4
Treebank+AVN+AWN	77.6	92.2
Treebank+arTenTen	77.3	92.7
Treebank+Gigaword	75.2	92.6
Treebank+arTenTen+Gigaword	77.8	92.7
All	78.4	93

Table 5.2: Results of learning to rank models in Arabic. **Treebank** features are extracted from the Arabic treebank. Lexical features include lemmas of candidate head, preposition, and child; POS identifies the part-of-speech of the candidate head; Distance is the candidate head normalized distance from the PP; context and morph-syntax features include other words in the sentence and their morphology. **Treebank+AVN/AWN** includes all treebank features and the Arabic Verbnet/Wordnet features; and **Treebank+corpus** includes all treebank features and cosine similarity of word vectors created from said corpus. **All** combines all features.

Chapter 6

Conclusions and Future Work

This work offers compositional models of word vector representations in order to disambiguate Prepositional Phrase attachment decisions, focusing on the Arabic language. Word vectors are initially trained from a large corpus of raw text. They are then combined in a recursive neural networks to produce a composite representation, which is used to score all possible attachment locations. The composition matrices and scoring vectors are trained in a max-margin framework using backpropagation through structure.

Various architectures of neural networks are suggested and experimentally investigated. It is shown that using the first child of the preposition is superior to exploiting the whole PP structure. The best model is achieved by combining two variations: first, training different composition matrices based on the distance of the candidate attachment location from the PP, and second, concatenating neighboring words to extend the vector representation.

Since the compositional models have access only to the initial word vectors and no other features, the results are compared with an SVM classifier which exploits a wide range of contextual and external features. Experiments show that the compositional models are able to perform as well as the SVM ranker even though their input is limited to word vectors.

There are several possible directions for future work. First, the success in disambiguating PP attachment decisions without solving the entire parsing problem can be exploited by combining such systems in complex parsers, either as additional features or by reranking

parser output. Second, the neural networks can be modified in a number of ways. For example, dropout is a technique that is known to reduce co-adaptation of features in feed-forward neural networks [12]. More work is needed to effectively update the word vectors, either by jointly learning word vectors and PP attachment parameters, or in a successive manner.

Appendix A

Algorithms

Table A.1 shows the notation used in the following algorithms.

Algorithm 1 Head-Prep-Child Cost

procedure HEADPREPCHILDCOST($\theta, \lambda, N, x_h, x_p, x_c, y$)
  $W, b, w \leftarrow \text{GetParams}(\theta)$
  $x_{p,c}, x_{h,pp} \leftarrow \text{Zeros}()$ ▷ Init PP parents and head+PP parents
  $s \leftarrow \text{Zeros}()$ ▷ Init scores
  for all $x_h^{(i)}, x_p^{(i)}, x_c^{(i)}$ **do**
    $x_{p,c}^{(i)} \leftarrow f(W[x_p^{(i)}; x_c^{(i)}] + b)$ ▷ Compose prep with child
    for all $x_h^{(i,j)}$ **do**
      $x_{h,pp}^{(i,j)} \leftarrow f(W[x_h^{(i,j)}; x_{p,c}^{(i)}])$ ▷ Compose candidate head with PP parent
      $s^{(i,j)} \leftarrow w^T x_{h,pp}^{(i,j)}$ ▷ Calculate score
    end for
  end for
  $J \leftarrow \frac{1}{N}[\sum_i s^{(i,y^{(i)})} - \arg\max_j(s^{(i,j)} + \Delta(x_h^{(i,j)}, x^{(i,y^{(i)})}))] - \frac{\lambda}{2}\sum_\theta \theta^2$
  return J ▷ The cost is J
end procedure

Algorithm 2 Head-Prep-Child Gradient

procedure HEADPREPCHILDGRAD($x_h, x_p, x_c, x_{h,pp}, x'_{h,pp}, x_{p,c}, x'_{p,c}, y, y_{\max}$)

  $\mathrm{d}w \leftarrow \sum_i [x_{h,pp}^{(i,y^{(i)})} - x_{h,pp}^{(i,y_{\max}^{(i)})}]$ ▷ Calculate gradient for scoring vector w

  for all $x_{h,pp}^{\prime(i)}$ **do** ▷ Back-propagate for all examples

    $\delta_{h,pp}^{(i),\mathrm{gold}} \leftarrow f'(x_{h,pp}^{\prime(i)}) \odot w$

    $\delta_{p,c}^{(i),\mathrm{gold}} \leftarrow (W^T \delta_{h,pp}^{(i),\mathrm{gold}})_{[n+1:2n]} \odot f'(x_{p,c}^{\prime(i)})$

  end for

  $\mathrm{d}W^{\mathrm{gold}} \leftarrow \delta_{h,pp}^{\mathrm{gold}} [x_h^y; x_{p,c}]^T + \delta_{p,c}^{\mathrm{gold}} [x_p; x_c]^T$ ▷ Calculate gold gradient

  $\mathrm{d}W^{\max} \leftarrow \delta_{h,pp}^{\max} [x_h^{y_{\max}}; x_{p,c}]^T + \delta_{p,c}^{\max} [x_p; x_c]^T$ ▷ Calculate max gradient

  $\mathrm{d}W \leftarrow \mathrm{d}W^{\mathrm{gold}} - \mathrm{d}W^{\max}$ ▷ Total gradient is the difference

  $\mathrm{d}b^{\mathrm{gold}} \leftarrow \sum_i [\delta_{h,pp}^{(i),\mathrm{gold}} + \delta_{p,c}^{(i),\mathrm{gold}}]$

  $\mathrm{d}b^{\max} \leftarrow \sum_i [\delta_{h,pp}^{(i),\max} + \delta_{p,c}^{(i),\max}]$

  $\mathrm{d}b \leftarrow \mathrm{d}b^{\mathrm{gold}} - \mathrm{d}b^{\max}$

  $\mathrm{d}\theta \leftarrow \frac{1}{N} [\mathrm{d}W; \mathrm{d}b; \mathrm{d}w] - \lambda\theta$ ▷ Normalize and regularize

  return $\mathrm{d}\theta$

end procedure

Algorithm 3 Forward Propagate Single Tree

procedure FORWARD-PROP-SINGLE-TREE(x, par, idx, W, b)
  $children \leftarrow$ GetChildren(par, idx) ▷ Get children of word at current index
  if Size$(children) = 0$ **then** ▷ Terminal node
    $tree \leftarrow$ MakeTree(x_{idx})
    return $tree$
  else
    $lastchild \leftarrow children(end)$ ▷ Start with farthest child
    $subtree \leftarrow$ Forward-Prop-Single-Tree$(x, par, lastchild, W, b, dir)$ ▷ Recurse
    $subtreeparent \leftarrow subtree.root$
    if IsNonterminal$(subtree.root)$ **then**
      $subtreeparent \leftarrow f(subtreeparent)$ ▷ Apply non-linearity to nonterminals
    end if
    $parent \leftarrow W[x_{idx}; subtreeparent] + b$ ▷ Compose word with subtree parent
    $tree \leftarrow$ MakeTree$(parent, x_{idx}, subtree)$ ▷ Create the initial tree
    $parent \leftarrow f(parent)$
    for $child \leftarrow children(end - 1) \ldots children(1)$ **do** ▷ Recurse tree
      $subtree \leftarrow$ Forward-Prop-Single-Tree$(x, par, child, W, b, dir)$
      $subtreeparent \leftarrow subtree.root$
      if IsNonterminal$(subtree.root)$ **then**
        $subtreeparent \leftarrow f(subtreeparent)$
      end if
      $parent \leftarrow W[parent; subtreeparent] + b$
      $tree \leftarrow$ MakeTree$(parent, tree, subtree)$
      $parent \leftarrow f(parent)$
    end for
    return $tree$ ▷ This is the resulting tree
  end if
end procedure

Algorithm 4 Back Propagate Single Tree

procedure BACK-PROP-SINGLE-TREE($tree, idx, W, \delta$)
  if IsTerminal($tree.idx$) **then** ▷ Terminal node
    $\mathrm{d}W \leftarrow$ Zeros()
    $\mathrm{d}b \leftarrow$ Zeros()
    return $\mathrm{d}W, \mathrm{d}b$
  end if
  $children \leftarrow tree.children(idx)$ ▷ Get children
  $child1 \leftarrow children(1)$
  $child2 \leftarrow children(2)$
  $\delta_{children} \leftarrow W^T delta \odot f'([child1; child2])$ ▷ Calculate children's δ
  $\delta_{child1} \leftarrow (\delta_{children})_{[1:n]}$
  $\delta_{child2} \leftarrow (\delta_{children})_{[n+1:2n]}$
  ▷ Recurse each of the two children
  $\mathrm{d}W_{child1}, \mathrm{d}b_{child1} \leftarrow$ Back-Prop-Single-Tree($tree, child1, W, \delta_{child1}$)
  $\mathrm{d}W_{child2}, \mathrm{d}b_{child2} \leftarrow$ Back-Prop-Single-Tree($tree, child2, W, \delta_{child2}$)
  if IsNonterminal($tree.child1$) **then** ▷ Apply non-linearity to nonterminals
    $child1 \leftarrow f(child1)$
  end if
  if IsNonterminal($tree.child2$) **then**
    $child2 \leftarrow f(child2)$
  end if
  $\mathrm{d}W \leftarrow \delta[child1; child2]^T$ ▷ Calculate gradients from current node
  $\mathrm{d}b \leftarrow \delta$
  $\mathrm{d}W \leftarrow \mathrm{d}W + \mathrm{d}W_{child1} + \mathrm{d}W_{child2}$ ▷ Add gradients from children
  $\mathrm{d}b \leftarrow \mathrm{d}b + \mathrm{d}b_{child1} + \mathrm{d}b_{child2}$
end procedure

Table A.1: Summary of Notation

θ	$\triangleq$	Vector of parameters to be learned
W	$\triangleq$	Composition matrix
b	$\triangleq$	Bias vector
w	$\triangleq$	Scoring vector
λ	$\triangleq$	Regularization constant
N	$\triangleq$	Training set size
$x_h^{(i,j)}$	$\triangleq$	The j-th candidate head in the i-th example.
$x_p^{(i)}$	$\triangleq$	The preposition in the i-th example.
$x_c^{(i)}$	$\triangleq$	The child of the preposition in the i-th example.
$y^{(i)}$	$\triangleq$	Index of the gold head in the i-th example.
$y_{\max}^{(i)}$	$\triangleq$	Index of the max head in the i-th example.
$s^{(i,j)}$	$\triangleq$	The score of the j-th candidate head in the i-th example.
$x'_{p,c}, x_{p,c}$	$\triangleq$	Composite representations of preposition p and child c before ($x'_{p,c}$) and after ($x_{p,c}$) non-linearity. Other compositions are indexed similarly.
f	$\triangleq$	Non-linearity function.
δ	$\triangleq$	Error during backpropagation.
x	$\triangleq$	Array of word vectors.
par	$\triangleq$	Indexes of word parents.
idx	$\triangleq$	Current word index in forward and backpropagation.
$tree$	$\triangleq$	Tree built during forward propagation.

Bibliography

[1] Eneko Agirre, Timothy Baldwin, and David Martinez. Improving Parsing and PP Attachment Performance with Sense Information. In *Proceedings of ACL-08: HLT*, pages 317–325, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[2] Rania Al-Sabbagh and Khaled Elghamry. A Web-Based Approach to Arabic PP Attachment. Unpublished manuscript.

[3] Yonatan Belinkov, Nizar Habash, Adam Kilgarriff, Noam Ordan, Ryan Roth, and Vt Suchomel. arTenTen: a new, vast corpus for Arabic. In Eric Atwell and Andrew Hardie, editors, *Proceedings of WACL2 Second Workshop on Arabic Corpus Linguistics*, pages 20–20, 2013.

[4] Yoshua Bengio, Jrme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *In ICML*, 2009.

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

[6] Michael Collins and James Brooks. Prepositional Phrase Attachment through a Backed-Off Model. *CoRR*, abs/cmp-lg/9506021, 1995.

[7] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[8] Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. Using latent semantic analysis to improve access to textual information. In *SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS*, pages 281–285. ACM, 1988.

[9] Spence Green. Improving Parsing Performance for Arabic PP Attachment Ambiguity. Unpublished manuscript.

[10] Nizar Habash and Owen Rambow. Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 573–580, 2005.

[11] Nizar Habash, Owen Rambow, and Ryan Roth. MADA+TOKAN: A Toolkit for Arabic Tokenization, Diacritization, Morphological Disambiguation, POS Tagging, Stemming and Lemmatization. In *Proceedings of the International Conference on Arabic Language Resources and Tools*, pages 573–580, 2005.

[12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[13] Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. Parser Showdown at the Wall Street Corral: An Empirical Investigation of Error Types in Parser Output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[14] Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, first edition, 22 April 2011.

[15] Yuval Marton, Nizar Habash, Owen Rambow, and Sarah Alkhulani. SPMRL'13 Shared Task System: The CADIM Arabic Dependency Parser. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 86–90, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

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

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

[18] Jeff Mitchell and Mirella Lapata. Composition in Distributional Models of Semantics. *Cognitive Science*, 34:1388–1429, 2010.

[19] Jaouad Mousser. A Large Coverage Verb Taxonomy for Arabic. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC-2010)*, pages 2675–2681, Valletta, Malta, May 2010. European Languages Resources Association (ELRA). ACL Anthology Identifier: L10-1529.

[20] Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. A Maximum Entropy Model for Prepositional Phrase Attachment. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 250–255, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[21] Horacio Rodríquez, David Farwell, Javi Ferreres, Manuel Bertran, Musa Alkhalifa, and M. Antonia Martí. Arabic WordNet: Semi-automatic Extensions using Bayesian Inference. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC-08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA). ACL Anthology Identifier: L08-1211.

[22] Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. Grounded Compositional Semantics for Finding and Describing Images with Sentences. *TACL*, 2014.

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

[24] Jiri Stetina and Makoto Nagao. Corpus Based PP Attachment Ambiguity Resolution with a Semantic Dictionary. In *Fifth Workshop on Very Large Corpora*, pages 66–80, 1997.

[25] Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, Uppsala, Sweden, July 2010. Association for Computational Linguistics.