Low-rank tensor approximations for compositional distributional semantics

Daniel Fried
Churchill College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: df345@cam.ac.uk

July 1, 2015

Declaration

I Daniel Fried of Churchill College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 13,886

Signed:

Date:

Abstract

This thesis explores compositional distributional semantics: methods for mapping words to feature vectors representing their meaning, and composing these word vectors to produce representations of the meanings of longer expressions such as phrases and sentences. Several compositional distributional semantic methods use matrices and their generalization, higher-order tensors, to model multi-way interactions between vectors. Unfortunately, the size of these higher-order tensors has been one obstacle to large-scale implementations of the compositional frameworks that would be able to produce representations for full-length sentences with a diverse vocabulary.

In this work, we investigate whether we can match the performance of full matrices and tensors with low-rank approximations that use a fraction of the original number of parameters. We compare low-rank matrices and tensors to full, unconstrained-rank matrices and tensors on standard semantic similarity tasks for two syntactic constructions: adjectives represented by matrices, and transitive verbs represented by third-order tensors. Using low-rank approximations allows us to reduce the number of the parameters in the models by about 40% for matrices, and by 99% (two orders of magnitude) for the third-order tensors. Despite this reduction in the size of the models, the low-rank matrices and tensors achieve performance comparable to, and occasionally surpassing, the full models. The parameters of these low-rank representations can be optimized directly using standard gradient-based methods, allowing them to be incorporated into existing machine learning models for compositional distributional semantics.

Acknowledgements

This thesis was done under the supervision of Stephen Clark and Tamara Polajnar. I've learned a great deal from them and would like to thank them for their encouragement and advice on this project, and for always being accessible. I'd also like to thank Tamara for allowing me to use and adapt her code from previous work for building the count vectors for words, adjective-noun pairs, and subject-verb-object triples.

Contents

Chapter 1

Introduction

How can a computer represent the meaning of human languages? A long tradition of work in linguistic *semantics*, the study of the meaning of language, has produced a variety of representations that have been used in computational linguistics (Liang and Potts, 2015). One *logic-based* approach, exemplified by the theories of Montague (1970; 1974) and recent computational implementations (Bos et al., 2004; Zettlemoyer and Collins, 2005), maps linguistic units, typically sentences, to logical representations. It provides an account of *compositionality*: how the meanings of words compose to form the meaning of phrases, sentences, and documents. However, while this logical approach allows applying theorem-provers and model theory to language, and provides an interpretation for words with direct logical meanings such as "all" and "or", it typically leaves the meanings of content words, such as "ran" or "car", symbolic and unspecified.

A recent wave of *statistical* approaches to semantics use algorithms and optimization to produce numeric representations for units of language, typically words. This work has usually been based on the *distributional hypothesis*, the idea that words which appear in similar contexts tend to be related in meaning (Harris, 1954; Firth, 1957; Turney and Pantel, 2010). For example, the semantic similarity of "automobile" and "car" is revealed by their usage with words such as "road", "drive", "wheels", and "fuel". Large text

corpora have made it possible to automatically produce representations of words' distributions based on this principle. Counting the co-occurrence of words with some type of context, such as nearby words within a fixed length window, produces vector representations for words, where the basis elements of these vectors correspond to possible contexts. These vectors can then be compared using distance in vector space to approximate the similarity of the words' meanings, or used as features in a machine learning system.

The distributional hypothesis has been borne out in practice: vector space representations of words' contextual distributions have been used to improve the performance of machine learning systems performing a wide variety of tasks in computational language processing (Schütze, 1998; Baker and McCallum, 1998). Such methods allow words to be compared using distance in the vector space, providing fine-grained representation of lexical relationships such as similarity, which correlate well with human similarity judgements (Mitchell and Lapata, 2010).

While there is, as of yet, no work that comprehensively combines the distributional, logical, and compositional accounts of semantics, there has been work on combining distributional and logical representations (Copestake and Herbelot, 2012; Beltagy et al., 2013), as well as the approach we follow in this thesis, *compositional distributional semantics* (CDS). CDS aims to develop methods for composing the distributional representations for words to produce representations of phrases and sentences (Mitchell and Lapata, 2008; Baroni and Zamparelli, 2010; Socher et al., 2012; Zanzotto and Dell'arciprete, 2012; Baroni et al., 2014a).

This thesis will be situated within the *Categorial framework* for CDS (Coecke et al., 2011), which uses linear algebra to compose vector-space representations for words into representations for phrases and sentences. Semantic composition in this framework is driven by the syntactic structure of the phrases and sentences, and the primary method of composition is function application. An adjective, for example, can be thought of as a function that takes a noun as input and returns a modified noun – so if we represent nouns (for example, *car*) as vectors, we could represent each adjective (such as

red) as a matrix, and multiply the matrix for *red* with the vector for *car* to produce a vector representing *red car*. Similarly, transitive verbs could be represented as tensors taking two noun phrase vectors and returning a vector representing a sentence.

A concrete implementation of the Categorial framework requires specifying the vectors, matrices, and higher-order tensors that represent words. There are a number of standard approaches for creating vectors to represent words based on their contexts. Similar contextual methods can also be used to learn vectors for phrases and sentences. However, there are several obstacles to learning the matrices and tensors used to transform the input word vectors into the output phrase vectors. These problems stem from the large number of parameters needed for the tensor models: even a model in relatively low-dimensional vector space will require hundreds or thousands of parameters in its tensors.

For example, the syntactic type $((S\backslash NP)\backslash(S\backslash NP))/((S\backslash NP)\backslash(S\backslash NP))$ (see §2.2.1 for a description of this notation) is relatively frequent, appearing 143 times in a sample of about 1900 sentences from an annotated corpus of newspaper text (Hockenmaier and Steedman, 2007). This syntactic type requires a seventh-order tensor to represent in the full form of the Categorial framework. If we use 50-dimensional vector spaces (most previous work has used at least this many dimensions to get good results on phrase similarity tasks, limited to tensors with much lower order) and store the entire tensor, we would need 50^7 or about 780 billion parameters for this single tensor, requiring terrabytes of storage space. If parameters are independent between words, so that each word has its own unique tensor, the number of parameters required is even higher.

Our primary goal in this thesis is to reduce the number of parameters required to represent these matrices and higher-order tensors, helping to lay the foundation for a full implementation of the framework. We use low-rank matrix decomposition and its higher-order generalization to tensors (Kolda and Bader, 2009) to approximate full matrices and tensors, and investigate optimization methods to set the parameters of the matrices and tensors. By

using these low-rank decompositions, we can both train the model and use it to produce representations for unseen text without ever constructing the full matrices and tensors. We are therefore able to improve substantially (by as much as two orders of magnitude for third-order tensors) on memory usage. Despite the large reduction in model complexity, we show that these low-rank approximations achieve comparable or even better performance than the full matrices and tensors on three different semantic comparison tasks, for two types of syntactic composition (adjectives and verbs) and two different methods of defining distributional vector spaces.

A paper presenting the low-rank tensor decomposition model and results for the verb disambiguation and sentence similarity tasks has been accepted to appear at the 2015 Meeting of the Association for Computational Linguistics (Fried et al., 2015).

Chapter 2

Background: vector space semantics

A foundational issue in natural language processing (NLP) is how to represent the meanings of words, phrases, sentences, and documents. Many NLP systems have successfully used the simple approach of treating words as atomic units, and counting the number of times each word occurs in a text. For example, a system designed to determine whether a movie review is positive or negative could use a large number of training reviews to learn that reviews where the word *exciting* appears multiple times have a higher probability of being positive than reviews containing *boring*.

However, this simple *bag-of-words* model has a number of shortcomings. First, it encodes no knowledge about the similarity in meanings between words: in our example, if the sentiment classifier is able to determine that *exciting* is a positive feature, it would ideally be able to treat words with similar meaning, such as *thrilling*, as positive features as well. Second, word meanings influence each other in complex, compositional ways: "despite a boring start, it turned out to be wonderful" and "despite a wonderful start, it turned out to be boring" have opposite meanings despite containing all the same words. A recent strand of research, *compositional distributional semantics*, attempts to address both of these problems: first by developing

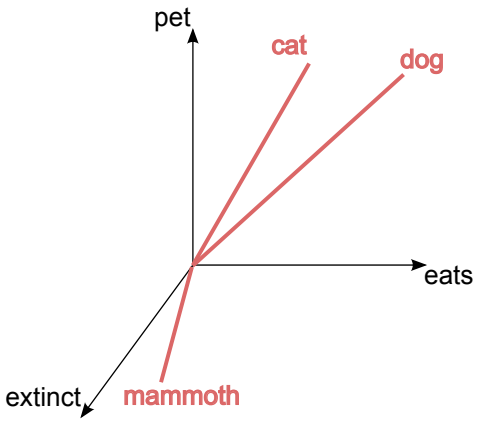

Figure 2.1: A three-dimensional subspace containing count vectors representing words. Each target word (*cat, dog, mammoth*) has a vector whose components count the number of times the word co-occurred with a given context word (*pet, extinct, eat*).

vector representations for word meanings that allow finer-grained comparisons of similarity and relationships between words, and then by composing these word representations into representations of the meanings of phrases and sentences.

2.1 Vector representations for words

The traditional methods for producing vector representations of word meanings directly encode words' contextual distributions in vectors by counting their contexts within a large corpus.[1] Each component of a word's vector corresponds to one possible context (or one dimension of a low-dimensional projection of this contextual space, as we will describe). For example, using other nearby words as context might produce vector representations like Fig. 2.1. Following Baroni et al. (2014b), we refer to these methods of producing distributional vectors for words as *count* models.

More recently, a family of methods based on machine learning have emerged under various names: distributed representations, neural language models,

[1]These counts are often re-weighted in the final vectors, as described later.

and word embedding methods. The common trait of all these *prediction* models is that they use machine learning to predict words given their contexts, or vice versa, and learn vectors in the process. The vectors produced by these prediction models have similar properties to the count vectors: words that appear in similar contexts have similar vectors. In fact, some work has shown that certain types of prediction models optimize objectives that produce the same vectors as modified count models (Levy and Goldberg, 2014b; Pennington et al., 2014)

Both the count and prediction models allow comparing the similarity of words by measuring how close their vectors are, providing empirical support for the distributional hypothesis. The most common measure is *cosine similarity* (Manning et al., 2008), the cosine of the angle between the two vectors, which for vectors $\mathbf{w_1}$ and $\mathbf{w_2}$ is given by

$$cos(\mathbf{w_1}, \mathbf{w_2}) = \frac{\mathbf{w_1} \cdot \mathbf{w_2}}{||\mathbf{w_1}||\ ||\mathbf{w_2}||} \tag{2.1}$$

where $\cdot$ is the vector dot product and $||\mathbf{w}||$ gives the magnitude of $\mathbf{w}$:

$$||\mathbf{w}|| = \sqrt{\mathbf{w} \cdot \mathbf{w}} \tag{2.2}$$

2.1.1 Count models

Count models typically use a pipeline of steps, each of which has several possible options and parameters, to produce distributional vectors for words. Curran (2004) and Turney and Pantel (2010) provide detailed descriptions and comparisons of these methods. Here, we outline the most common steps and give a brief introduction to the methods we use in our experiments.

Defining context

The first step is choosing the context used to define the words' distributions. A variety of contexts have been investigated in previous work: words sur-

rounding the target word within a fixed-length window (Lund and Burgess, 1996) or within a linguistic unit such as a sentence, identifiers for the documents a target word is contained in (Landauer and Dumais, 1997; Manning et al., 2008), words linked to the target word by syntactic dependencies (Curran, 2004; Padó and Lapata, 2007; Levy and Goldberg, 2014a), or even logical representations for sentences containing the word (Copestake and Herbelot, 2012). The intended application for the vectors often determines the context used; for measuring the semantic similarity of words, the standard choice is to use other words either within a window or a sentence as context (Turney and Pantel, 2010). Since some very common words (e.g. "the", "and", "to", "is") convey very little information about the meaning of nearby words, such words (known as *stopwords*) are usually not included as contexts.

Context weighting

Although it is possible to use vectors with raw frequency counts, these counts are typically re-weighted to assign higher values to more informative contexts for a given word. The motivation is similar to the removal of stopwords: words that occur frequently throughout a corpus convey less information about the meaning of a target word than words that may occur rarely, but have a strong association with the target word. For example, the similarity of "author" and "writer" is better revealed by a comparatively rare but strongly-associated context word such as "publish" than a word that appears as a context for many target words, such as "has". There are a wide variety of context weighting functions. We describe two here, but refer to Curran (2004) for a catalogue and discussion of other possible functions.

Let $C(w_i, c_j)$ be the number of times that the target word w_i co-occurs with context word c_j in the corpus. We use these frequencies to calculate the joint

probabilities $p(w_i, c_j)$ and marginal probabilities $p(w_i), p(c_j)$ given by:

$$p(w_i, c_j) = \frac{C(w_i, c_j)}{\sum_{k,l} C(w_k, c_l)} \tag{2.3}$$

$$p(w_i) = \frac{\sum_j C(w_i, c_j)}{\sum_{k,l} C(w_k, c_l)} \tag{2.4}$$

$$p(c_j) = \frac{\sum_i C(w_i, c_j)}{\sum_{k,l} C(w_k, c_l)} \tag{2.5}$$

A commonly-used context weighting method, *pointwise mutual information* (PMI) (Church and Hanks, 1990), is given by

$$pmi(w_i, c_j) = \log \frac{p(w_i, c_j)}{p(w_i)p(c_j)} \tag{2.6}$$

If word w_i's distribution of occurrence is independent from context c_j, $p(w_i, c_j) = p(w_i)p(c_j)$. Therefore the PMI will be 0 if the word and context are independent, higher if they are correlated (which by the distributional hypothesis implies there is some semantic information conveyed about the word by the context), and lower if they are anti-correlated. Since the degree of anti-correlation is less useful for detecting informative contexts, a variant called *positive pointwise mutual information* (PPMI) (Niwa and Nitta, 1994) replaces negative PMI values by 0:

$$ppmi(w_i, c_j) = \max(pmi(w_i, c_j), 0) \tag{2.7}$$

Curran (2004) compares PMI with a weighting function motivated by statistical hypothesis testing, the *t-test* (Manning and Schütze, 1999):

$$t - test(w_i, c_j) = \frac{p(w_i, c_j) - p(w_i)p(c_j)}{\sqrt{p(w_i)p(c_j)}} \tag{2.8}$$

Similarly to PMI and PPMI, this function compares the co-occurrence probabilities to the expected probabilities if words and contexts were independent.

Dimensionality reduction

Contextual vectors for words usually have high dimensionality, since they have one component corresponding to each possible context. For most types of context, the vectors are sparse (mostly zeros), since text resources are limited and only have some term-context co-occurrences out of all the ones that are linguistically plausible. *Dimensionality reduction* methods are often used on the term-context vectors as a method of removing noise, decreasing the number of vector components, and compensating for the fact that not all possible co-occurrences will actually be observed in a corpus (Turney and Pantel, 2010). In our experiments we will use latent semantic analysis (LSA) (Landauer et al., 1998), which formulates dimensionality reduction as matrix factorization using singular value decomposition (SVD, see §3.1.1).

First, LSA defines a matrix $\mathbf{C}$ whose rows are the contextual vectors for terms: that is, $\mathbf{C}_{i,j}$ is the weighted co-occurrence of target word w_i with the context c_j. This matrix is first decomposed into the SVD: $\mathbf{C} = \mathbf{U\Sigma V}^\top$. For a chosen value of k the best k-rank approximation $\mathbf{C_k} = \mathbf{U_k \Sigma_k V_k}^\top$ is produced as described in §3.1.1. This low-rank approximation removes noisy observations present in the matrix $\mathbf{C}$ by forcing word vectors in $\mathbf{C_k}$ to be a sum of some smaller number of basis vectors, which can be interpreted as latent senses or features of meaning.

Second, to reduce the number of components in the word vectors, rows in the matrix product $\mathbf{U_k \Sigma_k}$ are used to represent words. These vectors have k components, but still preserve the similarity relationships between word vectors in $\mathbf{C_k}$: the dot product between the ith and jth rows of $\mathbf{C_k}$ is given by the i, jth component of $\mathbf{C_k C_k}^\top$, and we have

$$
\begin{aligned}
\mathbf{C_k C_k}^\top &= (\mathbf{U_k \Sigma_k V_k}^\top)(\mathbf{U_k \Sigma_k V_k}^\top)^\top \\
&= \mathbf{U_k \Sigma_k V_k}^\top \mathbf{V_k \Sigma_k}^\top \mathbf{U_k} \\
&= (\mathbf{U_k \Sigma_k})(\mathbf{\Sigma}_k^\top \mathbf{U}_k^\top) && \text{since } \mathbf{V} \text{ is orthonormal, §3.1.1} \\
&= (\mathbf{U_k \Sigma_k})(\mathbf{U_k \Sigma_k})^\top
\end{aligned}
$$

so the dot product of rows (word vectors) in $\mathbf{C_k}$ is given by the dot product of rows in $\mathbf{U_k \Sigma_k}$.

2.1.2 Prediction models

The second type of methods for producing vector representations, *prediction* models, replace the pipeline of steps used in the traditional count vector models with a machine learning model. These methods typically attempt to predict words from their context (or vice-versa) using a model parameterized by vector representations for both words and contexts. By optimizing the components of these vectors to maximize prediction accuracy on a training corpus of text, the models learn vectors that encode word similarity, since words that appear in similar contexts will be assigned similar vectors during the optimization process.

These prediction models are often called *neural language models* because they originate in work on multi-layered neural network replacements for n-gram language models (Bengio et al., 2003; Collobert et al., 2011). When it was discovered that the word vectors learned as a side effect of these models could substitute well for traditional distributional vectors on a variety of tasks, the models were simplified to focus specifically on producing word vectors and allow training more efficiently on larger corpora (Mnih and Kavukcuoglu, 2013; Mikolov et al., 2013). Recent work has shown that prediction models often outperform count models across a variety of tasks (Baroni et al., 2014b) without careful tuning of the possible options of the count models (Levy et al., 2015).

The prediction model we will use, *skip-gram* (Mikolov et al., 2013), has recently become widely popular as a method of producing word vectors due to its scalability to large corpora and an efficient implementation, `word2vec`[2]. We briefly describe the method here, since we later modify it to learn representations for other constituents with atomic types in CCG, but refer to

[2] `http://code.google.com/p/word2vec/`

Goldberg and Levy (2014) for a more thorough exposition.

The skip-gram model defines a word's context as words contained in a fixed-length window around the word, for example, the previous 5 and next 5 words. Skip-gram predicts contexts from target words by modeling the conditional probability of contexts given targets, $p(c_i|w_j)$ for a context word c_i and target word w_j. The model has a vector for each context and target word, which we will denote by $\mathbf{c_i}$ and $\mathbf{w_j}$.[3] The conditional probability is modelled using a softmax function:

$$p(c_i|w_j) = \frac{\exp(\mathbf{c_i} \cdot \mathbf{w_j})}{\sum_k \exp(\mathbf{c_k} \cdot \mathbf{w_j})} \tag{2.9}$$

where $\exp(x)$ is the exponential function e^x, k indexes over all possible context words in the vocabulary, and $\cdot$ is the vector dot product. This equation fundamentally seeks to maximize the dot product of target word vectors with their context words. The intuition behind this is that if two target words w_i and w_j both appear frequently with context word c_k, $\mathbf{w_i}$ and $\mathbf{w_j}$ should be similar to $\mathbf{c_k}$, and therefore similar to each other. Maximizing the dot product, which is an unnormalized version of cosine similarity, is a method of enforcing this. To ensure a (conditional) probability distribution, the exponential function monotonically maps the dot product to a positive real value, and the denominator of the fraction is a normalization constant which ensures the values sum to one.

The skip-gram model attempts to set the components of the vectors for the target words $\mathbf{w}$ and context words $\mathbf{c}$ to maximize this conditional probability over the set of all target and context word pairs (w, c) in the corpus S:

$$\arg\max_{\mathbf{w},\mathbf{c}} \prod_{(w,c)\in S} p(c|w) \tag{2.10}$$

Directly optimizing this objective would be computationally intractable for a large corpus and vocabulary, so several methods are employed to make

[3]Separate vectors are maintained for targets and contexts, although the same words can appear as both targets and contexts.

it more efficient. We mention them briefly, but refer to Goldberg and Levy (2014) and Rong (2014) for detailed explanations. To deal with a large corpus (many word and context pairs), taking the logarithm of the loss to rewrite it as a sum allows training with an optimization method called stochastic gradient descent (SGD):

$$\arg\max_{\mathbf{w},\mathbf{c}} \log \prod_{(w,c)\in S} p(c|w) = \arg\max_{\mathbf{w},\mathbf{c}} \sum_{(w,c)\in S} \log p(c|w) \tag{2.11}$$

An individual word and context pair are sampled from the corpus, the gradients of the word and all context vectors (since Eq.2.10 uses all context vectors in the normalization context) with respect to this pair are calculated, and then the parameters of the vectors are updated by gradient descent (§3.2). This is then repeated for other pairs of words and context from the corpus.

For a large vocabulary, the gradient of the softmax normalization constant (the denominator of Eq. 2.9) is computationally expensive to compute (since it involves every possible context word's vector), so one of two approximation techniques is used during the training process to increase efficiency. The first, *negative sampling* (Mnih and Kavukcuoglu, 2013), generates a number of negative training examples for each actual training example by randomly replacing the context word. An unnormalized softmax is then used to score the actual training examples higher than the negative examples, so that during training only the vectors for the actual context and the negative context samples need to be updated for each term-context pair. The second method, *hierarchical softmax*, (Morin and Bengio, 2005), stores the vocabulary in a Huffman tree. This decreases the complexity of each training update from being linear to log-linear in the size of the vocabulary.

2.2 Compositional distributional semantics

Ideally, if we had distributional vectors representing longer units of text, such as phrases, sentences, and documents, the same vector similarity methods

currently applied to word vectors could be used to determine the similarities of phrases and sentences by comparing their vectors. This would be useful in a variety of applications such as classifying a sentence as having positive or negative sentiment (Socher et al., 2012), retrieving documents that match a query (Clark, 2015), or determining whether a sentence is nonsensical (Vecchi et al., 2011). However, sparsity makes it difficult to build distributional representations for expressions above the word level: while large corpora provide enough text to give a decent approximation of the possible contexts that a word could appear in, these corpora are insufficient to produce representations for longer phrases and for sentences (since the longer an expression is, the less likely we are to see it in a corpus). Indeed, there are an infinite number of possible phrases in language, so a given expression may not occur in a corpus at all.

One possible solution is to build representations for phrases and sentences in a bottom-up fashion. Motivated by the principle of compositionality, that "the meaning of an expression is a function of a meaning of its parts and of the way they are syntactically combined" (Partee, 1984), *compositional distributional semantics* (CDS) aims to develop methods for composing distributional vector representations for word meanings into vector representations for the meanings of phrases, sentences, and documents.

A simple approach to CDS is to simply treat a phrase as a mixture of the distributional vectors for its component words, for example by adding or multiplying their components element-wise (Mitchell and Lapata, 2010). This approach is limited because it ignores all syntax (the structure of language), and indeed is commutative: for example, while *dog bites man* and *man bites dog* have clearly different meanings, they will have the same composed vector in the additive or multiplicative model. Baroni et al. (2014a) and Clark (2015) describe other linguistic limitations of this model, and motivate the use of syntax in a compositional framework.

The Categorial framework (Coecke et al., 2011) describes one such syntax-driven process for composing representations for words in vector space. It relies on the fact that syntactic operations can be viewed as functions, for

example the meaning of a noun is represented as a vector, so each adjective is represented as a function in vector space. The Categorial framework chooses to use linear functions (or multi-linear, for words such as transitive verbs which take multiple syntactic arguments) for these words, which allows representing the words with matrices (or their higher-order generalizations, tensors, for words with multiple arguments). To compose the feature vectors representing words into vectors for longer expressions such as phrases and sentences, the framework uses vector-space operations such as matrix-vector multiplication and its higher-order generalization, tensor contraction.

We will focus on a variant of this framework based on a type of grammar called *Combinatory Categorial Grammar* (CCG) (Steedman, 2000) since existing annotated corpora (Hockenmaier and Steedman, 2007) and efficient statistical parsers (Curran et al., 2007) make it possible to generate CCG parses for the large corpora needed to produce distributional representations. The CCG version of the framework is also desirable because it has a simple vector composition function (Maillard et al., 2014), and some prior work on compositional semantics in vector space (Baroni and Zamparelli, 2010) can be seen as a special case of the CCG framework.

2.2.1 Combinatory categorial grammar

Categorial Grammars associate grammatical constituents (e.g. adjectives, noun phrases, or even entire sentences) with syntactic types. These grammars are highly lexicalized: each word in a sentence is assigned a syntactic type according to a *lexicon*. The grammar specifies a small number of rules describing how these syntactic types for words combine to form a derivation for the sentence.

Syntactic types are recursively defined. There are a small set of *atomic types*, such as N (noun), NP (noun phrase), and S (sentence). All other syntactic types are functions of these atomic types, or higher-order functions on functions. In *Combinatory Categorial Grammar* (CCG) (Steedman, 2000), these functions are directed: they specify which side in the sentence their

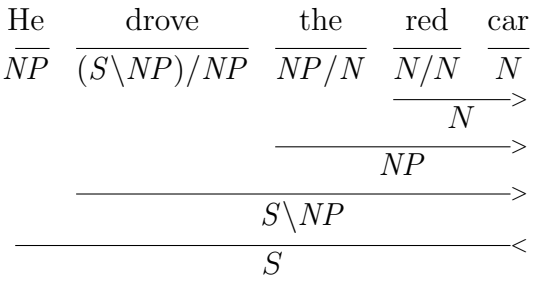

Figure 2.2: A CCG derivation for an example sentence.

argument(s) must appear on. These are denoted as *result / argument* for a function taking input on the right, and *result* $\backslash$ *argument* for a function taking input on the left. For example, a determiner such as *the* has type NP/N, which indicates it takes an argument of type N on the right, and produces an NP (Fig. 2.2). All functions take only one argument, but can be nested (this can be thought of as similar to currying a function of multiple arguments): for example, a transitive verb, such as "drove", has type $(S\backslash NP)/NP$, which means it first takes a NP to the right (the object noun phrase), and returns a function which takes an NP to the left (the subject noun phrase) to produce an S (Fig. 2.2).

In addition to function application, as shown above, CCG also defines several other rules for combining syntactic types, such as function composition and type raising. These combinatory rules are useful for dealing with certain linguistic phenomena such as non-constituent coordination (Steedman, 2000); however they will not be necessary for the simple syntactic constructions we consider in this work.

2.2.2 CCG in vector space

Maillard et al. (2014) describe a transparent method of integrating the Categorial framework with CCG. Atomic types each have their own vector space: e.g. a vector space N contains representations for nouns, and a vector space S has representations for sentences. These vector spaces need not have the same bases or even the same dimensionality. Although the framework is ag-

nostic about the semantic interpretation of these vectors, most prior work has used distributional vectors for nouns and one of several types of sentence space, such as a distributional space (Grefenstette et al., 2013) or a space measuring the plausibility of the sentence (Clark, 2013; Polajnar et al., 2014b).

CCG function types are then represented by functions in vector space. Following Coecke et al. (2011), Maillard et al. (2014) use multi-linear functions, which are well-suited to representing the combinatory rules of CCG. Consider again the case of adjectives, which are functions of type *N/N*: they take a noun, and return a modified noun. In vector-space terms, an adjective should be a function that takes a noun vector in N and returns another noun vector in N. For a n-dimensional vector space representing nouns, the function for a given adjective is then determined by a parameter matrix $\mathbf{A} \in N \otimes N$, that is $\mathbf{A} \in \mathbb{R}^{n \times n}$ (assuming again that all functions are linear).[4] Note that since each adjective is a distinct function, each has its own matrix. Function application is then given by matrix multiplication: multiplying the vector for *car* by the *red* matrix produces a vector representing *red car*. Formally, if an adjective A has parameter matrix $\mathbf{A}$ and is applied to a noun with vector $\mathbf{n}$, the resulting vector $\mathbf{An}$ has its i^{th} component given by:

$$\mathbf{An}_i = \sum_j \mathbf{A}_{i,j}\mathbf{n}_j \tag{2.12}$$

This generalizes in a straightforward way to syntactic functions of multiple arguments. For example, given a space S containing vectors for sentences, a transitive verb V, with type *(S\NP)/NP*, is a function parameterized by a third-order tensor $\mathcal{V} \in S \otimes N \otimes N$.[5] Given vectors $\mathbf{s}$ and $\mathbf{o}$ for subject and object nouns, respectively, the compositional representation for the subject, verb, and object is a vector $(\mathcal{V}\mathbf{o})\mathbf{s} \in S$, produced by *tensor contraction*, the higher-order analogue of matrix-vector multiplication. The i^{th} component of

[4] $N \otimes N$ is the vector space spanned by all possible tensor products of vectors in N, which for our purposes we can take to be $\mathbb{R}^{n \times n}$.

[5] For simplicity in this discussion, we assume that nouns and noun phrases have the same vector space.

the sentence-space vector is given by

$$((\mathcal{V}\mathbf{o})\mathbf{s})_i = \sum_{j,k} \mathcal{V}_{ljk}\mathbf{o}_k\mathbf{s}_j \tag{2.13}$$

$$= \sum_j \left[\left(\sum_k \mathcal{V}_{ljk}\mathbf{o}_k \right) \mathbf{s}_j \right] \tag{2.14}$$

Eq. 2.14 shows that this contraction can be viewed as first applying the verb tensor to the object noun, forming a matrix (by summing the matrices stacked in the tensor, weighted by the components of the object noun). This matrix is then multiplied by the subject noun to give a final vector.

As these simple examples (which will be sufficient for the syntactic constructions we examine in this work) demonstrate, CCG syntactic types map to vector spaces by replacing the slashes in function types with the tensor product operator (Maillard et al., 2014). Applying a function to an atomic type is done simply by summing the function's tensor along its final index, weighted by the components of the atomic vector.

Chapter 3

Background: factorization and optimization

In this chapter, we outline the matrix and tensor decompositions we use to reduce the number of parameters of the full compositional models, as well as the optimization methods used to fit the functions defined by matrices and tensors to distributional vectors obtained from text.

3.1 Matrix and tensor decompositions

We will use *rank factorizations* as our basic tool to reduce the number of parameters required to represent matrices and tensors. We first introduce some notation. Rank factorizations express matrices and tensors in terms of component vectors using *tensor products*. Given two column vectors (all vectors in our discussions will be column vectors) $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, the tensor product $\mathbf{u} \otimes \mathbf{v}$ is the $m \times n$ matrix with the component in the i^{th} row and j^{th} column given by

$$(\mathbf{u} \otimes \mathbf{v})_{i,j} = (\mathbf{u}\mathbf{v}^\top)_{i,j} = \mathbf{u}_i \mathbf{v}_j \tag{3.1}$$

where $\mathbf{u}_i$ is the i^{th} component of $\mathbf{u}$.[1]

The tensor product of more than two vectors is similarly defined: for example for $\mathbf{u} \in \mathbb{R}^l, \mathbf{v} \in \mathbb{R}^m, \mathbf{w} \in \mathbb{R}^n$, the tensor product $\mathbf{u} \otimes \mathbf{v} \otimes \mathbf{w}$ is a $l \times m \times n$ tensor with the component at position i, j, k given by

$$(\mathbf{u} \otimes \mathbf{v} \otimes \mathbf{w})_{i,j,k} = \mathbf{u}_i \mathbf{v}_j \mathbf{w}_k \tag{3.2}$$

In this section, we will describe the singular value decomposition (SVD) for matrices, which can be used to express a matrix in terms of a sum of weighted tensor products, and an analogue for tensors, canonical polyadic decomposition (CPD).

3.1.1 Singular value decomposition

Singular value decomposition (SVD) provides a method to factor any matrix into a form that is convenient for producing low-rank approximations. This approximation is useful, for example, to reduce the dimensionality of distributional word vectors (§2.1.1). Later, in §4.2.2, we will explicitly represent matrices in an SVD-like form during an optimization procedure to force them to have a low-rank.

A matrix $\mathbf{A} \in \mathbb{R}^{l \times n}$ with rank r has the singular value decomposition

$$\mathbf{A} = \mathbf{U}\hat{\mathbf{\Sigma}}\mathbf{V}^\top; \qquad \hat{\mathbf{\Sigma}} = \begin{pmatrix} \mathbf{\Sigma} & 0 \\ 0 & 0 \end{pmatrix} \tag{3.3}$$

where $\mathbf{U} \in \mathbb{R}^{l \times l}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices (the columns are mutually-perpendicular vectors of unit length, and so are the rows), and $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is a diagonal matrix with r positive entries in decreasing order, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. These diagonal entries $\sigma_1 \ldots \sigma_r$ are called the *singular values* of $\mathbf{A}$, and the number of them is equal to the rank r of the

[1] There is another definition of tensor product that produces a vector, but it is equivalent to this definition when the values in the vector are reshaped into a matrix.

matrix.

Given the SVD, it is simple to produce a low-rank approximation for $\mathbf{A}$ by truncating each of the three component matrices. Let $\mathbf{U_k} \in \mathbb{R}^{l \times k}$ and $\mathbf{V_k} \in \mathbb{R}^{n \times k}$ be the matrices given by taking the first k columns of $\mathbf{U}$ and $\mathbf{V}$, and let $\mathbf{\Sigma_k}$ be the diagonal matrix with the k largest singular values $\sigma_1 \ldots \sigma_k$ on the diagonal (that is, the upper-left $k \times k$ matrix of $\mathbf{\Sigma}$). Then the product

$$\mathbf{A_k} = \mathbf{U_k}\mathbf{\Sigma_k}\mathbf{V_k}^\top \tag{3.4}$$

has rank k. $\mathbf{A_k}$ is the matrix of rank k that is the least-squares best fit to $\mathbf{A}$: specifically, $\mathbf{A_k}$ minimizes the following *Frobenius norm* of the element-wise difference between it and $\mathbf{A}$:

$$||\mathbf{A} - \mathbf{A_k}||_F = \sqrt{\sum_{i=1}^{l}\sum_{j=1}^{n}\left[(\mathbf{A} - \mathbf{A_k})_{i,j}\right]^2} \tag{3.5}$$

SVD also provides a method to represent a matrix as a sum of weighted tensor products of vectors, which we will use to store low-rank matrices using fewer numerical components than the entire expanded matrix would require. Given the SVD for $\mathbf{A}$, we can rewrite

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{U}_i \otimes \mathbf{V}_i \tag{3.6}$$

where $\mathbf{U}_i \in \mathbb{R}^l$ and $\mathbf{V}_i \in \mathbb{R}^n$ are the ith columns of $\mathbf{U}$ and $\mathbf{V}$. This representation for $\mathbf{A}$ is known as the *Schmidt decomposition*.

3.1.2 Canonical polyadic decomposition

SVD for matrices computes a factorization with two properties: 1) it represents the matrix as a sum of r tensor products of vectors, and 2) these vectors can be partitioned into orthonormal sets. For higher-order tensors, *canonical polyadic decomposition* (CPD) has the first property, and *Tucker*

decomposition has the second (Kolda and Bader, 2009). In this work we use CPD, since it is a more compact representation and our primary focus is reducing the number of parameters required to store a tensor.

CPD factors a tensor into a sum of r tensor products of vectors. Given a third-order tensor $\mathcal{T} \in \mathbb{R}^{l \times m \times n}$, a CPD of $\mathcal{T}$ is:

$$\mathcal{T} = \sum_{i=1}^{r} \lambda_i \mathbf{U}_i \otimes \mathbf{V}_i \otimes \mathbf{W}_i \tag{3.7}$$

where $\lambda_1 \dots \lambda_r$ are scalar values, $\mathbf{U} \in \mathbb{R}^{l \times r}, \mathbf{V} \in \mathbb{R}^{m \times r}, \mathbf{W} \in \mathbb{R}^{n \times r}$ are parameter matrices, $\mathbf{U}_i$ gives the ith column of matrix $\mathbf{U}$, and $\otimes$ is the tensor product. The smallest r that allows the tensor to be expressed as this sum of outer products is the *rank* of the tensor (Kolda and Bader, 2009). This decomposition generalizes to tensors with order greater than 3 by simply maintaining one parameter matrix for each order of the tensor, increasing the number of vectors in each tensor product.

While the form of CPD looks similar to SVD for matrices, there are some key differences (Kolda and Bader, 2009). First, the parameter matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$ do not in general have orthonormal columns. Second, while for SVD, taking the vectors corresponding to the k-largest singular values produces the best approximation of rank k, this is not the case for tensors: for example the best approximation of rank k may share no common components with the best approximation of rank $k + 1$ for a given tensor. Third, there are no simple algorithms for calculating the tensor rank r, and the problem is in general NP-hard (Kolda and Bader, 2009). Generally, an approximate CPD for a tensor is determined numerically by fixing a rank r and optimizing the parameter matrices to minimize the squared loss. If this loss is too large, r is increased and the decomposition is recalculated.

3.2 Optimization

All of our learning algorithms involve fitting a parameterized model to some training data. This is done by defining a *loss function* that measures the error in the model's prediction of the training data as a function of the parameters. An optimization procedure can then be used to find parameters that produce a low value of the loss function.

3.2.1 Optimization methods

Consider a function L, called the loss function, that takes as input a vector of values $\mathbf{x}$ (the parameters of our model) and outputs a single scalar value $L(\mathbf{x})$, which evaluates how well the model fits our data (where a lower score is better). To use any of the optimization methods described below, L must be differentiable, which all of our loss functions will be. We aim to find a value for $\mathbf{x}$ that makes $L(\mathbf{x})$ as low as possible. Our primary tool will be the gradient, that is the vector of partial derivatives of L at $\mathbf{x}$, with i^{th} component given by

$$(\nabla L(\mathbf{x}))_i = \frac{\partial L(\mathbf{x})}{\partial \mathbf{x}_i} \tag{3.8}$$

We compared three gradient-based optimization methods: gradient descent, AdaGrad, and AdaDelta.

Gradient descent

Gradient descent (GD) (Boyd and Vandenberghe, 2004) is an iterative, greedy algorithm that attempts to update the parameters $\mathbf{x}$ slightly at each iteration to reach a lower value of the loss function. Let $\mathbf{x}^{(t)}$ be the value of the parameters at iteration t. Then the gradient of the loss evaluated at these parameter values, $\nabla L(\mathbf{x}^{(t)})$, is a vector pointing in the direction of fastest increase of L from the point $\mathbf{x}^{(t)}$. At each iteration, GD slightly adjusts the parameters in the direction *opposite* to the gradient (since we aim to minimize the function). A small constant, α, the *step-size*, determines how far to

adjust the parameters in this direction. Given the current parameter values at iteration t, $\mathbf{x}^{(t)}$, the GD update is

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla L(\mathbf{x}^{(t)}) \tag{3.9}$$

where $\nabla L(\mathbf{x}^{(t)})$ gives the gradient of f evaluated at $\mathbf{x}^{(t)}$, and α is the step size.

GD has two main flaws. First, the step size α has a large effect on the optimization procedure: if α is set too low, the algorithm will take a long time to reach the local minimum of the function; if α is set too high, GD may oscillate back and forth, repeatedly jumping over the local minimum. One solution is to use a step size that changes over the course of GD, starting out large and gradually decreasing. A second problem is that GD pays no attention to the relative importance of each parameter (component of $\mathbf{x}$) in the loss function's value: a step size that is too small for one parameter may be too large for another, causing oscillation. This can be addressed by maintaining a separate step size for each parameter.

AdaGrad

AdaGrad (Duchi et al., 2011) implements both of these solutions. It maintains a dynamically-changing step-size for each parameter, determined by that parameter's past gradient values: parameters that have had relatively large gradients will have a smaller step-size (to prevent overshooting a minimum) and parameters with small gradients will have a larger step-size (to allow faster descent). At each iteration, AdaGrad's update for the i^{th} parameter is given by

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{\alpha}{\sqrt{\sum_{k=1}^{t} \nabla L(\mathbf{x}^{(k)})^2}} \nabla L(\mathbf{x}^{(t)}) \tag{3.10}$$

where the denominator is the element-wise L2 norm of the gradient across all previous iterations (the square and square root are applied element-wise to

the gradient vector), and α is a base learning rate shared by all parameters.

AdaDelta

However, as AdaGrad training proceeds, gradients accumulate and the denominator increases, which can eventually slow the descent to a halt. The final gradient descent method we use, AdaDelta, aims to address this problem by allowing the stored gradient sum to decay over time. Suppose θ is a scalar variable whose squared value we want to track (such as a gradient component). A variable $D(\theta^2)$ stores a decaying average for θ^2, with value in iteration $t+1$ given by:

$$D(\theta^2)^{(t+1)} = \rho D(\theta^2)^{(t)} + (1-\rho)\theta^2 \tag{3.11}$$

for a decay constant $0 \leq \rho \leq 1$. Then an approximation of the root-mean-squared average for θ at iteration t is given by

$$Avg(\theta)^{(t)} = \sqrt{D(\theta^2)^{(t)} + \epsilon} \tag{3.12}$$

where ϵ is a small positive value to avoid numerical instability. Using this decaying average for the gradient, rather than the L2 norm of the full history, gives the following update value for the i^{th} parameter:

$$\Delta \mathbf{x}_i^{(t)} = -\frac{\alpha}{Avg(\nabla L(\mathbf{x})_i)^{(t)}} \tag{3.13}$$

Finally, to decrease the algorithm's sensitivity to the base learning rate α, AdaDelta multiplies this update by a decaying average of the $\Delta\mathbf{x}$ values as a corrective factor, accounting for the curvature of the loss function. We refer to Zeiler (2012) for details. The final update is given by

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{Avg(\Delta \mathbf{x})^{(t-1)}}{Avg(\nabla L(\mathbf{x}))^{(t)}} \nabla L(\mathbf{x}^{(t)}) \tag{3.14}$$

3.2.2 Preventing overfitting

The gradient descent methods described above all allow finding model parameters that achieve a local minimum of the loss function evaluated on training data. However, the training data generally includes some noise or variation, and if the model is forced to explain this noise, it will be less capable of making accurate predictions for data not used during the training procedure. This is known as *overfitting*. For this reason, it is not usually desirable to model the training data as closely as possible by strictly minimizing the loss function by itself. We compare two different methods to prevent overfitting.

Early stopping

Early stopping directly evaluates the model on a set of *validation data* not used in training, and stops the optimization process when the model's performance on the validation data is no longer decreasing. So long as gradient descent is working properly (i.e. there are no problems such as a large step-size leading to oscillation), the loss function evaluated on the training data will always decrease during optimization until a point close to a local minimum of the loss function is reached. If the validation data is similar to the training data (drawn from the same underlying distribution), the loss function evaluated on the validation data should also decrease as the model fits the underlying distribution. However, once the model begins to fit the noise in the training data (i.e. variation that is not present in the validation data), the loss on the validation data will stop decreasing, and may in fact increase. At this point, training should cease.

Previously, we defined a loss function of the model's parameters $L(\mathbf{x})$ by evaluating the model on the training data. We define a *validation loss* $L_{\text{valid}}(\mathbf{x})$ by evaluating the model on the held-out validation set. We then halt the iterative optimization procedure at the iteration t when

$$L_{\text{valid}}(\mathbf{x}^{(t+1)}) \geq f_{\text{valid}}(\mathbf{x}^{(t)}) + \epsilon \tag{3.15}$$

for a small constant value ϵ, which allows for very slight fluctuations in the score on the validation data due to noise.

Regularization

The second method for preventing overfitting, *regularization*, can be viewed as an application of Occam's Razor: the simplest model that explains the data should be preferred. Regularization adds a weighted penalty on model complexity directly into the loss function, forcing the optimization procedure to find a tradeoff between having a good fit to the training data and having a simple model. An advantage of regularization is that it does not require holding out a set of validation data, so more data can be used in the training procedure; a disadvantage is its sensitivity to the value chosen for the penalty weight.

Ridge regression (also known as weight decay), is based on the intuition that a complex model will have larger magnitude parameter values than a simple model. This assumption naturally depends on how the model is parameterized, but usually works well for the linear models we consider (Hastie et al., 2009). The sum of the squared values (the squared L2 norm) of the parameters $\mathbf{x}$, given by

$$P_{l2}(\mathbf{x}) = \sum_i (\mathbf{x}_i)^2 = \mathbf{x}^\top \mathbf{x} \tag{3.16}$$

is added as an additional penalty to the loss function. The augmented loss function used in optimization is then

$$L_{l2}(\mathbf{x}) = L(\mathbf{x}) + \lambda \; P_{l2}(\mathbf{x}) \tag{3.17}$$

where $\lambda \geq 0$ is the penalty weight that controls how strongly the parameters should be shrunk toward zero.

While ridge regression provides a method to shrink the values of all parameters, it may be desirable to train a model that does not consider some of the

parameters at all. In a linear model, this can be done using the *lasso* (least absolute shrinkage and selection operator) (Tibshirani, 1996). Similarly to ridge regression, the lasso penalizes the size of the parameters; however it uses the sum of their absolute values (the L1 norm) as a penalty:

$$P_{l1}(\mathbf{x}) = \sum_i |\mathbf{x}_i| \tag{3.18}$$

giving the augmented loss function

$$f_{l1}(\mathbf{x}) = f(\mathbf{x}) + \lambda\ P_{l1}(\mathbf{x}) \tag{3.19}$$

Whereas ridge regression shrinks all the parameters toward zero more or less equally, the geometry of the L1 norm makes the lasso more likely to find a solution to a linear model that sets some of the parameters to zero (Hastie et al., 2009). While this comes at the expense of other parameters having larger magnitude, it results in a simpler model because the variables corresponding to the zeroed-out parameters can then be excluded from the model entirely.

Chapter 4

Learning distributional models

The Categorial framework describes how to compose vector-space representations for word meanings: each syntactically-typed word has a vector, matrix, or higher-order tensor which represents its meaning. Given these mappings from words to representations in vector space, and a syntactic parse for a sentence, we can combine the word representations to produce representations for the sentence (along with all of the syntactic constituents which it contains) using linear algebraic operations, most commonly tensor contraction. However, the framework does not specify how these vectors, matrices, and tensors for words should be determined, or even what aspects of meaning they should represent.

In this chapter, we describe a procedure for producing vectors, matrices, and tensors for words using a corpus of text. First, we describe how to apply the count and prediction methods for producing word vectors (§2.1) to produce vectors for multi-word expressions with atomic CCG types, such as adjective-noun and transitive verb phrases. We then describe how to use syntactic parses for sentences in the corpus, the vector representations for words and multi-word expressions obtained previously, and standard optimization methods to produce matrices and tensors representing words with function types, such as adjectives and verbs. These methods for learning matrices and tensors are, like the Categorial framework itself, agnostic to

the type of vector representations used for the atomic types, and require only instances of the input and output vectors of the matrices and tensors. Finally, we explain modifications to these optimization methods that allow learning low-rank approximations of matrices and tensors.

4.1 Learning vectors

Choosing the vector spaces for atomic syntactic types (such as nouns and sentences) is the major decision in defining a compositional model within the Categorial framework, since the functions on atomic types are determined by their mappings from input vectors to output vectors. Although there have been alternate definitions such as a vector-space encoding of sentence plausibilities (Clark, 2013; Polajnar et al., 2014b), compositional frameworks generally use vector spaces (both for words and for phrases) defined contextually. In §2.1, we outlined commonly-used methods for producing these types of contextual vectors for words. Here, we describe how to adapt these same methods to produce contextual vectors for multi-word expressions found in a corpus.[1]

4.1.1 Defining context for multi-word expressions

The count (§2.1.1) and prediction (§2.1.2) models produce vectors for target words using context (typically words in the same sentence, or in a fixed-length window surrounding the target word), but can be adapted to produce vectors for target *expressions* consisting of multiple words by defining some context for the expressions. There is a great deal of potential flexibility in defining this context: for example, a sentence's context could be words in surrounding sentences or possible additions to the sentence that preserve the sentence's

[1]Since these methods require that the expressions occur in a corpus of text, they are not a replacement for a compositional framework that can build representations in a bottom-up fashion for previously unseen phrases and sentences. However, they can be used to produce data to train such a model, as we will describe.

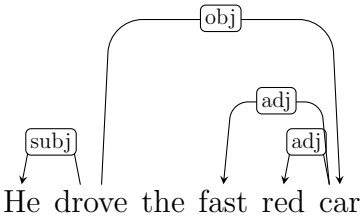

Figure 4.1: A simplified example dependency parse, used to extract syntactic constituents from sentences.

meaning (Baroni et al., 2014a). In our experiments, we are mainly concerned with comparing the performance of low-rank and full matrices and tensors, so we follow previous work (Grefenstette et al., 2013; Polajnar et al., 2015) by using an *intra-sentential* context space: words that appear within the same sentence as the multi-word expression. By using this type of contextual sentence space, we can use the same count and prediction methods we used to generate vectors for words.

As a motivating example, consider the sentence "He drove the fast red car". We will show how to extract identifiers and context for the adjective and transitive verb expressions. The syntactic *dependency parse* (Fig. 4.1) extracts the grammatical relations contained in the sentence from its CCG derivation (Clark et al., 2002; Briscoe et al., 2006; Clark and Curran, 2007). For example, *drove* has subject *he* and object *car*, and both adjectives *fast* and *red* modify the base noun *car*. Our multi-word expressions are then the adjective-noun pairs *(red, car)* and *(fast, car)*, and the subject-verb-object triple *(he, drove, car)*.

Given a sufficient amount of training data, it might be desirable to use the entire text that is spanned by a multi-word expression as an identifier for the expression, for example using *He_drove_the_fast_red_car* as the full expression for the verb with its arguments. However, we use only the words linked by the syntactic dependencies of interest which correspond to the base argument types (e.g. the subject noun and object noun for transitive verbs, and modified noun for adjectives) as an approximation, to reduce sparsity: we are much more likely to see multiple occurrences of the dependency triple *(he,*

drove, car) in a corpus than the n-gram *He_drove_the_fast_red_car*. While the training data produced by this method does not model nested applications of functions (for example recursive application of adjectives, i.e. *fast* modifies *red car* rather than *car*), given a large enough corpus with a diversity of examples it should be possible to train representative composition functions regardless.

One possible context for multi-word expressions would be all other words in the sentence. However, as the length of a multi-word expression increases, the amount of context available within the sentence decreases; in the limit, a sentence would have no available context words, since all words in the sentence would be part of the expression. Because of this trade-off between constituency and context, we use all words in the sentence other than the word with the functional type itself (e.g. the adjective in an adjective-noun pair, and the verb in a subject-verb-object triple) as context words, and additionally exclude the noun in the adjective-noun pairs for comparison with Polajnar et al. (2015).

4.1.2 Modifying word-based models

Having defined labels and contexts for the multi-word expressions, we can apply the count and prediction models to learn vectors for them with few modifications.

We produce count vectors for multi-word expressions through the pipeline of steps in 2.1.1. For example, the adjective-noun phrase *red car* has an initial vector where the i^{th} entry is given by the number of times the pair *red car* had word w_i in its context (defined as in §4.1.1) throughout the corpus. These counts are then re-weighted and the vectors reduced in dimensionality using SVD. We perform this dimensionality reduction separately for each type of constituent being learned: nouns (of CCG type N, which include single word nouns as well as adjective-noun pairs) have their own space which is distinct from the sentence space S containing subject-verb-object triples (as well as other constituents with type S).

For the prediction model, we use the Paragraph Vector distributed bag of words (PV-DBOW) method of Le and Mikolov (2014), which is an extension of the skip-gram model of Mikolov et al. (2013) (§2.1.2). The model uses the same softmax representation for conditional probability (Eq. 2.9) as the skip-gram model, but conditions the context word prediction on the multi-word expression label. For a corpus S consisting of co-occurrences of context words and multi-word expressions (c, e), this probability is modelled by

$$\prod_{(c_i,e_j)\in S} p(c_i|e_j) = \prod_{(c_i,e_j)\in S} \frac{\exp(\mathbf{c_i} \cdot \mathbf{e_j})}{\sum_k \exp(\mathbf{c_k} \cdot \mathbf{e_j})} \tag{4.1}$$

where $\mathbf{c_i}$ is the vector for context word c_i, $\mathbf{e_j}$ is the vector for the multi-word expression e_j, and k indexes over all the possible context words. As in the skip-gram model, this objective is optimized by SGD: sampling a multi-word expression and one of its content words and updating embeddings using gradient descent.

4.2 Learning matrices and tensors

The matrices and tensors of the Categorial framework can be viewed simply as parameterizations of functions: for example, a matrix $\mathbf{A} \in \mathbb{R}^{l\times n}$ defines a linear function $A : \mathbb{R}^n \to \mathbb{R}^l$ given by $A(\mathbf{x}) = \mathbf{Ax}$ (matrix-vector multiplication) and a tensor $\mathcal{T} \in \mathbb{R}^{l\times m\times n}$ defines a bi-linear function $T : (\mathbb{R}^n \times \mathbb{R}^m) \to \mathbb{R}^l$ given by $T(\mathbf{x}, \mathbf{y}) = (\mathcal{T}\mathbf{x})\mathbf{y}$ (tensor contraction). We use the methods previously outlined (§4.1) to produce vectors for atomic types, which are the inputs and outputs to these functions, then optimize the parameters of the functions to map these input vectors to the outputs as closely as possible.

For example, if we aim to learn a representation for the adjective *red*, we search in a corpus for instances where *red* is applied to a noun (e.g. *red car*, *red balloon*). Distributional vectors for the nouns *car* and *balloon*, produced using the standard methods in §2.1, are inputs to the *red* function. Distributional vectors for the pairs (*red car*, *red balloon*), produced using the

extensions in §4.1, are the corresponding outputs of the function. If we have a sufficient number of examples of vectors representing the input and output to these functions, we can perform regression (linear regression for matrices, or its generalization, multi-linear regression, for higher-order tensors) to set the matrix of parameters of the "red" function (Baroni and Zamparelli, 2010). We do this by defining a loss function which measures how well the parameters of the function predict the training outputs vectors from the input vectors, then optimizing the parameters to minimize this loss.

4.2.1 Full matrices and tensors

Loss functions

For the matrix case, suppose we have a set of N input vectors paired with output vectors, where the i^{th} input vector is $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and the i^{th} output vector is $\mathbf{z}^{(i)} \in \mathbb{R}^l$. We aim to find a matrix $\mathbf{A} \in \mathbb{R}^{l \times n}$ that minimizes the difference between the predicted output vectors $\mathbf{A}\mathbf{x}^{(i)}$ and the training output vectors $\mathbf{z}^{(i)}$.[2] Motivated by linear regression, we define a loss function as the sum of squared Euclidean distances between the predicted output and training output vectors:

$$L(\mathbf{A}) = \sum_{i=1}^{N} ||\mathbf{A}\mathbf{x}^{(i)} - \mathbf{z}^{(i)}||^2 \tag{4.2}$$

The tensor case is similar. We now have two input vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n, \mathbf{y}^{(i)} \in \mathbb{R}^m$ for each output vector $\mathbf{z}^{(i)} \in \mathbb{R}^l$, and aim to find a tensor $\mathcal{T} \in \mathbb{R}^{l \times m \times n}$ that minimizes

$$L(\mathcal{T}) = \sum_{i=1}^{N} ||(\mathcal{T}\mathbf{x}^{(i)})\mathbf{y}^{(i)} - \mathbf{z}^{(i)}||^2 \tag{4.3}$$

[2] In this discussion, we will have all vectors be column vectors.

Optimization

There are closed form solutions for minimizing these loss functions. For the matrix case, horizontally stacking the input vectors to construct a matrix $\mathbf{X} \in \mathbb{R}^{n \times N}$ and the output vectors to construct a matrix $\mathbf{Z} \in \mathbb{R}^{l \times N}$, we can reformulate Eq. 4.2 as finding $\mathbf{A}$ that minimizes the Frobenius norm (Eq. 3.5) between the output matrix $\mathbf{Z}$ and the matrix of predicted outputs $\mathbf{AX}$:

$$L(\mathbf{A}) = ||\mathbf{AX} - \mathbf{Z}||_F \tag{4.4}$$

By a generalization of the normal equations for linear regression to multiple outputs (Hastie et al., 2009), this loss is minimized by

$$\arg\min_{\mathbf{A}} L(\mathbf{A}) = \mathbf{Z}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \tag{4.5}$$

where the matrix product $(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$ is known as the Moore-Penrose pseudo-inverse of $\mathbf{X}$.[3] Because we can formulate tensor contraction as matrix multiplication using tensor products (Magnus and Neudecker, 1988), a similar technique can be used to minimize the loss in the tensor case by taking the pseudo-inverse of a matrix consisting of stacked tensor products of the input vectors $\mathbf{x}^{(i)}, \mathbf{y}^{(i)}$ and a flattened $l \times l$ identity matrix (Minka, 2000).

In our work, however, we use one of the gradient-based optimization methods (§3.2) to set $\mathbf{A}$. This is for two reasons: First, we were unable to find a closed-form solution minimizing the loss when matrices or tensors are expressed in decomposed, low-rank form, and we wanted the training procedures for the unconstrained and low-rank tensors to be as comparable as possible. Second, computational constraints make it less efficient to compute the pseudo-inverse of the training matrix $\mathbf{X}$ than to optimize the parameters using one of the iterative methods (§3.2). In the tensor case, the closed-form solution requires a prohibitive amount of memory for any reasonably large dimensionality of the input and output spaces, since we must first compute tensor products

[3] A closed form solution also exists if ridge regression is used on the parameters of $\mathbf{A}$, (Hastie et al., 2009)

resulting in a matrix with dimensions $N \times (l^2mn)$.

4.2.2 Low-rank matrices and tensors

The matrix and tensor decompositions described in §3.1 typically take a full matrix or tensor and decompose it into a vector-based form. Rather than learn a full matrix or tensor, as in 4.2.1, and then decompose it to try to reduce the number of parameters, we fix a maximal value for the rank and learn the parameters directly. This approach, which follows the work of Lei et al. (2014), allows us to increase computational and memory efficiency by never having to store the full matrix or tensor either during training or evaluation of the compositional model.

Matrix loss

Recall the matrix SVD, which gives a rank decomposition for a matrix $\mathbf{A} \in \mathbb{R}^{l \times n}$ (Eq. 3.6). To simplify the optimization, we will not maintain a separate σ_i for each vector pair (since it is a scalar value, it can be absorbed into either of the $\mathbf{U}_i$ or $\mathbf{V}_i$ vectors). This gives the form

$$\mathbf{A} = \sum_{i=1}^{r} \mathbf{U}_i \otimes \mathbf{V}_i = \mathbf{U}\mathbf{V}^\top \tag{4.6}$$

where $\mathbf{U}_i \in \mathbb{R}^l$ and $\mathbf{V}_i \in \mathbb{R}^n$ give the i^{th} column of the parameter matrices $\mathbf{U} \in \mathbb{R}^{l \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$. We also do not enforce that $\mathbf{U}$ and $\mathbf{V}$ are orthonormal since this is not necessary to reduce the number of parameters of the model; the existence of the SVD for an arbitrary matrix $\mathbf{A}$ implies that any matrix can be represented in this more general form.[4]

If the value of r is sufficiently small compared to l and n (i.e. if $rl + rn < ln$), this will reduce the number of parameters required to store the matrix.

[4]The matrix represented in this form may have rank less than r, since we do not require that the component vectors are orthonormal; however the rank will be at most r.

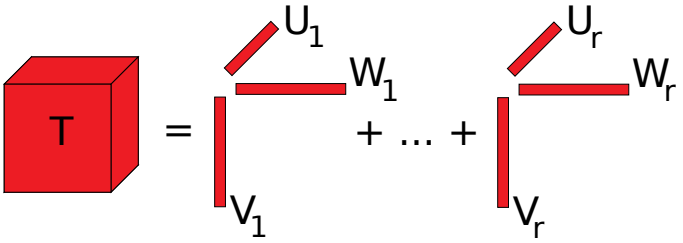

Figure 4.2: The CPD for a tensor $\mathcal{T}$ decomposes the tensor into the sum of r tensor products of vector triples.

We can then substitute this representation for the matrix into the matrix loss function (Eq. 4.2), giving

$$L(\mathbf{U}, \mathbf{V}) = \sum_{i=1}^{N} ||\mathbf{U}\mathbf{V}^\top \mathbf{x}^{(i)} - \mathbf{z}^{(i)}||^2 \tag{4.7}$$

Tensor loss

Similarly, we adapt the CPD for a tensor $\mathcal{T} \in \mathbb{R}^{l \times m \times n}$ (Eq. 3.7) by absorbing the scaling coefficients into the component vectors, giving

$$\mathcal{T} = \sum_{i=1}^{r} \mathbf{U}_i \otimes \mathbf{V}_i \otimes \mathbf{W}_i \tag{4.8}$$

for parameter matrices $\mathbf{U} \in \mathbb{R}^{l \times r}$, $\mathbf{V} \in \mathbb{R}^{m \times r}$, and $\mathbf{W} \in \mathbb{R}^{n \times r}$, with $\mathbf{U}_i$ giving the i^{th} column of $\mathbf{U}$ (Fig. 4.2).

If r is sufficiently small ($rl + rm + rn < lmn$), this decomposition will require fewer parameters than the full tensor.

The tensor contraction with vectors $\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m$ is then given by

$$(\mathcal{T}\mathbf{x})\mathbf{y} = \sum_{i=1}^{r} \mathbf{U}_i(\mathbf{V}_i\mathbf{y})(\mathbf{W}_i\mathbf{x}) \tag{4.9}$$

$$= \mathbf{U}[(\mathbf{V}^\top\mathbf{y}) \odot (\mathbf{W}^\top\mathbf{x})] \tag{4.10}$$

where $\odot$ is the element-wise vector product (Fig. 4.3)

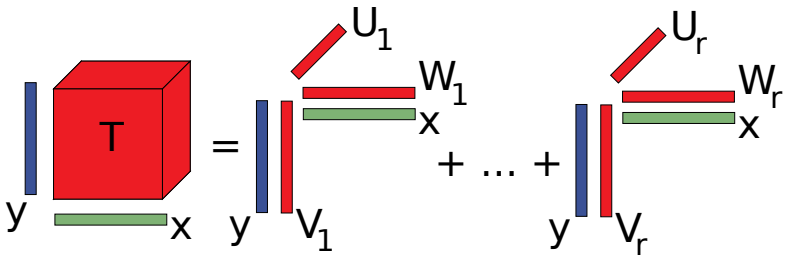

Figure 4.3: The contraction of a tensor in CPD form with two input vectors is given by a sum of one set of the vector components, weighted by dot products between the input vectors and the other components.

Substituting this into the tensor loss function (Eq. 4.3) gives

$$L(\mathbf{U}, \mathbf{V}, \mathbf{W}) = \sum_{i=1}^{N} ||\mathbf{U}[(\mathbf{V}^\top \mathbf{y}^{(i)}) \odot (\mathbf{W}^\top \mathbf{x}^{(i)})] - \mathbf{z}^{(i)}||^2 \tag{4.11}$$

Optimization

We compare two different methods of optimizing the parameters for the decomposed matrices and tensors. The first, *joint optimization*, optimizes all parameters (e.g. $\mathbf{U}, \mathbf{V}, \mathbf{W}$ for the tensor) simultaneously: at each iteration, the gradient for all parameters is computed, and each parameter is updated using the updated equation of the optimization method (e.g. AdaGrad or AdaDelta). The second is an *alternating optimization* method similar to previous tensor decomposition algorithms (Harshman, 1970; Carroll et al., 1980; Lei et al., 2014). We optimize one set of parameters (for example $\mathbf{U}$) for a fixed number of iterations while holding the other two fixed. This is then repeated, rotating through the other parameter matrices in turn until convergence or a fixed-number of iterations is reached (or until the loss on a validation increases if early stopping is being used, §3.2.2). Alternating optimization often converges more quickly and can be better at avoiding local minima than joint optimization, although this is difficult to predict and depends on the loss function and the way the parameters are partitioned (Bezdek and Hathaway, 2002).

Since the loss function is non-convex when a tensor in decomposed form is used, there may be multiple local minima of the loss function. The optimization methods we use are only capable of finding a local minimum of the function, specifically the one that will be reached by steepest descent (or alternating descent) from the initial values of the parameters. For this reason, initialization of the low-rank tensors can affect the final parameter values. In our experiments, we initialize the parameter values randomly by sampling from a Gaussian distribution centered at zero with small variance, but we suspect that better initialization schemes would produce better results.

Chapter 5

Experiments

We compare the performance of low-rank matrices and tensors to full, unconstrained rank matrices and tensors for two syntactic constructions: adjectives modifying nouns (such as "red car"), and transitive verb phrases (verbs with a subject and object, such as "dogs bite men"). This comparison is performed for both the count and prediction types of vectors (§§2.1,4.1). We evaluate our trained compositional models on three standard evaluation tasks, which require the model to rank the similarity of pairs of phrases, and compare the model's scores to similarity scores assigned by human evaluators.

5.1 Training

We train the compositional models for adjectives and transitive verbs in three steps: 1) extracting adjective-noun (AN) pairs and subject-verb-object (SVO) triples from a text corpus, 2) producing distributional vectors for the nouns, the AN pairs, and the SVO triples, and 3) learning parameters of the adjective and verb functions using these vectors.

5.1.1 Corpus data

We use the context extraction methods and dataset of Polajnar et al. (2015). Their corpus consists of an October 2013 download of Wikipedia, from which they extract AN and SVO pairs and contexts. The corpus is tokenized using the Stanford CoreNLP (Manning et al., 2014). Words are then mapped to lemmas (root forms, such as *walk* for *walking*) using the Morpha lemmatizer (Minnen et al., 2001), allowing syntactic dependency parses to be assigned to sentences using the C&C parser (Curran et al., 2007). The dependency parse structures are finally used to extract AN pairs and SVO triples from all the sentences.

The AN pairs and SVO triples are filtered to a set containing 400 distinct adjectives and 345 distinct verbs. These included the adjectives and verbs from the test datasets as well as some additional high-frequency adjectives and verbs included to produce more representative sentence spaces. For each adjective (or verb), they selected up to 600 pairs (or triples for verbs) that occurred more than once and contained a noun which occurred at least 100 times (both nouns in the SVO triples are required to have occurred at least this frequently), to allow sufficient context to produce a distributional representation for the triple. This resulted in approximately 178,000 AN pairs and 150,000 SVO triples overall.

5.1.2 Producing vectors for training

We use the Wikipedia corpus to produce vectors for both the nouns (inputs to the adjective and verb functions) and the AN pairs and SVO triples (outputs of the functions) using either a count model or a prediction model. We follow the steps outlined in §2.1 and §4.1, but describe details of the implementation here. All vectors used in our experiments, for both the count and prediction models, are 100-dimensional.

Count model

Words contained within the same sentence as the target word were used as context. Since some possible context words convey little information about the meaning of a target word, we exclude a set of *stopwords* containing determiners (e.g. "the", "a"), pronouns (e.g. "he", "they"), common prepositions (e.g. "of", "at"), and other uninformative words (e.g. "is", "could").[1] To reduce the impact of sparsity and rare words, only the top 10,000 words from the Wikipedia corpus (§5.1.1) by frequency (after removing stopwords) are used as context.

We use the t-test as a reweighting function in our experiments, following Polajnar and Clark (2014)'s finding that t-test weighting had the highest performance out of several weighting functions, including PPMI, when producing distributional vectors for the Mitchell and Lapata (2010) adjective-noun comparison task. Following Polajnar and Clark (2014), we further reduce noise in each word or phrase's vector using *context selection*: keeping only the 70 largest weighted values in the context vector (this value comes from previous work, and was not tuned for our task). Finally, we use SVD to reduce the vectors to 100-dimensions, performing SVD on the nouns and AN pairs separately from the SVO triples.

Prediction model

We use a modification of the `word2vec` software written by the authors of the skip-gram (Mikolov et al., 2013) and PV-DBOW models (Le and Mikolov, 2014).[2] This implementation combines the skip-gram and PV-DBOW models by taking a corpus of labelled sequences of words and predicting 1) words in each sequence from other words in the sequence (the skip-gram model, §2.1.2) and 2) words in each sequence from the sequence's label (the PV-

[1] We use Martin Porter's list of 127 stopwords, available in the NLTK Python library and at `http://snowball.tartarus.org/algorithms/english/stop.txt`.

[2] `https://groups.google.com/d/msg/word2vec-toolkit/Q49FIrNOQRo/J6KG8mUj45sJ`

DBOW model, §4.1.2). In the process, it produces vectors for the labels and for the words in the sequences.[3]

We use sentences from the Wikipedia corpus to produce these labelled sequences of words. A given sentence is used to generate one labelled sequence for each targeted AN pair or SVO triple found in the dependency parse for the sentence. The AN pair or SVO triple is used as the label, and the context words of the pair or triple (which are all words in the sentence, with the adjective removed in the case of AN pairs or the verb removed in the case of SVO triples, for the reasons outlined in §4.1.1) are used as the text sequence.[4]

Since most sentences in the Wikipedia corpus do not contain one of the targeted AN pairs or SVO triples, and discarding these sentences would waste a large amount of data that could be used to train the skip-gram model, we also include each sentence in its entirety with a special `*BLANK*` label which is later discarded. For example, the sentence "He drove the fast red car" has two AN pairs (`fast_car` and `red_car`) and one SVO triple (`he_drove_car`), so it would produce the labelled sequences in Fig. 5.1.

label	sequence
`*BLANK*`	he drove the fast red car
`red_car`	he drove the fast car
`fast_car`	he drove the red car
`he_drove_car`	he the fast red car

Figure 5.1: Sample training data used to produce word and constituent vectors for the sentence "He drove the fast red car".

The model then iterates through the corpus performing SGD: for each la-

[3]We also experimented with implementations of PV-DBOW by itself, `https://github.com/piskvorky/gensim/` and `https://bitbucket.org/yoavgo/word2vecf` and found that the compositional models trained on the resulting vectors achieved much lower performance on the evaluation tasks, highlighting the importance of having good base representations for words in the compositional model.

[4]Although the sequences with adjectives and verbs removed do not generally form grammatically correct sentences, this is acceptable because the skip-gram and PV-DBOW models sample individual words from the sequences and do not perform any concatenation of words.

belled sequence predicts the words in the sequence from the label, and the words in the sequence from other words in the sequence, updating the vector representations for the labels and the words to improve performance on these two prediction tasks (§2.1.2,§4.1.2).

We use the hierarchical sampling training procedure, perform 20 iterations through the training data, and do not remove any words from the context vocabulary.

5.1.3 Training methods

In both the low-rank and full-rank matrix and tensor learning, we use mini-batch AdaDelta optimization with batch-size set to 100 training instances. We use the values $\epsilon = 1e-6$ and $\rho = 0.95$ which produced the best results in the original AdaDelta description (Zeiler, 2012). We also tried using AdaGrad, but found that, without much tuning of its learning rate, it required longer training time to converge on a local minimum of the loss function than AdaDelta.

We compare seven different maximal ranks for the low-rank matrices and tensors, $R = 1$, 5, 10, 20, 30, 40 and 50. We found that alternating optimization generally converged on lower local minima of the loss functions than the joint optimization, so we use alternating optimization for all results reported here. In the full matrix and tensor experiments, we perform a maximum of 500 iterations of AdaDelta, which we found to be more than sufficient for the optimization to converge in all instances. In the low-rank alternating optimization experiments, we perform 10 AdaDelta iterations on each set of parameters during its alternation, and set a maximum of 50 outer batch iterations so that the total number of iterations per parameter set is the same as in the full tensor experiments.

We tried all of the regularization methods described in §3.2. We found that using an L2 regularization penalty on the parameters of the low-rank matrices and tensors produced very high loss scores even on the training sets,

likely because the interacting nature of the component parameters in the final tensor would require careful tuning of the regularization constant. L1 regularization produced good results on artificial datasets, when reconstructing generated tensors with low-rank (and actually allowed recovering the rank of the original artificially-generated tensor by thresholding the magnitudes of the component vectors), but overly constrained the optimization on vectors produced from the corpora. Instead, we used early stopping for both the low-rank and full matrices and tensors, using validation sets consisting of 10% of the available AN pairs for each adjective and SVO triples for each verb and an ϵ threshold of 1×10^{-6}.

5.2 Tasks

We compare the performance of the low-rank matrices and tensors against unconstrained-rank matrices and tensors on three tasks. All tasks require the model to produce a numeric score evaluating the semantic similarity of a pair of short phrases. These pairs have also been assigned scores by human evaluators, providing a ranking of most similar pairs to least similar pairs. The ranking of all pairs defined by the model's similarity scores is compared to the ranking given by the human similarity judgements using Spearman's rank correlation, ρ. The first dataset (ML10) evaluates the model's representation of adjective-noun composition; the second two (GS11 and KS14) test transitive verbs.

Adjective-Noun Phrase Similarity (ML10)

This dataset (Mitchell and Lapata, 2010) consists of 72 AN phrases, which are arranged into 36 pairs of phrases. The model must rank these pairs of phrases in terms of similarity, and these scores are compared to those produced by human annotators (on a 1-7 similarity scale). For example, the pair *important part – significant role* has a high similarity score, while *small house – old person* has a low score. The average human inter-annotator correlation (the

correlation of each annotator's scores with the scores produced by the other annotators), which provides an upper bound for how well a compositional model could do on the task, is 0.52.

We use our trained compositional model to produce a similarity score for each AN pair by multiplying the adjectives for the matrices with the vectors for the nouns to produce a vector for each phrase, for example multiplying the matrix for *important* and the vector for *part* to give a vector $\overrightarrow{\text{important part}}$. We then use the cosine similarity of the phrase vectors as the similarity score for the phrase pairs:

$$cos(\overrightarrow{\text{important part}}, \overrightarrow{\text{significant role}})$$

Verb Disambiguation (GS11)

This task (Grefenstette and Sadrzadeh, 2011) involves distinguishing between senses of an ambiguous verb, given subject and object nouns as context. For example, the verb *write* has the senses *publish* and *spell*, which are distinguished by the contexts *author write book* and *child write name*. The dataset consists of 200 phrase pairs, where the two phrases in each pair have the same subject and object but differ in the verb. Each of these pairs was ranked by human evaluators so that properly disambiguated pairs (e.g. *author write book – author publish book*) have higher similarity scores than improperly disambiguated pairs (e.g. *author write book – author spell book*). The average inter-annotator agreement on this task is 0.62.

We produce similarity scores for the phrase pairs in a similar manner to adjectives: first contracting the tensor for the verb with the object noun's vector, and then multiplying the resulting matrix with the subject noun. This gives a vector for the phrase, for example $\overrightarrow{\text{author write book}}$. Phrase similarity is again calculated using the cosine similarity of the phrase vectors.

Transitive Sentence Similarity (KS14)

This dataset (Kartsaklis and Sadrzadeh, 2014) consists of 72 SVO phrases arranged into 108 phrase pairs. As in GS11, each pair has a gold standard semantic similarity score assigned by human evaluators. In this dataset, however, the phrases in each pair have no lexical overlap: neither subjects, objects, nor verbs are shared. For example, the pair *medication achieve result – drug produce effect* has a high similarity rating, while *author write book – delegate buy land* has a low rating. Similarity scores are produced in the same manner as for GS11. The average inter-annotator agreement is 0.66.

5.3 Results

We compare low-rank and full matrices and tensors on the tasks presented above. We also compare these models to the additive and multiplicative composition methods of Mitchell and Lapata (2008), which have traditionally produced some of the highest results on these datasets. These methods produce a vector for a phrase using vectors for each of the component words, simply by performing element-wise addition or multiplication of the vectors. This requires having vectors for the functional words such as adjectives and verbs – which are not required to train our compositional models – so we produce these vectors at the same time as we produce noun and phrase vectors. For the count model, we use the same context and weighting methods for adjectives and verbs as for nouns, and apply SVD simultaneously to the nouns, adjectives, and verbs so that the vector components correspond and the vectors can be added. The prediction model automatically learns vector representations for all words in the corpus including the adjectives and verbs.

5.3.1 Adjective matrix results

Table 5.1 displays correlations between the systems' scores and human AN similarity judgements on the adjective-noun phrase similarity task (ML10),

	ML10		# of adj.
	CV	PV	params.
Add.	0.39	**0.50**	–
Mult.	0.27	0.36	–
R=1	0.28	0.35	200
R=5	0.35	0.44	1K
R=10	0.35	0.43	2K
R=20	0.37	0.45	4K
R=30	0.39	0.48	6K
R=40	0.37	0.45	8K
R=50	0.34	0.47	10K
Full	**0.40**	**0.50**	10K

Table 5.1: Matrix performance on the adjective-noun comparison task (ML10), and the number of parameters needed to represent each adjective's matrix. We show the highest matrix result for each task and vector set in bold (and also bold the Add. or Mult. methods when they outperform the matrix methods). Human inter-annotator agreement (an upper bound for the task) is 0.52.

for both the count (CV) and prediction vectors (PV).

We observe that, unsurprisingly, the rank-1 matrices have the lowest performance of all the matrix models, but are comparable to the element-wise multiplication baseline (Mult.) for both the CV and PV sets of vectors. Performance generally increases with the maximum rank R, but begins to decrease after $R = 30$ for both sets of vectors. The scores for $R = 30$ are still slightly lower than the scores of the full matrices (ρ values of 0.39 vs 0.40 for CV, and 0.45 vs 0.50 for PV), but the number of parameters required to store each verb has also been reduced by 40% (from 10,000 to 6,000).

While our main objective in these experiments was to compare the full and low-rank tensors, it is worth noting that the full matrices slightly outperform the additive baseline (Add.) on count vectors, and tie its scores for PV, which has not been achieved with many past compositional models (Blacoe and Lapata, 2012; Hermann and Blunsom, 2013). The highest score is 0.50, achieved by the Add and full matrix model using PV vectors, which comes close to the average human inter-annotator agreement of 0.52. We also see

	GS11		KS14		# tensor
	CV	PV	CV	PV	params.
Add.	0.13	0.14	**0.55**	**0.56**	–
Mult.	0.13	0.14	0.09	0.27	–
R=1	0.10	0.05	0.18	0.30	300
R=5	0.26	0.30	0.28	0.40	1.5K
R=10	0.29	0.32	0.26	0.45	3K
R=20	0.31	0.34	0.39	0.44	6K
R=30	0.28	0.33	0.32	0.46	9K
R=40	0.32	0.30	0.31	**0.52**	12K
R=50	**0.34**	0.32	**0.42**	0.51	15K
Full	0.29	**0.36**	0.41	**0.52**	1M

Table 5.2: Tensor performance on the verb disambiguation (GS11) and sentence similarity (KS14) tasks, and the number of parameters needed to represent each verb's tensor. We show the highest tensor result for each vector set in bold (and also bold Add. and Mult. if they outperform the tensor methods). Human inter-annotator agreement (an upper bound for the tasks) is 0.62 for GS11 and 0.66 for KS14.

that the prediction vectors outperform the count vectors in every instance, which is consistent with results on other tasks and datasets (Baroni et al., 2014b; Milajevs et al., 2014).

5.3.2 Verb tensor results

Table 5.2 displays results on the verb disambiguation (GS11) and sentence similarity (KS14) tasks. As is consistent with prior work, the tensor-based models are surpassed by vector addition on the KS14 dataset (Milajevs et al., 2014), but perform better than both addition and multiplication on the GS11 dataset.[5]

[5]The results in this table are not directly comparable with Milajevs et al. (2014), who differ from previous GS11 and KS14 evaluation methods by comparing against *averaged* annotator scores. Comparing against averaged annotator scores, our best result on GS11 is **0.47** for the full-rank tensor with PV vectors, and our best non-addition result on KS14 is **0.68** for the K=40 tensor with PV vectors (the best result is addition with PV vectors, which achieves **0.71**). These results exceed the tensor model results of Milajevs et al. (2014).

Similarly to the adjective matrix results, the rank-1 tensor has lowest performance for both tasks and vector sets and performance generally increases as we increase the maximal rank R. The full tensor achieves the best, or tied for the best, performance on both tasks when using the prediction vectors. However, for the count vectors (CV), low-rank tensors occasionally surpass the performance of the full-rank tensor. The reduction in the number of parameters is even more pronounced for the low-rank verb tensors than for the low-rank adjective matrices: the best performing low-rank tensors, $R = 40$ and $R = 50$, require nearly two orders of magnitude fewer parameters than the full tensors (12-15 thousand for these low-rank tensors; 1 million for the full tensor).

On GS11, the CV and PV vectors have varying but mostly comparable performance, with PV achieving higher performance on 5 out of 8 models. However, on KS14, the PV vectors have better performance than the CV vectors for every model, by at least 0.05 points, which is consistent with prior work comparing count and prediction vectors on these datasets (Milajevs et al., 2014). Levy et al. (2015) indicate that tuning hyperparameters of the count-based vectors may be able to produce comparable performance. Regardless, we show that the low-rank tensors are able to achieve performance comparable to the full rank for both types of vectors.

We also informally compared the efficiency of the low-rank and full tensor models, and found that the low-rank tensor models are at least twice as fast to train as the full tensors: on a single core, training a rank-1 tensor takes about 5 seconds for each verb on average, ranks 5-50 each take between 1 and 2 minutes, and the full tensors each take about 4 minutes. There is a large gap between rank-1 training time and higher-rank training time because we compute batched rank-1 tensor updates as optimized matrix multiplications.

Chapter 6

Related work

Early work on combining distributional representations for words relied on adding the words' vectors (Landauer and Dumais, 1997; Foltz et al., 1998; Kintsch, 2001). To evaluate this additive method and its variations for a variety of vector spaces, Mitchell and Lapata (2010) defined the adjective-noun similarity task which we use for evaluation in this thesis. Since in vector addition the magnitudes of the vectors influence the direction of the resulting vector, they also introduce element-wise vector multiplication as an alternative composition method.

These *vector mixture* models are widely applicable because they require no training other than the production of word vectors from a text corpus. However, as mentioned previously, they they do not account for the syntactic properties of composition needed to represent the semantics of more complex expressions (Baroni et al., 2014a). While these models seem ill-suited for capturing complex linguistic phenomena, they have traditionally produced some of the best results on similarity tasks on short phrases that do not involve complex syntax (Erk and Padó, 2008; Vecchi et al., 2011; Boleda et al., 2012), which we also found to be true in most of our experiments.

A variety of CDS models based on syntax emerged to deal with these concerns about the vector mixture models. Socher and co-authors (2011; 2012; 2013)

define a succession of compositional models using recursive neural networks. As in the Categorial framework, composition is applied syntactically, mirroring the constituency parse of the sentence; however, they use non-linear composition functions. The best-performing models use tensors to allow interaction between the components of the vectors being composed. A feature of this work is that it has been implemented for full-length sentences, allowing its application to a variety of tasks including syntactic parsing and sentiment classification. Our work on parameter reduction aims to help make a similar implementation of the Categorial framework feasable.

A number of pieces of work which use linear composition functions, including this thesis, can be viewed as implementations of parts of the Categorial framework. Baroni and Zamparelli (2010) treat adjectives as functions represented by matrices, and introduce the linear regression method for setting the adjective function parameters which we extend to learn low-rank approximations of adjectives. The tensor model for the subject-verb-object construction is first explored by Grefenstette et al. (2013), although they use a multi-step regression algorithm while we optimize the tensor parameters directly, following Polajnar et al. (2015). Our performance scores on the GS11 task using the single step regression are about the same as their multi-step regression results, although not directly comparable given differences in the vectors we use to train the tensors.

Other work has also sought to reduce the number of parameters of the tensors in a compositional semantic framework. Polajnar et al. (2014a) introduce several alternative ways of reducing the number of tensor parameters for a verb by using matrices. The best performing method uses two matrices, one representing the subject-verb interactions and the other the verb-object interactions. Some interaction between the subject and the object is reintroduced through a non-linear softmax layer. A similar method is presented in Paperno et al. (2014).

Tensor decompositions in various forms have been used in a wide variety of applications (Kolda and Bader, 2009). The most relevant to our work are low-rank tensor-based models for dependency parsing (Lei et al., 2014) and

semantic role labeling (Lei et al., 2015) tasks. These models also use tensors stored in a low-rank CPD form, and optimize the component parameters of the tensor directly to overcome large feature spaces, but are not comparable to our evaluation results since the models are intended for different tasks.

Our decomposition approach is also closely related to the notion of *entanglement* explored by Blacoe et al. (2013) and Kartsaklis and Sadrzadeh (2014): a matrix or tensor with high rank has high entanglement, and allows more information to be passed from the arguments to the output of the function it represents. Kartsaklis and Sadrzadeh (2014) find that matrices for transitive verbs, constructed by summing outer products of distributional vectors for their subject and objects, have surprisingly low entanglement. Our finding that low-rank tensors match or surpass the performance of unconstrained rank tensors suggests that verb tensors trained to output a distrbutional representation also tend to have relatively low entanglement.

Chapter 7

Conclusions

7.1 Summary

We have used low-rank matrix and tensor decompositions to reduce the number of parameters required by a compositional vector space model of linguistic meaning. We train these low-rank models efficiently using gradient-based methods, allowing approximations of multi-linear functions of unconstrained rank without ever producing the full matrices and tensors used to represent these functions. This allows us to improve on both runtime and memory usage. Our best performing low-rank models reduce the memory requirements of the full models by 40% for matrices and 99% for tensors, and are at least twice as fast to train. Despite these increases in efficiency, we find that the low-rank models achieve comparable or better performance than the full models on several standard compositional similarity tasks.

7.2 Future Work

In this work, we compared the performance of various fixed rank decompositions, where all adjectives and verbs being modelled have the same maximal rank. We saw that increasing the rank of all matrices and tensors generally

increased performance on the semantic evaluation tasks up to a point, after which performance stabilized or decreased. With this in mind, it would be natural to use set maximal ranks independently for different words in the vocabulary to optimize the trade-off between accuracy and efficiency. One possible method is to attempt to determine these ranks automatically using a optimization procedure such as nuclear norm minimization (Jaggi et al., 2010) or structured sparsity (Huang et al., 2011), which would use regularization to force some vector components of a larger-rank representation toward zero. Another possibility is to use linguistic information such as measuring the entanglement of an adjective or verb in a corpus (Kartsaklis and Sadrzadeh, 2014). Indeed, in some preliminary experiments we found a slight but statistically significant correlation between the approximate ranks of a verb's full tensor learned distributionally and the number of word senses defined for the verb in the WordNet database (Miller, 1995).

The larger goal of this work is to help enable a full implementation of the Categorial framework, capable of representing the compositional meaning of real sentences. In this work we deal only with relatively low-order types and one form of syntactic composition, function application. A concrete implementation will need to model many more grammatical types, such as adverbs and relative clauses, which have corresponding tensors of higher order. The decomposition methods we use here can approximate these tensors, but there are a few challenges in adapting the training algorithms to learn the parameters of such a model.

First, if distributional spaces are used to train this model, training data will be difficult to acquire. This is for reasons of sparsity: these more complex grammatical types will tend to appear in longer phrases and sentences, which will occur fewer times in a text corpus. One possible solution is an extension of the tensor skip-gram model of Maillard and Clark (2015) to constructions beyond adjectives and nouns. The context of multi-word expressions would be predicted from a composed representation of the expressions, and back-propagation used to update the parameters of all contained words to maximize the accuracy of this prediction. All sentences that a given word

(including those with a high-order syntactic type) occurs in could then be used to train the representation for the word.

Second, these higher-order types are often used in syntactic constructions beyond function application, such as function composition, which requires contracting two higher-order tensors (Coecke et al., 2011; Maillard et al., 2014). While tensor rank is not preserved during tensor contraction (e.g. a third-order tensor of rank m, contracted along one index with another third-order tensor of rank n, produces a tensor with maximal rank $m \times n$), it would be possible to store the individual tensors in their low-rank form and update the parameters using back-propagation. Such a model would be similar to the recursive neural networks of Socher et al. (2013) which parallel a sentence's syntactic composition, but would use linear functions parameterized independently for each word in the lexicon. This type of wide-coverage implementation of the Categorial framework would allow it to be applied to the same sort of tasks (e.g. parsing and sentiment classification) as the recursive neural networks, and allow a direct comparison of the linear (Categorial) and non-linear (neural network) approaches.

References

L. Douglas Baker and Andrew Kachites McCallum. 1998. Distributional clustering of words for text classification. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia.

Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, Cambridge, Massachusetts.

Marco Baroni, Raffaela Bernardi, and Roberto Zamparelli. 2014a. Frege in space: A program of compositional distributional semantics. *Linguistic Issues in Language Technology*, 9.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014b. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Baltimore, Maryland.

Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond Mooney. 2013. Montague meets Markov: Deep semantics with probabilistic logical form. In *Proceedings of the 2nd Joint Conference on Lexical and Computational Semantics (*Sem 2013)*, Atlanta, Georgia.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, (3):1137–1155.

James C Bezdek and Richard J Hathaway. 2002. Some notes on alternating optimization. In *Advances in Soft Computing (AFSS 2002)*, pages 288–300. Springer.

William Blacoe and Mirella Lapata. 2012. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012)*, Jeju Island, Korea.

William Blacoe, Elham Kashefi, and Mirella Lapata. 2013. A quantum-theoretic approach to distributional semantics. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Compu-*

tational Linguistics: Human Language Technologies (NAACL-HLT 2013), Atlanta, Georgia.

Gemma Boleda, Eva Maria Vecchi, Miquel Cornudella, and Louise McNally. 2012. First-order vs. higher-order modification in distributional semantics. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012)*, Jeju Island, Korea.

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, Geneva, Switzerland.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge University Press.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, Sydney, Australia.

J Douglas Carroll, Sandra Pruzansky, and Joseph B Kruskal. 1980. CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika*, 45(1):3–24.

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

Stephen Clark and James R Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, Philadelphia, Pennsylvania.

Stephen Clark. 2013. Type-driven syntax and semantics for composing meaning vectors. *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse*, pages 359–377.

Stephen Clark. 2015. Vector space models of lexical meaning. In *Handbook of Contemporary Semantics, 2nd Edition*. Wiley-Blackwell.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. 2011. Mathematical

foundations for a compositional distributional model of meaning. *Linguistic Analysis*, 36(1-4):345–384.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Ann Copestake and Aurelie Herbelot. 2012. Lexicalised compositionality. `http://www.cl.cam.ac.uk/~ah433/lc-semprag.pdf`.

James R Curran, Stephen Clark, and Johan Bos. 2007. Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the Demonstration Session of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007)*, Prague, Czech Republic.

James R. Curran. 2004. *From distributional to semantic similarity.* Ph.D. thesis, University of Edinburgh.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Katrin Erk and Sebastian Padó. 2008. A structured vector space model for word meaning in context. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, Waikiki, Honolulu, Hawaii.

John R. Firth. 1957. A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*, pages 1–32.

Peter W Foltz, Walter Kintsch, and Thomas K Landauer. 1998. The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2-3):285–307.

Daniel Fried, Tamara Polajnar, and Stephen Clark. 2015. Low-rank tensors for verbs in compositional distributional semantics. In *Proceedings of the Short Papers of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*, Beijing, China.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Edward Grefenstette and Mehrnoosh Sadrzadeh. 2011. Experimenting with transitive verbs in a DisCoCat. In *Proceedings of the 2011 Workshop on Ge-*

ometrical Models of Natural Language Semantics (GEMS 2011), Edinburgh, Scotland.

Edward Grefenstette, Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh, and Marco Baroni. 2013. Multi-step regression learning for compositional distributional semantics. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)*, Pottsdam, Germany.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(23):146–162.

Richard A Harshman. 1970. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition.

Karl Moritz Hermann and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, Sofia, Bulgaria.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Junzhou Huang, Tong Zhang, and Dimitris Metaxas. 2011. Learning with structured sparsity. *The Journal of Machine Learning Research*, 12:3371–3412.

Martin Jaggi, Marek Sulovsk, et al. 2010. A simple algorithm for nuclear norm regularized problems. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 471–478.

Dimitri Kartsaklis and Mehrnoosh Sadrzadeh. 2014. A study of entanglement in a categorical framework of natural language. In *Proceedings of the 11th Workshop on Quantum Physics and Logic (QPL 2014)*, Kyoto, Japan, June.

Walter Kintsch. 2001. Predication. *Cognitive Science*, 25(2):173–202.

Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Thomas K Landauer and Susan T Dumais. 1997. A solution to Plato's

problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2):211.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, Beijing, China.

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

Tao Lei, Yuan Zhang, Lluis Marquez, Alessandro Moschitti, and Regina Barzilay. 2015. High-order low-rank tensors for semantic role labeling. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2015)*, Denver, Colorado.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Baltimore, Maryland.

Omer Levy and Yoav Goldberg. 2014b. Neural word embedding as implicit matrix factorization. *Advances in Neural Information Processing Systems*, 27:2177–2185.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Percy Liang and Christopher Potts. 2015. Bringing machine learning and compositional semantics together. *Annual Reviews of Linguistics*, 1(1):355–376.

Kevin Lund and Curt Burgess. 1996. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208.

Jan R Magnus and Heinz Neudecker. 1988. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons.

Jean Maillard and Stephen Clark. 2015. A tensor skip-gram model for

learning adjective and noun representations. In *Advances in Distributional Semantics Workshop*, London, UK.

Jean Maillard, Stephen Clark, and Edward Grefenstette. 2014. A type-driven tensor-based semantics for CCG. In *Proceedings of the EACL 2014 Type Theory and Natural Language Semantics Workshop (TTNLS)*, Gothenburg, Sweden.

Christopher D Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT press.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014): System Demonstrations*, pages 55–60, Baltimore, Maryland.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Neural Information Processing Systems (NIPS 2013)*.

Dmitrijs Milajevs, Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Matthew Purver. 2014. Evaluating neural word representations in tensor-based compositional settings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

George A Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Thomas P Minka. 2000. Old and new matrix algebra useful for statistics. `http://research.microsoft.com/en-us/um/people/minka/papers/matrix/minka-matrix.pdf`.

Guido Minnen, John Carroll, and Darren Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(03):207–223.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of the 46th Annual Meeting of the Assocation for Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, Columbus, Ohio.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. *Advances in Neural Information Processing Systems*, 26:2265–2273.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Richard Montague. 1974. The proper treatment of quantification in ordinary English. In *Formal Philosophy*, pages 247–270. Yale University Press.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, Barbados.

Yoshiki Niwa and Yoshihiko Nitta. 1994. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, Kyoto, Japan.

Sebastian Padó and Mirella Lapata. 2007. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199.

Denis Paperno, Nghia The Pham, and Marco Baroni. 2014. A practical and linguistically-motivated approach to compositional distributional semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Baltimore, Maryland.

Barbara Partee. 1984. Compositionality. *Varieties of Formal Semantics*, 3:281–311.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empiricial Methods in Natural Language Processing (EMNLP 2014)*, Doha, Qatar.

Tamara Polajnar and Stephen Clark. 2014. Improving distributional semantic vectors through context selection and normalisation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2014)*, Gothenburg, Sweden.

Tamara Polajnar, Luana Fagarasan, and Stephen Clark. 2014a. Reducing dimensions of tensors in type-driven distributional semantics. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, Doha, Qatar.

Tamara Polajnar, Laura Rimell, and Stephen Clark. 2014b. Using sentence

plausibility to learn the semantics of transitive verbs. In *Proceedings of the NIPS Workshop on Learning Semantics*, Montreal, Quebec.

Tamara Polajnar, Laura Rimell, and Stephen Clark. 2015. Learning verb tensors with contextual sentence spaces. *Under review.*

Xin Rong. 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738.*

Hinrich Schütze. 1998. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–124.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, Bellevue, Washington.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012)*, Jeju Island, Korea.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, Seattle, Washington.

Mark Steedman. 2000. *The Syntactic Process.* MIT Press, Cambridge, MA.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

Peter D Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188.

Eva Maria Vecchi, Marco Baroni, and Roberto Zamparelli. 2011. (Linear) maps of the impossible: capturing semantic anomalies in distributional space. In *Proceedings of the Workshop on Distributional Semantics and Compositionality (DISCo-11)*, Portland, Oregon.

Fabio M Zanzotto and Lorenzo Dell'arciprete. 2012. Distributed tree kernels.

In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, Edinburgh, Scotland.

Matthew D Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in AI (UAI 2005)*, Edinburgh, Scotland.