The Categorial Framework for Compositional Distributional Semantics

Stephen Clark, Laura Rimell, Tamara Polajnar, Jean Maillard
University of Cambridge Computer Laboratory

January 7, 2016

1 Introduction

This technical report is concerned with the integration of formal semantics [52, 18] and distributional semantics [62, 70, 11]. Several current attempts at this integration [42, 23, 29] effectively take the logic of formal semantics as a starting point, and then attempt to fill the acknowledged gaps in that framework by using distributional semantics to enrich the logical representations with lexical meaning representations. Here we adopt a different strategy: we start with a framework in which vectors, the building blocks of distributional semantics, are native elements, and which also provides a natural operation for many of the compositional processes that combine words into larger phrases and sentences [12, 16, 5, 6].

Formal semantics provides established accounts of many semantic phenomena which appear to be missing from distributional approaches, for example compositionality, quantification, negation, conjunction, and inference. Here the focus is on the question of how to integrate a compositional process into a vector-based semantics, which allows the derivation of linear-algebraic objects for phrases and sentences. The starting point is the compositional framework of Coecke et al. [16], which shows how a type-driven process based on categorial grammar, similar to that used by Montague, can be defined in terms of tensors (higher order vectors). The framework, which we refer to as the Categorial framework, provides an elegant solution to the question of how compositionality can be introduced into a vector-based semantics; in particular, a solution in which the relational nature of word types such as verbs and adjectives is reflected in their linear algebraic type.

One feature of the Categorial framework is its abstract nature. While it provides a natural framework for phrase and sentence composition, it leaves open many key questions about the optimal vector spaces, parameter learning methods, and instantiations of more complex compositional processes required for any natural language application. These questions must be answered by particular implementations of the framework, and their effectiveness evaluated against suitable datasets. Three key questions involve: the choice of parameter learning methods for the tensor represenions of relational words such as verbs and adjectives; the realisation of compositional processes involving closed-class function words such as quantifiers, negation, conjunction, and relative pronouns; and the choice of vector spaces for composed sentence representations. From a practical point of view, these questions are inextricably linked, since the target semantic representation of a composed phrase or sentence influences the representation and parameter learning method for its constutient relational and functional word types.

Regarding the third question, the Categorial framework makes no commitment to the nature of the sentence space, other than that it should be a vector space, potentially distinct from the other vector spaces corresponding to basic categories (primarily the noun space). Given that one of the main motivations of the framework is to derive phrase- and sentence-level representations [11], this is an urgent question. Some sentence spaces have been implemented, for example a two-dimensional "plausibility space" [57], and an $N \otimes N$ space consisting of outer products of a verb's nominal arguments [27, 26, 34], but more work is required in this area.

There are currently two contenders for how to build vector-based sentence representations compositionally. The first, in the neural networks tradition, is to derive sentence representations using a task-based objective, for example as a means to perform sentiment analysis or syntactic parsing [67, 66]. The second is to extend the distributional hypothesis from words to phrases and even whole sentences [15, 6]. The attraction of this approach is that it provides a unified account of meaning: all meanings are defined in terms of contextual usage, even for larger phrases. The role of compositionality is then to derive these contextual representations for phrases which are unattested, or rarely seen, in corpora, since simple counting techniques will no longer apply. However, the strong intuition that the meanings of words can be defined in terms of contexts is perhaps less applicable in the case of sentences, and so the question arises of what the relevant contexts of sentences should be. Polajnar et al. [59] investigate several novel N-dimensional sentence spaces where the features are co-occurrence counts of sentences with words in the surrounding discourse.

In the remainder of this report, Section 2 describes existing work attempting to unite formal and distributional semantics, with a focus on the problem of compositionality. The current strand of work in Computational Linguistics, which started around 2007 [14, 12, 50, 5], is also related to work in Cognitive Science addressing a similar problem in the 1980s and 90s [20, 64]. Section 2 also describes how vector representations are composed in neural network models which build sentence representations.

Section 3, which is the focus of this report, describes the Categorial framework [16] with a Computational Linguistics audience in mind. Hence the presentation will be based on multilinear algebra, rather than the category theory of the original papers, and the underlying syntactic formalism will be assumed to be Combinatory Categorial Grammar rather than pregroup grammars. Some existing implementations within the framework are also described.

2 Background

In this section we look at the various ways in which vector-based representations have been combined in order to produce compositional phrasal representations. First we survey work in which vectors are combined using standard operators from linear algebra, such as vector addition and elementwise multiplication. Then we consider various neural network architectures, which introduce the notion of matrices acting as operators on pairs of vectors given as input. This leads to a model in which the words themselves can be matrices; i.e. some words have "operator semantics", where the meaning of a word is largely determined by the effect that it has on another word it combines with. Taking this idea one step further leads neatly into Section 3, where the meanings of some words, such as verbs, are multilinear operators, i.e. tensors, where the shape of the tensor is determined by the word's syntactic type. But before describing the tensor-based framework in Section 3, this section concludes with a historical perspective, contrasting current work in Computational Linguistics with that from Cognitive Science in the 1980s and 90s.

Note that, in this report, we are agnostic about how the vector representations are built. Many of the composition techniques we describe can be equally applied to "classical" distributional vectors built using counting methods [70, 11] and vectors built using neural embedding techniques, such as the skip-gram model [48]. It may be that some composition techniques work better with one type of vector or the other — for example there is anecdotal evidence that elementwise multiplication performs better with counting methods and the tensor product performs better with neural embeddings — although research comparing the two vector-building methods is still in its infancy [4, 49, 40, 41].

2.1 Simple Operators on Vectors

The simplest method of combining two distributional representations, for example vectors for *fast* and *car*, is to perform some elementwise combination, such as vector addition or elementwise multiplication. As a method of integrating formal and distributional semantics, these operators appear to be inadequate *a priori*, not least because they are commutative, and so do not respect word order. Despite this fact, there is a large literature investigating how such operators can be used for phrasal composition, starting with the work of Mitchell and Lapata [50, 51] (M&L subsequently).

The additive model from M&L has the general form:

$$\mathbf{p} = A\mathbf{u} + B\mathbf{v}$$

where $\mathbf{u}$ and $\mathbf{v}$ are word (column) vectors in $\mathbb{R}^n$ (e.g. $\overrightarrow{\textit{fast}}$ and $\overrightarrow{\textit{car}}$), $\mathbf{p} \in \mathbb{R}^n$ is the vector for the phrase resulting from composition of $\mathbf{u}$ and $\mathbf{v}$ ($\overrightarrow{\textit{fast car}}$), and $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$ are matrices which determine the contribution of $\mathbf{u}$ and $\mathbf{v}$ to $\mathbf{p}$. However, M&L make the simplifying assumption that only the ith components of $\mathbf{u}$ and $\mathbf{v}$ contribute to the ith component of $\mathbf{p}$, which yields the form:

$$\mathbf{p} = \alpha\mathbf{u} + \beta\mathbf{v}$$

The parameters $\alpha, \beta \in \mathbb{R}$ allow the contributions of $\mathbf{u}$ and $\mathbf{v}$ to be weighted differently, providing a minimal level of syntax-awareness to the model. For example, if $\mathbf{u}$ is the vector representation for an adjective, and $\mathbf{v}$ the vector representation for a noun, $\mathbf{v}$ may be given a higher weight since it provides the linguistic head for the resulting noun phrase; whereas if $\mathbf{u}$ is an intransitive verb and $\mathbf{v}$ a noun, $\mathbf{u}$ may be weighted more highly.

The multiplicative model presented by M&L has the general form:

$$\mathbf{p} = C\ (\mathbf{u} \otimes \mathbf{v})$$

where $\mathbf{u} \otimes \mathbf{v}$ is the tensor, or outer, product of $\mathbf{u}$ and $\mathbf{v}$, and C is a tensor of order three, which projects the tensor product of $\mathbf{u}$ and $\mathbf{v}$ onto the space of $\mathbf{p}$ (which is assumed to be the same space as that containing $\mathbf{u}$ and $\mathbf{v}$). Under similar simplifying assumptions to the additive model, the multiplicative model has the form:

$$\mathbf{p} = \mathbf{u} \odot \mathbf{v}$$

where $\odot$ represents elementwise multiplication, that is, $\mathbf{p}_i = \mathbf{u}_i \cdot \mathbf{v}_i$, where $\mathbf{p}_i$ is the ith coefficient of $\mathbf{p}$. In this simpler model, C simply picks out the diagonal in the $\mathbf{u} \otimes \mathbf{v}$ tensor.

Vector multiplication is intersective, while vector addition is not, in the sense that under multiplication $\mathbf{p}$ has a non-zero component $\mathbf{p}_i$ only when $\mathbf{u}_i$ and $\mathbf{v}_i$ are both non-zero, while under addition each component of both $\mathbf{u}$ and $\mathbf{v}$ contributes to $\mathbf{p}$. For example, when deriving a phrase vector for *horse gallop* using addition (and assuming classical count vectors), if *horse* has a non-zero co-occurrence with *tail*, but *gallop* has a zero co-occurrence, then $\overrightarrow{\textit{horse gallop}}$ will still have a non-zero coefficient for the basis vector corresponding to *tail*. On the other hand, if both *horse* and *gallop* have high co-occurrence values with *run*, then the coefficient of $\overrightarrow{\textit{horse gallop}}$ for the basis vector corresponding to *run* will be larger than that for *tail*; i.e. vector addition emphasizes components common to $\mathbf{u}$ and $\mathbf{v}$. As M&L point out, intersectivity of this kind is not necessarily an advantage, since some information is discarded; however, multiplicative models have typically performed well on various phrasal composition tasks.

One obvious question to consider is how well such simple operators scale when applied to phrases or sentences longer than a few words. Polajnar et al. [58, 56] suggest that elementwise multiplication performs badly in this respect, with the quality of the composed representation degrading quickly as the number of composition operations increases. Given the intersective nature of multiplication, as described above, perhaps this is not surprising. Vector addition is more stable, but Polajnar et al. [58] suggest that the quality of the composed representation degrades after around ten binary composition operations, even with addition.

Of course there are theoretical reasons to believe that the simple nature of vector addition, which is commutative and, in some sense, a rather blunt operation, could not hope to capture the subtleties of natural language semantics. However, it is proving surprisingly difficult to create datasets and define tasks in which vector addition does not provide an extremely competitive baseline. Note also that, in the field of Information Retrieval (IR, [45]), which has been using vector representations since its inception in the 1960s [11], vector addition has been the default composition operator for decades and still continues to provide state-of-the-art performance for tasks in IR. A similar comment could be made regarding empirical studies in Cognitive Science [21, 39, 35].

M&L also introduce a method that aims to interpret one of the constituent vectors as an operator, despite its lying in the same semantic space as the argument. A dilation operation decomposes **v** (the argument) into two components, one parallel to **u** and the other perpendicular, and dilates **v** in the direction of **u**:

$$\mathbf{p} = (\mathbf{u} \cdot \mathbf{u})\mathbf{v} + (\lambda - 1)(\mathbf{u} \cdot \mathbf{v})\mathbf{u}$$

M&L also consider the tensor product, which is an instance of the general multiplicative model described above (when C is the identity operator), and described below in Section 2.3, but find that it performs poorly on their phrasal composition tasks and datasets, using classical count vectors. One of the potential problems with the tensor product is that the resulting phrase vector lives in a different vector space to the argument vectors, so that $\overrightarrow{fast} \otimes \overrightarrow{car}$, for example, lives in a different space to $\overrightarrow{car}$. Hence M&L also discuss circular convolution, a technique for projecting a vector in a tensor product space onto the original vector space components [54].

2.2 Neural Network Models

This section considers how vector representations are combined in neural network models. The focus is on the composition operations, rather than training, although of course how to learn the parameters of the network is crucial in obtaining good performance. We will also not discuss the experimental results, but refer readers to the relevant papers.

We start with the work of Socher et al. [67], who apply recursive neural networks to the classic Penn Treebank parsing task [46, 17, 9]. Figure 1, taken from [67], shows how word vectors, and the resulting phrase vectors, are combined. For example, the vectors for words x3 and x4 are concatenated and then combined using the weight matrix W, the output of which is put through a hyperbolic tangent function. There are two features of this process which differ to the tensor-based framework described in Section 3. First, the neural network defines a non-linear function, whereas the tensor-based framework defines linear maps. Whether there are elements of natural language semantics which

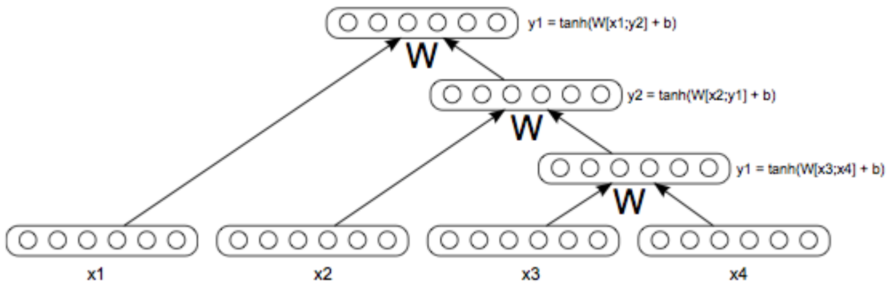

Figure 1: Composition using a recursive neural network, taken from Socher et al. 2010

are inherently non-linear and could not be reasonably modelled with multilinear maps is an open question [68, 31]. A second crucial feature of the recursive neural network is that the output of the composition operation is a vector in the same space as the input vectors; the meanings of words, phrases and sentences have to live in the same space, in order that the composition operation can be applied recursively. In contrast, the type-driven tensor-based framework defines many vector and tensor spaces, in which the meanings of words and phrases with different syntactic types live in different spaces. On the one hand, having all word and phrase types live in the same space allows the meanings of words and phrases with different syntactic types to be easily compared; on the other hand, the recursive neural network is less flexible in terms of the composition operator: all word pairs are combined using the matrix W, irrespective of the type of the syntactic combination (for example verb-object and subject-verb combinations are dealt with in the same way).

One natural way to extend this model is to have more than one weight matrix, W, parameterising W by the type of combination. Hermann and Blunsom take this approach [30], using Combinatory Categorial Grammar (CCG) as the grammar formalism. One of their models defines distinct W matrices for each combinatory rule, for example forward application or forward composition, whereas another, more expressive model has a distinct matrix for every CCG category produced through a binary combination.[1] Note that this is a different use of CCG to the tensor-based framework described in Section 3, which is much more closely tied to the grammar formalism, and adheres to the principle of combinatory type transparency [69]; i.e. the type of the reduction in the syntax, for example forward composition, is exactly mirrored in the semantics [44]. Hermann and Blunsom effectively have a weaker form of this principle, in that different types of combination are governed by different weight matrices,

[1]Or at least a matrix for every (frequent) category seen in training data, since the number of categories is in principle unbounded for some implementations of CCG, depending on the rule schema used.

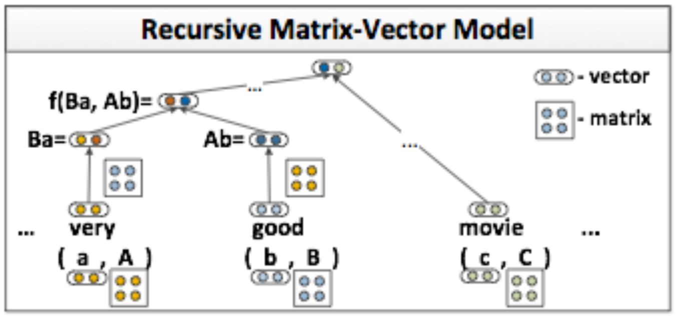

Figure 2: Composition using a matrix-vector representation, from Socher et al. 2012

but ultimately all combinations are forms of matrix multiplication followed by a pointwise non-linearity. Note that, in all the neural network models discussed so far, the weight matrices represent abstract composition operators and do not correspond to any lexical word or phrase; at their most specific, they are tied to a syntactic type.

Socher et al. [66] extends this idea further by having separate matrix operators for every word and phrase, rather than having one operator W handle the composition of every word/phrase pair (perhaps parameterised by the composition type, as described above). They define "matrix-vector spaces", referring to the fact that every word and phrase has both a matrix and a vector representation. The matrix operators can be thought of as capturing how words or phrases affect the meanings of other words or phrases, whereas the vectors are the meanings of the words or phrases themselves. The vector p for a parent node with two vector children a and b is calculated as follows (see also Figure 2):

$$p = f_{A,B}(a,b) = f(Ba, Ab) = g\left(W \begin{bmatrix} Ba \\ Ab \end{bmatrix}\right)$$

where $A \in \mathbb{R}^{n\times n}$ is the matrix for a, $B \in \mathbb{R}^{n\times n}$ is the matrix for b, and $W \in \mathbb{R}^{n\times 2n}$ is a global matrix which maps the pair of Ba and Ab back into the original n-dimensional word space. Note that all words and phrases have an operator semantics, so that child a is multiplied by matrix B and child b is multiplied by matrix A. The function g is a non-linear function such as the hyperbolic tangent. The matrix operator P for the parent p is obtained via another global matrix $W_M \in \mathbb{R}^{n\times 2n}$:

$$P = f_M(A,B) = W_M \begin{bmatrix} A \\ B \end{bmatrix}$$

The final part of the model, separate from the composition operation, is a softmax function which assigns a class distribution $d(p)$ to each parent node p, again requiring a global matrix W^{label} which operates on the parent's vector representation:

$$d(p) = \text{softmax}(W^{label}p)$$

If there are K possible labels, then $d(p) \in \mathbb{R}^K$ is a K-dimensional categorical distribution. The task in [66] is not syntactic parsing but predicting the sentiment of adverb-adjective pairs, such as *not great*, *pretty good* and *pretty terrible*. Hence the K labels range over the possible sentiment values, for example positive or negative. Here the adverbs are thought of as largely having an operator semantics, affecting the meanings of the adjectives with which they combine.

2.3 Historical Perspective

The general question of how to combine vector-based meaning representations in a compositional framework is a relatively new question for Computational Linguistics, but one which has been actively researched in Cognitive Science since the 1980s. The question arose because of the perceived failure of distributed models in general, and connectionist models in particular, to provide a suitable account of compositionality in language [20]. The question is also relevant to the broad enterprise of Artificial Intelligence in that connectionist or distributed representations may seem to be in opposition to the more traditional symbolic systems often employed in AI.

Smolensky argues for the tensor product as the operation which binds a predicate to an argument in the distributed meaning space [64, 65]. Smolensky represents a structure as possessing a set of *roles*, $\mathbf{r}_i$, which, for a particular instance of the structure, also has a set of *fillers*, $\mathbf{f}_i$ (the roles are *bound* to fillers). The vector representation of a role-filler pair is obtained using the tensor product, and the representation for a complete structure, $\mathbf{s}$, of role-filler pairs is the sum of the tensor products for each pair:

$$\mathbf{s} = \sum_i \mathbf{f}_i \otimes \mathbf{r}_i$$

A dependency tree, for example, is naturally represented this way, with the roles being predicates paired with grammatical relations, such as *object of eat*, and the fillers being heads of arguments, such as *hotdog*. The vector representation of *eat hotdog* is $\overrightarrow{hotdog} \otimes \overrightarrow{eat\_\text{dobj}}$. The vector representation of a whole dependency tree is the sum of the tensor products over all the dependency edges in the tree.

Smolensky and Legendre [65] argue at length for why the tensor product is appropriate for combining connectionist and symbolic structures. A number of other proposals for realizing symbolic structures in vector representations are described, including that of Plate [55] and Pollack [60]. Smolensky's claim is that, despite perhaps appearances to the contrary, all are special cases of a generalized tensor product representation.

Clark and Pulman [14] suggest that a similar scheme to Smolensky's could be applied to parse trees for NLP applications, perhaps with the use of a convolution kernel [28] to calculate similarity between two parse trees with distributed representations at each node. A similar idea has been fleshed out in Zanzotto et al. [72], showing how kernel functions can be defined for a variety of composition operations (including most of the operations discussed in this report), allowing the similarity between two parse trees to be computed based on the distributional representations at the leaves and the composed representations on the internal nodes.

Aerts and Gabora [1] also propose use of the tensor product, in order to combine vector-based representations of concepts. More specifically they are interested in the "pet fish" problem, a well-known phenomenon in psychology where subjects typically rate a guppy as a poor example of a fish, and a poor example of a pet, but a good example of a pet fish. This observation has been used to attack the claim that prototypes [61] can serve as concepts. Aerts and Gabora, however, claim that the tensor product can be used to combine distributed representations of pet and fish to yield the desired properties of pet fish.

Note there is an important difference between the use of the tensor product as suggested by Smolensky and how it is used in the Categorial framework in Section 3. The difference is that the Categorial framework uses tensors to represent the meanings of relational words such as verbs, and the generalisation of matrix multiplication (tensor contraction) to combine a verb with an argument; whereas Smolensky uses the tensor product as the compositional operation itself. Crudely put, the representations in Smolensky's scheme get bigger as words and phrases are combined, whereas in the Categorial framework they get smaller through the tensor contraction reduction mechanism.

Another difference, which also applies to the work of Zanzotto et al. [72], is that, in the Categorial framework, the syntactic structure does not form part of the semantic representation. Rather, in line with the philosophy of CCG [69], the derivation is seen merely as a "trace" of the parsing algorithm which combined the syntactic types. The composed semantic representation is built in step with the syntactic derivation, but the syntactic derivation is not part of the semantic representation itself. This is true of both the logical representations favoured by Steedman and others working on semantic parsing with CCG [69, 8, 73] and the vector-based representations in this report.

3 The Compositional Framework

In this section we show how an idea from the previous section — that words can have "operator semantics" represented by matrices, which are objects from linear algebra — can be extended to words with more complex types using multilinear algebra, an extension of linear algebra to functions with more than one argument. It is important to note that the Categorial framework makes no commitment to how the multilinear algebraic objects are realised, only a commitment to their shape and how they are combined. In particular, the vector spaces corresponding to atomic categories, such as noun and sentence, do not have to be *distributional* in the classical sense; they could be *distributed* in the connectionist sense [64], where the basis vectors themselves are not readily interpretable (the neural network models of [66] decribed above fall into this category). Another category of model could include vector representations where the basis vectors are interpretable, but not contextual, for example models based on feature norms [47, 63, 19]. Distributional vector spaces may also be modified, for example by dimensionality reduction techniques, so that the basis vectors are not directly interpretable as contextual. Hence perhaps the most appropriate theory-neutral terms to describe the framework are simply *vector-based* and *tensor-based*, since in principle the framework can accommodate all the above-mentioned representations.

The grammatical formalism assumed in this section will be a variant of Categorial Grammar, which was also the grammar formalism used by Montague [52]. The original papers setting out the tensor-based compositional framework [12, 16] used pregroup categorial grammars [38], largely because they share an abstract mathematical structure with vector spaces. However, other forms of categorial grammar can be used equally as well in practice, and here we use Combinatory Categorial Grammar (CCG, [69]). This decision is based on the ease with which CCG can be used within the framework [24, 44], the prevalence of CCG in the Computational Linguistics literature [33, 13, 74, 71], and the existence of wide-coverage CCG parsers [32, 13, 2, 43]. With a Computational Linguistics audience in mind, we also present the framework entirely in terms of multilinear algebra, rather than the Category Theory of the original papers.

3.1 Composition as Matrix Multiplication

A useful starting point in describing the framework is adjectival modification. In fact, the proposal in Baroni and Zamparelli [5] (B&Z hereafter) to model the meanings of adjectives as matrices can be seen as an instance of the Categorial framework (and also an instance of the general framework described in [6]). The insight in B&Z is that, in theoretical linguistics, adjectives are typically thought of as having a functional role, mapping noun denotations to noun denotations. In CCG, the syntactic type of an adjective in English is N/N, which can be read as: an adjective requires a noun as an argument to the right, and, once it has found such a noun, will return another noun as the result. B&Z make the transition to vector spaces by arguing that, in linear algebra, functions are represented as matrices (the *linear maps*). There is now an obvious answer to the question of what the composition operator should be in this framework, namely matrix multiplication. Figure 3 shows how the context vector for *car* is multiplied by the matrix for *red* to give the vector for *red car*. In this simple example, the noun space containing $\overrightarrow{car} = (c_1, \ldots, c_5)^{\mathbf{T}}$ and $\overrightarrow{red\ car} = (rc_1, \ldots, rc_5)^{\mathbf{T}}$ has 5 basis vectors, which means that a 5×5 matrix, $\overline{red}$, is required for the adjective.

The second contribution of B&Z is to propose a method for learning the $\overline{red}$ matrix from supervised training data. What should the gold-standard representation for $\overrightarrow{red\ car}$ be? B&Z argue that, given large enough corpora, it should ideally be the context vector for the compound *red car*; i.e. that the vector for

$$\begin{array}{ccc} \overline{red} & \overrightarrow{car} & \overrightarrow{red\ car} \\ \begin{pmatrix} R_{11} & R_{12} & R_{13} & R_{14} & R_{15} \\ R_{21} & R_{22} & R_{23} & R_{24} & R_{25} \\ R_{31} & R_{32} & R_{33} & R_{34} & R_{35} \\ R_{41} & R_{42} & R_{43} & R_{44} & R_{45} \\ R_{51} & R_{52} & R_{53} & R_{54} & R_{55} \end{pmatrix} & \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} = & \begin{pmatrix} rc_1 \\ rc_2 \\ rc_3 \\ rc_4 \\ rc_5 \end{pmatrix} \end{array}$$

Figure 3: Matrix mulitiplication for adjective-noun combinations

red car should be built in exactly the same way as the vector for *car*. The reason for the adjective matrix is that it allows the generalisation of adjective-noun combinations beyond those seen in the training data. The details of the B&Z training process will not be covered in this section, but briefly the process is to find all examples of adjective-noun pairs in a large training corpus, and then use standard linear regression techniques to obtain matrices for each adjective in the training data. These learnt matrices can then be used to generalise beyond the seen adjective-noun pairs. So if *tasty artichoke*, for example, has not been seen in the training data (or perhaps seen infrequently), the prediction for the context vector $\overrightarrow{tasty\ artichoke}$ can be obtained, as long as there are enough examples of *tasty X* in the data to obtain the $\overline{tasty}$ matrix (via linear regression), and enough occurrences of *artichoke* in order to obtain the $\overrightarrow{artichoke}$ vector. Polajnar et al. [59] use a similar training technique to obtain contextual vector representations for subject-verb-object sentences.

Testing is performed by using held-out context vectors for some of the adjective-noun pairs in the data, and seeing how close — according to the cosine measure — the predicted context vector is to the actual context vector for each adjective-noun pair in the test set. The learning method using linear regression was compared against various methods of combining the context vectors of the adjective and noun, such as vector addition and elementwise multiplication, and was found to perform significantly better at this prediction task.

3.2 An Extension using Multilinear Algebra: tensor-based CCG semantics

A matrix is a second-order tensor; for example, the $\overline{red}$ matrix above lives in the $\mathsf{N} \otimes \mathsf{N}$ space, meaning that two indices — each corresponding to a basis vector in the noun space N — are needed to specify an entry in the matrix.[2] Noting that $\mathsf{N} \otimes \mathsf{N}$ is structurally similar to the syntactic type N/N — both specify functions from nouns to nouns — a recipe for translating a syntactic type into a semantic type suggests itself: replace all slash operators in the syntactic type

[2]The tensor product $V \otimes W$ of two vector spaces $V \in \mathbb{R}^n$ and $W \in \mathbb{R}^m$ is an nm-dimensional space spanned by elements of the form $v \otimes w$, i.e. pairs of basis vectors of V and W. An element $a \in V \otimes W$ can be written as $\sum a_{ij} v_i \otimes w_j$ where a_{ij} is a scalar.

with tensor product operators. With this translation, the combinators employed by CCG to combine syntactic categories carry over seamlessly to the meaning spaces, maintaining what is often described as CCG's "transparent interface" between syntax and semantics. The seamless integration with CCG arises from the (somewhat trivial) observation that tensors are linear maps — a particular kind of function — and hence can be manipulated using CCG's combinatory rules.

Here are some example syntactic types, and the corresponding tensor spaces in which words with those types are semantically represented (using the notation *syntactic type* : *semantic type*). We first assume that all atomic types have meanings living in distinct vector spaces:[3]

- noun phrases, $NP : \mathsf{N}$
- sentences, $S : \mathsf{S}$

Replacing each slash in a complex syntactic type with a tensor product operator results in the following meaning spaces, following the CCG result-leftmost convention for both the syntactic and semantic types:

- Intransitive verb, $S\backslash NP : \mathsf{S} \otimes \mathsf{N}$
- Transitive verb, $(S\backslash NP)/NP : \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$
- Ditransitive verb, $((S\backslash NP)/NP)/NP : \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N} \otimes \mathsf{N}$
- Adverbial modifier, $(S\backslash NP)\backslash(S\backslash NP) : \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{S} \otimes \mathsf{N}$
- Preposition modifying NP, $(NP\backslash NP)/NP : \mathsf{N} \otimes \mathsf{N} \otimes \mathsf{N}$

Hence the meaning of an intransitive verb, for example, is a particular matrix, or second-order tensor, in the tensor product space $\mathsf{S} \otimes \mathsf{N}$. The meaning of a transitive verb is a "cuboid", or 3rd-order tensor, in the tensor product space $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$. In the same way that the syntactic type of an intransitive verb can be thought of as a function — taking an NP and returning an S — the meaning of an intransitive verb is also a function (linear map) — taking a vector in N and returning a vector in S. Another way to think of this function is that each element of the matrix specifies, for a pair of basis vectors (one from N and one from S), how a value on the given N basis vector contributes to the result for the given S basis vector.

Now we describe how the combinatory rules carry over to the meaning spaces, for a number of common rules. For a more detailed mathematical description, and more examples of combinatory rules, including backward-crossed composition and type-raising, the reader is referred to [44].

[3]We make no distinction in this section between nouns and noun phrases, as far as the corresponding semantic space is concerned.

3.2.1 Forward and Backward Application

The function application rules of CCG are forward (>) and backward (<) application:

$$\begin{array}{llll} X/Y & Y & \Longrightarrow X & (>) \\ Y & X\backslash Y & \Longrightarrow X & (<) \end{array}$$

where X and Y can be any CCG category (atomic or complex). Because of the transparent nature of CCG's syntax-semantics interface, the corresponding semantic rule is also function application. For example, the meaning of a subject *NP* combines with the meaning of an intransitive verb via matrix multiplication, which is equivalent to applying the linear map corresponding to the intransitive verb matrix to the subject *NP* vector.

Consider the following derivation involving an intransitive verb, with the corresponding semantic type below its syntactic type:

$$\begin{array}{cc} \textit{Basil} & \textit{sleeps} \\ \hline NP & S\backslash NP \\ \mathsf{N} & \mathsf{S}\otimes\mathsf{N} \\ \hline \multicolumn{2}{c}{S} \\ \multicolumn{2}{c}{\mathsf{S}} \end{array} <$$

Let the meaning of *Basil* be $\overrightarrow{Basil} \in \mathsf{N}$ and *sleeps* be $\overline{sleeps} \in \mathsf{S}\otimes\mathsf{N}$; then applying the $\overline{sleeps}$ operator (matrix) to its argument $\overrightarrow{Basil}$ is an example of the more general *tensor contraction* operation from multilinear algebra. A useful notation for tensor contraction, described in [44], is Einstein notation, shown here on the left of the equality for the matrix multiplication:

$$\overline{sleeps}_{ij}\ \overrightarrow{Basil}_j = \sum_j \overline{sleeps}_{ij}\ \overrightarrow{Basil}_j \qquad (1)$$

$$= \overrightarrow{Basil\ sleeps}_i \qquad (2)$$

The *Einstein summation convention* implicitly assumes summation over the relevant range on every component index that occurs twice, in this case the j index. Pairs of indices that are summed over are known as *contracted*, while the remaining indices are known as *free*. Here we use the convention that the indices on the function maintain the same order as the syntactic and semantic types. That is why, when performing the contraction corresponding to *Basil sleeps*, $\overrightarrow{Basil} \in \mathsf{N}$ is contracted with the *second* index of $\overline{sleeps} \in \mathsf{S}\otimes\mathsf{N}$, and not the first. The first index of $\overline{sleeps}$ is then the only free index, telling us that the above operation yields a first-order tensor (vector). Since this index corresponds to S, we know that using backward application to combine *Basil* and *sleeps* yields a meaning vector in S.

Forward application is performed in the same manner. Consider the following derivation involving a transitive verb:

$$\begin{array}{ccc}
\textit{Basil} & \textit{chases} & \textit{Tommy} \\
\overline{NP} & \overline{(S\backslash NP)/NP} & \overline{NP} \\
\mathsf{N} & \mathsf{S}\otimes\mathsf{N}\otimes\mathsf{N} & \mathsf{N} \\
 & \multicolumn{2}{c}{\overline{\qquad\qquad\qquad\qquad}{>}} \\
 & \multicolumn{2}{c}{S\backslash NP} \\
 & \multicolumn{2}{c}{\mathsf{S}\otimes\mathsf{N}}
\end{array}$$

The forward application deriving the type of *chases Tommy* corresponds to:

$$\overline{chases}_{ijk}\ \overrightarrow{Tommy}_k$$

where $\overrightarrow{Tommy}$ is contracted with the third index of $\overline{chases}$ because we have maintained the order defined by the type $(S\backslash NP)/NP$: the third index then corresponds to an argument NP coming from the right. Counting the number of free indices in the above expression tells us that it yields a second-order tensor. Looking at the types corresponding to the free indices tells us that this second-order tensor is of type $\mathsf{S}\otimes\mathsf{N}$, which is the semantic type of a verb phrase (or intransitive verb), as we have already seen in the *sleeps* example.

3.2.2 Forward and Backward Function Composition

The forward ($>_{\mathbf{B}}$) and backward ($<_{\mathbf{B}}$) composition rules are:

$$\begin{array}{llll}
X/Y & Y/Z & \Longrightarrow X/Z & (>_{\mathbf{B}}) \\
Y\backslash Z & X\backslash Y & \Longrightarrow X\backslash Z & (<_{\mathbf{B}})
\end{array}$$

Again, because of the transparent interface, the corresponding semantic rule is function composition, which for multilinear algebra also reduces to a form of tensor contraction. Consider the following example, in which *might* can combine with *chase* using forward composition:

$$\begin{array}{cccc}
\textit{Basil} & \textit{might} & \textit{chase} & \textit{Tommy} \\
\overline{NP} & \overline{(S\backslash NP)/(S\backslash NP)} & \overline{(S\backslash NP)/NP} & \overline{NP} \\
\mathsf{N} & \mathsf{S}\otimes\mathsf{N}\otimes\mathsf{S}\otimes\mathsf{N} & \mathsf{S}\otimes\mathsf{N}\otimes\mathsf{N} & \mathsf{N} \\
 & \multicolumn{2}{c}{\overline{\qquad\qquad\qquad\qquad\qquad\qquad}{>_{\mathbf{B}}}} & \\
 & \multicolumn{2}{c}{(S\backslash NP)/NP} & \\
 & \multicolumn{2}{c}{\mathsf{S}\otimes\mathsf{N}\otimes\mathsf{N}} &
\end{array}$$

with tensors $\overline{might} \in \mathsf{S}\otimes\mathsf{N}\otimes\mathsf{S}\otimes\mathsf{N}$ and $\overline{chase} \in \mathsf{S}\otimes\mathsf{N}\otimes\mathsf{N}$. Combining the meanings of *might* and *chase* corresponds to the following operation using Einstein notation:

$$\overline{might}_{ijkl}\ \overline{chase}_{klm}$$

yielding a tensor in $\mathsf{S}\otimes\mathsf{N}\otimes\mathsf{N}$ for the meaning of *might chase* (with free indices i, j and m), which is the correct semantic type for a phrase with syntactic type $(S\backslash NP)/NP$. Backward composition is performed analogously.

Figure 4 gives some further intuition regarding the correspondence between the syntactic and semantic combination rules. Earlier work in Categorial Grammar [3] noted how the syntactic combination rules are analgous to rules of "cancellation". In the function composition example in the figure, the categories in

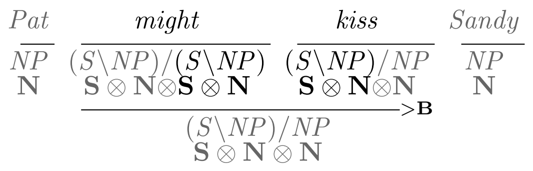

Figure 4: Combinatory rules correspond to "cancellation" in both the syntax and semantics.

black are cancelling. Figure 5 shows how this cancellation corresponds to taking inner products. The meaning of *might* is a fourth-order tensor living in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{S} \otimes \mathsf{N}$. For the purpose of exposition of how the inner products work, we choose to show the fourth-order tensor (on the left) as a matrix of matrices. Note how four indices are needed to specify an entry in the fourth-order tensor. The meaning of *kiss* is a third-order tensor living in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$, shown here on the right as a (column) vector of matrices. Note how three indices are needed to specify an entry in the third-order tensor. Finally, the meaning of *might kiss* is also a third-order tensor living in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$, shown at the bottom as a (row) vector of matrices.

The point of the diagram is to show how the value for $\overline{might\ kiss}_{111}$ is calculated (the mk_{111} value in black at the bottom of the figure). Considering the large matrix-of-matrices for $\overline{might}$, we take the component matrix corresponding to position $\overline{might}_{11}$ (the m_{11lm} matrix shown in black at the top left).[4] For the vector of $\overline{kiss}$, we take the component matrix corresponding to position $\overline{kiss}_1$ (the k_{1lm} matrix in black at the top right); notice how these three indices, for $\overline{might}$ and $\overline{kiss}$, match the indices on the result for $\overline{might\ kiss}_{111}$. The value for $\overline{might\ kiss}_{111}$ is obtained by taking an inner product between the two matrices in black (also known as a *Frobenius inner product*). The Frobenius inner product is analagous to vector inner product: a sum of the products of the corresponding components of the two matrices.

A point of clarification is required here: in order to represent the cancelling matrices in diagrammatic form, the $\overline{kiss}$ tensor has been manipulated by changing the canonical order of the indices: the first index on the $\overline{kiss}$ vector of matrices corresponds to the *NP* object, which would be the *last* index using the convention in which the indices are ordered by argument order. In terms of tensor contraction, and using the earlier ordering convention, the forward composition would be represented as follows:

$$m_{abcd}\, k_{cde} = mk_{abe}$$

where e corresponds to the *NP* object, b to the *NP* subject, and a to the final S.

[4]We are abusing notation slightly here, since $\overline{might}$ is a fourth-order tensor; specifying just the first two indices is a convenient way to denote one of the component matrices when the fourth-order tensor is represented as a matrix of matrices.

$$\begin{pmatrix} m_{1111} & m_{1112} & m_{1113} \\ m_{1121} & m_{1122} & m_{1123} \\ m_{1131} & m_{1132} & m_{1133} \end{pmatrix} \begin{pmatrix} m_{1211} & m_{1212} & m_{1213} \\ m_{1221} & m_{1222} & m_{1223} \\ m_{1231} & m_{1232} & m_{1233} \end{pmatrix} \begin{pmatrix} m_{1311} & m_{1312} & m_{1313} \\ m_{1321} & m_{1322} & m_{1323} \\ m_{1331} & m_{1332} & m_{1333} \end{pmatrix} \quad \begin{pmatrix} k_{111} & k_{112} & k_{113} \\ k_{121} & k_{122} & k_{123} \\ k_{131} & k_{132} & k_{133} \end{pmatrix}$$

$$\begin{pmatrix} m_{2111} & m_{2112} & m_{2113} \\ m_{2121} & m_{2122} & m_{2123} \\ m_{2131} & m_{2132} & m_{2133} \end{pmatrix} \begin{pmatrix} m_{2211} & m_{2212} & m_{2213} \\ m_{2221} & m_{2222} & m_{2223} \\ m_{2231} & m_{2232} & m_{2233} \end{pmatrix} \begin{pmatrix} m_{2311} & m_{2312} & m_{2313} \\ m_{2321} & m_{2322} & m_{2323} \\ m_{2331} & m_{2332} & m_{2333} \end{pmatrix} \quad \begin{pmatrix} k_{211} & k_{212} & k_{213} \\ k_{221} & k_{222} & k_{223} \\ k_{231} & k_{232} & k_{233} \end{pmatrix}$$

$$\begin{pmatrix} m_{3111} & m_{3112} & m_{3113} \\ m_{3121} & m_{3122} & m_{3123} \\ m_{3131} & m_{3132} & m_{3133} \end{pmatrix} \begin{pmatrix} m_{3211} & m_{3212} & m_{3213} \\ m_{3221} & m_{3222} & m_{3223} \\ m_{3231} & m_{3232} & m_{3233} \end{pmatrix} \begin{pmatrix} m_{3311} & m_{3312} & m_{3313} \\ m_{3321} & m_{3322} & m_{3323} \\ m_{3331} & m_{3332} & m_{3333} \end{pmatrix} \quad \begin{pmatrix} k_{311} & k_{312} & k_{313} \\ k_{321} & k_{322} & k_{323} \\ k_{331} & k_{332} & k_{333} \end{pmatrix}$$

$$\begin{pmatrix} mk_{111} & mk_{112} & mk_{113} \\ mk_{121} & mk_{122} & mk_{123} \\ mk_{131} & mk_{132} & mk_{133} \end{pmatrix} \begin{pmatrix} mk_{211} & mk_{212} & mk_{213} \\ mk_{221} & mk_{222} & mk_{223} \\ mk_{231} & mk_{232} & mk_{233} \end{pmatrix} \begin{pmatrix} mk_{311} & mk_{312} & mk_{313} \\ mk_{321} & mk_{322} & mk_{323} \\ mk_{331} & mk_{332} & mk_{333} \end{pmatrix}$$

Figure 5: Cancellation rules in the semantics correspond to taking inner products. For simplicity, we assume that noun vectors are three-dimensional.

The point of these examples is to show how cancellation in the semantics corresponds to taking two tensors and collapsing them into a single real value. Similar explanations can be given for all combinatory rules, and cancellation of all syntactic types, irrespective of the number of slashes. This comment applies to simpler cases, such as the adjective-noun example from earlier, and more complex cases with more slashes; for example, a combination where the *cancelling* categories are $(N/N)/(N/N)$ would require inner products between 4th-order tensors in $\mathsf{N} \otimes \mathsf{N} \otimes \mathsf{N} \otimes \mathsf{N}$.

3.3 Existing Implementations

There are a few existing implementations of the Categorial framework, focusing mostly on adjectives and transitive verbs. The adjective implementation is that of B&Z as described in Section 3.1, where linear regression is used to learn each adjective as a mapping from noun contexts to adjective-noun contexts, with the observed context vectors for the noun and adjective-noun instances as training data. Since an adjective-noun combination has noun phrase meaning, the noun space is the obvious choice for the space in which the composed meanings should live.

For verb-argument composition, the question of sentence spaces arises.[5] A transitive verb such as $\overline{\textit{chase}}$ lives in $S \otimes N \otimes N$, and can therefore be represented as

[5]We consider a verb with all of its argument positions saturated, for example a subject-verb-object triple, to be a sentence, although real-world sentences are much longer and contain determiners and other function words.

$$\sum_{ijk} C_{ijk}(\overrightarrow{s_i} \otimes \overrightarrow{n_j} \otimes \overrightarrow{n_k})$$

for some choice of coefficients C_{ijk}. Grefenstette et al. [27] choose $S = N \otimes N$, using the tensor product of the noun space with itself as the sentence space. Their concrete implementation learns the verb by counting observed $n^{(s)}$, $n^{(o)}$ pairs where $n^{(s)}$ is the subject and $n^{(o)}$ the object of V. The final representation for V is then

$$V = \sum_{i=1}^{N_V} \overrightarrow{n_i^{(s)}} \otimes \overrightarrow{n_i^{(o)}}$$

where N_V is the number of instances of V in the corpus, and $(n_i^{(s)}, n_i^{(o)})$ are the subject and object nouns for the ith instance of V. This method of learning the verb tensor is known as the relational method, since it captures relations between subject and object features. Composition with a particular subject and object reduces to taking the tensor product of the subject and object, and performing pointwise multiplication with the verb matrix, yielding a sentence meaning in $N \otimes N$. Since the tensor product is not the composition operation, this approach does not suffer from the 'grammatical explosion' problem of Smolensky, and the size of the sentence representation is the same as the size of the verb. Sentence meanings can be compared to one another by taking their angle in sentence space. One problem with this approach is that sentences with transitive verbs live in a different space than sentences with intransitive verbs, so sentences with different grammatical structures cannot be directly compared.

Kartsaklis et al. [34] introduce another sentence space, by setting $S = N$. To achieve this, while training the verb in the same way as the relational method, they embed the $N \otimes N$ tensor into $N \otimes N \otimes N$ space, by copying the subject or object slices of the original tensor; for example, in the copy-object method the representation for V is

$$V = \sum_i c_i \, \overrightarrow{n_i^{(s)}} \otimes \overrightarrow{n_i^{(o)}} \otimes \overrightarrow{n_i^{(o)}}$$

Representing sentence meaning in the noun space confers the advantage that words, phrases, and sentences of any grammatical structure can be compared with one another, but it is an open question whether the features in the noun space, whether distributional or not, are appropriate or adequate for representing sentence meaning.

An alternative way of learning the transitive verb tensor parameters is presented in [25], using a process analogous to B&Z's process for learning adjectives. Two linear regression steps are performed. In the first step, a matrix is learned representing verb-object constructions, that is, a verb that has already been paired with its object. For example, the matrix for $\overline{\textit{eat meat}}$ is learned as a mapping from corpus instances such as *dogs* and *dogs eat meat*. The full tensor for $\overline{\textit{eat}}$ is then learned with *meat* as input and the $\overline{\textit{eat meat}}$ matrix as output.

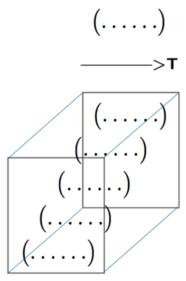

Figure 6: Type-raising places the vector of a subject *NP* along the diagonal of a third-order tensor.

Essentially, the subject mapping is learned in the first step, and the object mapping in the second. The contexts for the noun and verb-object vectors are words that co-occur with them intra-sententially.

Polajnar et al. [57], following Krishnamurthy and Mitchell [37], investigate a non-distributional sentence space, the “plausibility space” described by Clark [10, 11]. Here the sentence space is one- or two-dimensional, with sentence meaning either a real number between 0 and 1, or a probability distribution over the classes *plausible* and *implausible*. A verb tensor is learned using single-step linear regression, with the training data consisting of positive (plausible) and negative (implausible) subject-verb-object examples. Positive examples are attested in the corpus, and negative examples for a given verb have frequency-matched random nouns substituted in the argument positions. One tensor investigated has dimensions $K \times K \times S$, where K is the number of noun dimensions and S has two dimensions, each ranging from 0 to 1. The parameters of the tensor are learned so that when combined with a subject and object, a plausibility judgement is produced. Another method proposed by Polajnar et al. [57] is KKMat, in which the verb is learned as a $K \times K$ matrix. In this case composition with the verb and object results in a scalar plausibility value.

3.4 Summary of the Tensor-based Semantics

Since the multilinear maps employed here are functions, manipulated in a standard way by CCG’s combinatory rules, the standard features of CCG carry over to the tensor-based semantics. For example, Maillard et al. [44] show how subject type-raising has an elegant interpretation in which the vector of the subject *NP* being type-raised is placed “in steps” along a diagonal of a third-order tensor of type $\mathsf{S} \otimes \mathsf{S} \otimes \mathsf{N}$ (corresponding to the syntactic type $S/(S\backslash NP)$), with zeroes everywhere else. Figure 6 shows this in diagrammatic form, with the subject noun vector at the top. Type-raising, in combination with function composition, results in a form of additional, often-called “spurious”, ambiguity in which even simple subject-verb-object sentences have multiple derivations.

Crucially, the sentence vector resulting from the tensor contractions is the same in both cases, whether obtained through two function applications, or through the type-raising and composition route. The *Pat might kiss Sandy* example earlier also exhibits multiple derivations, depending on whether function composition is used, or just application. Again, the resulting S vector is the same in both cases.

The framework we have described derives a sentence vector for *any* syntactic derivation resluting in S, including those which use the additional combinatory rules of CCG [44], providing a complete recipe for the meaning composition process for any sentence. One way to consider the framework is that it provides a "homomorphic passage" from word to sentence meaning [16], in a similar spirit to the work of Montague [52], but using the mathematics of multilinear algebra rather than set theory.

In practice, syntactic categories such as $((N/N)/(N/N))/((N/N)/(N/N))$ are not uncommon in the wide-coverage grammar of CCGBank [33]; such a category would require an *8th-order* tensor. The combination of many word-category pairs and higher-order tensors results in a huge number of parameters to be learned in any implementation. As a solution to this problem, various ways to reduce the number of parameters are being investigated, for example using tensor decomposition techniques [36, 22], and removing some of the interactions encoded in the tensor by using only matrices to encode each predicate-argument combination [57, 53]. Many challenges remain before a type-driven compositional distributional semantics can be realised in full, similar to the work of Bos for the model-theoretic case [8, 7], but in this section we have set out the theoretical framework for such an implementation.

References

[1] Diederik Aerts and Liane Gabora. A state-context-property model of concepts and their combinations II: A Hilbert space representation. *Kybernetes*, 34(1&2):192–221, 2005.

[2] Michael Auli and Adam Lopez. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of the EMNLP Conference*, Edinburgh, UK, 2011.

[3] Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

[4] M. Baroni, G. Dinu, and G. Kruszewski. 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, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 238–247, 2014.

[5] M. Baroni and R. Zamparelli. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the EMNLP Conference*, pages 1183–1193, Boston, MA, 2010.

[6] Marco Baroni, Raffaella Bernardi, and Roberto Zamparelli. Frege in space: A program for compositional distributional semantics. *Linguistic Issues in Language Technology*, 9:5–110, 2014.

[7] Johan Bos. Towards wide-coverage semantic interpretation. In *Proceedings of the Sixth International Workshop on Computational Semantics (IWCS-6)*, pages 42–53, Tilburg, The Netherlands, 2005.

[8] Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. Wide-coverage semantic representations from a CCG parser. In *Proceedings of COLING-04*, pages 1240–1246, Geneva, Switzerland, 2004.

[9] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the NAACL*, pages 132–139, Seattle, WA, 2000.

[10] Stephen Clark. Type-driven syntax and semantics for composing meaning vectors. In Chris Heunen, Mehrnoosh Sadrzadeh, and Edward Grefenstette, editors, *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse.* Oxford University Press, 2013.

[11] Stephen Clark. Vector space models of lexical meaning. In Shalom Lappin and Chris Fox, editors, *Handbook of Contemporary Semantics - second edition*, chapter 16. Wiley-Blackwell, 2015.

[12] Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. A compositional distributional model of meaning. In *Proceedings of QI-2008*, pages 133–140, Oxford, UK, 2008.

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

[14] Stephen Clark and Stephen Pulman. Combining symbolic and distributional models of meaning. In *Proceedings of AAAI Spring Symposium on Quantum Interaction*, Stanford, CA, 2007. AAAI Press.

[15] Daoud Clarke. A context-theoretic framework for compositionality in distributional semantics. *Computational Linguistics*, 38(1):41–71, 2012.

[16] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis 36: A Festschrift for Joachim (Jim) Lambek*, 2010.

[17] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, 1999.

[18] D.R. Dowty, R.E. Wall, and S. Peters. *Introduction to Montague Semantics.* Dordrecht, 1981.

[19] Luana Fagarasan, Eva Maria Vecchi, and Stephen Clark. From distributional semantics to feature norms: grounding semantic models in human perceptual data. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS-15)*, London, UK, 2015.

[20] Jerry Fodor and Zenon Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.

[21] P.W. Foltz, W. Kintsch, and T.K. Landauer. The measurement of textual coherence with latent semantic analysis. *Discourse Process*, 15:285–307, 1998.

[22] Daniel Fried, Tamara Polajnar, and Stephen Clark. 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)*, pages 731–736, Beijing, China, 2015.

[23] Dan Garrette, Katrin Erk, and Raymond Mooney. Integrating logical representations with probabilistic information using Markov Logic. In *Proceedings of IWCS*, Oxford, UK, 2011.

[24] Edward Grefenstette. *Category-Theoretic Quantitative Compositional Distributional Models of Natural Language Semantics.* PhD thesis, University of Oxford, 2013.

[25] Edward Grefenstette, Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh, and Marco Baroni. Multi-step regression learning for compositional distributional semantics. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS-13)*, Potsdam, Germany, 2013.

[26] Edward Grefenstette and Mehrnoosh Sadrzadeh. Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1394–1404, Edinburgh, Scotland, UK, July 2011.

[27] Edward Grefenstette, Mehrnoosh Sadrzadeh, Stephen Clark, Bob Coecke, and Stephen Pulman. Concrete sentence spaces for compositional distributional models of meaning. In *Proceedings of the 9th International Conference on Computational Semantics (IWCS-11)*, pages 125–134, Oxford, UK, 2011.

[28] David Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, University of California in Santa Cruz, 1999.

[29] Aurelie Herbelot and Ann Copestake. Lexicalised compositionality. University of Cambridge Computer Laboratory, unpublished manuscript, 2015.

[30] Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. *Proceedings of ACL, Sofia, Bulgaria, August. Association for Computational Linguistics*, 2013.

[31] Karl Moritz Hermann, Edward Grefenstette, and Phil Blunsom. "Not not bad" is not "bad": A distributional account of negation. In *Proceedings of the ACL Workshop on Continuous Vector Space Models and their Compositionality*, 2013.

[32] Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA, 2002.

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

[34] Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Stephen Pulman. A unified sentence space for categorical distributional-compositional semantics: Theory and experiments. In *Proceedings of COLING*, pages 549–558, 2012.

[35] W. Kintsch. Predication. *Cognitive Science*, 25:173–202, 2001.

[36] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[37] Jayant Krishnamurthy and Tom M. Mitchell. Vector space semantic parsing: A framework for compositional vector space models. In *Proceedings of the 2013 ACL Workshop on Continuous Vector Space Models and their Compositionality*, Sofia, Bulgaria, 2013.

[38] Joachim Lambek. *From Word to Sentence. A Computational Algebraic Approach to Grammar.* Polimetrica, 2008.

[39] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: the latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104(2):211–240, 1997.

[40] Omer Levy and Yoav Goldberg. Neural word embeddings as implicit matrix factorization. In *Proceedings of NIPS*, 2014.

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

[42] Mike Lewis and Mark Steedman. Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192, 2013.

[43] Mike Lewis and Mark Steedman. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP-14)*, Doha, Qatar, 2014.

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

[45] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[46] Mitchell Marcus, Beatrice Santorini, and Mary Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[47] Ken McRae, George S Cree, Mark S Seidenberg, and Chris McNorgan. Semantic feature production norms for a large set of living and nonliving things. *Behavior research methods*, 37(4):547–559, 2005.

[48] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013.*, pages 3111–3119, Lake Tahoe, Nevada, United States, 2013.

[49] Dmitrijs Milajevs, Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Matthew Purver. Evaluating neural word representations in tensor-based compositional settings. In *Proceedings of EMNLP*, Doha, Qatar, 2014.

[50] J. Mitchell and M. Lapata. Vector-based models of semantic composition. In *Proceedings of ACL*, pages 236–244, Columbus, OH, 2008.

[51] Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive Science*, 34:1388–1439, 2010.

[52] R. Montague. English as a formal language. In R. Montague, editor, *Formal Philosophy*. Yale University Press, New Haven, CT, 1974.

[53] D. Paperno, N. Pham, and M. Baroni. A practical and linguistically-motivated approach to compositional distributional semantics. In *Proceedings of ACL 2014 (52nd Annual Meeting of the Association for Computational Linguistics)*, pages 90–99, 2014.

[54] Tony A. Plate. Holographic reduced representations: Convolution algebra for compositional distributed representations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 30–35, Sydney, Australia, 1991.

[55] Tony A. Plate. Analogy retrieval and processing with distributed vector representations. *Expert Systems*, 17(1):29–40, 2000.

[56] Tamara Polajnar and Stephen Clark. 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)*, Gothenburg, Sweden, 2014.

[57] Tamara Polajnar, Luana Fagarasan, and Stephen Clark. Reducing dimensions of tensors in type-driven distributional semantics. In *Proceedings of EMNLP 2014*, Doha, Qatar, 2014.

[58] Tamara Polajnar, Laura Rimell, and Stephen Clark. Evaluation of simple distributional compositional operations on longer texts. In *Proceedings of the 9th Language Resources and Evaluation Conference (LREC 2014)*, Reykjavik, Iceland, 2014.

[59] Tamara Polajnar, Laura Rimell, and Stephen Clark. An exploration of discourse-based sentence spaces for compositional distributional semantics. In *Proceedings of the EMNLP 2015 Workshop on Linking Models of Lexical, Sentential and Discourse-Level Semantics (LSDSem 2015)*, pages 1–11, Lisbon, Portugal, 2015.

[60] J. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[61] E Rosch and C B Mervis. Family resemblances: Studies in the internal structures of categories. *Cognitive Psychology*, 7:573–605, 1975.

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

[63] Carina Silberer, Vittorio Ferrari, and Mirella Lapata. Models of semantic representation with visual attributes. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 572–582, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[64] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.

[65] Paul Smolensky and Geraldine Legendre. *The Harmonic Mind: from neural computation to optimality-theoretic grammar.* MIT Press, Cambridge, MA, 2006.

[66] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1201–1211, Jeju, Korea, 2012.

[67] Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of NIPS Deep Learning and Unsupervised Feature Learning Workshop*, Whistler, BC, Canada, 2010.

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

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

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

[71] Michael White and Rajakrishnan Rajkumar. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 410–419, Singapore, August 2009. Association for Computational Linguistics.

[72] Fabio Massimo Zanzotto, Lorenzo Ferrone, and Marco Baroni. When the whole is not greater than the combination of its parts: A "decompositional" look at compositional distributional semantics. *Computational Linguistics*, 41(1), 2015.

[73] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, Edinburgh, UK, 2005.

[74] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Conference on Empirical Methods in Natural Language Processing (EMNLP-07)*, Prague, Czech Republic, 2007.