A Type-Driven Tensor-Based Semantics for CCG

Jean Maillard
University of Cambridge
Computer Laboratory
jm864@cam.ac.uk

Stephen Clark
University of Cambridge
Computer Laboratory
sc609@cam.ac.uk

Edward Grefenstette
University of Oxford
Department of Computer Science
edward.grefenstette@cs.ox.ac.uk

Abstract

This paper shows how the tensor-based semantic framework of Coecke et al. can be seamlessly integrated with Combinatory Categorial Grammar (CCG). The integration follows from the observation that tensors are linear maps, and hence can be manipulated using the combinators of CCG, including type-raising and composition. Given the existence of robust, wide-coverage CCG parsers, this opens up the possibility of a practical, type-driven compositional semantics based on distributional representations.

1 Intoduction

In this paper we show how tensor-based distributional semantics can be seamlessly integrated with Combinatory Categorial Grammar (CCG, Steedman (2000)), building on the theoretical discussion in Grefenstette (2013). Tensor-based distributional semantics represents the meanings of words with particular syntactic types as tensors whose semantic type matches that of the syntactic type (Coecke et al., 2010). For example, the meaning of a transitive verb with syntactic type $(S\backslash NP)/NP$ is a 3rd-order tensor from the tensor product space $N \otimes S \otimes N$. 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.

Tensor-based semantics arises from the desire to enhance distributional semantics with some compositional structure, in order to make distributional semantics more of a complete semantic theory, and to increase its utility in NLP applications. There are a number of suggestions for how to add compositionality to a distributional semantics (Clarke, 2012; Pulman, 2013; Erk, 2012). One approach is to assume that the meanings of all words are represented by context vectors, and then combine those vectors using some operation, such as vector addition, element-wise multiplication, or tensor product (Clark and Pulman, 2007; Mitchell and Lapata, 2008). A more sophisticated approach, which is the subject of this paper, is to adapt the compositional process from formal semantics (Dowty et al., 1981) and attempt to build a distributional representation in step with the syntactic derivation (Coecke et al., 2010; Baroni et al., 2013). Finally, there is a third approach using neural networks, which perhaps lies in between the two described above (Socher et al., 2010; Socher et al., 2012). Here compositional distributed representations are built using matrices operating on vectors, with all parameters learnt through a supervised learning procedure intended to optimise performance on some NLP task, such as syntactic parsing or sentiment analysis. The approach of Hermann and Blunsom (2013) conditions the vector combination operation on the syntactic type of the combinands, moving it a little closer to the more formal semantics-inspired approaches.

The remainder of the Introduction gives a short summary of distributional semantics. The rest of the paper introduces some mathematical notation from multi-linear algebra, including Einstein notation, and then shows how the combinatory rules of CCG, including type-raising and composition, can be applied directly to tensor-based semantic representations. As well as describing a tensor-based semantics for CCG, a further goal of this paper is to present the compositional framework of Coecke et al. (2010), which is based on category theory, to a computational linguistics audience using only the mathematics of multi-linear algebra.

1.1 Distributional Semantics

We assume a basic knowledge of distributional semantics (Grefenstette, 1994; Schütze, 1998). Re-

cent inroductions to the topic include Turney and Pantel (2010) and Clark (2014).

A potentially useful distinction for this paper, and one not commonly made, is between *distributional* and *distributed* representations. Distributional representations are inherently *contextual*, and rely on the frequently quoted dictum from Firth that "you shall know a word from the company it keeps" (Firth, 1957; Pulman, 2013). This leads to the so-called distributional hypothesis that words that occur in similar contexts tend to have similar meanings, and to various proposals for how to implement this hypothesis (Curran, 2004), including alternative definitions of context; alternative weighting schemes which emphasize the importance of some contexts over others; alternative similarity measures; and various dimensionality reduction schemes such as the well-known LSA technique (Landauer and Dumais, 1997). An interesting conceptual question is whether a similar distributional hypothesis can be applied to phrases and larger units: is it the case that sentences, for example, have similar meanings if they occur in similar contexts? Work which does extend the distributional hypothesis to larger units includes Baroni and Zamparelli (2010), Clarke (2012), and Baroni et al. (2013).

Distributed representations, on the other hand, can be thought of simply as vectors (or possibly higher-order tensors) of real numbers, where there is no *a priori* interpretation of the basis vectors. Neural networks can perhaps be categorised in this way, since the resulting vector representations are simply sequences of real numbers resulting from the optimisation of some training criterion on a training set (Collobert and Weston, 2008; Socher et al., 2010). Whether these distributed representations can be given a contextual interpretation depends on how they are trained.

One important point for this paper is that *the tensor-based compositional process makes no assumptions about the interpretation of the tensors*. Hence in the remainder of the paper we make no reference to how noun vectors or verb tensors, for example, can be acquired (which, for the case of the higher-order tensors, is a wide open research question). However, in order to help the reader who would prefer a more grounded discussion, one possibility is to obtain the noun vectors using standard *distributional* techniques (Curran, 2004), and learn the higher-order tensors using recent techniques from "recursive" neural networks (Socher et al., 2010). Another possibility is suggested by Grefenstette et al. (2013), extending the learning technique based on linear regression from Baroni and Zamparelli (2010) in which "gold-standard" distributional representations are assumed to be available for some phrases and larger units.

2 Mathematical Preliminaries

The tensor-based compositional process relies on taking dot (or inner) products between vectors and higher-order tensors. Dot products, and a number of other operations on vectors and tensors, can be conveniently written using Einstein notation (also referred to as the Einstein summation convention). In the rest of the paper we assume that the vector spaces are over the field of real numbers.

2.1 Einstein Notation

The squared amplitude of a vector $\boldsymbol{v} \in \mathbb{R}^n$ is given by:

$$|\boldsymbol{v}|^2 = \sum_{i=1}^{n} v_i v_i$$

Similarly, the dot product of two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ is given by:

$$\boldsymbol{v} \cdot \boldsymbol{w} = \sum_{i=1}^{n} v_i w_i$$

Denote the components of an $m \times n$ real matrix $\mathbf{A}$ by A_{ij} for $1 \leq i \leq m$ and $1 \leq j \leq n$. Then the matrix-vector product of $\mathbf{A}$ and $\boldsymbol{v} \in \mathbb{R}^n$ gives a vector $\mathbf{A}\boldsymbol{v} \in \mathbb{R}^m$ with components:

$$(\mathbf{A}\boldsymbol{v})_i = \sum_{j=1}^{n} A_{ij} v_j$$

We can also multiply an $n \times m$ matrix $\mathbf{A}$ and an $m \times o$ matrix $\mathbf{B}$ to produce an $n \times o$ matrix $\mathbf{AB}$ with components:

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

The previous examples are some of the most common operations in linear algebra, and they all involve sums over repeated indices. They can be simplified by introducing the *Einstein summation convention*: summation over the relevant range is implied on every component index that occurs

twice. Pairs of indices that are summed over are known as *contracted*, while the remaining indices are known as *free*. Using this convention, the above operations can be written as:

$|\boldsymbol{v}|^2 = v_i v_i$

$\boldsymbol{v} \cdot \boldsymbol{w} = v_i w_i$

$(\mathbf{A}\boldsymbol{v})_i = A_{ij} v_j$, i.e. the contraction of $\boldsymbol{v}$ with the second index of $\mathbf{A}$

$(\mathbf{AB})_{ij} = A_{ik} B_{kj}$, i.e. the contraction of the second index of $\mathbf{A}$ with the first of $\mathbf{B}$

Note how the number of free indices is always conserved between the left- and right-hand sides in these examples. For instance, while the last equation has two indices on the left and four on the right, the two extra indices on the right are contracted. Hence *counting the number of free indices* can be a quick way of determining what type of object is given by a certain mathematical expression in Einstein notation: no free indices means that an operation yields a scalar number, one free index means a vector, two a matrix, and so on.

2.2 Tensors

Linear Functionals Given a finite-dimensional vector space $\mathbb{R}^n$ over $\mathbb{R}$, a *linear functional* is a linear map $a : \mathbb{R}^n \to \mathbb{R}$.

Let a vector $\boldsymbol{v}$ have components v_i in a fixed basis. Then the result of applying a linear functional a to $\boldsymbol{v}$ can be written as:

$$a(\boldsymbol{v}) = a_1 v_1 + \cdots + a_n v_n = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}$$

The numbers a_i are the components of the linear functional, which can also be pictured as a row vector. Since there is a one-to-one correspondence between row and column vectors, the above equation is equivalent to:

$$v(\boldsymbol{a}) = a_1 v_1 + \cdots + a_n v_n = \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

Using Einstein convention, the equations above can be written as:

$$a(\boldsymbol{v}) = v_i a_i = v(\boldsymbol{a})$$

Thus every finite-dimensional vector is a linear functional, and vice versa. Row and column vectors are examples of *first-order tensors*.

Definition 1 (First-order tensor)**.** *Given a vector space V over the field $\mathbb{R}$, a first-order tensor T can be defined as:*

- *an element of the vector space V,*
- *a linear map $T : V \to \mathbb{R}$,*
- *a $|V|$-dimensional array of numbers T_i, for $1 \leq i \leq |V|$.*

These three definitions are all equivalent. Given a first-order tensor described using one of these definitions, it is trivial to find the two other descriptions.

Matrices An $n \times m$ matrix $\mathbf{A}$ over $\mathbb{R}$ can be represented by a two-dimensional array of real numbers A_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

Via matrix-vector multiplication, the matrix $\mathbf{A}$ can be seen as a linear map $\mathbf{A} : \mathbb{R}^m \to \mathbb{R}^n$. It maps a vector $\boldsymbol{v} \in \mathbb{R}^m$ to a vector

$$\begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix},$$

with components

$$\mathbf{A}(\boldsymbol{v})_i = A_{ij} v_j.$$

We can also contract a vector with the first index of the matrix, which gives us a map $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}^m$. This corresponds to the operation

$$\begin{pmatrix} w_1 & \cdots & w_n \end{pmatrix} \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{pmatrix},$$

resulting in a vector with components

$$(\boldsymbol{w}^T \mathbf{A})_i = A_{ji} w_j.$$

We can combine the two operations and see a matrix as a map $\mathbf{A} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, defined by:

$$\boldsymbol{w}^T \mathbf{A} \boldsymbol{v} = \begin{pmatrix} w_1 & \cdots & w_n \end{pmatrix} \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix}$$

In Einstein notation, this operation can be written as

$$w_i A_{ij} v_j,$$

which yields a scalar (constant) value, consistent with the fact that all the indices are contracted.

Finally, matrices can also be characterised in terms of Kronecker products. Given two vectors $\boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^m$, their Kronecker product $\boldsymbol{v} \otimes \boldsymbol{w}$ is a matrix

$$\boldsymbol{v} \otimes \boldsymbol{w} = \begin{pmatrix} v_1 w_1 & \cdots & v_1 w_m \\ \vdots & \ddots & \vdots \\ v_n w_1 & \cdots & v_n w_m \end{pmatrix},$$

with components

$$(\boldsymbol{v} \otimes \boldsymbol{w})_{ij} = v_i w_j.$$

It is a general result in linear algebra that any $n \times m$ matrix can be written as a finite sum of Kronecker products $\sum_k \boldsymbol{x^{(k)}} \otimes \boldsymbol{y^{(k)}}$ of a set of vectors $\boldsymbol{x^{(k)}}$ and $\boldsymbol{y^{(k)}}$. Note that the sum over k is written explicitly as it would not be implied by Einstein notation: this is because the index k does not range over vector/matrix/tensor components, but over a set of vectors, and hence that index appears in brackets.

An $n \times m$ matrix is an element of the tensor space $\mathbb{R}^n \otimes \mathbb{R}^m$, and it can also be seen as a linear map $\mathbf{A} : \mathbb{R}^n \otimes \mathbb{R}^m \to \mathbb{R}$. This is because, given a matrix $\mathbf{B}$ with decomposition $\sum_k \boldsymbol{x^{(k)}} \otimes \boldsymbol{y^{(k)}}$, the matrix $\mathbf{A}$ can act as follows:

$$\begin{aligned} \mathbf{A}(\mathbf{B}) &= A_{ij} \sum_k x_i^{(k)} y_j^{(k)} \\ &= \sum_k \begin{pmatrix} x_1^{(k)} & \cdots & x_n^{(k)} \end{pmatrix} \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{pmatrix} \begin{pmatrix} y^{(k)} \\ \vdots \\ y_m^{(k)} \end{pmatrix} \\ &= A_{ij} B_{ij}. \end{aligned}$$

Again, counting the number of free indices in the last line tells us that this operation yields a scalar.

Matrices are examples of *second-order tensors*.

Definition 2 (Second-order tensor)**.** *Given vector spaces V, W over the field $\mathbb{R}$, a second-order tensor T can be defined as:*

- *an element of the vector space $V \otimes W$,*
- *a $|V| \times |W|$-dimensional array of numbers T_{ij}, for $1 \leq i \leq |V|$ and $1 \leq j \leq |W|$,*
- *a (multi-) linear map:*
  - $T : V \to W$,
  - $T : W \to V$,
  - $T : V \times W \to \mathbb{R}$ *or* $T : V \otimes W \to \mathbb{R}$.

Again, these definitions are all equivalent. Most importantly, the four types of maps given in the definition are isomorphic. Therefore specifying one map is enough to specify all the others.

Tensors We can generalise these definitions to the more general concept of *tensor*.

Definition 3 (Tensor)**.** *Given vector spaces $V_1, \ldots, V_k$ over the field $\mathbb{R}$, a k^{th}-order tensor T is defined as:*

- *an element of the vector space $V_1 \otimes \cdots \otimes V_k$,*
- *a $|V_1| \times \cdots \times |V_k|$, k^{th}-dimensional array of numbers $T_{i_1 \cdots i_k}$, for $1 \leq i_j \leq |V_j|$,*
- *a multi-linear map $T : V_1 \times \cdots \times V_k \to \mathbb{R}$.*

3 Tensor-Based CCG Semantics

In this section we show how CCG's syntactic types can be given tensor-based meaning spaces, and how the combinator's employed by CCG to combine syntactic categories carry over to those meaning spaces, maintaining what is often described as CCG's "transparent interface" between syntax and semantics. Here are some example syntactic types, and the corresponding tensor spaces containing the meanings of the words with those types (using the notation *syntactic type* : *semantic type*).

We first assume that all atomic types have meanings living in distinct vector spaces:

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

The recipe for determining the meaning space of a complex syntactic type is to replace each atomic type with its corresponding vector space and the slashes with tensor product operators:

- 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 matrix 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), what the result is on the S basis vector given a value on the N basis vector.

Now we describe how the combinatory rules carry over to the meaning spaces.

3.1 Application

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

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

In a traditional semantics for CCG, if function application is applied in the syntax, then function application applies also in the semantics (Steedman, 2000). This is also true of the tensor-based semantics. 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 matrix to the vector representing the meaning of the NP. Applying (multi-)linear maps in (multi-)linear algebra is equivalent to applying tensor contraction to the combining tensors. Here is the case for an intransitive verb:

Pat	*walks*
NP	$S\backslash NP$
N	$\mathsf{S} \otimes \mathsf{N}$

Let *Pat* be assigned a vector $P \in \mathsf{N}$ and *walks* be assigned a second-order tensor $W \in \mathsf{S} \otimes \mathsf{N}$. Using the backward application combinator corresponds to feeding P, an element of N, into W, seen as a function $\mathsf{N} \to \mathsf{S}$. In terms of tensor contraction, this is the following operation:

$$W_{ij} P_j.$$

Here we use the convention that the indices maintain the same order as the syntactic type. Therefore, in the tensor of an object of type X/Y, the first index corresponds to the type X and the second to the type Y. That is why, when performing the contraction corresponding to *Pat walks*, $P \in \mathsf{N}$ is contracted with the *second* index of $W \in \mathsf{S} \otimes \mathsf{N}$, and not the first.[1] The first index of W 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 applying backward application to *Pat walks* yields a meaning vector in S.

Forward application is performed in the same manner. Consider the following example:

Pat	*kisses*	*Sandy*
NP	$(S\backslash NP)/NP$	NP
N	$\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$	N

with corresponding tensors $P \in \mathsf{N}$ for *Pat*, $K \in \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$ for *kisses* and $Y \in \mathsf{N}$ for *Sandy*.

The forward application deriving the type of *kisses Sandy* corresponds to

$$K_{ijk} Y_k,$$

where Y is contracted with the third index of K 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 *walks* example.

3.2 Composition

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

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

Composition in the semantics also reduces to a form of tensor contraction. Consider the following example, in which *might* can combine with *kiss* using forward composition:

Pat	*might*	*kiss*	*Sandy*
NP	$(S\backslash NP)/(S\backslash NP)$	$(S\backslash NP)/NP$	NP
N	$\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{S} \otimes \mathsf{N}$	$\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$	N

[1] The particular order of the indices is not important, as long as a convention such as this one is decided upon and consistently applied to all types (so that tensor contraction contracts the relevant tensors from each side when a combinator is used).

with tensors $M \in \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{S} \otimes \mathsf{N}$ for *might* and $K \in \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$ for *kiss*. Combining the meanings of *might* and *kiss* corresponds to the following operation:

$$M_{ijkl}K_{klm},$$

yielding a tensor in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$, which is the correct semantic type for a phrase with syntactic type $(S\backslash NP)/NP$. Backward composition is performed analogously.

3.3 Backward-Crossed Composition

English also requires the use of backward-crossed composition (Steedman, 2000):

$$X/Y \quad Z\backslash X \quad \Longrightarrow Z/Y \qquad (<_{\mathbf{B}_\times})$$

In tensor terms, this is the same as forward composition; we just need to make sure that the contraction matches up the correct parts of each tensor correctly. Consider the following backward-crossed composition:

$$(S\backslash NP)/NP \quad (S\backslash NP)\backslash(S\backslash NP) \quad \Rightarrow_{<_{\mathbf{B}_\times}} (S\backslash NP)/NP$$

Let the two items on the left-hand side be represented by tensors $A \in \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$ and $B \in \mathsf{S} \otimes \mathsf{N} \otimes \mathsf{S} \otimes \mathsf{N}$. Then, combining them with backward-crossed composition in tensor terms is

$$B_{ijkl}A_{klm},$$

resulting in a tensor in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$ (corresponding to the indices i, j and m). Note that we have reversed the order of tensors in the contraction to make the matching of the indices more transparent; however, tensor contraction is commutative (since it corresponds to a sum over products) so the order of the tensors does not affect the result.

3.4 Type-raising

The forward ($>_\mathbf{T}$) and backward ($<_\mathbf{T}$) type-raising rules are:

$$\begin{array}{lll} X & \Longrightarrow T/(T\backslash X) & (>_\mathbf{T}) \\ X & \Longrightarrow T\backslash(T/X) & (<_\mathbf{T}) \end{array}$$

where T is a variable ranging over categories.

Suppose we are given an item of atomic type Y, with corresponding vector $A \in \mathsf{Y}$. If we apply forward type-raising to it, we get a new tensor of type $A' \in \mathsf{T} \otimes \mathsf{T} \otimes \mathsf{Y}$. Now suppose the item of type Y is followed by another item of type $X\backslash Y$, with tensor $B \in \mathsf{X} \otimes \mathsf{Y}$. A phrase consisting of two words with types Y and $X\backslash Y$ can be parsed in two different ways:

- $Y \;\; X\backslash Y \Rightarrow X$, by backward application;
- $Y \;\; X\backslash Y \Rightarrow_\mathbf{T} X/(X\backslash Y) \;\; X\backslash Y$, by forward type-raising, and $X/(X\backslash Y) \;\; X\backslash Y \Rightarrow X$, by forward application.

Both ways of parsing this sentence yield an item of type X, and crucially the meaning of the resulting item should be the same in both cases.[2] This property of type-raising provides an avenue into determining what the tensor representation for the type-raised category should be, since the tensor representations must also be the same:

$$A_jB_{ij} = A'_{ijk}B_{jk}.$$

Moreover, this equation must hold for all items, B. As a concrete example, the requirement says that a subject NP combining with a verb phrase $S\backslash NP$ must produce the same meaning for the two alternative derivations, irrespective of the verb phrase. This is equivalent to the requirement that

$$A_jB_{ij} = A'_{ijk}B_{jk}, \quad \forall B \in \mathsf{X} \otimes \mathsf{Y}.$$

So to arrive at the tensor representation, we simply have to solve the tensor equation above. We start by renaming the dummy index j on the left-hand side:

$$A_kB_{ik} = A'_{ijk}B_{jk}.$$

We then insert a Kronecker delta ($\delta_{ij} = 1$ if $i = j$ and 0 otherwise):

$$A_kB_{jk}\delta_{ij} = A'_{ijk}B_{jk}.$$

Since the equation holds for all B, we are left with

$$A'_{ijk} = \delta_{ij}A_k,$$

which gives us a recipe for performing type-raising in a tensor-based model. The recipe is particularly simple and elegant: it corresponds to inserting the vector being type-raised into the 3rd-order tensor at all places where the first two indices are equal (with the rest of the elements in the 3rd-order tensor being zero). For example, to type-raise a subject NP, its meaning vector in N is placed in the 3rd-order tensor $\mathsf{S}\otimes\mathsf{S}\otimes\mathsf{N}$ at all places where the indices of the two S dimensions are the same. Visually, the 3rd-order tensor corresponding to the meaning of the type-raised category is

[2] This property of CCG resulting from the use of type-raising and composition is sometimes referred to as "spurious ambiguity".

a cubiod in which the noun vector is repeated a number of times (once for each sentence index), resulting in a series of "steps" progressing diagonally from the bottom of the cuboid to the top (assuming a particular orientation).

The discussion so far has been somewhat abstract, so to finish this section we include some more examples with CCG categories, and show that the tensor contraction operation has an intuitive similarity with the "cancellation law" of categorial grammar which applies in the syntax.

First consider the example of a subject NP with meaning A, combining with a verb phrase $S\backslash NP$ with meaning B, resulting in a sentence with meaning C. In the syntax, the two NPs cancel. In the semantics, for each basis of the sentence space S we perform an inner product between two vectors in N:

$$C_i = A_j B_{ij}$$

Hence, *inner products in the tensor space correspond to cancellation in the syntax*.

This correspondence extends to complex arguments, and also to composition. Consider the subject type-raising case, in which a subject NP with meaning A in $\mathsf{S} \otimes \mathsf{S} \otimes \mathsf{N}$ combines with a verb phrase $S\backslash NP$ with meaning B, resulting in a sentence with meaning C. Again we perform inner product operations, but this time the inner product is between two matrices:[3]

$$C_i = A_{ijk} B_{jk}$$

Note that two matrices are "cancelled" for each basis vector of the sentence space (i.e. for each index i in C_i).

As a final example, consider the forward composition from earlier, in which a modal verb with meaning A in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{S} \otimes \mathsf{N}$ combines with a transitive verb with meaning B in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$ to give a transitive verb with meaning C in $\mathsf{S} \otimes \mathsf{N} \otimes \mathsf{N}$. Again the cancellation in the syntax corresponds to inner products between matrices, but this time we need an inner product for each combination of 3 indices:

$$C_{ijk} = A_{ijlm} B_{lmk}$$

[3]To be more precise, the two matrices can be thought of as vectors in the tensor space $\mathsf{S} \otimes \mathsf{N}$ and the inner product is between these vectors. Another way to think of this operation is to "linearize" the two matrices into vectors and then perform the inner product on these vectors.

For each i, j, k, two matrices — corresponding to the l, m indices above — are "cancelled".

This intuitive explanation extends to arguments with any number of slashes. For example, a composition where the cancelling categories are $(N/N)/(N/N)$ would require inner products between 4th-order tensors in $\mathsf{N} \otimes \mathsf{N} \otimes \mathsf{N} \otimes \mathsf{N}$.

4 Related Work

The tensor-based semantics presented in this paper is effectively an extension of the Coecke et al. (2010) framework to CCG, re-expressing in Einstein notation the existing categorical CCG extension in Grefenstette (2013), which itself builds on an earlier Lambek Grammar extension to the framework by Coecke et al. (2013).

This work also bears some similarity to the treatment of categorial grammars presented by Baroni et al. (2013), which it effectively encompasses by expressing the tensor contractions described by Baroni *et al.* as Einstein summations. However, this paper also covers CCG-specific operations not discussed by Baroni *et al.*, such as type-raising and composition.

One difference between this paper and the original work by Coecke et al. (2010) is that they use pregroups as the syntactic formalism (Lambek, 2008), a context-free variant of categorial grammar. In pregroups, cancellation in the syntax is always between two atomic categories (or more precisely, between an atomic category and its "adjoint"), whereas in CCG the arguments in complex categories can be complex categories themselves. To what extent this difference is significant remains to be seen. For example, one area where this may have an impact is when non-linearities are added after contractions. Since the CCG contractions with complex arguments happen "in one go", whereas the corresponding pregroup cancellation in the semantics would be a series of contractions, many more non-linearities would be added in the pregroup case.

Krishnamurthy and Mitchell (2013) is based on a similar insight to this paper – that CCG provides combinators which can manipulate functions operating over vectors. Krishnamurthy and Mitchell consider the function application case, whereas we have shown how the type-raising and composition operators apply naturally in this setting also.

5 Conclusion

This paper provides a theoretical framework for the development of a compositional distributional semantics for CCG. Given the existence of robust, wide-coverage CCG parsers (Clark and Curran, 2007; Hockenmaier and Steedman, 2002), together with various techniques for learning the tensors, the opportunity exists for a practical implementation. However, there are significant engineering difficulties which need to be overcome.

Consider adapting the neural-network learning techniques of Socher et al. (2012) to this problem.[4] In terms of the number of tensors, the lexicon would need to contain a tensor for every word-category pair; this is at least an order of magnitude more tensors then the number of matrices learnt in existing work (Socher et al., 2012; Hermann and Blunsom, 2013). Furthermore, the order of the tensors is now higher. Syntactic categories such as $((N/N)/(N/N))/((N/N)/(N/N))$ are not uncommon in the wide-coverage grammar of Hockenmaier and Steedman (2007), which in this case would require an 8th-order tensor. This combination of many word-category pairs and higher-order tensors results in a huge number of parameters.

As a solution to this problem, we are investigating ways to reduce the number of parameters, for example using tensor decomposition techniques (Kolda and Bader, 2009). It may also be possible to reduce the size of some of the complex categories in the grammar. Many challenges remain before a type-driven compositional distributional semantics can be realised, similar to the work of Bos for the model-theoretic case (Bos et al., 2004; Bos, 2005), but in this paper we have set out the theoretical framework for such an implementation.

Finally, we repeat a comment made earlier that the compositional framework makes no assumptions about the underlying vector spaces, or how they are to be interpreted. On the one hand, this flexibility is welcome, since it means the framework can encompass many techniques for building word vectors (and tensors). On the other hand, it means that a description of the framework is necessarily abstract, and it leaves open the question of what the meaning spaces represent. The latter question is particularly pressing in the case of the sentence space, and providing an interpretation of such spaces remains a challenge for the distributional semantics community, as well as relating distributional semantics to more traditional topics in semantics such as quantification and inference.

[4] Non-linear transformations are inherent to neural networks, whereas the framework in this paper is entirely linear. However, as hinted at earlier in the paper, non-linear transformations can be applied to the output of each tensor, turning the linear networks in this paper into extensions of those in Socher et al. (2012) (extensions in the sense that the tensors in Socher et al. (2012) do not extend beyond matrices).

Acknowledgments

Jean Maillard is supported by an EPSRC MPhil studentship. Stephen Clark is supported by ERC Starting Grant DisCoTex (306920) and EPSRC grant EP/I037512/1. Edward Grefenstette is supported by EPSRC grant EP/I037512/1. We would like to thank Tamara Polajnar, Laura Rimell, Nal Kalchbrenner and Karl Moritz Hermann for useful discussion.

References

M. Baroni and R. Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Conference on Empirical Methods in Natural Language Processing (EMNLP-10)*, Cambridge, MA.

M. Baroni, R. Bernardi, and R. Zamparelli. 2013. Frege in space: A program for compositional distributional semantics (to appear). *Linguistic Issues in Language Technologies*.

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

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

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

Stephen Clark. 2014. Vector space models of lexical meaning (to appear). In Shalom Lappin and Chris Fox, editors, *Handbook of Contemporary Semantics second edition*. Wiley-Blackwell.

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

B. Coecke, M. Sadrzadeh, and S. Clark. 2010. Mathematical foundations for a compositional distributional model of meaning. In J. van Bentham, M. Moortgat, and W. Buszkowski, editors, *Linguistic Analysis (Lambek Festschrift)*, volume 36, pages 345–384.

Bob Coecke, Edward Grefenstette, and Mehrnoosh Sadrzadeh. 2013. Lambek vs. Lambek: Functorial vector space semantics and string diagrams for Lambek calculus. *Annals of Pure and Applied Logic*.

R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, Helsinki, Finland.

James R. Curran. 2004. *From Distributional to Semantic Similarity*. Ph.D. thesis, University of Edinburgh.

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

Katrin Erk. 2012. Vector space models of word meaning and phrase meaning: a survey. *Language and Linguistics Compass*, 6(10):635–653.

J. R. Firth. 1957. A synopsis of linguistic theory 1930-1955. In *Studies in Linguistic Analysis*, pages 1–32. Oxford: Philological Society.

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

Gregory Grefenstette. 1994. *Explorations in Automatic Thesaurus Discovery*. Kluwer.

Edward Grefenstette. 2013. *Category-Theoretic Quantitative Compositional Distributional Models of Natural Language Semantics*. Ph.D. thesis, University of Oxford.

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

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

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.

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

Jayant Krishnamurthy and Tom M. Mitchell. 2013. 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.

Joachim Lambek. 2008. *From Word to Sentence. A Computational Algebraic Approach to Grammar*. Polimetrica.

T. K. Landauer and S. 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):211–240.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of ACL-08*, pages 236–244, Columbus, OH.

Stephen Pulman. 2013. Distributional semantic models. In Sadrzadeh Heunen and Grefenstette, editors, *Compositional Methods in Physics and Linguistics*. Oxford University Press.

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

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

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

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

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