Vector Space Semantic Parsing: A Framework for Compositional Vector Space Models

Jayant Krishnamurthy
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
jayantk@cs.cmu.edu

Tom M. Mitchell
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
tom.mitchell@cmu.edu

Abstract

We present *vector space semantic parsing* (VSSP), a framework for learning compositional models of vector space semantics. Our framework uses Combinatory Categorial Grammar (CCG) to define a correspondence between syntactic categories and semantic representations, which are vectors and functions on vectors. The complete correspondence is a direct consequence of minimal assumptions about the semantic representations of basic syntactic categories (e.g., nouns are vectors), and CCG's tight coupling of syntax and semantics. Furthermore, this correspondence permits nonuniform semantic representations and more expressive composition operations than previous work. VSSP builds a CCG semantic parser respecting this correspondence; this semantic parser parses text into lambda calculus formulas that evaluate to vector space representations. In these formulas, the meanings of words are represented by parameters that can be trained in a task-specific fashion. We present experiments using noun-verb-noun and adverb-adjective-noun phrases which demonstrate that VSSP can learn composition operations that RNN (Socher et al., 2011) and MV-RNN (Socher et al., 2012) cannot.

1 Introduction

Vector space models represent the semantics of natural language using vectors and operations on vectors (Turney and Pantel, 2010). These models are most commonly used for individual words and short phrases, where vectors are created using distributional information from a corpus. Such models achieve impressive performance on standardized tests (Turney, 2006; Rapp, 2003), correlate well with human similarity judgments (Griffiths et al., 2007), and have been successfully applied to a number of natural language tasks (Collobert et al., 2011).

While vector space representations for individual words are well-understood, there remains much uncertainty about how to compose vector space representations for phrases out of their component words. Recent work in this area raises many important theoretical questions. For example, should all syntactic categories of words be represented as vectors, or are some categories, such as adjectives, different? Using distinct semantic representations for distinct syntactic categories has the advantage of representing the operational nature of modifier words, but the disadvantage of more complex parameter estimation (Baroni and Zamparelli, 2010). Also, does semantic composition factorize according to a constituency parse tree (Socher et al., 2011; Socher et al., 2012)? A binarized constituency parse cannot directly represent many intuitive intra-sentence dependencies, such as the dependence between a verb's subject and its object. What is needed to resolve these questions is a comprehensive theoretical framework for compositional vector space models.

In this paper, we observe that we *already have such a framework*: Combinatory Categorial Grammar (CCG) (Steedman, 1996). CCG provides a tight mapping between syntactic categories and semantic types. If we assume that nouns, sentences, and other basic syntactic categories are represented by vectors, this mapping prescribes semantic types for all other syntactic categories.[1] For example, we get that adjectives are functions from noun vectors to noun vectors, and that prepo-

[1] It is not necessary to assume that sentences are vectors. However, this assumption simplifies presentation and seems like a reasonable first step. CCG can be used similarly to explore alternative representations.

Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, pages 1–10,
Sofia, Bulgaria, August 9 2013.

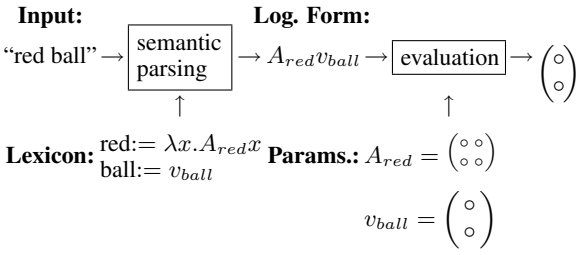

Figure 1: Overview of vector space semantic parsing (VSSP). A semantic parser first translates natural language into a logical form, which is then evaluated to produce a vector.

sitions are functions from a pair of noun vectors to a noun vector. These semantic type specifications permit a variety of different composition operations, many of which cannot be represented in previously-proposed frameworks. Parsing in CCG applies these functions to each other, naturally deriving a vector space representation for an entire phrase.

The CCG framework provides function type specifications for each word's semantics, given its syntactic category. Instantiating this framework amounts to selecting particular functions for each word. *Vector space semantic parsing* (VSSP) produces these per-word functions in a two-step process. The first step chooses a parametric functional form for each syntactic category, which contains as-yet unknown per-word and global parameters. The second step estimates these parameters using a concrete task of interest, such as predicting the corpus statistics of adjective-noun compounds. We present a stochastic gradient algorithm for this step which resembles training a neural network with backpropagation. These parameters may also be estimated in an unsupervised fashion, for example, using distributional statistics.

Figure 1 presents an overview of VSSP. The input to VSSP is a natural language phrase and a *lexicon*, which contains the parametrized functional forms for each word. These per-word representations are combined by CCG semantic parsing to produce a logical form, which is a symbolic mathematical formula for producing the vector for a phrase – for example, $A_{red}v_{ball}$ is a formula that performs matrix-vector multiplication. This formula is evaluated using learned per-word and global parameters (values for A_{red} and v_{ball}) to produce the language's vector space representation.

The contributions of this paper are threefold. First, we demonstrate how CCG provides a theoretical basis for vector space models. Second, we describe VSSP, which is a method for concretely instantiating this theoretical framework. Finally, we perform experiments comparing VSSP against other compositional vector space models. We perform two case studies of composition using noun-verb-noun and adverb-adjective-noun phrases, finding that VSSP can learn composition operations that existing models cannot. We also find that VSSP produces intuitively reasonable parameters.

2 Combinatory Categorial Grammar for Vector Space Models

Combinatory Categorial Grammar (CCG) (Steedman, 1996) is a lexicalized grammar formalism that has been used for both broad coverage syntactic parsing and semantic parsing. Like other lexicalized formalisms, CCG has a rich set of syntactic categories, which are combined using a small set of parsing operations. These syntactic categories are tightly coupled to semantic representations, and parsing in CCG simultaneously derives both a syntactic parse tree and a semantic representation for each node in the parse tree. This coupling between syntax and semantics motivates CCG's use in semantic parsing (Zettlemoyer and Collins, 2005), and provides a framework for building compositional vector space models.

2.1 Syntax

The intuition embodied in CCG is that, syntactically, words and phrases behave like functions. For example, an adjective like "red" can combine with a noun like "ball" to produce another noun, "red ball." Therefore, adjectives are naturally viewed as functions that apply to nouns and return nouns. CCG generalizes this idea by defining most parts of speech in terms of such functions.

Parts of speech in CCG are called *syntactic categories*. CCG has two kinds of syntactic categories: atomic categories and functional categories. Atomic categories are used to represent phrases that do not accept arguments. These categories include N for noun, NP for noun phrase, S for sentence, and PP for prepositional phrase. All other parts of speech are represented using functional categories. Functional categories are written as X/Y or $X\backslash Y$, where both X and Y are syn-

Part of speech	Syntactic category	Example usage	Semantic type	Example log. form
Noun	N	**person** : N	$\mathbb{R}^d$	v_{person}
Adjective	N/N_x	**good** person : N	$\langle \mathbb{R}^d, \mathbb{R}^d \rangle$	$\lambda x.A_{good}x$
Determiner	NP/N_x	**the** person : NP	$\langle \mathbb{R}^d, \mathbb{R}^d \rangle$	$\lambda x.x$
Intrans. Verb	$S\backslash NP_x$	the person **ran** : S	$\langle \mathbb{R}^d, \mathbb{R}^d \rangle$	$\lambda x.A_{ran}x + b_{ran}$
Trans. Verb	$S\backslash NP_y/NP_x$	the person **ran** home : S	$\langle \mathbb{R}^d, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$\lambda x.\lambda y.(\mathcal{T}_{ran}x)y$
Adverb	$(S\backslash NP)\backslash(S\backslash NP)$	ran **lazily** : $S\backslash NP$	$\langle \langle \mathbb{R}^d, \mathbb{R}^d \rangle, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$[\lambda y.Ay \rightarrow \lambda y.(\mathcal{T}_{lazy}A)y]$
	$(S\backslash NP)/(S\backslash NP)$	**lazily** ran : $S\backslash NP$	$\langle \langle \mathbb{R}^d, \mathbb{R}^d \rangle, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$[\lambda y.Ay \rightarrow \lambda y.(\mathcal{T}_{lazy}A)y]$
	$(N/N)/(N/N)$	**very** good person : N	$\langle \langle \mathbb{R}^d, \mathbb{R}^d \rangle, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$[\lambda y.Ay \rightarrow \lambda y.(\mathcal{T}_{very}A)y]$
Preposition	$(N\backslash N_y)/N_x$	person **in** France : N	$\langle \mathbb{R}^d, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$\lambda x.\lambda y.(\mathcal{T}_{in}x)y$
	$(S\backslash NP_y)\backslash(S\backslash NP)_f/NP_x$	ran **in** France : $S\backslash NP$	$\langle \mathbb{R}^d, \langle \langle \mathbb{R}^d, \mathbb{R}^d \rangle, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle \rangle$	$\lambda x.\lambda f.\lambda y.(\mathcal{T}_{in}x)(f(y))$

Table 1: Common syntactic categories in CCG, paired with their semantic types and example logical forms. The example usage column shows phrases paired with the syntactic category that results from using the exemplified syntactic category for the bolded word. For ease of reference, each argument to a syntactic category on the left is subscripted with its corresponding semantic variable in the example logical form on the right. The variables x, y, b, v denote vectors, f denotes a function, A denotes a matrix, and $\mathcal{T}$ denotes a tensor. Subscripted variables (A_{red}) denote parameters. Functions in logical forms are specified using lambda calculus; for example $\lambda x.Ax$ is the function that accepts a (vector) argument x and returns the vector Ax. The notation $[f \rightarrow g]$ denotes the higher-order function that, given input function f, outputs function g.

tactic categories. These categories represent functions that accept an argument of category Y and return a phrase of category X. The direction of the slash defines the expected location of the argument: X/Y expects an argument on the right, and $X\backslash Y$ expects an argument on the left.[2] For example, adjectives are represented by the category N/N – a function that accepts a noun on the right and returns a noun.

The left part of Table 1 shows examples of common syntactic categories, along with example uses. Note that some intuitive parts of speech, such as prepositions, are represented by multiple syntactic categories. Each of these categories captures a different use of a preposition, in this case the noun-modifying and verb-modifying uses.

2.2 Semantics

Semantics in CCG are given by first associating a semantic type with each syntactic category. Each word in a syntactic category is then assigned a semantic representation of the corresponding semantic type. These semantic representations are known as *logical forms*. In our case, a logical form is a fragment of a formula for computing a vector space representation, containing word-specific parameters and specifying composition operations.

In order to construct a vector space model, we associate all of the atomic syntactic categories, N, NP, S, and PP, with the type $\mathbb{R}^d$. Then, the logical form for a noun like "ball" is a vector $v_{ball} \in \mathbb{R}^d$. The functional categories X/Y and $X\backslash Y$ are associated with functions from the semantic type of X to the semantic type of Y. For example, the semantic type of N/N is $\langle \mathbb{R}^d, \mathbb{R}^d \rangle$, representing the set of functions from $\mathbb{R}^d$ to $\mathbb{R}^d$.[3] This semantic type captures the same intuition as adjective-noun composition models: semantically, adjectives are functions from noun vectors to noun vectors.

The right portion of Table 1 shows semantic types for several syntactic categories, along with example logical forms. All of these mappings are a direct consequence of the assumption that all atomic categories are semantically represented by vectors. Interestingly, many of these semantic types contain functions that cannot be represented in other frameworks. For example, adverbs have type $\langle \langle \mathbb{R}^d, \mathbb{R}^d \rangle, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$, representing functions that accept an adjective argument and return an adjective. In Table 1, the example logical form applies a 4-mode tensor to the adjective's matrix. Another powerful semantic type is $\langle \mathbb{R}^d, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$, which corresponds to transitive verbs and prepo-

[2] As a memory aid, note that the top of the slash points in the direction of the expected argument.

[3] The notation $\langle A, B \rangle$ represents the set of functions whose domain is A and whose range is B. Somewhat confusingly, the bracketing in this notation is backward relative to the syntactic categories – the syntactic category $(N\backslash N)/N$ has semantic type $\langle \mathbb{R}^d, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$, where the inner $\langle \mathbb{R}^d, \mathbb{R}^d \rangle$ corresponds to the left $(N\backslash N)$.

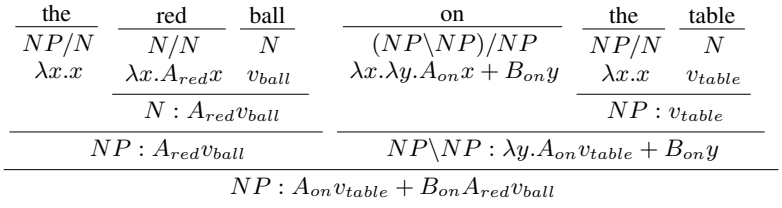

Figure 2: Syntactic CCG parse and corresponding vector space semantic derivation.

sitions. This type represents functions from two argument vectors to an output vector, which have been curried to accept one argument vector at a time. The example logical form for this type uses a 3-mode tensor to capture interactions between the two arguments.

Note that this semantic correspondence permits a wide range of logical forms for each syntactic category. Each logical form can have an arbitrary functional form, as long as it has the correct semantic type. This flexibility permits experimentation with different composition operations. For example, adjectives can be represented nonlinearly by using a logical form such as $\lambda x. \tanh(Ax)$. Or, adjectives can be represented nonparametrically by using kernel regression to learn the appropriate function from vectors to vectors. We can also introduce simplifying assumptions, as demonstrated by the last entry in Table 1. CCG treats prepositions as modifying intransitive verbs (the category $S\backslash N$). In the example logical form, the verb's semantics are represented by the function f, the verb's subject noun is represented by y, and $f(y)$ represents the sentence vector created by composing the verb with its argument. By only operating on $f(y)$, this logical form assumes that the action of a preposition is conditionally independent of the verb f and noun y, given the sentence $f(y)$.

2.3 Lexicon

The main input to a CCG parser is a *lexicon*, which is a mapping from words to syntactic categories and logical forms. A lexicon contains entries such as:

$$\text{ball} := N : v_{ball}$$
$$\text{red} := N/N : \lambda x.A_{red}x$$
$$\text{red} := N : v_{red}$$
$$\text{flies} := ((S\backslash NP)/NP) : \lambda x.\lambda y.(\mathcal{T}_{flies}x)y$$

Each entry of the lexicon associates a word (ball) with a syntactic category (N) and a logical form (v_{ball}) giving its vector space representation. Note that a word may appear multiple times in the lexicon with distinct syntactic categories and logical forms. Such repeated entries capture words with multiple possible uses; parsing must determine the correct use in the context of a sentence.

2.4 Parsing

Parsing in CCG has two stages. First, a category for each word in the input is retrieved from the lexicon. Second, adjacent categories are iteratively combined by applying one of a small number of combinators. The most common combinator is function application:

$$X/Y : f \quad Y : g \quad \Longrightarrow X : f(g)$$
$$Y : g \quad X\backslash Y : f \quad \Longrightarrow X : f(g)$$

The function application rule states that a category of the form X/Y behaves like a function that accepts an input category Y and returns category X. The rule also derives a logical form for the result by applying the function f (the logical form for X/Y) to g (the logical form for Y). Figure 2 shows how repeatedly applying this rule produces a syntactic parse tree and logical form for a phrase. The top row of the parse represents retrieving a lexicon entry for each word in the input. Each following line represents a use of the function application combinator to syntactically and semantically combine a pair of adjacent categories. The order of these operations is ambiguous, and different orderings may result in different parses – a CCG parser's job is to find a correct ordering. The result of parsing is a syntactic category for the entire phrase, coupled with a logical form giving the phrase's vector space representation.

3 Vector Space Semantic Parsing

Vector space semantic parsing (VSSP) is an approach for constructing compositional vector space models based on the theoretical framework of the previous section. VSSP concretely instantiates CCG's syntactic/semantic correspondence by adding appropriately-typed logical forms to a syntactic CCG parser's lexicon. Parsing a sentence with this lexicon and evaluating the resulting logi-

Semantic type	Example syntactic categories	Logical form template
$\mathbb{R}^d$	N, NP, PP, S	v_w
$\langle \mathbb{R}^d, \mathbb{R}^d \rangle$	N/N, NP/N, S/S, $S \backslash NP$	$\lambda x.\sigma(A_w x)$
$\langle \mathbb{R}^d, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$(S \backslash NP)/NP$, $(NP \backslash NP)/NP$	$\lambda x.\lambda y.\sigma((\mathcal{T}_w x)y)$
$\langle \langle \mathbb{R}^d, \mathbb{R}^d \rangle, \langle \mathbb{R}^d, \mathbb{R}^d \rangle \rangle$	$(N/N)/(N/N)$	$[\lambda y.\sigma(Ay) \rightarrow \lambda y.\sigma((\mathcal{T}_w A)y)]$

Table 2: Lexicon templates used in this paper to produce a CCG semantic parser. σ represents the sigmoid function, $\sigma(x) = \frac{e^x}{1+e^x}$.

cal form produces the sentence's vector space representation.

While it is relatively easy to devise vector space representations for individual nouns, it is more challenging to do so for the fairly complex function types licensed by CCG. VSSP defines these functions in two phases. First, we create a lexicon mapping words to parametrized logical forms. This lexicon specifies a functional form for each word, but leaves free some per-word parameters. Parsing with this lexicon produces logical forms that are essentially functions from these per-word parameters to vector space representations. Next, we train these parameters to produce good vector space representations in a task-specific fashion. Training performs stochastic gradient descent, backpropagating gradient information through the logical forms.

3.1 Producing the Parametrized Lexicon

We create a lexicon using a set of manually-constructed templates that associate each syntactic category with a parametrized logical form. Each template contains variables that are instantiated to define per-word parameters. The output of this step is a CCG lexicon which can be used in a broad coverage syntactic CCG parser (Clark and Curran, 2007) to produce logical forms for input language.[4]

Table 2 shows some templates used to create logical forms for syntactic categories. To reduce annotation effort, we define one template per semantic type, covering all syntactic categories with that type. These templates are instantiated by replacing the variable w in each logical form with the current word. For example, instantiating the second template for "red" produces the logical form $\lambda x.\sigma(A_{red}x)$, where A_{red} is a matrix of parameters.

[4]In order to use the lexicon in an existing parser, the generated syntactic categories must match the parser's syntactic categories. Then, to produce a logical form for a sentence, simply syntactically parse the sentence, generate logical forms for each input word, and retrace the syntactic derivation while applying the corresponding semantic operations to the logical forms.

Note that Table 2 is a only starting point – devising appropriate functional forms for each syntactic category is an empirical question that requires further research. We use these templates in our experiments (Section 4), suggesting that they are a reasonable first step. More complex data sets will require more complex logical forms. For example, to use high-dimensional vectors, all matrices and tensors will have to be made low rank. Another possible improvement is to tie the parameters for a single word across related syntactic categories (such as the transitive and intransitive forms of a verb).

3.2 Training the Logical Form Parameters

The training problem in VSSP is to optimize the logical form parameters to best perform a given task. Our task formulation subsumes both classification and regression: we assume the input is a logical form, and the output is a vector. Given a data set of this form, training can be performed using stochastic gradient descent in a fashion similar to backpropagation in a neural network.

The data set for training consists of tuples, $\{(\ell^i, y^i)\}_{i=1}^n$, where ℓ is a logical form and y is a label vector representing the expected task output. Each logical form ℓ is treated as a function from parameter vectors θ to vectors in $\mathbb{R}^d$. For example, the logical form $A_{red}v_{ball}$ is a function from A_{red} and v_{ball} to a vector. We use θ to denote the set of all parameters; for example, $\theta = \{A_{red}, v_{ball}\}$. We further assume a loss function $\mathcal{L}$ defined over pairs of label vectors. The training problem is therefore to minimize the objective:

$$O(\theta) = \sum_{i=1}^{n} \mathcal{L}(y^i, g(\ell^i(\theta)) + \frac{\lambda}{2}||\theta||^2$$

Above, g represents a global postprocessing function which is applied to the output of VSSP to make a task-specific prediction. This function may also be parametrized, but we suppress these parameters for simplicity. As a concrete example, consider a classification task (as in our evaluation). In this case, y represents a target distribution over labels, $\mathcal{L}$ is the KL divergence between the pre-

dicted and target distributions, and g represents a softmax classifier.

We optimize the objective O by running stochastic gradient descent. The gradients of the parameters θ can be computed by iteratively applying the chain rule to ℓ, which procedurally resembles performing backpropagation in a neural network (Rumelhart et al., 1988; Goller and Küchler, 1996).

4 Comparing Models of Semantic Composition

This section compares the expressive power of VSSP to previous work. An advantage of VSSP is its ability to assign complex logical forms to categories like adverbs and transitive verbs. This section examines cases where such complex logical forms are necessary, using synthetic data sets. Specifically, we create simple data sets mimicking expected forms of composition in noun-verb-noun and adverb-adjective-noun phrases. VSSP is able to learn the correct composition operations for these data sets, but previously proposed models cannot.

We compare VSSP against RNN (Socher et al., 2011) and MV-RNN (Socher et al., 2012), two recursive neural network models which factorize composition according to a binarized constituency parse tree. The RNN model represents the semantics of each parse tree node using a single vector, while the MV-RNN represents each node using both a matrix and a vector. These representations seem sufficient for adjectives and nouns, but it is unclear how they generalize to other natural language constructions.

In these experiments, each model is used to map an input phrase to a vector, which is used to train a softmax classifier that predicts the task output. For VSSP, we use the lexicon templates from Table 2. All nouns are represented as two-dimensional vectors, and all matrices and tensors are full rank. The parameters of each model (i.e., the per-word vectors, matrices and tensors) and the softmax classifier are trained as described in Section 3.2.

4.1 Propositional Logic

The propositional logic experiment examines the impact of VSSP's representation of transitive verbs. VSSP directly represents these verbs as two-argument functions, allowing it to learn operations with complex interactions between both arguments. In contrast, the RNN and MV-RNN models learn a set of global weights which are used to combine the verb with its arguments. The functional forms of these models limit the kinds of interactions that can be captured by verbs.

false and false	0,1	false or false	0,1	false xor false	0,1
true and false	0,1	true or false	1,0	true xor false	1,0
false and true	0,1	false or true	1,0	false xor true	1,0
true and true	1,0	true or true	1,0	true xor true	0,1

Table 3: Data for propositional logic experiment.

Composition Formula	KL divergence
RNN	0.44
MV-RNN	0.12
VSSP	0.01

Table 4: Training error on the propositional logic data set. VSSP achieves zero error because its verb representation can learn arbitrary logical operations.

We evaluated the learnability of argument interactions using the simple data set shown in Table 3. In this data set, the words "and," "or," and "xor" are treated as transitive verbs, while "true" and "false" are nouns. The goal is to predict the listed truth values, which are represented as two-dimensional distributions over true and false.

Table 4 shows the training error of each model on this data set, measured in terms of KL divergence between the model's predictions and the true values. VSSP achieves essentially zero training error because its 3-mode tensor representation of transitive verbs is trivially able to learn arbitrary logical operations. RNN and MV-RNN can learn each logical operation independently, but cannot learn all three at the same time – this phenomenon occurs because XOR requires different global weight matrices than AND/OR. As a result, these models learn both AND and OR, but fail to learn XOR. This result suggests that much of the learning in these models occurs in the global weight matrices, while the verb representations can have only limited influence.

Although this data set is synthetic, the interaction given by XOR seems necessary to represent real verbs. To learn AND and OR, the arguments need not interact – it is sufficient to detect a set of appropriate subject and object arguments, then threshold the number of such arguments. This information is essentially type constraints for the subject and object of a verb. However, type constraints are insufficient for real verbs. For example, consider the verb "eats." All animals eat and

very big elephant	1,0	very big mouse	0.3,0.7
pretty big elephant	0.9,0.1	pretty big mouse	0.2,0.8
pretty small elephant	0.8,0.2	pretty small mouse	0.1,0.9
very small elephant	0.7,0.3	very small mouse	0,1

Table 5: Data for adverb-adjective-noun composition experiment. Higher first dimension values represent larger objects.

Composition Model	KL divergence
RNN	0.10
MV-RNN	0.10
VSSP	0.00

Table 6: Training error of each composition model on the adverb-adjective-noun experiment.

can be eaten, but not all animals eat all other animals; whether or not "X eats Y" is true depends on an interaction between X and Y.

4.2 Adverb-Adjective-Noun Composition

Adverbs can enhance or attenuate the properties of adjectives, which in turn can enhance or attenuate the properties of nouns. The adverb-adjective-noun experiment compares each model's ability to learn these effects using a synthetic object size data set, shown in Table 5. The task is to predict the size of each described object, which is represented as a two-dimensional distribution over big and small. The challenge of this data set is that an adverb's impact on size depends on the adjective being modified – a very big elephant is *bigger* than a big elephant, but a very small elephant is *smaller* than a small elephant. Note that this task is more difficult than adverb-adjective composition (Socher et al., 2012), since in this task the adverb has to enhance/attenuate the enhancing/attenuating properties of an adjective.

Table 6 shows the training error of each model on this data set. VSSP achieves zero training error because its higher-order treatment of adverbs allows it to accurately represent their enhancing and attenuating effects. However, none of the other models are capable of representing these effects. This result is unsurprising, considering that the RNN and MV-RNN models essentially *add* the adverb and adjective parameters using a learned linear operator (followed by a nonlinearity). Such additive combination forces adverbs to have a consistent direction of effect on the size of the noun, which is incompatible with the desired enhancing and attenuating behavior.

Examining VSSP's learned parameters clearly demonstrates its ability to learn enhancing and attenuating phenomena. Figure 3 demonstrates VSSP's learned treatment of "very." In the figure, a high first dimension value represents a large object, while a high second dimension value represents a small object; hence the vectors for elephant and mouse show that, by default, elephants are larger than mice. Similarly, the matrices for big and small scale up the appropriate dimension while shrinking the other dimension. Finally, we show the computed matrices for "very big" and "very small" – this operation is possible because these phrases have an adjective's syntactic category, N/N. These matrices have the same direction of effect as their unenhanced versions, but produce a larger scaling in that direction.

"elephant" $\begin{pmatrix} 1.6 \\ -0.1 \end{pmatrix}$ "mouse" $\begin{pmatrix} -0.1 \\ 1.6 \end{pmatrix}$

"small" $\begin{pmatrix} 0.22 & 0 \\ 0 & 1.7 \end{pmatrix}$ "big" $\begin{pmatrix} 1.7 & -1.1 \\ 0 & 0.22 \end{pmatrix}$

"very small" $\begin{pmatrix} 0.25 & -.12 \\ -1.34 & 2.3 \end{pmatrix}$ "very big" $\begin{pmatrix} 2.3 & -1.34 \\ -0.12 & 0.25 \end{pmatrix}$

Figure 3: Parameters for nouns, adjectives and adjective phrases learned by VSSP. When the adverb "very" is applied to "small" and "big," it enhances their effect on a modified noun.

5 Related Work

Several models for compositionality in vector spaces have been proposed in recent years. Much work has focused on evaluating composition operations for word pairs (Mitchell and Lapata, 2010; Widdows, 2008). Many operations have been proposed, including various combinations of addition, multiplication, and linear operations (Mitchell and Lapata, 2008), holographic reduced representations (Plate, 1991) and others (Kintsch, 2001). Other work has used regression to train models for adjectives in adjective-noun phrases (Baroni and Zamparelli, 2010; Guevara, 2010). All of this work is complementary to ours, as these composition operations can be used within VSSP by appropriately choosing the logical forms in the lexicon.

A few comprehensive frameworks for composition have also been proposed. One approach is to take tensor outer products of word vectors, following syntactic structure (Clark and Pulman, 2007). However, this approach results in differently-shaped tensors for different grammatical structures. An improvement of this framework uses a categorial grammar to ensure that similarly-

typed objects lie in the same vector space (Clark et al., 2008; Coecke et al., 2010; Grefenstette and Sadrzadeh, 2011). VSSP generalizes this work by allowing nonlinear composition operations and considering supervised parameter estimation. Several recent neural network models implicitly use a framework which assumes that composition factorizes according to a binarized constituency parse, and that words and phrases have uniform semantic representations (Socher et al., 2011; Socher et al., 2012). Notably, Hermann and Blunsom (2013) instantiate such a framework using CCG. VSSP generalizes these approaches, as they can be implemented within VSSP by choosing appropriate logical forms. Furthermore, our experiments demonstrate that VSSP can learn composition operations that cannot be learned by these approaches.

The VSSP framework uses semantic parsing to define a compositional vector space model. Semantic parsers typically map sentences to logical semantic representations (Zelle and Mooney, 1996; Kate and Mooney, 2006), with many systems using CCG as the parsing formalism (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2011; Krishnamurthy and Mitchell, 2012). Although previous work has focused on logical semantics, it has demonstrated that semantic parsing is an elegant technique for specifying models of compositional semantics. In this paper, we show how to use semantic parsing to produce compositional models of vector space semantics.

6 Discussion and Future Work

We present vector space semantic parsing (VSSP), a general framework for building compositional models of vector space semantics. Our framework is based on Combinatory Categorial Grammar (CCG), which defines a correspondence between syntactic categories and semantic types representing vectors and functions on vectors. A model in VSSP instantiates this mapping in a CCG semantic parser. This semantic parser parses natural language into logical forms, which are in turn evaluated to produce vector space representations. We further propose a method for constructing such a semantic parser using a small number of logical form templates and task-driven estimation of per-word parameters. Synthetic data experiments show that VSSP's treatment of adverbs and transitive verbs can learn more functions than prior work.

An interesting aspect of VSSP is that it highlights cases where propositional semantics seem superior to vector space semantics. For example, compare "the ball that I threw" and "I threw the ball." We expect the semantics of these phrases to be closely related, differing only in that one phrase refers to the ball, while the other refers to the throwing event. Therefore, our goal is to define a logical form for "that" which appropriately relates the semantics of the above expressions. It is easy to devise such a logical form in propositional semantics, but difficult in vector space semantics. Producing vector space solutions to such problems is an area for future work.

Another direction for future work is joint training of both the semantic parser and vector space representations. Our proposed approach of adding logical forms to a broad CCG coverage parser has the advantage of allowing VSSP to be applied to general natural language. However, using the syntactic parses from this parser may not result in the best possible factorization of semantic composition. Jointly training the semantic parser and the vector space representations may lead to better models of semantic composition.

We also plan to apply VSSP to real data sets. We have made some progress applying VSSP to SemEval Task 8, learning to extract relations between nominals (Hendrickx et al., 2010). Although our work thus far is preliminary, we have found that the generality of VSSP makes it easy to experiment with different models of composition. To swap between models, we simply modify the CCG lexicon templates – all of the remaining infrastructure is unchanged. Such preliminary results suggest the power of VSSP as a general framework for learning vector space models.

Acknowledgments

This research has been supported in part by DARPA under award FA8750-13-2-0005, and in part by a gift from Google. We thank Matt Gardner, Justin Betteridge, Brian Murphy, Partha Talukdar, Alona Fyshe and the anonymous reviewers for their helpful comments.

References

Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: representing adjective-noun constructions in semantic space. In

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing.

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*.

Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. 2008. A compositional distributional model of meaning. *Proceedings of the Second Symposium on Quantum Interaction*.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. 2010. Mathematical Foundations for a Compositional Distributed Model of Meaning. *Lambek Festschirft, Linguistic Analysis*, 36.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, November.

Christoph Goller and Andreas Küchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of the International Conference on Neural Networks (ICNN-96)*, pages 347–352. IEEE.

Edward Grefenstette and Mehrnoosh Sadrzadeh. 2011. Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Thomas L. Griffiths, Joshua B. Tenenbaum, and Mark Steyvers. 2007. Topics in semantic representation. *Psychological Review 114*.

Emiliano Guevara. 2010. A regression model of adjective-noun compositionality in distributional semantics. In *Proceedings of the 2010 Workshop on Geometrical Models of Natural Language Semantics*.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó. Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2010. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation*.

Karl Moritz Hermann and Phil Blunsom. 2013. The Role of Syntax in Vector Space Models of Compositional Semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*.

Walter Kintsch. 2001. Predication. *Cognitive Science*, 25(2).

Jayant Krishnamurthy and Tom M. Mitchell. 2012. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of ACL-08: HLT*.

Jeff Mitchell and Mirella Lapata. 2010. Composition in Distributional Models of Semantics. *Cognitive Science*, 34(8):1388–1429.

Tony Plate. 1991. Holographic reduced representations: convolution algebra for compositional distributed representations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*.

Reinhard Rapp. 2003. Word sense discovery based on sense descriptor dissimilarity. In *Proceedings of the Ninth Machine Translation Summit*.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Neurocomputing: foundations of research. chapter Learning internal representations by error propagation.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

Mark Steedman. 1996. *Surface Structure and Interpretation*. The MIT Press.

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

Peter D. Turney. 2006. Similarity of semantic relations. *Computational Linguistics*, 32(3), September.

Dominic Widdows. 2008. Semantic vector products: Some initial investigations. In *Proceedings of the Second AAAI Symposium on Quantum Interaction*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*.