Machine learning methods for vector-based compositional semantics

Jean Maillard

St John's College

University of Cambridge
Department of Computer Science and Technology

January 2019

This dissertation is submitted
for the degree of Doctor of Philosophy

Declaration

Machine learning methods for vector-based compositional semantics

Jean Maillard

Rich semantic representations of linguistic data are an essential component to the development of machine learning algorithms for natural language processing. This thesis explores techniques to model the meaning of phrases and sentences as dense vectors, which can then be further analysed and manipulated to perform any number of tasks involving the understanding of human language. Rather than seeing this task purely as an engineering problem, this thesis will focus on linguistically-motivated approaches, based on the principle of compositionality.

The first half of the thesis will be dedicated to categorial compositional models, which are based on the observation that certain types of grammars share the structure of the algebra of vector spaces. This leads to an approach where the meanings of words are modelled as multilinear maps, encoded as tensors. In this framework, the meaning of a composite linguistic phrase can be computed via the tensor multiplication of its constituents, according to the phrase's syntactic structure. I contribute two categorial compositional models: the first, an extension of a popular method for learning semantic representation of words, models the meanings of adjective-noun phrases as matrix-vector multiplications; the second uses higher-order tensors to represent the meaning of relative clauses.

In contrast, the models presented in the second half of the thesis do away with traditional syntactic structures. Rather than using the standard syntax trees of linguistics to drive the compositional process, these models treat the compositional structure as a latent variable. I contribute two models that automatically induce trees for a downstream task, without ever being shown a 'real' syntax tree: one model based on chart parsing, and one based on shift-reduce parsing. While these proposed approaches induce trees that do not resemble traditional syntax trees, they do lead to models with higher performance on downstream tasks – opening up avenues for future research.

Acknowledgments

I would like to express my profound gratitude to my family, and especially to my parents Emilia and Vincent. I am deeply grateful for their unconditional love and encouragement throughout all of my pursuits. This thesis is dedicated to them. I would also like to express my gratitude to my grandmother Giulia, my aunt Marina, Daniel Bec, and all the members of my extended family for their continuous support.

Special thanks go to my supervisor, Steve Clark, who introduced me to the field of Computational Linguistics. His guidance and support over the past years have made me grow as a researcher and a scientist. I could not have hoped for a better mentor. I am also sincerely thankful to the members of my thesis committee, Ted Briscoe and Edward Grefenstette; my advisor, Ann Copestake; the graduate education manager, Lise Gough; and the EPSRC, St John's College, and the Computer Laboratory for their generous funding.

My years as a doctoral student were greatly enriched by interacting and collaborating with a number of brilliant colleagues and friends: Luana Bulat, Kris Cao, Vesna Djokic, Guy Emerson, Douwe Kiela, Ekaterina Kochmar, Alexander Kuhnle, Ewa Muszyńska, Tamara Polajnar, Marek Rei, Laura Rimell, Diarmuid Ó Séaghdha, Ekaterina Shutova, Yiannos Stathopoulos, Eva Maria Vecchi, Anita Vero, Andreas Vlachos, Wenduan Xu, Helen Yannakoudakis, Dani Yogatama, and all the other current and former members of the Computer Laboratory's NLIP group at the University of Cambridge.

Finally, my time in Cambridge would not have been the same without the incredible group of friends I was lucky enough to find, and who have been extremely supportive throughout this journey. My sincere gratitude goes out to Sandro Bauer, Johannes Bausch, Laura Convertino, Conrad Koziol, Mircea Micu, Christopher Pulte, Emilie Skulberg, and Sofiya Zlateva. I am grateful to Ludovica Gonella, Andrea Odone, and Davide Testuggine for their friendships, which are still lasting after over 15 years through very long distances. I would also like to thank *team awesome*, the MPhil crew, my college friends, and all the wonderful people I met while sailing and dancing.

Thank you!

Contents

List of Figures

List of Tables

Part I

Introduction

1 Introduction

Before a machine learning algorithm can perform any task which involves language understanding, it needs to obtain some internal characterisation of the meaning of its linguistic input. Effective semantic representations of linguistic data can then be further analysed and manipulated to perform any number of Natural Language Processing (NLP) tasks, ranging from machine translation to the understanding of verbal commands (e.g. in the context of a virtual assistant such as Apple's Siri or Amazon's Alexa). In this thesis, we will explore ways in which computational models of human language can effectively represent the meaning of phrases and sentences as vectors. More specifically, we will focus on vector-based semantic models that respect the *principle of compositionality* – the fundamental idea in linguistics that the meaning of an expression is fully determined by its structure and the meaning of its constituent parts. The principle has been stated in a variety of ways in the literature, and there is no universally accepted definition of what is meant by terms such as *meaning* or *structure* (Goldberg, 2003). Dowty (2007), for instance, states it as "The meaning of a sentence is a function of the meanings of the words in it and the way they are combined syntactically",[1] and further discusses how most authors have taken this to suggest that there exists a homomorphism (a structure-preserving map) between syntax and semantics, under which semantics can be seen as the image of syntax. He illustrates this idea for the sentence *Fido barks* with the following equation:

$$\begin{aligned} &\textit{meaning-of}\ (\textsc{Syntactic-Combination-Of}\ (\textbf{Fido}, \textbf{barks})) \\ &\qquad = \qquad \textsc{Semantic-Function-of}\ \big(\textit{meaning-of}(\textbf{Fido}), \textit{meaning-of}(\textbf{barks})\big)\,, \end{aligned}$$

where Semantic-Function-of and Syntactic-Combination-Of are in direct correspondence under the homomorphism.

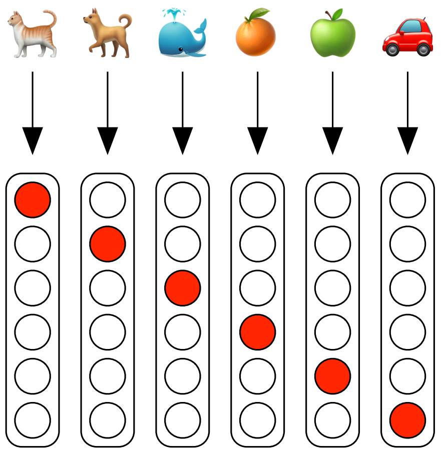
In this chapter we discuss how vectors, which are ubiquitous structures throughout machine learning, represent an effective way of encoding the meaning of words, phrases, and sentences; and that they offer advantages compared to more traditional approaches. This chapter further serves as an overview of the existing literature on vector-based semantics within NLP. Additional in-depth reviews of categorial models of composition and tree-based neural composition are presented in Chapters 3 and 6, respectively.

[1]It should be noted, however, that this does not readily apply to metaphorical and idiomatic language, or constructional meaning (Goldberg, 2003). These aspects are beyond the scope of this thesis, and we refer interested readers to Westerståhl (2002) and Goldberg (2015).

Having presented several approaches to model the meaning of words, the rest of the thesis discusses ways to combine these to obtain the meaning of the larger expressions they form. These effectively amount to providing different concrete definitions of Dowty's "Semantic-Function-of", given a syntactic structure. My own contributions are characterised by structural priors and strong inductive biases, resulting in compositional, linguistically-motivated models which closely adhere to a view of semantics as a homomorphism – where vector representations are built in step with the syntactic derivation. Broadly speaking, this thesis aims to compare these models to more mainstream phrase and sentence encoding approaches, which lack these inductive biases. We shall see that, while there remain issues of scalability, these linguistically motivated approaches look promising in various ways.

1.1 Encoding words in computers

For the majority of NLP algorithms, words are the simplest indivisible unit. Therefore, in accordance with compositionality, it is with them that our story must begin. How do computers represent words? When encoded digitally, a word is nothing more than a sequence of bytes representing characters. For most of today's computers *cat* is stored as `63 61 74`, *dog* as `64 6F 67`, and *ozone* as `6F 7A 6F 6E 65`.[2] These representations are unsuitable for our purposes: they do not intrinsically contain any semantic information; they are space-inefficient, as the encoding is meant to represent more entities than just words, such as control characters and drawing elements; and they are variable in size – which complicates their handling. How, then, are the meanings of words represented in NLP?


(a) Symbolic *one-hot* representations.

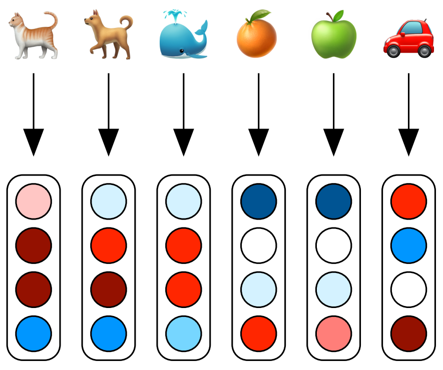

(b) Distributed representations.

Figure 1.1: Hypothetical representations of word meanings.

[2]For more information on the background and peculiarities of this encoding scheme, ASCII, see Bemer (1980) and references therein.

Symbolic representations The traditional approach to the representation of meaning in NLP follows formal semantics (Dowty et al., 1981). Words are associated with unique, atomic symbols which are then combined, according to syntax, into some form of logical structure. Examples of such systems include Bos et al. (2004), who use CCG derivations to construct semantic representations, and Briscoe and Carroll (2002), who generate underspecified semantic representations. As a concrete example of this approach, using the formalism of first-order logic with a neo-Davidsonian analysis of events, the sentence *a dog chased a cat* might get translated to

$$\exists x\, \exists y\, \exists e\; \texttt{dog}(x) \wedge \texttt{cat}(y) \wedge \texttt{chase}(e) \wedge \text{agent}(e, x) \wedge \text{patient}(e, y). \tag{1.1}$$

where `cat`, `dog`, and `chase` are the symbolic representations of the respective words, denoting objects in a set-theoretic model.

In practical terms, these opaque symbols might be implemented by simply mapping the string representation of words to unique numerical identifiers, e.g. *cat* $\mapsto$ 1, *dog* $\mapsto$ 2, and so forth for the rest of the vocabulary; or, if using a machine learning system where vector representations are more convenient, the natural solution would be to turn these into a *one-hot* encoding: vectors of zeroes with a single one in the position corresponding to the word's index (illustrated in Figure 1.1a). One obvious shortcoming of this approach is that word representations are completely independent from one another, and have no intrinsic notion of similarity: there is therefore no way of knowing from the representations that *cat* and *kitty* can be synonyms, or that both are relatively close in meaning to *dog*, at least when compared to e.g. *car*. Such knowledge would be very important for applications such as information retrieval systems and search engines, which could return a wider range of relevant results by considering similar queries. This information could be learned separately, for instance by using a lexical database such as WordNet (Fellbaum, 1998), as suggested by Bos (2005). However, any such resources need to be hand-crafted by trained experts, and their coverage is limited. A more fundamental problem is that the size of the parameter space associated with these representations grows with the size of the vocabulary. This issue, known as the *curse of dimensionality*, becomes particularly obvious when thinking about modelling joint distributions of words: modelling 10 consecutive words with a vocabulary of size 100,000 leads to potentially $\sim 10^{50}$ free parameters (Bengio et al., 2003).

Distributed representations Instead of using a symbolic representation of words, an alternative strategy is to distribute the information content for each word across all dimensions of a vector space, moving from discrete sparse representations to dense continuous ones. These are known as *distributed representations*, and are illustrated in Figure 1.1b. They have the obvious advantage of being more space efficient, thus requiring fewer parameters and getting around the curse of dimensionality (Hinton et al., 1986). Further, they can be built in such a way that similar words will have similar representations, which has two main advantages: a very useful intrinsic notion of similarity, as discussed in the previous paragraph; and more broadly, a better generalisation ability, as many algorithms can be expected to have local smoothness properties (Bengio et al.,

```
… domesticated breed of cat, with a distinctive phy…
…the simplest type is a cat flap, or pet door, whic…
…breeds, the maine coon cat has longer whiskers tha…
… factors, the breed of cat will affect pet insuran…
…f whisker fatigue in a cat whom I pet sit for, and…
…e a look at the top 26 cat whisker facts which you…

…in the role of the pet dog, such as the increased …
… majority of breeds of dog are at most a few hundr…
…emporary people with a dog describe their pet as p…
…ost vocal canid is the dog: its tendency to bark a…
…have trained their pet dog not to bark whenever th…
… a pet is a particular dog breed that is best for …
```

Figure 1.2: Highlighted occurrences of some words neighbouring *cat* and *dog* in a symmetric window around them. The example sentences come from a web search.

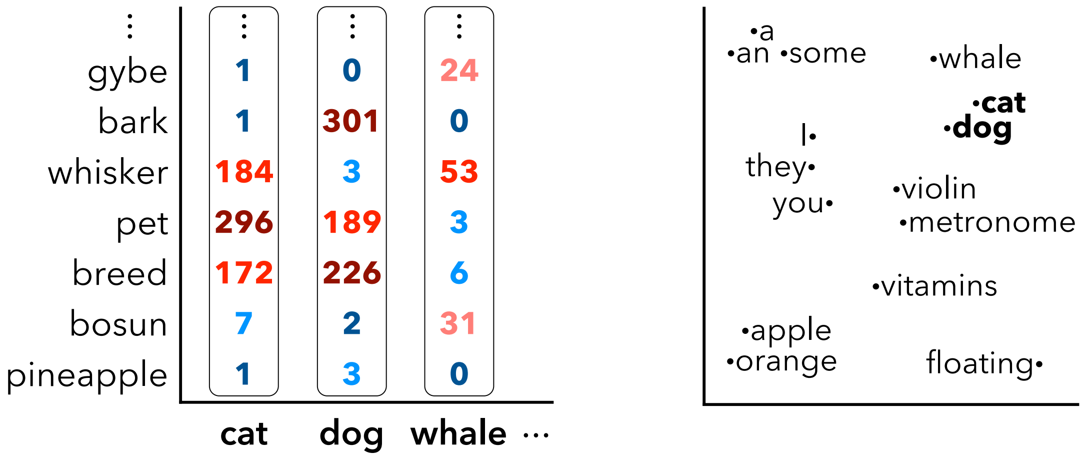
2003), such that similar inputs lead to similar outputs. Finally, these representations have the advantage that new words can be added to the vocabulary without having to increase the dimensionality of the vector space.

1.2 Distributed word representations

While we have listed some advantages of distributed word representations, no mention was made of how these vectors can be obtained. In this section, we discuss several methods to automatically compute such representations.

Distributional models The *distributional hypothesis*, popularised by Firth (1957), states that words used in similar contexts share a similar meaning. This suggests that the semantics of words can be characterised by harvesting their co-occurrence statistics from large corpora, a process which is easily automated.[3] As illustrated by the example in Figure 1.2, co-occurrence counts are readily collected by inspecting the neighbours of words in a linguistic corpus, and then collated into a matrix whose columns correspond to the words in the vocabulary (Figure 1.3a). After some post-processing of the data such as normalisation, smoothing of the counts, and dimensionality reduction, the columns can then be seen as vectors representing the semantics of words. Looking at a two-dimensional projection of the resulting vector space will reveal that words with similar usage are clustered together (Figure 1.3b), showing that geometrical methods can be used on the vectors to measure the similarity of meaning of the corresponding words. This approach has had a considerable impact on natural language processing (Turney and Pantel, 2010; Baroni et al., 2013; Clark, 2015; and references therein), as it provides vector representations of the

[3]For three seminal papers exploiting these ideas, see Salton et al. (1975), Deerwester et al. (1990), and Schütze (1998).



	cat	dog	whale	⋯
⋮	⋮	⋮	⋮	
gybe	1	0	24	
bark	1	301	0	
whisker	184	3	53	
pet	296	189	3	
breed	172	226	6	
bosun	7	2	31	
pineapple	1	3	0	

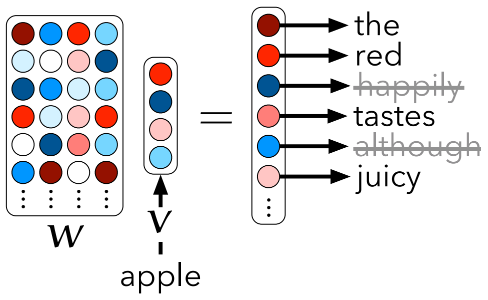
(a) Co-occurrence counts of words. (b) Resulting vector space.

Figure 1.3: The co-occurrence counts in a corpus can be used to produce semantic vector representations of words. The resulting vector space captures some notion of semantic similarity.

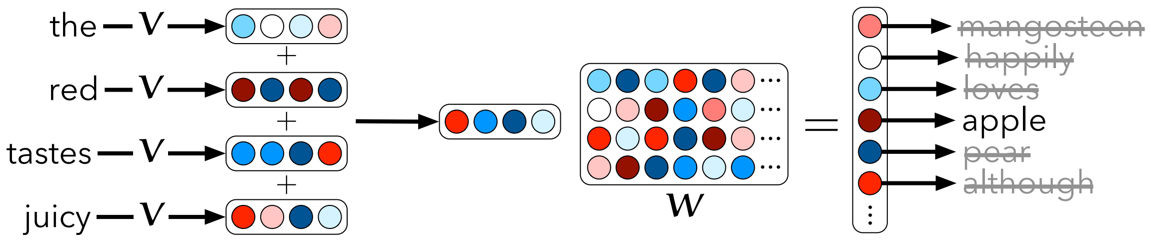
semantics of words which can be readily harvested from large unannotated corpora, and are easily used in downstream tasks.

Prediction-based models More recently, a new family of methods to train distributed representations of words has emerged: rather than explicitly collecting counts, these methods learn a vector parameter for each word that maximises the probability of predicting its neighbours, or related quantities. Bengio et al. (2003) and Collobert and Weston (2008) are some of the earliest published approaches, although some of the key insights can already be found in the literature of the 80s (Hinton et al., 1986; Rumelhart et al., 1986a; Rumelhart et al., 1986b). However it was arguably the models of Mikolov et al. (2013b), along with their implementation `word2vec`,[4] that have had the greater impact. The paper presented two popular models, skip-gram and CBOW, which are sketched in Figure 1.4. Baroni et al. (2014) perform an extensive evaluation of the traditional count-based methods and the new prediction-based methods, showing that the latter perform better on a wide range of lexical semantics evaluations. Some authors take yet another approach: instead of taking word representations trained with skip-gram or similar methods and using them for some NLP task, they use randomly initialised vectors, relying on the downstream task to learn optimal values for them (Sutskever et al., 2014; Bahdanau et al., 2015; Gehring et al., 2017; inter alia). This approach, in which word representations are learned directly as a by-product of solving the desired task, is generally effective and has the benefit of simplifying the experimental pipeline (Kocmi and Bojar, 2017). Vectors obtained with any of the these methods are often called *embeddings* in the literature, and we shall also use this term.

[4] `http://word2vec.googlecode.com/`



(a) Skip-gram is given a word and predicts its neighbours.



(b) CBOW predicts a word based on the averaged representations of its neighbours.

Figure 1.4: Sketch of the skip-gram and CBoW models of Mikolov et al. (2013a) processing the word *apple* in the sentence *the red apple tastes juicy*.

Other approaches Latent semantic analysis, known as LSA (Deerwester et al., 1990; Landauer and Dumais, 1997), is a historical method which has had a large impact on the fields of NLP and Information Retrieval. Starting from a set of documents, it calculates a word/document co-occurrence matrix, and uses a dimensionality reduction technique to obtain distributed representations for words. A hybrid method which has been shown to perform well is that of Pennington et al. (2014), which borrows ideas from both count-based and prediction-based approaches, and is widely used. Faruqui and Dyer (2015) show how lexical databases can also be exploited to obtain distributed vectors, which the authors call *non-distributional* as they do not encode any word co-occurrence information. A related effort by Faruqui et al. (2015) is to refine pre-existing distributed vectors with information from semantic lexicons. Bojanowski et al. (2017) show an interesting extension of skip-gram, where each word is further represented as an (unordered) set of character n-grams, thus allowing the building of representations for words not present in the training data. Finally, a very powerful recent method which has improved the state-of-the-art on a number of tasks is ELMo (Peters et al., 2018), which learns contextualised word representations as the hidden layers of a deep recurrent bidirectional neural language model.

1.3 Compositional distributed semantics

How is the meaning of a sentence determined? According to the *principle of compositionality*, which is at the basis of most contemporary work in semantics, it is obtained recursively from

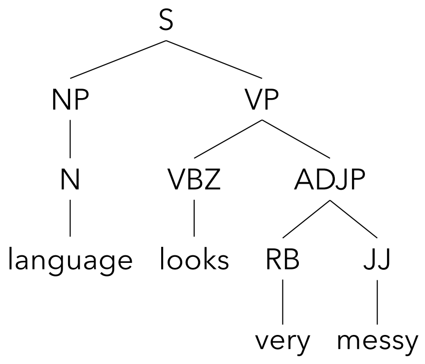


Figure 1.5: Syntactic tree of the sentence *language looks very messy*.

the meaning of its constituents, and the ways in which they are combined (Dowty et al., 1981). Thus, the meaning of the sentence *language looks very messy* is given by the meaning of its words, combined into larger and larger constituents as determined by its syntax (Figure 1.5). It is then a function of the meaning of the noun *language*, combined as a subject with the meaning of the verb phrase *looks very messy*, which is in turn given by the meaning of the verb *looks* modified by the meaning of the predicative adjective phrase *very messy*. Finally, the latter is given by the meaning of the adjective *messy*, modified by the meaning of the adverb *very*.

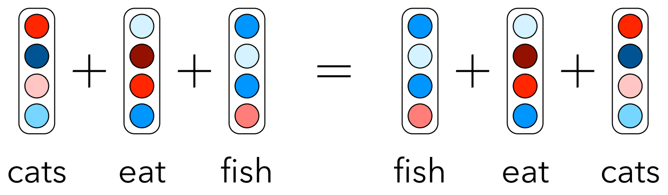
Semantic productivity is often brought forward as an argument supporting the principle of compositionality, with Frege (1980) claiming

> The possibility of our understanding [sentences] which we have never heard before rests evidently on this, that we can construct the sense of a [sentence] out of parts that correspond to words.[5]

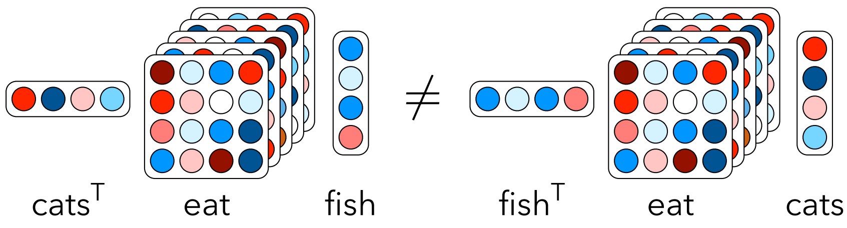
Standard distributed models of semantics, described in the previous section, only deal with the meaning of words as individual units. However, a number of extensions have been developed in recent years to model the meaning of larger linguistic units, in line with the principle of compositionality.

Bag-of-words models The simplest models of composition compute the meaning of multi-word constituents by combining individual word vectors using simple mathematical operations such as vector addition. It is a very common approach which yields good performance in simple phrase similarity tasks (Mitchell and Lapata, 2008, 2010; and references therein), but it has the drawback of being based on a commutative operation, making this a fundamentally order- and syntax-insensitive *bag-of-words* model. While this does not contradict the principle of compositionality, this simplistic approach leads to absurd situations, such as the one illustrated in Figure 1.6a. This method, as well as closely related models using operations such as element-wise multiplication and circular convolution, is reviewed in Mitchell and Lapata (2008, 2010) and Polajnar et al. (2014b), inter alia.

[5]The square brackets are part of the amended translation given in Szabó (2001).



(a) Who's eating whom? The additive compositional model is insensitive to word order.



(b) A hypothetical implementation of the categorial framework where sentence meanings live in a five-dimensional space, and noun meanings in a four-dimensional one.

Figure 1.6: Representation of the sentences *cats eat fish* and *fish eat cats* in an additive and a categorial compositional model.

Categorial framework One feature that brings the above models together, other than their use of simple mathematical operations, is their assumption that words, as well as larger linguistic units, are represented by vectors living in the same semantic space. An alternative approach is to more closely follow the compositional process of formal semantics (Dowty et al., 1981), by building a semantic representation in step with the syntactic derivation, and letting the representations of words be determined by their syntactic type. Coecke et al. (2011) achieve this by treating relational words such as verbs and adjectives as functions in the semantic space. The functions are assumed to be multilinear maps, and are therefore realised as tensors, with composition being achieved through tensor contraction.[6] Thus, for instance, words are represented as vectors, adjectives as matrices,[7] transitive verbs as third-order tensors, and so forth. A simple example illustrating the differences between the representations of the sentences *cats eat fish* and *fish eat cats* is shown in Figure 1.6b and should be contrasted with Figure 1.6a. The figure shows the transitive verb *eat*, which is a relational word taking two arguments (subject and object), represented as a third-order tensor. In this particular example, we have taken the space of noun representations to be four-dimensional, and the space of sentence representations to be five-dimensional, making the *eat* tensor $4\times5\times4$-dimensional. Following Clark et al. (2016), we will call this approach the *categorial framework*. A more in-depth review of this specific family of models, as well as its performance gains compared to bag-of-word models, will be presented in the opening chapter of Part II of this thesis.

[6] Baroni et al. (2013) and Paperno et al. (2014) develop similar approaches.
[7] An approach also suggested by Baroni and Zamparelli (2010).

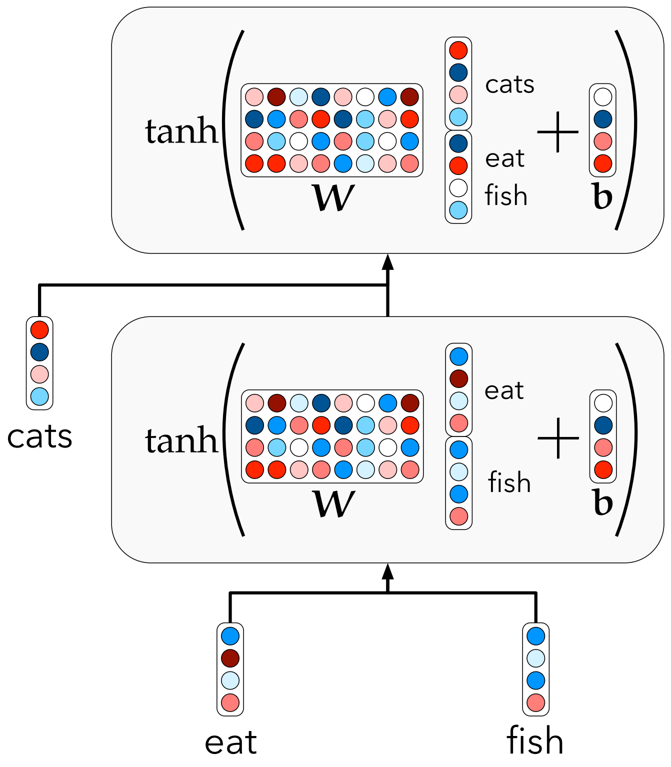


Figure 1.7: Sketch of the simple RNN model from Socher et al. (2010), processing the sentence *cats eat fish* according to the tree `( cats ( eat fish ) )`.

Neural compositional models Both the addition model and the categorial models rely on multilinear functions to perform composition. More recent compositional models are based on *neural networks*, which are inherently nonlinear. Good examples are the recursive neural network architectures of Socher et al. (2010), the simplest of which we illustrate in Figure 1.7: the model uses vectors to represent the meaning of words, and combines them according to a given parse tree using an affine transformation and a nonlinear function. Further examples are variants of this recursive architecture that are lexicalised (Socher et al., 2012) or parametrised on the syntactic categories (Socher et al., 2013; Hermann and Blunsom, 2013); convolutional networks (Collobert and Weston, 2008; Kalchbrenner et al., 2014; Kim, 2014; Mou et al., 2016); or more complex tree-structured approaches based on the treeLSTM architecture (Tai et al., 2015; Zhu et al., 2015; Bowman et al., 2016). These models, as well as the categorial ones, use the syntactic structure of sentences to drive their composition, and therefore will require parse trees to be provided at runtime, either from an automatic parser or in the form of annotations from trained experts. Chapter 6 will contain a more thorough review of some of these models. *Latent tree learning* models, which automatically induce a composition structure which is optimal for the downstream task and therefore do not require parse trees, are the subject of Part III of this thesis.

Linear-chain RNNs Finally, a very large number of works have been published which obtain sentence representations by using *recurrent neural networks* with a *linear chain* structure. They have been successfully applied to a wide range of linguistic tasks: language modelling (Sundermeyer

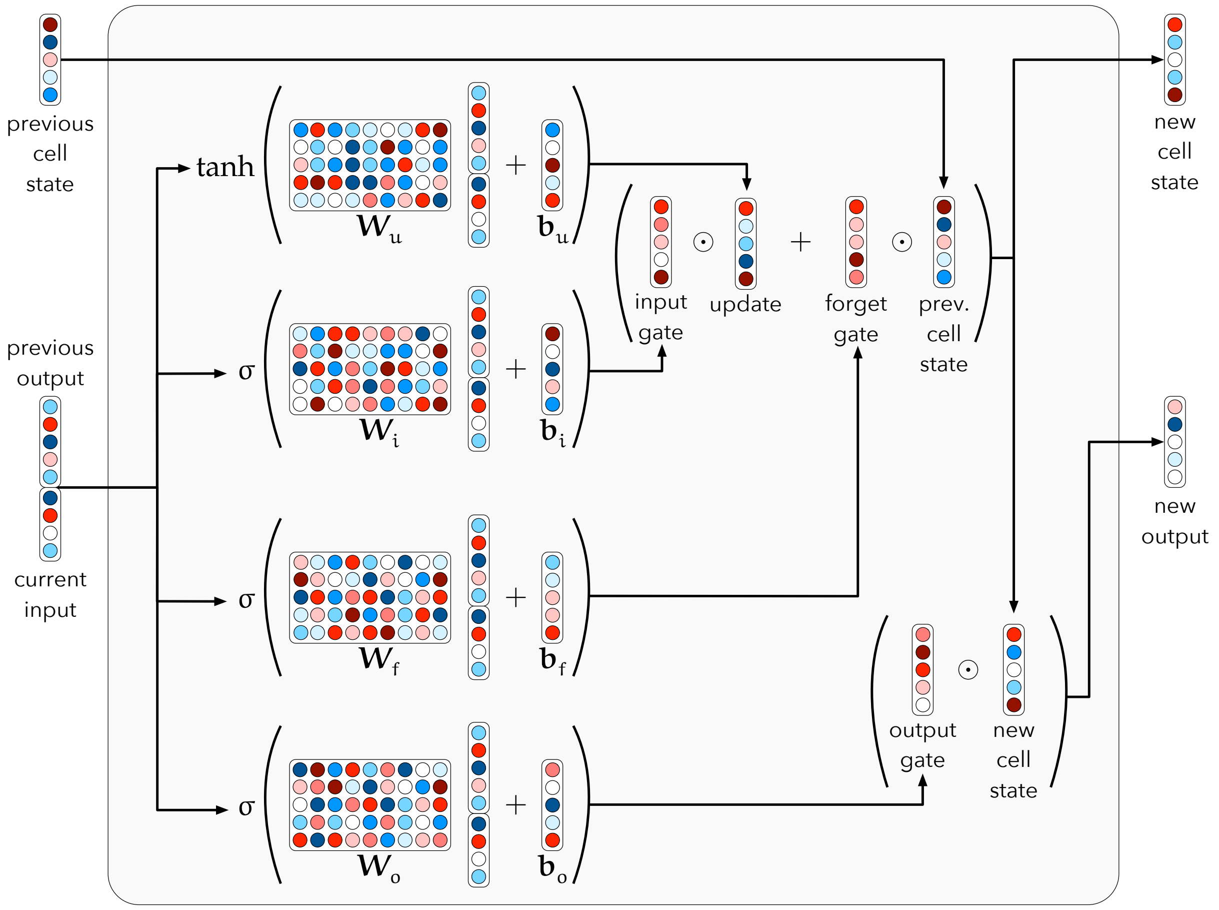
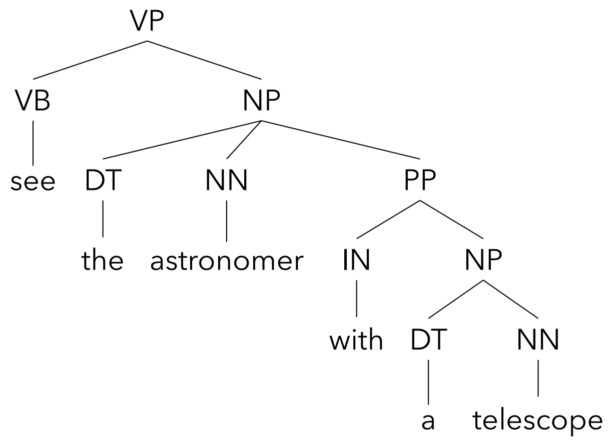


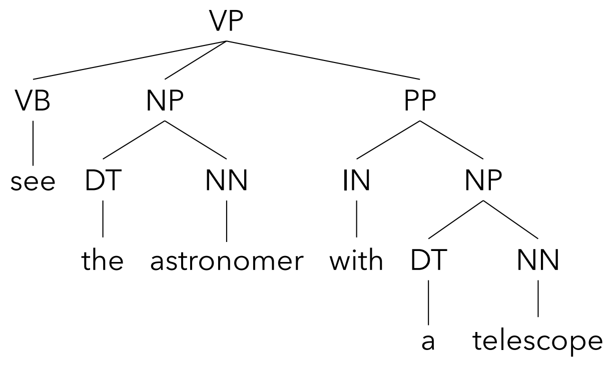
Figure 1.8: Long Short-Term Memory (LSTM) cell (Hochreiter and Schmidhuber, 1997). At each time step it receives an input, the previous output, and its internal state.

et al., 2012; Jozefowicz et al., 2016), word sense disambiguation (Yuan et al., 2016), and machine translation (Sutskever et al., 2014) amongst others. The most commonly used architectures are LSTMs (Hochreiter and Schmidhuber, 1997; illustrated in Figure 1.8), GRUs (Cho et al., 2014), as well as their bidirectional variants (Schuster and Paliwal, 1997; Graves and Schmidhuber, 2005) and extensions using attention mechanisms (Bahdanau et al., 2015). They differ from simple RNNs by having mechanisms which help them better model long-range dependencies, which we will look at in Chapter 6. LSTM and variants continue to be a very popular choice when needing to encode the meaning of a sentence into a vector, even though the treeLSTM architecture (Tai et al., 2015) is starting to outperform them on an increasing number of tasks. Indeed, Manning (2017) recently stated that 'the de facto consensus in NLP in 2017 is that no matter what the task, you throw a [bidirectional LSTM] at it'. Recent work which demonstrates the effectiveness of these architectures is the attempt to learn general-purpose sentence representations. Of particular note are the approaches of Subramanian et al. (2018), based on the GRU architecture, and Conneau et al. (2017), based on a bidirectional LSTM encoder.

It should be noted that while LSTM and variants can be seen as conforming to the principle of compositionality in a strict sense, they do so in a somewhat simplistic way. As their composition order is always linear, typically left-to-right, they are effectively assuming that words are always



(a) See the astronomer who is carrying a telescope.



(b) See the astronomer by looking through a telescope.

Figure 1.9: Two equally valid syntatic trees for *see the astronomer with a telescope*.

put together according to a fully left-branching tree. While there exist formalisms that would allow a fully left-branching composition order (e.g. Lambek, 2008; Steedman, 2000), standard recurrent neural networks cannot be parametrised to support the mechanisms, such as type-raising, that are required by these formalisms. As such, this family of models can only be considered to be compositional in a weak sense.

1.4 Summary

We began this chapter with a brief introduction to traditional symbolic methods in NLP, which represent the meaning of phrases and sentences as logical forms with a model-theoretic treatment.[8] These approaches can provide a precise treatment of aspects such as logical connectives and quantifiers (Montague, 1973), and shine in applications that can make use of the associated inference mechanisms (Blackburn and Bos, 2005). What they lack, however, is the ability

[8] For a more thorough discussion see Dowty et al. (1981), Cann (1993), and Blackburn and Bos (2005).

to easily characterise the meaning of content words – i.e. words such as nouns, verbs, and adjectives that have intrinsic meaning, as opposed to function words such as conjunctions and articles. These systems are brittle, as they can only deal with words that are part of their ontologies, and grammatical constructs that match their rules. Further, they also lack an intrinsic notion of similarity between constituents which, as discussed in § 1.1, is useful for NLP tasks.[9]

The rest of the chapter was dedicated to the presentation of distributed semantic models, starting from the meaning of individual words, and extending it to the meaning of larger linguistic units with the use of compositional models. Distributed models have a robust, comprehensive, data-driven approach to the characterisation of the meaning of content words, and provide ways of dealing with unknown tokens – either via a special embedding (see e.g. the implementation of Pennington et al., 2014) or by other approaches such as character- or *n*-gram-based algorithms (e.g. Bojanowski et al., 2017).

Are distributed models, then, better in every way? While traditional symbolic methods are able to represent the meaning of phrases and whole sentences through composition in a well-defined way, despite recent successes it is not yet clear how to best achieve this with distributed models. This will be the main topic of my dissertation. In § 1.3 we reviewed the existing literature on the subject. We saw that there are approaches, such as *bag-of-words* models and linear-chain RNNs, that follow the principle of compositionality only in a weak sense. Other approaches, e.g. categorial models and Socher et al. (2010), take a stronger stance and use linguistically-motivated, syntax-driven composition functions. In Parts II and III we will look at my own proposals for compositional models, which belong to this second class of linguistically-motivavetd approaches. Two will be based on the categorial framework, and two on the treeLSTM architecture.

A further open question is how to incorporate concepts from formal semantics into distributed models, to be used in applications such as question answering and inference. Distributed methods are still not able to reason about language in the elaborate and interpretable ways which are possible in a logic-based system such as the one described by Blackburn and Bos (2005). While there has been promising recent work on inference about entailment and contradiction in sentences (Bowman et al., 2015; Williams et al., 2018b) and question answering (Lewis and Steedman, 2013; Rajpurkar et al., 2018), this remains an open problem.

[9]A good discussion of the limitations of formal semantic approaches can be found in Boleda and Herbelot (2016) and the references therein. For a more informal treatment, see also the scenarios presented in Gazdar (1996).

2 Thesis contributions

Table 2.1: A taxonomy of vector-based compositional semantic models, showing a sample of representative papers. My contributions are highlighted.

		Composition function	
		Multilinear	**Nonlinear**
Composition structure	**Provided**	Baroni and Zamparelli (2010)	Socher et al. (2012)
		Grefenstette et al. (2013)	Socher et al. (2013)
		Paperno et al. (2014)	Tai et al. (2015)
		Maillard and Clark (2015)	Zhu et al. (2015)
		Rimell et al. (2016)[1]	Bowman et al. (2016)
	Induced	——	Socher et al. (2011)
			Yogatama et al. (2017)
			Maillard et al. (2017)
			Choi et al. (2018)
			Maillard and Clark (2018)

This thesis contributes a number of vector-based models of semantics that respect the principle of compositionality. In Part II, Chapters 3 to 5, we explore new categorial models that can be seen as implementations of the theoretical framework of Coecke et al. (2011). First, in Chapter 4, we look at the specific case of subject and object relative clauses, and implement a number of tensor-based models for it, which have been separately published in Rimell et al. (2016).[1] Then, in Chapter 5, we look at a new approach to learning matrices for categorial models, which was published separately in Maillard and Clark (2015). This differs from the more standard approaches of Baroni and Zamparelli (2010), Guevara (2010), and Paperno et al. (2014) by being the analogue, for matrices, of the prediction-based models of word semantics already discussed in § 1.3.

The final part of this thesis, Part III, Chapters 6 to 8, deals with latent tree learning models. These models are similar to the recursive neural networks already discussed in § 1.3, with one crucial

[1]My contributions in Rimell et al. (2016) are as follows: designing, coding, and running the algorithms for learning holistic vectors together with word embeddings, described in § 5.1.3 of the article; coding and running of the verb matrix learning algorithms described in § 5.2.2, as well as the code for learning relative pronoun tensors; coding and running the categorial compositional methods described in § 5.5. This will be further clarified in Chapter 4.

difference: rather than relying on externally provided parse trees similar to the ones that would be assigned by trained linguists, these models automatically induce a task-specific 'grammar' based on their downstream task (Williams et al., 2018a). They achieve this by integrating, within the compositional model, a natural language parser which is either differentiable (and therefore trainable via backpropagation), or trained via reinforcement learning. We will look at two new models: in Chapter 7 we will examine the first latent tree learning model based on chart parsing; and in Chapter 8 the first one based on the combination of shift-reduce parsing and beam search. We will see how they are able to outperform compositional models which use traditional syntax trees. Finally, we will analyse the trees that they induce, comparing and contrasting them to traditional ones.

An alternative way of navigating through parts of the sea of literature describing compositional models is to classify them based on two features: the nature of the composition function, which can either be multilinear (e.g. for the categorial models) or nonlinear (e.g. for the recursive neural models); and the origin of the composition structure, which can be either externally provided (e.g. by a parser, for the categorial and recursive neural models) or automatically induced (as is the case for the latent tree learning models). This view is illustrated schematically in Table 9.1.[2] If we take this picture as a guiding map, then in Part II of this thesis we will explore the upper left section of the table, and in Part III the lower right section.

2.1 Publications

This thesis builds on the following papers, which I list in reverse chronological order:

- **J. Maillard**, S. Clark, D. Yogatama (In press). 'Jointly Learning Sentence Embeddings and Syntax with Unsupervised Tree-LSTMs'. *Natural Language Engineering*. (Extended version of `arXiv:1705.09189`).
- **J. Maillard**, S. Clark (2018). 'Latent Tree Learning with Differentiable Parsers: Shift-Reduce Parsing and Chart Parsing'. In: *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*. Melbourne, Australia.
- **J. Maillard**, S. Clark, D. Yogatama (2017). 'Jointly Learning Sentence Embeddings and Syntax with Unsupervised Tree-LSTMs'. `arXiv:1705.09189`.
- L. Rimell, **J. Maillard**, T. Polajnar, S. Clark (2016). 'RELPRON: A Relative Clause Evaluation Data Set for Compositional Distributional Semantics'. *Computational Linguistics*, 42.4, pp. 661–701.

[2]Kartsaklis (2014; p. 26) presents a related taxonomy of vector-based compositional semantics. They differentiate models just based on the nature of their composition function, and follow with an interesting discussion of their theoretical power and distinguishing features.

- **J. Maillard**, S Clark (2015). 'Learning Adjective Meanings with a Tensor-Based Skip-Gram Model'. In: *Proceedings of the 19th Conference on Computational Natural Language Learning*. Beijing, China.

The following papers, while not included in this thesis, relate to its topical matter:

- V. G. Djokic, **J. Maillard**, L. Bulat, E. Shutova (2019). 'Modeling Affirmative and Negated Action Processing in the Brain with Lexical and Compositional Semantic Models'. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy.
- E. Shutova, D. Kiela, **J. Maillard** (2016). 'Black Holes and White Rabbits: Metaphor Identification with Visual Features'. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California. **Best paper runner up**.
- S. Clark, L. Rimell, T. Polajnar, **J. Maillard** (2016). *The Categorial Framework for Compositional Distributional Semantics*. Technical report, University of Cambridge.

Finally, the source code used in the experiments is available on my personal website, `www.maillard.it`.

Part II

Categorial compositional models

3 Categorial models

Having given a general overview of vector-based word semantic models and their compositional extensions in Chapter 1, we will now look into more detail at the framework introduced by Coecke et al. (2011), and its application to Combinatory Categorial Grammar (CCG, see Steedman, 2000). Following Clark et al. (2016), we will call the framework of Coecke et al. *categorial*, and its implementations *categorial models*. We will introduce the main theoretical result of Coecke et al. (2011), and discuss some of its first concrete implementations. Following a presentation of the core aspects of CCG, we will see how this formalism can be seamlessly integrated with the categorial framework, to obtain a practical tensor-based compositional distributed model of semantics. This chapter will serve as an introduction to Chapters 4 and 5, which describe my contributions in this area.

3.1 The categorial framework

Coecke et al. (2011) make the crucial observation that pregroup categorial grammars (Lambek, 2008) – a context-free variant of categorial grammars (Buszkowski, 2001) – share a common structure with the vector spaces of distributed models from the point of view of category theory: both can be represented as *compact closed categories*, a special case of categories which are additionally equipped with the notion of tensor product, and where each object has a left and right adjoint, subject to coherence conditions.[1] Indeed vector spaces – which are used in distributed models to represent the semantics of words – together with linear maps and tensor products, can be seen as a compact closed category. The same is also true of the syntactic types of a pregroup grammar, together with adjoints and multiplication. These categories also admit a practical form of diagrammatic calculus, which can be useful to intuitively depict grammatical reductions (as demonstrated in Clark et al., 2008; Coecke et al., 2011; inter alia).

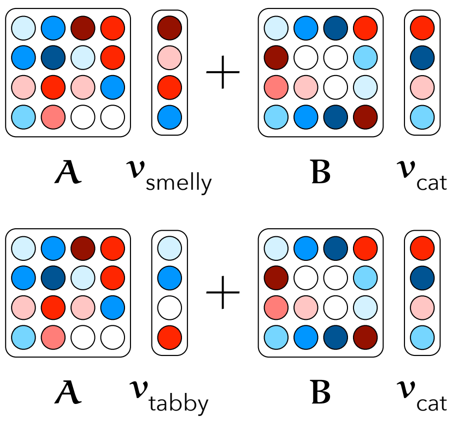
Using this unifying structure, Coecke et al. describe a category-theoretic framework in which the compositional nature of pregroup grammars is mapped to the category of vector spaces. This allows the assignment of meaning to well-formed compound linguistic units, starting from the meaning of their constituents.

[1]A discussion of these concepts is beyond the scope of this thesis. Several excellent introductions to category theory exist, such as the classic textbook by Mac Lane (1978), or the more recent presentations by Lawvere and Schanuel (2009) and Coecke and Paquette (2010).

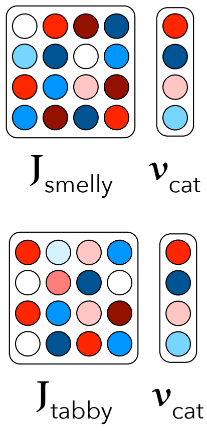
Adjective-noun composition One of the earliest proposals that can be seen as an implementation of this theoretical framework, even though it was developed separately, is the adjective-noun model of composition by Baroni and Zamparelli (2010). The authors start from the idea from formal semantics that attributive adjectives can be seen as functions from the meaning of a noun to the meaning of the modified noun – in mathematical terms, $f_{\text{smelly}} : \texttt{cat} \mapsto \texttt{smelly\_cat}$. They then propose a concrete implementation of such functions for distributional models of semantics, in the form of linear maps acting on word vectors, e.g. square matrices. The matrices are trained via linear regression to approximate a set of adjective-noun semantic vectors. These vectors are extracted from the corpus, in an analogous way to how the individual word vectors are learned, by considering the adjective-noun as a single token. The model is illustrated in Figure 3.1b for the two phrases *smelly cat* and *tabby cat*, and the resulting transformations in a hypothetical semantic space are shown in Figure 3.1c. A related model by Guevara (2010), which uses two separate matrices (shared by all adjectives), is shown in Figure 3.1a. A third model which can be applied to adjective-noun composition is by Maillard and Clark (2015). It can be seen as an extension of the skip-gram model of Mikolov et al. (2013b), and will be described in detail in Chapter 5.

General composition Grefenstette and Sadrzadeh (2011a) propose an implementation of the categorial framework for relational words, and demonstrate its application to transitive and intransitive verbs. Concretely, letting N be the vector space of noun meanings, they propose that the tensor of a relational word with n arguments live in the space $N^{\otimes n}$, i.e. n tensor products of the noun space with itself. Composition is performed via the element-wise product of the relational word tensor R with the Kronecker product of the vectors of its arguments $\mathbf{a}_i$, i.e. $R \odot (\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_n)$. Therefore, transitive verb tensors live in $N \otimes N$. The authors propose calculating their representation as $V = \sum_i \mathbf{s}_i \otimes \mathbf{o}_i$, where $(\mathbf{s}_i, \mathbf{o}_i)$ are the pairs of vectors for the corpus-observed subjects and objects of the corresponding verb. An alternative way of calculating the verb tensor representations, which is shown to perform better in a disambiguation task, is to simply compute the Kronecker product of the verb's word vector with itself (Grefenstette and Sadrzadeh, 2011b).

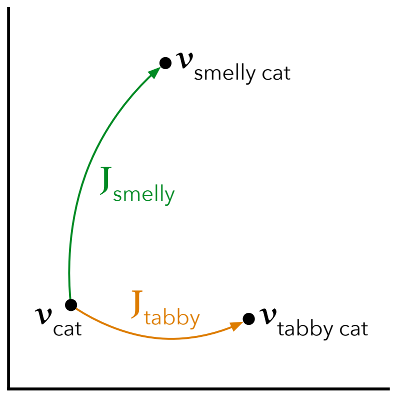
The two methods presented above are based on Kronecker products of arguments and element-wise multiplication. Due to their use of multiplication, they cannot be meaningfully applied to vectors which contain negative values (Grefenstette et al., 2013) which are, however, very common in NLP due to methods such as skip-gram (Mikolov et al., 2013b) or GloVe (Pennington et al., 2014), or the use of dimensionality reduction techniques such as SVD. A different family of verb composition, which does not have this limitation and has been shown to perform better, is the extension of the adjective-noun regression approach of Baroni and Zamparelli (2010) to relational words of greater arity. Grefenstette et al. (2013) propose an approach using *multi-step regression*, and demonstrate learning third-order transitive verb tensors: this involves first learning matrices for verb phrases such that, when multiplied with subject vectors, they approximate corpus-extracted sentence vectors; and then learning a verb tensor which,


$\mathbf{A}$
$\mathbf{v}_{\text{smelly}}$
$\mathbf{B}$
$\mathbf{v}_{\text{cat}}$
$\mathbf{A}$
$\mathbf{v}_{\text{tabby}}$
$\mathbf{B}$
$\mathbf{v}_{\text{cat}}$

(a) Guevara


$\mathbf{J}_{\text{smelly}}$
$\mathbf{v}_{\text{cat}}$
$\mathbf{J}_{\text{tabby}}$
$\mathbf{v}_{\text{cat}}$

(b) Baroni and Zamparelli


$\mathbf{v}_{\text{smelly cat}}$
$\mathbf{J}_{\text{smelly}}$
$\mathbf{v}_{\text{cat}}$
$\mathbf{J}_{\text{tabby}}$
$\mathbf{v}_{\text{tabby cat}}$

(c) Linear transformations

Figure 3.1: Adjective-noun phrases *smelly cat* and *tabby cat* according to the compositional models of Guevara (2010) and Baroni and Zamparelli (2010). Below, the linear transformations corresponding to Baroni and Zamparelli's adjective-specific matrices on a 2D projection of a hypothetical noun phrase vector space.

when multiplied with object vectors, approximates the corresponding verb phrase matrix. Composition is then performed by multiplying the verb tensor with the vectors of its object and subject.

Polajnar et al. (2014a) propose to avoid issues with data sparseness and computational complexity that are associated with learning high-order tensors by proposing several low-dimensional approximations, and Fried et al. (2015) attempt the same by using tensor rank decompositions, representing transitive verb tensors as sums of tensor products of vectors. In the same spirit, Paperno et al. (2014) propose the Practical Lexical Function model (henceforth PLF): in their approach, relational words of n arguments are not represented as an n^{th} order tensor, but rather as a single vector together with a set of n matrices. In order to perform composition, each matrix is multiplied with the corresponding argument's vector, and all resulting vectors are added together, along with the relational word's vector. This is illustrated in Figure 3.3 for the case of a transitive verb *eat*. In a task involving composition of sentences involving verbs and adjectives,

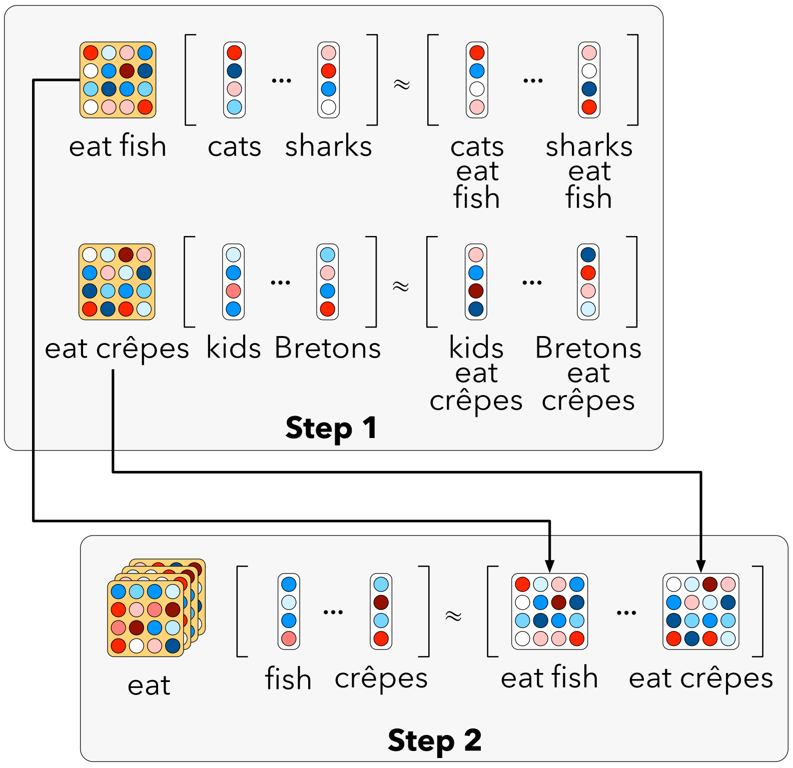


Figure 3.2: Multi-step regression (Grefenstette et al., 2013). Highlighted in orange are the tensors which are being learned via regression: in step 1, the verb phrase matrices; and in step 2, the verb tensor.

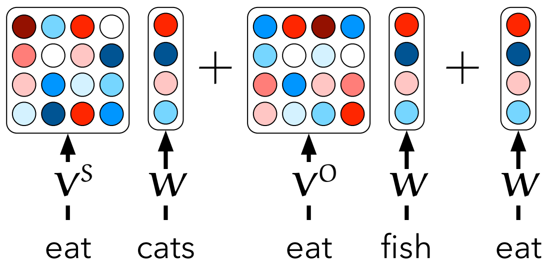


Figure 3.3: Transitive verb composition for *cats eat fish* in the PLF model. Gupta et al. (2015) suggest dropping the last summand, i.e. the word embedding of the verb.

PLF was shown to outperform baselines such as addition and element-wise multiplication, as well as composition with tensors learned via multi-step regression (see Paperno et al., 2014; whose results we report in Table 3.1).

Further developments Other aspects of natural language have been investigated in the context of the categorial framework. While categorial models have been mainly applied to tasks involving phrase similarity, Balkır et al. (2018) argue that they can also be used for textual entailment tasks, and provide preliminary results in support of this position. Grefenstette (2013b) shows how aspects of predicate logic, such as connectives and quantifiers, can be replicated in a tensor-based framework. Sadrzadeh et al. (2013, 2016) discuss relative pronouns and perform

Table 3.1: Performance of transitive verb composition models on two tasks involving predicting the similarity of sentences with the structure adjective-noun-verb-adjective-noun. The numbers given are the Spearman rank correlation coefficients with human similarity ratings. ANVAN1 is by Kartsaklis et al. (2013); ANVAN2 is by Grefenstette (2013a). All numbers are as reported in Paperno et al. (2014; table 5). We also show the inter-annotator agreement (denoted *humans* below), which serves as an upper bound on the performance.

Model	ANVAN1	ANVAN2
Addition	0.08	0.22
Multi-step	0.15	0.30
PLF	**0.20**	**0.36**
Humans	*0.38*	*0.48*

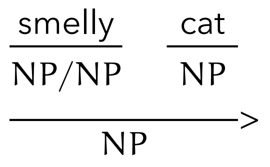
preliminary experiments. Taking this work as inspiration, Rimell et al. (2016) implement a number of categorial models for relative clause composition, including an extension of the PLF model of Paperno et al. (2014) and a tensor-based model. The work of Rimell et al. will be discussed in detail in Chapter 4, including a full description of the models used and the results of an evaluation involving term retrieval from relative clauses.

3.2 Combinatory Categorial Grammar

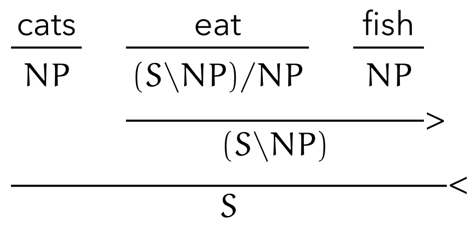
The original formulation of the categorial framework was made in terms of pregroup grammars, due to their sharing common structure with vector spaces and linear maps. In this thesis, we prefer to implement the framework in terms of a different grammatical formalism, CCG. This idea was originally suggested by Grefenstette (2013a), and later expanded upon by Maillard et al. (2014) and Grefenstette and Sadrzadeh (2015). While CCG and pregroup grammars are theoretically different formalisms, and are therefore not generally interchangeable, Buszkowski and Moroz (2008) and Grefenstette and Sadrzadeh (2015) observe how CCGs are equivalent to pregroup grammars, and can therefore be used within the categorial framework.

Our choice is motivated by the wide use of CCG within NLP (Hockenmaier and Steedman, 2007; Krishnamurthy and Mitchell, 2014; Zettlemoyer and Collins, 2007; Nadejde et al., 2017; inter alia), the availability of several high-quality parsers (Clark and Curran, 2007; Xu et al., 2014; Xu, 2016; Lewis et al., 2016) and, finally, the ease with which this grammatical formalism can be integrated with the categorial framework (Grefenstette, 2013a; pp. 136–147; Maillard et al., 2014).

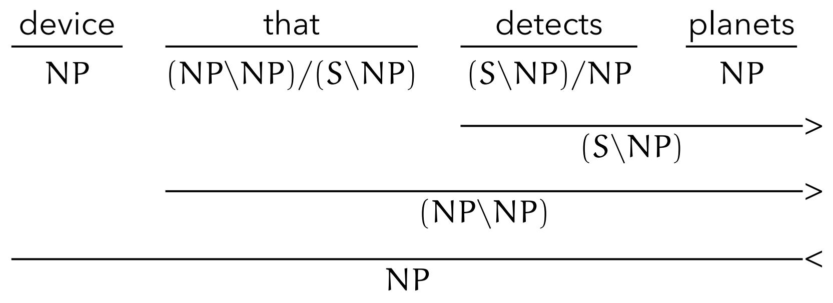
In CCG (Steedman, 2000), all constituents are assigned a *type*, which identifies them as either *functions* or *arguments*. These combine in various ways according to *combinatory rules*, to ultimately produce the syntactic derivation of a linguistic unit. Types in CCG are defined recursively



(a) Adjective-noun phrase composition with forward application.



(b) Subject-verb-object composition with forward and backward application.



(c) Subject relative clause composition with forward and backward application.

Figure 3.4: CCG derivations of *smelly cat*, *cats eat fish*, and *device that detects planets*, demonstrating the use of forward application (>) and backward application (<).

in terms of the primitive types, which are the types of atomic arguments. Here we will assume there are only two: the type of nouns and noun phrases, NP; and the type of sentences, S.

Functions have composite types, which specify the number, type, and direction of their arguments, and the type that results from their application (their codomain). A function taking an argument of some type Y to the right, and resulting in some type X, has type $\mathrm{X/Y}$. A function taking an argument of type X to the left, and resulting in type Y, has type $\mathrm{X\backslash Y}$. Brackets are used to avoid ambiguity in the precedence of slashes. Adjacent constituents can be reduced by applying the combinatory rules of the grammar. In this thesis, we will mainly be concerned with two rules: forward (rightward) application and backward (leftward) application,[2] denoted by $>$ and $<$ respectively.

$$\begin{aligned} \mathrm{X/Y} \quad \mathrm{Y} \quad &\Longrightarrow \mathrm{X} \qquad (>) \\ \mathrm{Y} \quad \mathrm{X\backslash Y} \quad &\Longrightarrow \mathrm{X} \qquad (<) \end{aligned}$$

Forward application combines any function with type $\mathrm{X/Y}$ with an immediately following Y, yielding X. Backward application works analogously, but for the case of function words expecting arguments to their left.

To illustrate these concepts, let us consider the case of adjective-noun phrase composition. As

[2]In the parser of Clark and Curran (2007), for example, a larger set of rules is implemented including (generalized) composition, leading to a mildly context sensitive grammar. In this thesis we only consider a small set of grammatical constructions, and so do not need the full grammar. However, the categorial framework applies to CCG more generally, including type-raising and composition rules (Maillard et al., 2014).

discussed above, nouns and noun phrases have primitive type NP. Adjectives combine with nouns to their right to produce a noun phrase, and thus have type NP/NP. Figure 3.4a shows how, through the rule of forward application, adjective *smelly* combines with noun phrase *cat* to yield a noun phrase, which has type NP. A similar example is given in Figure 3.4b depicting the composition of a subject-verb-object sentence. Transitive verb *eat*, which expects a direct object to the right and a subject to the left, is combined with two noun phrases through forward and backward application. The result is a sentence, which has primitive type S. Finally, Figure 3.4c demonstrates composition of a relative clause.

Additional combination rules are forward and backward composition, denoted $B_>$ and $B_<$ respectively, which are analogous to the composition of functions in mathematics; and forward and backward type-raising, denoted $T_>$ and $T_<$ respectively, which are used to turn argument types into function types. They are defined as follows:

$$\begin{array}{rcll} X/Y \quad Y/Z & \Longrightarrow & X/Z & (B_>) \\ Y\backslash Z \quad X\backslash Y & \Longrightarrow & X\backslash Z & (B_<) \\ X & \Longrightarrow & T/(T\backslash X) & (T_>) \\ X & \Longrightarrow & T\backslash(T/X) & (T_<) \end{array}$$

CCG tightly couples syntax with semantics, as each category can be augmented with its semantic type, and the combinatory rules apply in the same way to the semantic representations. Using λ-calculus to encode the semantics (as in Steedman, 2000) the meaning of the adjective *smelly* can be represented as $\lambda x\, \mathtt{smelly}(x)$, and the noun *cat* as the symbol $\mathtt{cat}$. Application rules correspond to function application, so that combining *smelly cat* via forward application will lead to a semantic interpretation of $\mathtt{smelly}(\mathtt{cat})$. The slightly more involved case of subject-verb-object composition is demonstrated in Figure 3.5 for the sample sentence *cats eat fish*. First, a forward application is performed, replacing the bound variable y in the semantic representation of *eat* with the semantic representation of *fish*. Then, a backward application replaces the bound variable x with the semantic representation of *cats*.

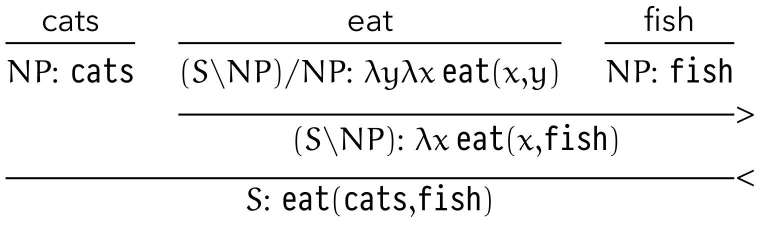


Figure 3.5: Subject-verb-object CCG derivation with semantic composition.

3.3 Tensor-based semantics

The semantic representations introduced in the previous section in terms of abstract λ-calculus are not vector-based, but are closer to the logical forms of the symbolic tradition of NLP described in Chapter 1. We will now show how the CCG formalism can be integrated with distributed models of semantics, following the framework of Maillard et al. (2014) to which we refer readers for a more complete treatment.

We start by assuming that the meanings of primitive types live in potentially distinct vector spaces. Thus, the type NP of nouns and noun phrases will have an associated semantic vector space N, the type S of sentences will have a space S, and in general $N \neq S$. Having done this, we further assume that words which are identified as functions in the syntax will also be treated as functions in the semantics. More specifically, we choose to model the semantics of all function words as a multilinear maps (encoded as tensors) of order $n + 1$, where n is the number of primitive types making up its CCG type. The specific tensor space of these maps is easily determined by taking the syntactic type of a function word, replacing all CCG primitive types by their associated vector spaces, and replacing all slashes by tensor products. Thus an adjective, of syntactic type NP/NP, has a semantic representation living in the space $N \otimes N$; and a transitive verb, of type $(S\backslash NP)/NP$, has a semantic representation living in $S \otimes N \otimes N$. Finally, forward and backward composition rules correspond, in the semantics, to tensor contraction. We can thus take a derivation such as the one shown in Figure 3.5 and turn it into a tensor formula, as shown in Figure 3.6.

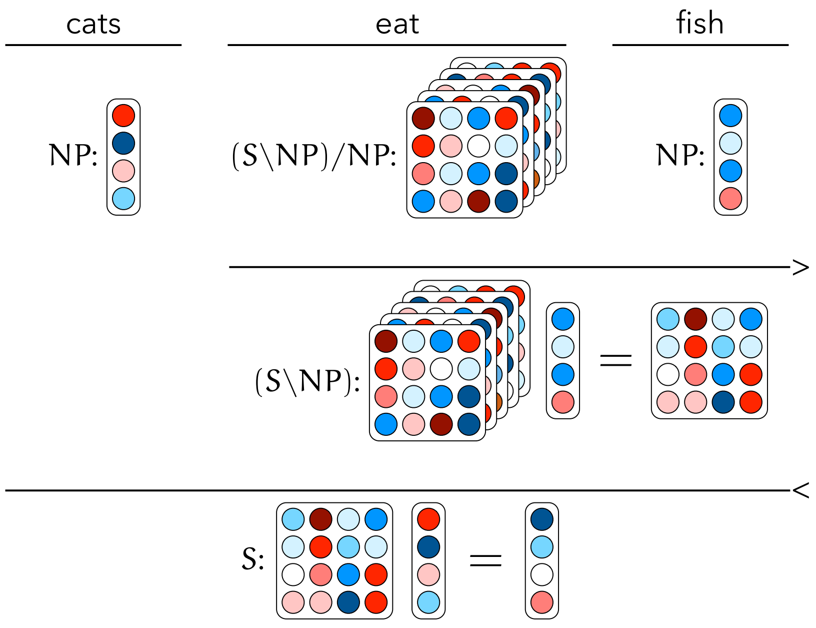


Figure 3.6: Subject-verb-object composition in the tensor-based CCG semantic framework.

Note how, for the adjective case, this fits perfectly with the model of Baroni and Zamparelli (2010) discussed in § 3.1: adjective meanings are represented as tensors in $N \otimes N$, i.e. they are $n \times n$ matrices; nouns meanings are represented as n-dimensional vectors; and adjective-noun composition corresponds to matrix multiplication. What is still missing, however, is how these vectors and matrices are learned: on this topic, this CCG-flavoured variant of the categorial framework – as described in Maillard et al. (2014) and Clark et al. (2016) – makes no assumptions. As such, it is still a theoretical framework, requiring practical implementations.

Several models fit this picture well, providing concrete implementations. Other than the aforementioned Baroni and Zamparelli (2010), we also have Grefenstette et al. (2013) who model transitive verbs as third-order tensors and use tensor multiplication. Two interesting approaches are those of Paperno et al. (2014) and Polajnar et al. (2014a) who, in order to get around the data sparsity issues that arise with tensors of higher orders, use simpler, lower-dimensional representations. Apart from the last, all these models have in common the way in which they learn the tensors and matrices corresponding to function words, initially proposed by Baroni and Zamparelli (2010): first a number of phrase vectors, all containing the same function word, are extracted from the corpus; then, the function word tensor or matrix is learned via regression to be close to the corpus-extracted phrase vectors when multiplied by the appropriate argument vectors.

The next two chapters will describe my contributions in this area. Both can be seen as practical implementations of the CCG-flavoured categorial framework described in this section. In Chapter 4 I will focus on relative clause composition, and test several models: the PLF model as proposed by Paperno et al. (2014), a proposed extension of PLF specific to relative clauses, and a new model based on a third-order relative clause tensor. Then, in Chapter 5, I will propose an alternative to regression for learning function word tensors, and evaluate it on two tasks involving adjective-noun compositionality.

4 Relative clause composition

Evaluation tasks for compositional distributed models have mostly involved measuring the similarity of simple phrases: adjective-noun phrases (Baroni and Zamparelli, 2010; Mitchell and Lapata, 2010; Vecchi et al., 2011; Boleda et al., 2012; Vecchi et al., 2017), subject-verb and verb-object combinations (Mitchell and Lapata, 2008, 2010; Grefenstette and Sadrzadeh, 2011a), or simple transitive sentences (Kartsaklis et al., 2013; Grefenstette, 2013a). While these datasets have been fundamental in evaluating the first generation of compositional distributed models, the simplicity of the grammatical structures that they evaluate means that a wide range of compositional phenomena is left untested. Other evaluation tasks involve measuring the similarity of pairs of full sentences (Agirre et al., 2012; Marelli et al., 2014; Agirre et al., 2015; Pham et al., 2013). While the grammatical constructions in these tasks are more complex, reducing sentence similarity to a single numerical value offers no insight into the performance of models on specific phrase types, and thus might mask areas where models need improvement (Rimell et al., 2016).

Feeling that a wider range of compositional phenomena should be investigated in compositional distributed models, my co-authors in Rimell et al. (2016) decided to build RELPRON, a dataset for the evaluation of subject and object relative clause composition. In the next section, § 4.1, I will describe the nature of RELPRON and how to evaluate compositional distributed models with it. In the following sections, §§ 4.2 to 4.4, I will discuss the experiments I ran using RELPRON, and the distributed models I evaluated. Namely, these are the PLF model of Paperno et al. (2014) and several ablated versions; a new extension of PLF to relative clauses; and a new approach involving third-order tensors, learned with a strategy reminiscent of Grefenstette et al. (2013), but without the ad hoc intermediate regression. Training these models required having semantic vectors for phrases, which are used as training targets when learning matrices and tensors for function words via linear regression. These phrase vectors were learned with an extension of skip-gram (Mikolov et al., 2013b) that I designed, which will be described in § 4.3. All results presented in this chapter were published separately in Rimell et al. (2016).

4.1 Dataset description

One of the inspirations for the RELPRON dataset was the toy experiment by Sadrzadeh et al. (2013) involving relative clauses. Starting from the motivation that relative clauses are often used to describe words, Sadrzadeh et al. chose a set of nine words and manually described them

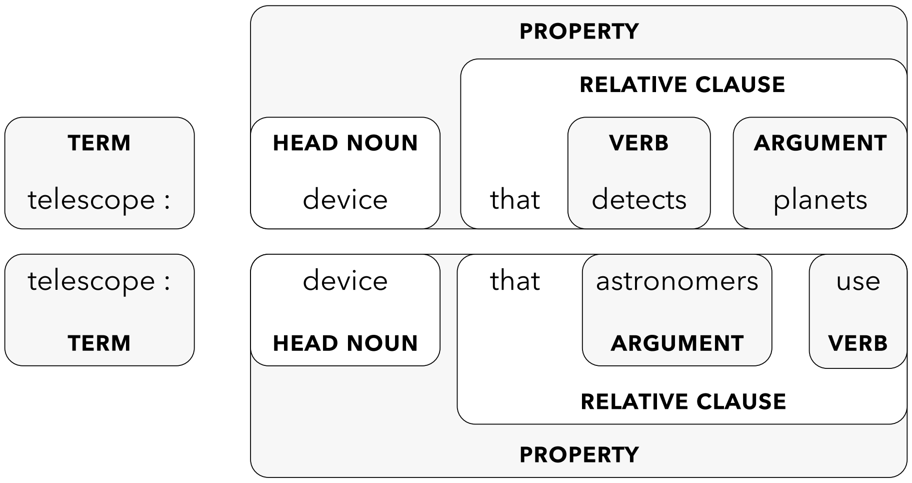


Figure 4.1: Terminology used to describe the terms and properties in the dataset. Above, the noun phrase with subject relative clause *device that detects planets*; below, the noun phrase with object relative clause *device that astronomers use*. Both refer to the term *telescope*.

using a noun phrase which includes a relative clause – for instance, they described *mammal* as *animal which gives birth*. The goal of their task was then to compute, according to the model being tested, how many of the words were closest in meaning to their descriptions.

The RELPRON dataset is made up of similar tuples, but is larger, and with more realistic noun phrases which are based on data extracted from a corpus. Rather than being definitions, the noun phrases in RELPRON express representative properties of various words, and are thus called *properties* in this dataset. Two examples, with a subject and an object relative clause, are '*telescope: device that detects planets*' and '*telescope: device that astronomers use*'. These are illustrated in Figure 4.1 along with the terminology used to describe various parts of the properties: we call *telescope* the *term*, *device* the *head noun*, and the verb's dependent the *argument* (this is the verb's object in a subject relative clause, and its subject in an object relative clause). Between four and ten properties were included in the dataset for each term.

Rather than directly extracting relative clauses from a corpus, which would have led to few examples due to sparsity issues, my co-authors extracted subject-verb-object triples from a 2010 Wikipedia dump and the British National Corpus,[1] using the C&C tools (Clark and Curran, 2007) to parse the data and extract the subject and object relations. Then, they joined the extracted triples with appropriate hypernyms as head nouns: for example, the extracted triple *astronomers use telescope* was joined with head noun *device* (a hypernym of *telescope*) to form the term-property tuple '*telescope: device that astronomers use*', which contains an object relative clause; and similarly the triple *telescope detects planets* formed the tuple '*telescope: device that detects planets*', which contains a subject relative clause. For simplicity, *that* was used as the only relative pronoun throughout the dataset. In order to make the task more challenging, and ensure that models could not simply rely on the similarity between terms and head nouns,

[1]The British National Corpus, version 3 (2007). `http://www.natcorp.ox.ac.uk/`.

multiple terms that shared the same head noun were chosen: for example, the terms *telescope*, *watch*, *button*, and *pipe*, being all hyponyms of *device*, were all assigned properties that start with *device that*. Finally, frequency cutoffs were applied, and the data was manually filtered to ensure that only terms with four or more good identifying properties were retained; and that only head nouns with at least four terms were retained.[2]

This procedure resulted in a total of 1087 tuples, including both subject and object relative clauses, modifying both abstract and concrete nouns. This was further divided into a development set of 518 properties (comprising 7 head nouns and 65 terms) and a test set of 569 properties (comprising 8 head nouns and 73 terms). The canonical task for RELPRON is, given a term, to rank all properties corresponding to it above other unrelated properties. For all experiments described in the following sections I use as evaluation measure the *mean average precision* (henceforth MAP), as suggested by Rimell et al. (2016).[3] It is defined as

$$\text{MAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}(t_i), \tag{4.1}$$

where N is the number of terms to be ranked, and AP is the *average precision*, defined as

$$\text{AP} = \frac{1}{P_t}\sum_{k=1}^{M} \text{Prec}(k) \cdot \mathbb{I}_t(k),$$

where P_t is the number of properties corresponding to term t according to the dataset; M is the total number of properties in the dataset; $\text{Prec}(k)$ is the precision at rank k; and $\mathbb{I}_t(k)$ is an indicator function, equal to one if the property at rank k is valid for term t, and zero otherwise.

4.2 Compositional models

I will now describe the various models that were used to compute distributed semantic representations of noun phrases modified by relative clauses. Apart from the baseline methods described below, all other models, which can be seen as implementations of the categorial framework described in Chapter 3, were implemented and evaluated by me.

Arithmetic methods My co-authors trained and evaluated a number of methods using three different sets of distributed vectors: my 100-dimensional skip-gram vectors; a set of 2000-dimensional count-based vectors built on Wikipedia, using the same settings as Grefenstette and Sadrzadeh (2011a); and count-based vectors reduced to 300 dimensions with SVD, using the same settings as Polajnar et al. (2014a). These methods, called here *arithmetic*, included using only the vector of

[2]For a more detailed step-by-step description of how my co-authors built RELPRON, see the paper, Rimell et al. (2016).

[3]This was chosen so that results would be comparable to those published in Rimell et al. (2016). As MAP is difficult to interpret, readers are advised to consider alternative, more standard metrics from information retrieval, such as recall and mean reciprocal rank.

the argument or the verb, and adding or element-wise multiplying the vectors for the argument, verb, and head noun. All these methods were evaluated on the development set. Only addition with the skip-gram vectors was found to be competitive with the other methods:

$$\mathbf{N}_h + \mathbf{N}_a + \mathbf{N}_v$$

where $\mathbf{N}$ is the word embedding matrix, and h, a, and v represent the indices of the head noun, argument, and verb respectively (see Figure 4.1 and the previous section for the definitions of these terms).

PLF and ablated variants The first model I implemented and evaluated was the Practical Lexical Function model (PLF) of Paperno et al. (2014). Transitive verbs have CCG type $(S\backslash NP)/NP$ which, according to the tensor-based semantic framework described in § 3.3, leads to them being represented as third-order tensors living in $S \otimes N \otimes N$. Paperno et al. simplify this in order to make learning more tractable, by modelling a transitive verb as two matrices in $S \otimes N$. One matrix, the verb-subject matrix, models the interaction of the verb with its subject, and the other, the verb-object matrix, with its object. Therefore, the composition of a subject-verb-object sentence is modelled in PLF as

$$\mathbf{V}^S_v \mathbf{N}_s + \mathbf{V}^O_v \mathbf{N}_o,$$

where $\mathbf{N}$ is the word embedding matrix as before; $\mathbf{V}^S$ and $\mathbf{V}^O$ are verb-subject and verb-object embedding tensors, which contain the full set of verb-subject and verb-object matrices; and s, v, and o are the indices of the subject, verb, and object respectively. The original paper by Paperno et al. also included a third term to the addition above, i.e. the word vector of the verb. Gupta et al. (2015) found that removing this third term yielded better results, and I also adopted this modification.

In PLF most grammatical words, including relative pronouns, are treated as 'empty' elements that do not project into semantics. This leads to noun phrases modified by relative clauses being treated as subject-verb-object sentences: for example, *device that detects planets* is equivalent to *device detects planets*, and *device that astronomers use* is equivalent to *astronomers use device*. Therefore, noun phrases modified by a subject and object relative clause are represented in PLF respectively as

$$\mathbf{V}^S_v \mathbf{N}_h + \mathbf{V}^O_v \mathbf{N}_a \quad (\text{SPLF, subject}) \qquad \text{and} \qquad \mathbf{V}^O_v \mathbf{N}_h + \mathbf{V}^S_v \mathbf{N}_a \quad (\text{SPLF, object}),$$

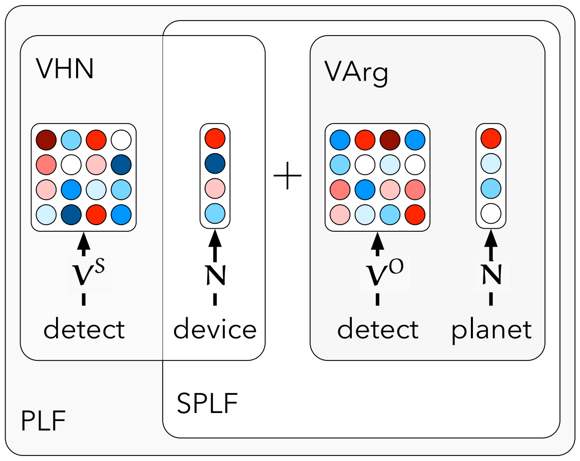
where h, a, and v represent again the indices of the head noun, argument, and verb.

I further evaluated three ablated variants of PLF. The first, called *simplified PLF* (henceforth SPLF), drops the interaction of the verb with the head noun:

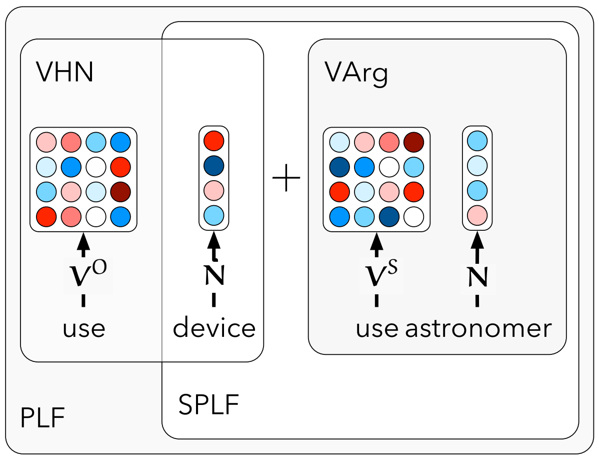
$$\mathbf{N}_h + \mathbf{V}^O_v \mathbf{N}_a \quad (\text{subject}) \qquad \text{and} \qquad \mathbf{N}_h + \mathbf{V}^S_v \mathbf{N}_a \quad (\text{object}).$$

The second, VArg, only models the verb-argument composition, effectively dropping the head noun altogether. The third, VHN, drops the argument. They are defined, respectively, as

$$\mathbf{V}^O_v \mathbf{N}_a \quad (\text{VArg, subject}) \qquad \text{and} \qquad \mathbf{V}^S_v \mathbf{N}_a \quad (\text{VArg, object}),$$



(a) Noun with subject relative clause: *device that detects planets*.



(b) Noun with object relative clause: *device that astronomers use*.

Figure 4.2: PLF model, as presented in Paperno et al. (2014) with the modification proposed by Gupta et al. (2015). Also shown are two ablated versions: a simplified alternative which effectively replaces the verb matrix modifying the head noun with the identity (SPLF); a version which drops the head noun altogether (VArg); and one which drops the argument (VHn)

$$\mathbf{V}^S_v \mathbf{N}_h \quad (\text{VHN, subject}) \qquad \text{and} \qquad \mathbf{V}^O_v \mathbf{N}_h \quad (\text{VHN, object}).$$

These ablated models, as well as the standard PLF, are illustrated in Figure 4.2.

Full PLF We then extended the PLF intuition to relative pronouns, with an approach that captures the interaction of the relative pronoun with the head and the verb-argument phrase via two matrices. In this model – which we call *full* PLF (FPLF) – instead of ignoring relative pronouns, a noun modified by a subject relative clause is composed as

$$\mathbf{R}^{S,va}\mathbf{V}^O_v \mathbf{N}_o + \mathbf{R}^{S,n}\mathbf{N}_h \quad (\text{RPTensor, subject}),$$

where $\mathbf{R}^{S,n}$ and $\mathbf{R}^{S,va}$ are the relative pronoun matrices capturing interactions with the head noun and verb-argument phrase respectively; and all other variables are defined as previously. Analogously, the composition of a noun modified by an object relative clause is modelled as

$$\mathbf{R}^{O,va}\mathbf{V}^S_v \mathbf{N}_s + \mathbf{R}^{O,n}\mathbf{N}_h \quad (\text{RPTensor, object}).$$

See Figure 4.3 for a graphical representation of FPLF composing a noun with a subject and an object relative clause.

Relative pronoun tensor Finally, the last approach models relative pronouns as tensors. Subject relative pronouns have CCG type $(NP\backslash NP)/(S\backslash NP)$, and object relative pronouns have type $(NP\backslash NP)/(S/NP)$, which should both result in fourth-order tensors living in $N \otimes N \otimes S \otimes N$. In order to make the learning of relative pronoun tensors tractable, we decided to make use of

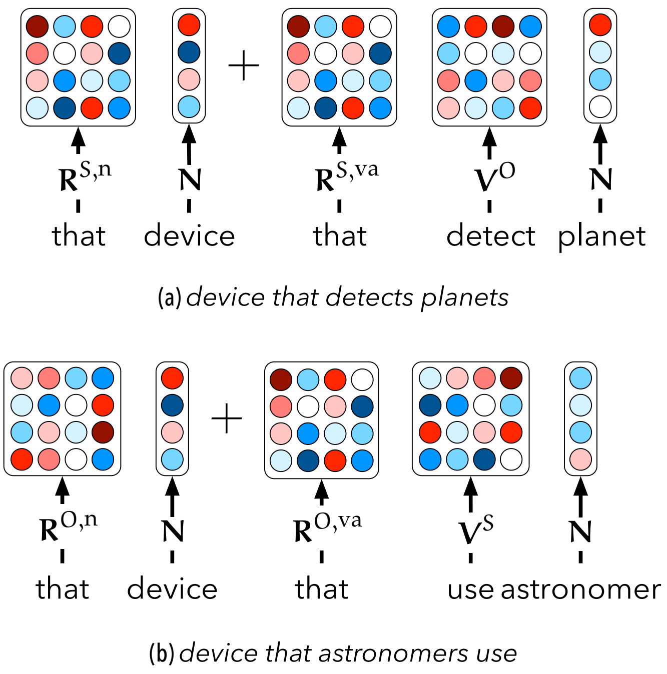


Figure 4.3: FPLF model composing subject and object relative phrases.

the PLF approach, which reduces the semantic space of transitive verbs from $S \otimes N \otimes N$ to $S \otimes N$. This, in turn, leads to a reduction of the semantic space of both subject and object relative pronouns from $N \otimes N \otimes S \otimes N$ to $N \otimes N \otimes S$, which was tractable given the computing power available to me. Using the relative pronoun tensor approach (henceforth RPTensor), nouns modified by subject and object relative phrases are composed respectively as

$$\mathbf{R}^{S} \cdot \mathbf{V}_{v}^{O}\mathbf{N}_{o} \cdot \mathbf{N}_{h}, \tag{4.2}$$

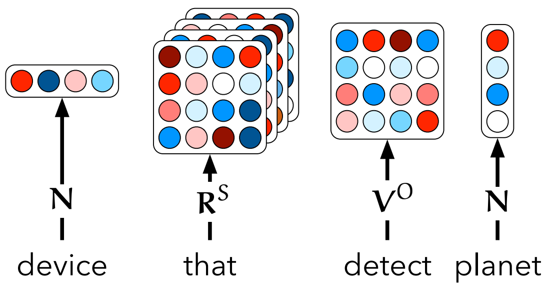
$$\mathbf{R}^{O} \cdot \mathbf{V}_{v}^{S}\mathbf{N}_{s} \cdot \mathbf{N}_{h}, \tag{4.3}$$

where $\mathbf{R}^{S}$ and $\mathbf{R}^{O}$ are the subject and object relative pronoun tensors; the dots represent tensor contractions with the rightmost index; and all other variables are defined as in previous examples. This is illustrated in Figure 4.4.

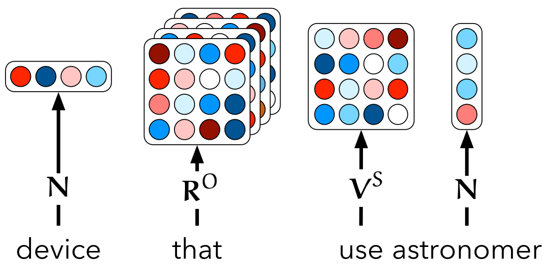
4.3 Experimental setup

In the previous section, I described a number of compositional models. I will now explain how their parameters were learned. All code was written in Python 3, using the 11.1 release of NumPy.

Word vectors I learned 100-dimensional word vectors for my experiments using skip-gram with negative sampling (Mikolov et al., 2013b), a method which was shown to provide the best results in early experiments on the development data, when compared to traditional count-



(a) *device that detects planets*



(b) *device that astronomers use*

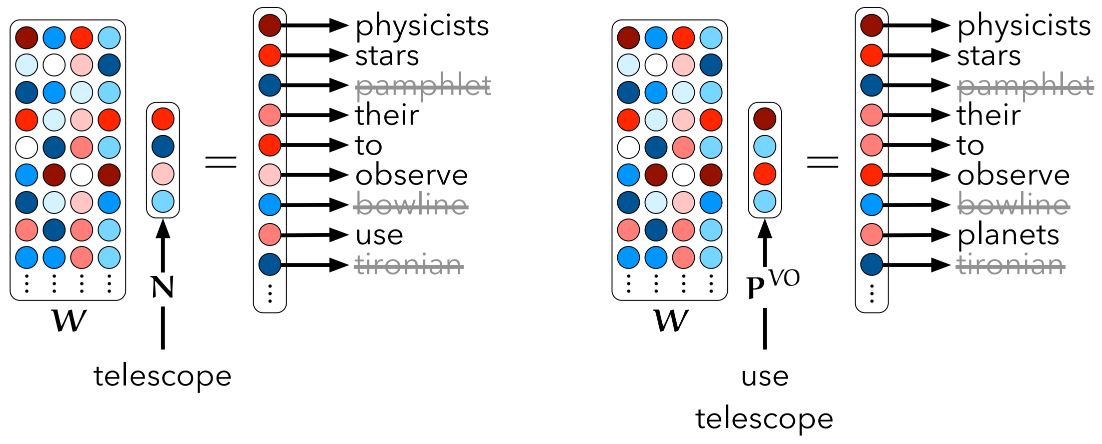
Figure 4.4: RPTensor model composing subject and object relative phrases.

based methods as well as the neural-based CBOW (Mikolov et al., 2013a). It is based on optimising the following cost function

$$\frac{1}{\left|\mathcal{D}^W\right|} \sum_{(t,c)\in\mathcal{D}^W} \left(\log \sigma(\mathbf{N}_t \cdot \mathbf{W}_c) + \sum^{k} \mathbb{E}_{c'\sim P} \left[\log \sigma(-\mathbf{N}_t \cdot \mathbf{W}_{c'}) \right] \right), \tag{4.4}$$

where $\mathcal{D}^W$ is the corpus, seen as pairs (t, c) of target words and context (neighbour) words; σ is the standard logistic function; $\mathbf{N}$ and $\mathbf{W}$ are, respectively, the word embedding and context word matrices, both learned parameters; and P is the noise distribution used to draw random words, set to the unigram distribution raised to the power of 3/4 (as recommended by Goldberg and Levy, 2014). The second term in the outer summation, called *negative sampling* by the authors, can be seen as a simplified variant of noise-contrastive estimation (Gutmann and Hyvärinen, 2012), an approximation to a full hierarchical softmax objective which is more efficient.[4] This process is illustrated in Figure 4.5a, for target word *telescope*. I ran skip-gram on a 2015 dump of Wikipedia, lemmatised using the Stanford CoreNLP tools (Manning et al., 2014). I used a window of ten words on either side of the target, with ten negative samples per word, and 100-dimensional target and context vectors. Lemmas with fewer than 100 occurrences in the corpus were ignored.

[4]Readers should note that, although the use of negative sampling was necessary when these experiments were originally run, given the current state of technology it should now be feasible to directly maximise the likelihood.



(a) First, we learn the noun vectors using the standard skip-gram algorithm.

(b) Then, we learn the corpus-based phrase embeddings, keeping $\boldsymbol{W}$ fixed.

Figure 4.5: The two-step skip-gram method for learning corpus-based phrase vectors with skip-gram. Shown above is the special case for verb-object phrases, with the example sentence *physicists use telescope to observe stars*.

Phrase vectors The categorial models I tested on the RELPRON dataset use matrices and tensors to model the meaning of function words. To learn these matrices and tensors, phrase vectors are needed as targets for linear regression. They are analogous to word vectors, but rather than encoding information on neighbours co-occurring with a word, they encode information on the neighbours co-occurring with the head word of a given phrase: for example, the vector for the verb-object phrase *use telescope* encodes information on the co-occurring neighbours of *use* when the verb appears in the corpus with object *telescope*.

To learn phrase vectors, I extracted from the same Wikipedia corpus subject-verb phrases, verb-object phrases, and noun phrases modified by subject and object relative clauses, as well as the words neighbouring the head word of each phrase. This was achieved by parsing the corpus with the Stanford CoreNLP tools using Universal Dependencies (Marneffe et al., 2014), and extracting relations of type `nsubj` and `obj` for subject-verb and verb-object phrases respectively; and `acl:relcl` with `PronType=Rel` for relative clauses, looking at the verb argument's role to determine whether they were subject or object relative clauses. Then, I learned phrase vectors using the same objective as skip-gram with negative sampling, described in Equation 4.4, but replacing the word embedding matrix $\mathbf{N}$ with a phrase embedding matrix, where each row is the embedding of a particular phrase: these were called $\mathbf{P}^{VO}$ for verb-object phrases, $\mathbf{P}^{SRC}$ for noun phrases modified by subject relative clauses, and so forth. For example, for the case of verb-object phrases, the cost function was defined as

$$\frac{1}{\left|\mathcal{D}^{VO}\right|}\sum_{(t,c)\in\mathcal{D}^{VO}}\left[\log\sigma\left(\mathbf{P}^{VO}_{t}\cdot\boldsymbol{W}_{c}\right)+\sum^{k}\mathbb{E}_{c'\sim P}\left(\log\sigma\left[-\mathbf{P}^{VO}_{t}\cdot\boldsymbol{W}_{c'}\right]\right)\right],$$

where $\mathcal{D}^{VO}$ is the set of verb-object phrases and their neighbours.

Crucially, in order to ensure that both word and phrase vectors encoded information in a similar way, and could thus be seen as living in the same vector space, I found it necessary to freeze the context word matrix $\boldsymbol{W}$. I therefore took the matrix $\boldsymbol{W}$ that was learned while training the word embeddings and re-used it, keeping it constant this time, also for training the phrase embeddings. Practically, this removed the influence of the random initialisation of $\boldsymbol{W}$, which would have been different for this second round of skip-gram. Phrases occurring fewer than twice in the corpus were discarded, and we used a wider window of 15 as suggested by Paperno et al. (2014). All other training parameters were the same as used for training word embeddings. I will call this method *two-step skip-gram*. The whole process is illustrated in Figure 4.5b for the case of verb-object phrase *use telescope*, whose vector is stored in one of the rows of the verb-object phrase embedding matrix $\mathbf{P}^{\mathrm{VO}}$.

Verb matrices In order to learn verb-subject and verb-object matrices described in the previous section, I followed the procedure in Paperno et al. (2014), with two differences: (1) instead of using count-based vectors, I used the aforementioned skip-gram word embeddings, and the phrase embeddings obtained with my two-step skip-gram method; (2) matrices were learned via ℓ_2-regularised regression (also known as ridge regression), additionally weighting each phrase-argument tuple by the logarithm of the number of occurrences of the phrase in the corpus. Given, for example, a verb-object matrix $\mathbf{V}_v^{\mathrm{O}}$ to be learned, and a set of tuples of vectors $(\mathbf{P}_i^{\mathrm{VO}}, \mathbf{N}_{o_i})_i$, where $\mathbf{P}_i^{\mathrm{VO}}$ is a corpus-extracted verb-object phrase vector and $\mathbf{N}_{o_i}$ is the word vector of the corresponding object, the weighted ridge regression problem is then formulated as

$$\mathbf{V}_v^{\mathrm{O}} = \arg\min_{\mathbf{M}} \sum_i w_i \left\| \mathbf{P}_i^{\mathrm{VO}} - \mathbf{M}\mathbf{N}_{o_i} \right\| + \gamma \left\| \mathbf{M} \right\|^2 ,$$

where γ is the regularisation weight, and w_i is the weight of the i^{th} phrase-argument tuple, defined as the logarithm of the number of occurrences of the phrase in the corpus. This is solved analytically as

$$\mathbf{V}_v^{\mathrm{O}} = \mathbf{P}^{\mathsf{T}}\boldsymbol{\Omega}^{-1}\mathbf{O}^{\mathsf{T}} \left(\mathbf{O}^{\mathsf{T}}\boldsymbol{\Omega}^{-1}\mathbf{O} + \boldsymbol{\Gamma}^{\mathsf{T}}\boldsymbol{\Gamma} \right)^{-1} ,$$

where $\mathbf{P}$ and $\mathbf{O}$ are matrices obtained by stacking as columns the phrase and object vectors, respectively; $\boldsymbol{\Omega} = \mathrm{diag}(w_1, w_2, \ldots)$ is the diagonal matrix of the weights; and $\boldsymbol{\Gamma} = \gamma\mathbf{I}$ is the identity matrix scaled by the regularisation weight. After tuning on the development set, the regularisation weight was set to $\gamma = 75$.

Weighting was introduced as an attempt to capture two ideas, which I illustrate using the verb *write* as an example: (1) when learning the verb-object matrix for *write* via regression, recreating the phrase vectors for *write music* and *write article*, which appear over 100 000 times in the corpus, should be more important than recreating the vector for *write opioid*, which only has three occurrences; (2) furthermore, the corpus-extracted phrase vector for *write opioid* is likely to be of low quality, due to the sparsity of the phrase, and this should be accounted for. Different weighting schemes were tested on the development set: standard unweighted ridge regression, and ridge regression using the simple counts as weights, as well as the logarithm, square root, and various powers of the counts. I found logarithms to yield the best result.

Relative pronoun tensor As with the verb matrices discussed above, relative pronoun tensors for the RPTensor model were also learned via weighted ridge regression. Training data for nouns modified by relative clauses consisted of tuples of head noun, verb, and argument, together with the corresponding corpus-extracted vectors for the whole phrase. Instead of using the multi-step regression method of Grefenstette et al. (2013), I found it was simpler in this case to reformulate the problem into a matrix regression problem, exploiting the verb-object and verb-subject matrices already learned for PLF. I did this by matricising the tensors $\mathbf{R}^S$ and $\mathbf{R}^O$ into matrices $\mathbf{M}^S_R$ and $\mathbf{M}^O_R$, flattening their last two dimensions into one. I then combined all of their vector arguments with a Kronecker product, and flattened the result. This turned the subject relative clause example of Equation 4.2 into the mathematically equivalent

$$\mathbf{M}^S_R \operatorname{vec}\left(\mathbf{V}^O_v \mathbf{N}_o \otimes \mathbf{N}_h\right),$$

where vec(·) represents the flattening of a matrix into a vector; and similarly for the case of object relative clauses. Regression was then performed on the matrix form of the equations analogously to the case of verb matrices, using the flattened Kronecker product as input, and the vector of the full phrase as target.

Relative pronoun matrices Finally, for the FPLF model we learned the two pairs of matrices $(\mathbf{R}^{S,n}, \mathbf{R}^{S,va})$ and $(\mathbf{R}^{O,n}, \mathbf{R}^{O,va})$ in an analogous way to the verb matrices, using weighted ridge regression with corpus-extracted phrase vectors as targets.

4.4 Results and discussion

We evaluated all models on the RELPRON development and test sets, using the MAP measure as defined in Equation 4.1. Results are shown in Table 4.1. The addition and SPLF methods achieved the highest performance on both development and test sets. The performance of SPLF was higher on the test data, although the difference is not significant. The MAP scores can be made more easily interpretable by realising that a model ranking, on average, a correct property in every second place, would achieve a score of 50%.

The next highest performing model is standard PLF. Recall that, while SPLF models the interaction with the head noun as an addition, PLF uses a verb-subject or verb-object matrix, depending on the type of relative clause. Effectively, this means that PLF treats nouns modified by relative clauses as subject-verb-object sentences. Its significantly lower performance when compared to

Table 4.1: MAP scores of various methods on the RELPRON development and test set. Results marked with * are significantly higher than the next lowest result ($p < 0.005$).

Model	**Development** (%)	**Test** (%)
SPLF	**49.6**	**49.7**
addition	**49.6***	**47.2***
PLF	44.4	43.3*
FPLF	40.5	38.7*
RPTensor	38.0	35.4
VArg	44.8	39.7
VHN	21.9	20.4

SPLF suggests that this approach might be suboptimal. The representations produced by PLF can also be seen as the addition of two terms: one representing the interaction of the verb with the argument (VArg), and one representing the interaction of the verb with the head noun (VHN). It is interesting to look at the performance of these two components separately, which are shown at the bottom of table Table 4.1: VHN is the worst-performing of the listed models, while VArg performs better. The head noun does however contribute to the composed representation, as evidenced by the fact that both SPLF (which includes the head noun via addition) and PLF (which includes it after multiplication with the verb matrix) achieve a significantly higher MAP on the development data than VArg.

The two models which learn subject and object relative pronoun functions by regression, FPLF and RPTensor, perform poorly; and both are outperformed by the VArg baseline. One possible explanation is that these methods might be lacking enough training data to learn their higher number of parameters when compared to the other, simpler methods. In order to investigate this, we looked at the performance of these models split by grammatical function, as there was a heavy imbalance in the availability of training data for relative pronouns. Indeed, the extracted relative clauses from the corpus were heavily skewed towards the subject case, with 771 k subject relative clauses found versus only 21 k object relative clauses. The results, shown in Table 4.2, support this hypothesis: for SPLF, addition, and VArg, the scores are relatively well-balanced, while FPLF and RPTensor show a higher discrepancy in the results.

4.5 Summary

In this chapter I introduced RELPRON, a relative clause evaluation dataset built by my collaborators (Rimell et al., 2016). I described the motivations behind its creation, and how it was built. Then, I discussed how I implemented a number of categorial models, and their evaluation on RELPRON. Several new ideas were presented: (1) a two-step skip-gram method for the training of word and

Table 4.2: MAP scores of various methods on the RELPRON development set, split by grammatical function. The training data for FPLF and RPTensor consisted of 771 k subject relative clauses and 21 k object relative clauses.

Model	**Subject RC** (%)	**Object RC** (%)
SPLF	58.5	54.1
addition	57.1	57.4
FPLF	57.0	42.8
RPTensor	52.0	40.0
VArg	53.0	48.9

arbitrary phrase vectors; (2) FPLF, an extension of the PLF compositional model of Paperno et al. (2014) to relative pronouns; (3) SPLF, an ablated variant of PLF, which was one of the two best-performing models on the RELPRON task; and (4) RPTensor, the first concrete implementation of a relative pronoun tensor.

When evaluating these models we saw two simple methods, SPLF and basic addition, taking the top spots. This suggests the hypothesis that there may not be enough data to effectively train the parameters of FPLF and RPTensor, the two more advanced models that learn relative pronoun functions via regression. A further experiment (Table 4.2) supports this idea, showing that FPLF and RPTensor perform better on subject relative clauses, which benefit from an amount of training data greater by an order of magnitude.

Addition – a trivially simple model – was one of the best performing approaches. It is likely, however, that substantial improvements over the presented results will require more sophisticated models. There are two reasons to believe this. First, any gains for methods as simple as addition will be limited to what can be achieved by improving the underlying word vectors, which would also benefit other distributed models. Second – and more important – the quality of additive vector representations has been shown to degrade with sentences longer than about ten words (Polajnar et al., 2014b). While this factor does not play a role with the simple relative clauses in RELPRON, it is likely to have a greater effect on longer linguistic constructions.

While addition has little obvious room for gains, the more complex categorial models have an obvious avenue for increasing performance by improving the quality and amount of training data. It is also possible that methods designed specifically for the RELPRON task, as opposed to the general-purpose phrase embeddings models tested here, will perform better.

5 Tensor-based skip-gram for adjective-noun composition

Baroni and Zamparelli, in their seminal 2010 paper, successfully demonstrated how, in distributed semantics, function words could be modelled as functions acting on the semantic representations of their arguments. Analogously to formal semantics, their approach treats adjectives in distributed semantics as endomorphic functions[1] on the vector space of noun meanings. These functions are encoded as matrices, learned via regression to approximate corpus-extracted vectors of a target phrase. This idea has been influential in NLP, giving rise to a number of models based on the same principles – see § 3.1 for a review, and the previous chapter for several more examples of models fitting this picture.

The original approach by Baroni and Zamparelli used count-based vectors, which were standard at the time of its publication, but have since been superseded by neural-based approaches (Baroni et al., 2014) such as skip-gram and CBOW (Mikolov et al., 2013b,a), and more recently FastText (Bojanowski et al., 2017) and ELMo (Peters et al., 2018). In this chapter I will describe tensor-based skip-gram (TBSG), a model first published in Maillard and Clark (2015), which can be seen as the neural-based counterpart of the count-based approach of Baroni and Zamparelli (2010).

Following the steps of Baroni and Zamparelli, I will also test TBSG on adjective-noun composition. Despite evaluation being limited here to the adjective-noun case, TBSG is a general method, that can be readily extended to other types of composition and higher order tensors.

5.1 A tensor-based skip-gram model

Adjectives have CCG type NP/NP[2] and thus, according to the tensor-based semantic framework previously described in § 3.3, should have semantic type $\mathsf{N} \otimes \mathsf{N}$. Therefore, like previous approaches (Baroni and Zamparelli, 2010; Paperno et al., 2014), the tensor-based skip-gram (henceforth TBSG) described in this chapter will treat adjectives as linear maps, encoded as matrices, over the vector space of noun meanings.

[1]That is, functions from a set onto itself.

[2]In the Clark and Curran (2007) parser, their actual type is N/N. In this thesis, we take the simplified approach of dropping the distinction between the N and NP types, as in Maillard et al. (2014).

The algorithm relies on standard skip-gram (Mikolov et al., 2013b), and works in two stages: (1) the first stage learns the noun vectors, as in a standard skip-gram model, but additionally storing the matrix of neighbour word parameters; and (2) the second stage learns the adjective matrices, re-using (as constants) the parameters learned in the first step.

In this section, I will start by describing the two stages of the TBSG algorithm. I will then conclude by defining a new similarity measure between adjective matrices since, as I will discuss, the standard cosine similarity function of vectors does not translate well to matrices.

5.1.1 Training of nouns

As the first step of TBSG, noun vectors are learned by using a skip-gram model with negative sampling[3] (Mikolov et al., 2013b). Given a training corpus $\mathcal{D}^W$ – seen as a set of tuples (t, c) indexing target nouns and neighbour words (contexts) – the algorithm learns a noun embedding matrix $\mathbf{N}$, whose rows constitute the noun embeddings; and a context word matrix $\boldsymbol{W}$. It does so by optimising the following objective, previously described in § 4.3 and Equation 4.4:

$$\frac{1}{\left|\mathcal{D}^W\right|} \sum_{(t,c)\in\mathcal{D}^W} \left(\log \sigma(\mathbf{N}_t \cdot \boldsymbol{W}_c) + \sum^{k} \mathbb{E}_{c'\sim P} \left[\log \sigma(-\mathbf{N}_t \cdot \boldsymbol{W}_{c'})\right] \right), \tag{5.1}$$

where, for each target noun index t, neighbours c are the indices corresponding to words in a fixed symmetrical window of size ten around it; and P is the noise distribution from which negative samples are drawn, set to the unigram distribution raised to the power of 3⁄4 (as recommended by Goldberg and Levy, 2014). The parameter k, which controls how many negative samples are drawn per positive example, was set to 5 in these experiments.

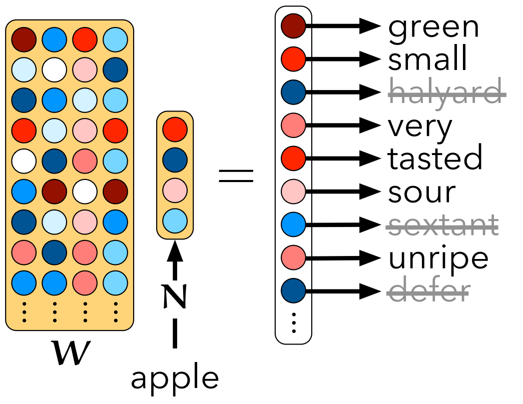
Intuitively, this procedure leads to noun embeddings having a high inner product with the context vectors of words appearing in their neighbourhood within the corpus; and a low inner product with context vectors of negatively sampled words. Figure 5.1a shows this intuition. It should be noted that at this stage of the algorithm, both $\mathbf{N}$ and $\boldsymbol{W}$ are being updated.

5.1.2 Training of adjectives

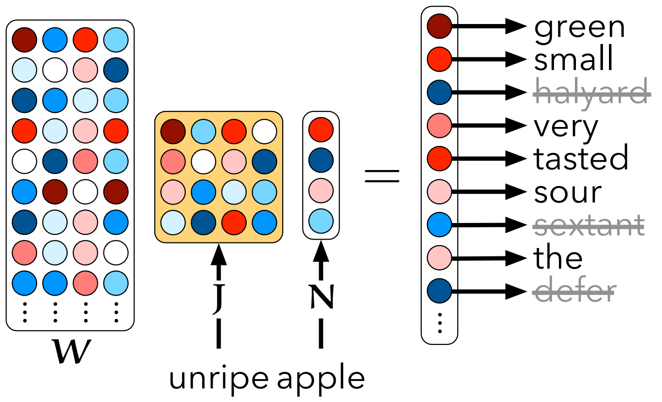
For the training of adjectives, the training data $\mathcal{D}^J$ is a set of tuples (j, n, c) of adjective, noun, and neighbour word indices. These are extracted from a corpus: the adjectives and nouns are those found in adjective-noun phrases in the corpus, while their neighbours are words found in a symmetric window around the corresponding phrase.

Adjective matrices are learned with a variation of the skip-gram algorithm, which trains an adjective embedding tensor $\mathbf{J}$, whose rows constitute the adjective matrices. It takes as input the

[3]While this was a necessary simplification at the time these experiments were run, given the fall in costs of computational power, it should now be possible to directly maximise the likelihood.



(a) Learning the word vector for *apple* with context *green small unripe apple tasted very sour*.



(b) Learning the adjective matrix for *unripe* with the context *the green small unripe apple tasted very sour*.

Figure 5.1: Skip-gram with negative sampling. Parameters are updated to increase the inner product with rows of $\boldsymbol{W}$ corresponding to positive examples, and decrease it with negative examples drawn from the noise distribution. On the left, standard skip-gram with negative sampling. On the right, TBSG applied to the adjective-noun case, where nouns vectors are kept fixed. Highlighted in orange are the parameters that are being updated: $\boldsymbol{W}$ and $\mathbf{N}$ on the left, $\mathbf{J}$ on the right.

noun embedding matrix $\mathbf{N}$ and the context word matrix $\boldsymbol{W}$ learned by Equation 5.1, and optimises the following cost function

$$\frac{1}{\left|\mathcal{D}^{J}\right|}\sum_{(j,n,c)\in\mathcal{D}^{J}}\left(\log\sigma(\mathbf{J}_j\mathbf{N}_n\cdot\boldsymbol{W}_c)+\sum^{k}\mathbb{E}_{c'\sim P}\left[\log\sigma(-\mathbf{J}_j\mathbf{N}_n\cdot\boldsymbol{W}_{c'})\right]\right), \tag{5.2}$$

where P and k are defined as in Equation 5.1. Matrices in $\mathbf{J}$ are initialised to the identity plus noise, while $\mathbf{N}$ and $\boldsymbol{W}$ are kept constant.

Intuitively, optimising Equation 5.2 means that the induced matrices will have the following property: when multiplying the matrix with a noun vector, the resulting adjective-noun vector will have a high inner product with words which are neighbours of the adjective-noun phrase in the corpus; and low inner product with the negatively sampled words. This is exemplified in Figure 5.1b.

5.1.3 Similarity measure

In distributed semantic models, the similarity of two vectors n and m is generally measured using the cosine similarity function (Turney and Pantel, 2010; Baroni et al., 2014), which is defined as

$$\mathrm{vecsim}(\mathbf{n}, \mathbf{m}) = \frac{\mathbf{n} \cdot \mathbf{m}}{\|\mathbf{n}\| \, \|\mathbf{m}\|}.$$

Based on tests using a development set, I found that using cosine to measure the similarity of adjective matrices leads to no correlation with gold-standard similarity judgements. Cosine similarity, while suitable for vectors as it is related to the angle between them, does not capture any information about the function of matrices as linear maps. While several ways of measuring matrix similarity could be devised using matrix norms, we postulate that a suitable measure of the similarity of two adjective matrices should be related to how similarly they transform the vectors of valid nouns.

Consider two adjective matrices $\mathbf{A}$ and $\mathbf{B}$. If $\mathbf{A}\mathbf{n}$ and $\mathbf{B}\mathbf{n}$ are similar vectors for every noun vector $\mathbf{n}$, then we deem the adjectives to be similar. Therefore, one possible measure involves calculating the cosine distance between the images of all nouns under the two adjectives, and taking an average or median of these distances. Rather than working on every noun in the vocabulary, which is expensive, we instead take the most frequent nouns, cluster them, and use the cluster centroids (obtained in our case using k-means). The resulting distance function is given by

$$\mathrm{matsim}(\mathbf{A}, \mathbf{B}) = \underset{\mathbf{n} \in \mathcal{N}}{\mathrm{median}} \ \mathrm{vecsim}(\mathbf{A}\mathbf{n}, \mathbf{B}\mathbf{n}),$$

where the median is taken over the set of cluster centroids $\mathcal{N}$. The median was chosen instead of the average as it is more resistant to outliers in the data.

5.2 Evaluation

I trained the TBSG model on a dump of the English Wikipedia, automatically parsed with the C&C parser (Clark and Curran, 2007). The corpus contains around 200 million noun examples, and 30 million adjective-noun examples. For every neighbour word in the corpus, 5 negative words were sampled from the unigram distribution, as described in the previous section. Subsampling was used to decrease the number of frequent words (Mikolov et al., 2013b). I trained 100-dimensional noun vectors and 100×100-dimensional adjective matrices.

5.2.1 Word Similarity

First I tested word similarity, as opposed to phrase similarity, on the MEN test collection (Bruni et al., 2014), which contains a set of part-of-speech tagged word pairs together with gold-standard human similarity judgements. I used the part-of-speech tags included in the dataset to select all noun-noun and adjective-adjective pairs, resulting in a set of 643 noun-noun pairs and 96 adjective-adjective pairs.

Table 5.1: Spearman rank correlation of the two models on the noun and adjective similarity tasks of the MEN test collection.

Model	Nouns	Adjectives
300D skip-gram	**0.78**	0.64
100D TBSG	0.77	**0.65**

For the noun-noun dataset, I tested the quality of the 100-dimensional noun vectors learned in the first stage of TBSG, which is equivalent to the standard skip-gram shipped with `word2vec`, but limited to learning just the vectors for nouns. I compared these to the 300-dimensional skip-gram vectors available from the `word2vec` page, which were trained on a very large, proprietary 100 billion token dataset.[4]

The adjective-adjective pairs from the MEN dataset were then used to test the 100×100D matrices obtained from the second step of the TBSG procedure, and these were again compared to the 300-dimensional skip-gram vectors. The resulting Spearman correlations between human judgements and the similarity of vectors are reported in Table 5.1. Note that for adjectives I used the similarity measure described in § 5.1.3.

The results show that, despite the lower dimensionality and smaller training corpus, the noun vectors used in TBSG were of a high quality, performing comparably to the skip-gram noun vectors on the noun-noun similarity data. The TBSG adjective matrices, used with the new similarity measure, also performed comparably to the skip-gram adjective vectors on the adjective-adjective similarity data.

5.2.2 Phrase Similarity

The TBSG model aims to learn matrices that act in a compositional manner. Therefore, a more interesting evaluation of its performance is to test how well the matrices combine with noun vectors. To do this, I used the Mitchell and Lapata (2010) adjective-noun similarity dataset, which contains pairs of adjective-noun phrases such as *last number – vast majority*, together with gold-standard human similarity judgements. For the evaluation, I calculated the Spearman correlation between human similarity judgements (individual ones, rather than their average) and the cosine similarity of the vectors produced by various compositional models.

The results in Table 5.2 show that TBSG has the best correlation with human judgements of the other models tested. It outperforms skip-gram vectors with both addition and element-wise multiplication as composition functions (the latter not shown in the table, as it is worse than addition). Also reported is the baseline performance of skip-gram and TBSG when using only nouns to compute similarity, i.e. ignoring the adjectives. It is interesting to note that TBSG

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

Table 5.2: Spearman rank correlation for the various models on the adjective-noun similarity task of Mitchell and Lapata. In the last row, the inter-annotator agreement, calculated by the dataset's authors via leave one-out resampling, which can be seen as an upper bound for the task.

Model	Correlation
100D TBSG	**0.50**
300D skip-gram (add)	0.48
300D skip-gram (N only)	0.43
100D TBSG (N only)	0.42
600D LR	0.37
humans	*0.52*

also outperforms the result of the matrix-vector linear regression method (LR) of Baroni and Zamparelli (2010) on this dataset, as implemented and reported by Vecchi et al. (2017). The Baroni and Zamparelli method is the most similar to TBSG and, as described in the introduction to this chapter, inspired its development.

5.2.3 Semantic Anomaly

As a further evaluation of the new model, I used TBSG to attempt to distinguish between semantically acceptable adjective-noun phrases and anomalous ones. This was done using the dataset provided by Vecchi et al. (2011), which consists of two sets of phrases: a set of unobserved acceptable phrases (e.g. *ethical statute*) and one of deviant phrases (e.g. *cultural acne*). Following Vecchi et al. (2011), I used two measures of semantic anomaly. The first, denoted *cosine*, is the cosine similarity between the adjective-noun vector and the noun vector. This index is based on the hypothesis that deviant adjective-noun vectors will form a wider angle with the noun vector. The second, denoted *density*, is the average cosine distance between the adjective-noun vector and its 10 nearest noun neighbours. This measure is based on the hypothesis that nonsensical adjective-nouns should not have many neighbours in the space of (meaningful) nouns.[5] These two measures are computed for the acceptable and deviant sets, and compared using a two-tailed Welch's t-test as in the dataset's paper.

Table 5.3 shows the results of this experiment, including the performances of TBSG, count-based vectors using addition and element-wise multiplication as implemented and reported by Vecchi et al. (2011), as well as the matrix-vector linear regression method (LR) of Baroni and Zamparelli (2010). With both the *cosine* and the *density* index, TBSG obtains the highest score.

[5]Vecchi et al. (2011) also used a third measure of semantic anomaly, based on the length of adjective-noun vectors. We omitted this measure as we deemed it unsuitable for models not based on counts and element-wise vector operations.

Table 5.3: Correlation on test data for semantic anomalies. Significance levels are marked *** for $p < 0.001$, ** for $p < 0.01$.

	Cosine		Density	
Model	t	sig.	t	sig.
100D TBSG	**5.16**	***	**5.72**	***
300D addition	0.31		2.63	**
300D multiplication	-0.56		2.68	**
300D LR	0.48		3.12	**

5.3 Summary

In this chapter, I described TBSG, a tensor-based extension of skip-gram with negative sampling, a popular neural word embedding model (Mikolov et al., 2013b). Like other models in Part II of this thesis, it can be seen as an implementation of the categorial framework of Coecke et al. (2011), which also fits into the CCG-flavoured tensor-based semantic framework described in § 3.3.

Further, I wrote and evaluated a concrete implementation of TBSG for the specific case of adjective-noun composition. The model was shown to produce high quality noun and adjective embeddings, when compared to standard skip-gram embeddings on single-word similarity tasks. It also compared favourably to other approaches on a phrase similarity task and a task involving the detection of semantically anonmalous adjective-noun combinations.

A few interesting lines of research suggest themselves for future work. First, while adjectives and nouns are learned separately in this study, an obvious extension is to learn these embeddings jointly. Second, while TBSG was tested here only on adjective-noun combinations, there are obvious ways in which it can be extended to other parts of speech in line with the framework described in § 3.3. The approach of reshaping the contraction of higher-order tensors into simple matrix-vector multiplications, as already demonstrated in § 4.3, would be useful here for simplifying the implementation of higher-order versions of TBSG; and for allowing them to exploit the highly optimised linear algebra libraries which are available on modern computers, such as BLAS. Finally, the generality of this approach would lend itself well to the implementation of multi-modal versions of TBSG, incorporating information from e.g. the visual domain. This could be achieved by including feature vectors for the appropriate images within the neighbours of a target word, as illustrated in Figure 5.2.

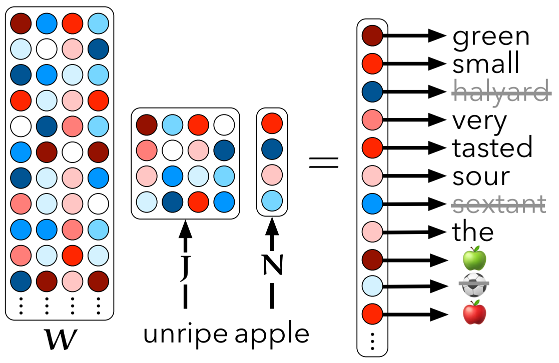


Figure 5.2: A multi-modal version of TBSG, currently processing the adjective phrase *unripe apple*, using for context the fragment *green small unripe apple tasted sour* and two images of an apple.

Part III

Latent tree learning models

6 Tree-structured recurrent neural networks

Part III of this thesis is devoted to latent tree learning (henceforth LTL) models, a family of neural networks that jointly learn to induce trees and compose sentences according to them. This chapter serves as an introduction to the subject. We will start with a presentation of the LSTM neural architecture, which forms the basis of the treeLSTM composition function used by the LTL models discussed Chapters 7 and 8 of this thesis. This will be followed by some essential theoretical background; a discussion of a number of tree-structured recurrent neural network models from the literature; and finally a review of related work within latent tree learning.

6.1 The Long Short-Term Memory architecture

Recurrent neural networks, in particular the Long Short-Term Memory architecture (henceforth LSTM) of Hochreiter and Schmidhuber (1997) and some of its variants (Graves and Schmidhuber, 2005; Bahdanau et al., 2015; Cho et al., 2014), have been widely applied to problems in NLP, such as textual entailment (Bowman et al., 2015; Williams et al., 2018b), question answering (Suster and Daelemans, 2018; Hermann et al., 2015), and machine translation (Bahdanau et al., 2015; Sutskever et al., 2014), amongst others. An LSTM is a recurrent network which, given a sentence represented by a sequence of word vectors $\boldsymbol{w}_1, \dots, \boldsymbol{w}_T \in \mathbb{R}^d$, runs for T time steps $t = 1 \dots T$ and computes

$$\begin{aligned}
\begin{bmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{u}_t \\ \mathbf{o}_t \end{bmatrix} &= \boldsymbol{W}\boldsymbol{w}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}, \\
\mathbf{c}_t &= \mathbf{c}_{t-1} \odot \sigma(\mathbf{f}_t) + \tanh(\mathbf{u}_t) \odot \sigma(\mathbf{i}_t), \\
\mathbf{h}_t &= \sigma(\mathbf{o}_t) \odot \tanh(\mathbf{c}_t), \\
\text{LSTM}_\theta(\boldsymbol{w}_t, \mathbf{c}_{t-1}, \mathbf{h}_{t-1}) &= (\mathbf{c}_t, \mathbf{h}_t),
\end{aligned} \tag{6.1}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the standard logistic function, and the square bracket notation is used to represent vector concatenation. The architecture is parametrised by a collection θ of learned matrices $\boldsymbol{W} \in \mathbb{R}^{4D \times d}$, $\mathbf{U} \in \mathbb{R}^{4D \times d}$, and a learned bias vector $\mathbf{b} \in \mathbb{R}^{4D}$. The vectors $\sigma(\mathbf{i}_t), \sigma(\mathbf{f}_t), \sigma(\mathbf{o}_t) \in \mathbb{R}^D$ are known as *input*, *forget*, and *output* gates respectively; and the vector $\tanh(\mathbf{u}_t) \in \mathbb{R}^D$ is the *candidate update*. The vectors $\mathbf{c}_t, \mathbf{h}_t \in \mathbb{R}^D$ are known as the *cell*

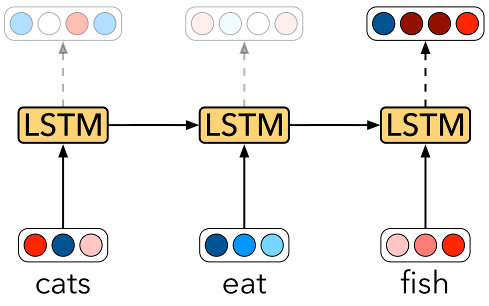


Figure 6.1: LSTM composing the sentence *cats eat fish* from left to right. The final output vector is often taken as the representation of the whole sentence. Intermediate outputs are shown as faded in the image, as they are commonly discarded in simpler models.

state and *output* of time step t, respectively. Commonly $\mathbf{h}_T$, the final output, is taken as the vector representation of the full sentence. The computational process involved in a single time step was already illustrated in Figure 1.8. The composition of a whole sentence is shown in Figure 6.1, using a single 'LSTM' box to represent the whole cell in simplified form.

As can be seen from the schematic illustration, the topology of an LSTM is linear: words are read sequentially, typically in left-to-right order. However, language is known to have an underlying hierarchical, tree-like structure (Chomsky, 1957). How to best capture this structure in a neural network, and whether doing so leads to improved performance on common linguistic tasks, remains an open question. In this and the following chapters, we will explore potential answers to this issue. The treeLSTM architecture (Tai et al., 2015; Zhu et al., 2015) – which generalises the LSTM to tree-structured topologies – provides a possible solution, and will be discussed in more depth in § 6.2.

Despite their superior performance on a number of tasks, treeLSTM networks have the drawback of needing to know the order in which to compose the sentences – i.e., they require an extra labelling of the input sentences, in the form of parse trees. This is often problematic to obtain, requiring either costly manual annotation of data by experts, or the use of automatic parsers which have been trained on the appropriate language and domain. Yogatama et al. (2017) proposed to remove this requirement, by including a shift-reduce parser in their sentence encoding model, to be optimised alongside the treeLSTM composition function based on a downstream task. This type of approach, where treeLSTMs learn to parse without ever being given an example of a correct parse tree, has been called a *latent tree learning model* by Williams et al. (2018a), and we shall adopt this term. We will discuss three such models in § 6.3: the recursive auto-encoder model of Socher et al. (2011), the aforementioned model by Yogatama et al., and a model by Choi et al. (2018) based on best-first parsing.

In the following chapters, Chapters 7 and 8, we will explore two new proposed latent tree learning models. They differ from the work of Yogatama et al. (2017) by being trainable via simple backpropagation, which removes the need to use reinforcement learning. We will show how this leads to better downstream performance, and non-trivial induced tree structures.

6.2 TreeLSTM

The treeLSTM is a generalisation of the popular LSTM architecture to tree topologies (Tai et al., 2015; Zhu et al., 2015), which has been shown to be more effective than a standard LSTM in semantic relatedness and sentiment analysis tasks.

Multiple variants of the treeLSTM architecture exist. We will focus here on the most popular version, commonly known as *binary treeLSTM*. Tai et al. (2015) also describe other variants for trees with different branching numbers, as well as a variant that supports trees with a variable number of children. Henceforth, we shall use 'treeLSTM' to refer to the binary variant, unless otherwise specified.

The binary treeLSTM is defined by the following recurrent relations,

$$\begin{aligned}
\begin{bmatrix} \mathbf{i} \\ \mathbf{f}_\ell \\ \mathbf{f}_r \\ \mathbf{u} \\ \mathbf{o} \end{bmatrix} &= \mathbf{W}\boldsymbol{w} + \mathbf{U}\mathbf{h}_\ell + \mathbf{V}\mathbf{h}_r + \mathbf{b}, \\
\mathbf{c} &= \mathbf{c}_\ell \odot \sigma(\mathbf{f}_\ell) + \mathbf{c}_r \odot \sigma(\mathbf{f}_r) + \tanh(\mathbf{u}) \odot \sigma(\mathbf{i}), \\
\mathbf{h} &= \sigma(\mathbf{o}) \odot \tanh(\mathbf{c}), \\
\text{treeLSTM}_\theta(\boldsymbol{w}, \mathbf{c}_\ell, \mathbf{h}_\ell, \mathbf{c}_r, \mathbf{h}_r) &= (\mathbf{c}, \mathbf{h}),
\end{aligned} \tag{6.2}$$

where all parameters are defined analogously to the LSTM of Equation 6.1. The linear chain architecture of the LSTM, with different words being input at different time steps, is replaced here with a binary tree-structured architecture: nodes can therefore have a left and right child, whose memory cells and outputs are denoted by subscripts ℓ and r, respectively. Much like in a parse tree, tree nodes can be either terminals (leaves), in which case the equation above has left and right children set to zero ($\mathbf{c}_{\ell/r} = \mathbf{h}_{\ell/r} = 0$) and the input $\boldsymbol{w}$ set to the appropriate word embedding; or they can be nonterminals (branches), in which case the input $\boldsymbol{w}$ is set to zero, and the left and right children are set to the outputs of the appropriate nodes.

This process is exemplified in Figure 6.2 with the sentence *colourless green ideas sleep furiously* (Chomsky, 1957), using one particular choice of composition order. It should be noted that we have said nothing about how the tree structure is chosen: commonly, either gold standard trees or the outputs of an external parser are used to drive the composition process (Tai et al., 2015; Bowman et al., 2016; inter alia).

6.3 Latent tree learning

By not committing to a specific composition order, but rather treating parse trees as another input, treeLSTMs remain simple and flexible. What could be seen as a strength is, however, also

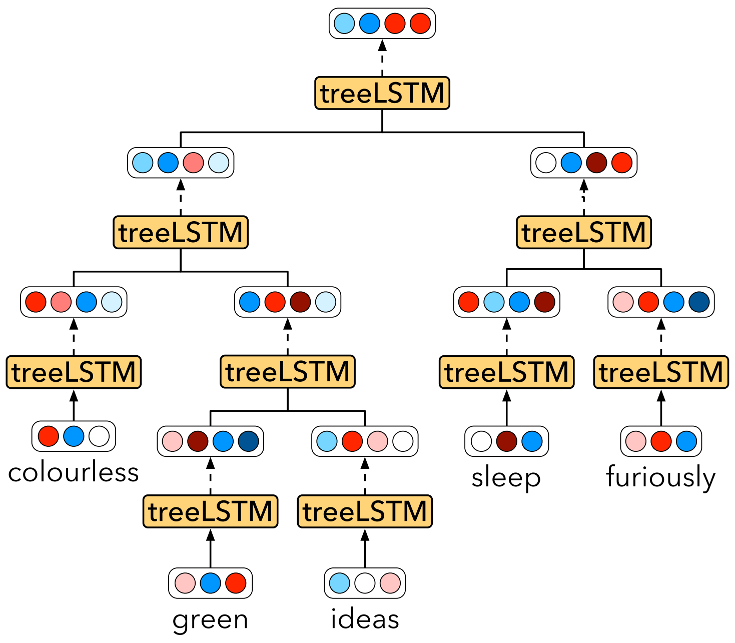


Figure 6.2: The treeLSTM architecture of Tai et al. (2015), showing both *leaf* and *branch* transformations, composing the sentence *colourless green ideas sleep furiously* according to the tree `( ( colourless ( green ideas ) ) ( sleep furiously ) )`.

a weakness: annotating sentences requires experts, is expensive, and is slow; while using an automatic parser may introduce errors, and is only a feasible route for high-resource languages and domains. There is one further question to consider: do the standard parse trees produced by automatic parsers really represent the best order in which to compose words, or might there be better structures, perhaps specific to the downstream task at hand?

Latent tree learning models (Williams et al., 2018a) address these points by learning not only the composition function (usually a treeLSTM) but also the order in which it should be applied – all solely driven by the downstream task. They can be said to learn 'grammars' (seen here as mere strategies for assigning tree structures to sentences) that are optimised for a given task, without ever being shown a single parse tree as input. In the following sections we will review the models that populate this recently developed area of research.

6.3.1 Semi-supervised recursive autoencoders

Back in 2011, Socher et al. proposed a sentence embedding model for sentiment analysis based on a treeRNN architecture, which they called *semi-supervised recursive autoencoder*. After representing the words as vector embeddings, their model recursively composes them according to a binary tree structure. Given two child nodes with corresponding vectors $\mathbf{h}_\ell$ and $\mathbf{h}_r$, the parent is represented as the vector

$$\mathbf{h}_p = \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_\ell \\ \mathbf{h}_r\end{bmatrix} + \mathbf{b}\right),$$

where the matrix $\mathbf{W}$ and the vector $\mathbf{b}$ are learned parameters. In order to assess how well the resulting parent vector represents the composition of its children, the model attempts to reconstruct them as

$$\begin{bmatrix} \mathbf{h}'_\ell \\ \mathbf{h}'_r \end{bmatrix} = \mathbf{W}'\mathbf{h}_p + \mathbf{b}',$$

where $\mathbf{W}'$ and $\mathbf{b}'$ are learned parameters. During training, on top of optimising for the sentiment analysis task, the model also attempts to minimise the reconstruction error, defined as

$$E(\mathbf{h}_\ell, \mathbf{h}_r) = \frac{1}{2} \left\| \begin{bmatrix} \mathbf{h}_\ell \\ \mathbf{h}_r \end{bmatrix} - \begin{bmatrix} \mathbf{h}'_\ell \\ \mathbf{h}'_r \end{bmatrix} \right\|^2 .$$

The novelty of Socher et al.'s approach consists in the way the composition order is chosen. Unlike other similar architectures which take external parse trees (Socher et al., 2010; Hermann and Blunsom, 2013; inter alia) here the model merges, at each time step, the pair of nodes which minimises the total reconstruction error of the tree. Letting $\mathcal{T}$ be the set of all possible trees for a given sentence, and $\mathcal{C}(\mathbf{t})$ be the set of pairs of sibling nodes for a tree $\mathbf{t}$, the model will choose the tree $\mathbf{t}^*$ given by

$$\mathbf{t}^* = \arg\min_{\mathbf{t} \in \mathcal{T}} \sum_{(\mathbf{h}_\ell, \mathbf{h}_r) \in \mathcal{C}(\mathbf{t})} E(\mathbf{h}_\ell, \mathbf{h}_r).$$

This process is sketched in Figure 6.3.

The semi-supervised recursive autoencoder model can be seen as a *proto-LTL* approach: it shares with other LTL models the fundamental idea of using the same vector representations ($\mathbf{h}_p$) to both encode the sentence and select a parse tree for it. However, it differs from other LTL models by driving the parsing component using an additional objective – a recursive autoencoder – rather than using the downstream task itself.

6.3.2 Stack-augmented parser-interpreter neural network

The *stack-augmented parser-interpreter neural network* (henceforth SPINN) is a proposal by Bowman et al. (2016) to solve the problem of treeLSTMs being only applicable to already-parsed sentences. The authors propose to include a shift-reduce parser directly into the model, so that the complete model is able to both parse and compose sentences.

Like a standard shift-reduce parser,[1] SPINN contains a buffer and a stack. Unlike a standard parser, these data structures are used to store vectors: the buffer contains the embeddings of words that are still to be processed; while the stack contains the embeddings of nodes that have been composed. An LSTM, known as the *tracking* LSTM, is given inputs from the buffer and the stack at each time step – intuitively, this is meant to keep a summary of what has been parsed so far. The outputs of the tracking LSTM are used to drive the transition classifier, and are also

[1]For examples of the application of this parsing technique to NLP, see Shieber (1983) and Nivre (2003).

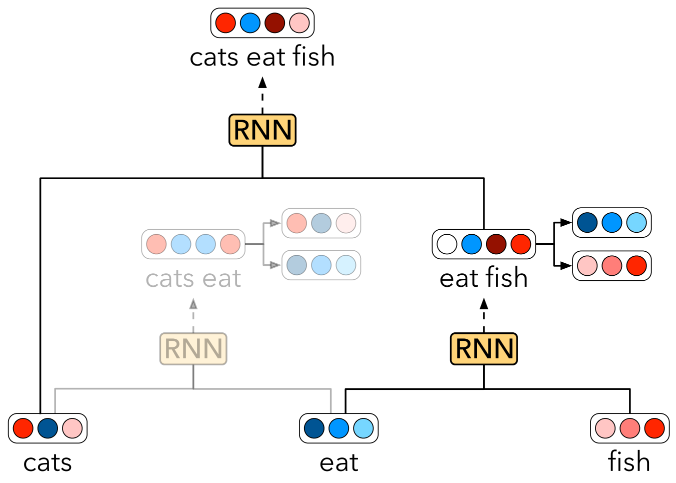


Figure 6.3: Semi-supervised recursive autoencoder RNN by Socher et al. (2011). At each step, the model attempts every pairwise composition, and picks whichever one gives the best performance on the auxiliary autoencoder loss. In the illustration, the model settles on the parse tree `( cats ( eat fish ) )`.

used as an additional input to the treeLSTM, with the aim of allowing the composition function to be influenced by context. The transition classifier is trained to match the transitions emitted by an external parser – the Stanford PCFG parser (Manning et al., 2014) – allowing the model to operate on unparsed data. The architecture of SPINN is sketched in Figure 6.4, showing the last three transitions of the sentence *cats eat fish*. We refer readers to Bowman et al. (2016) for more details on the model.

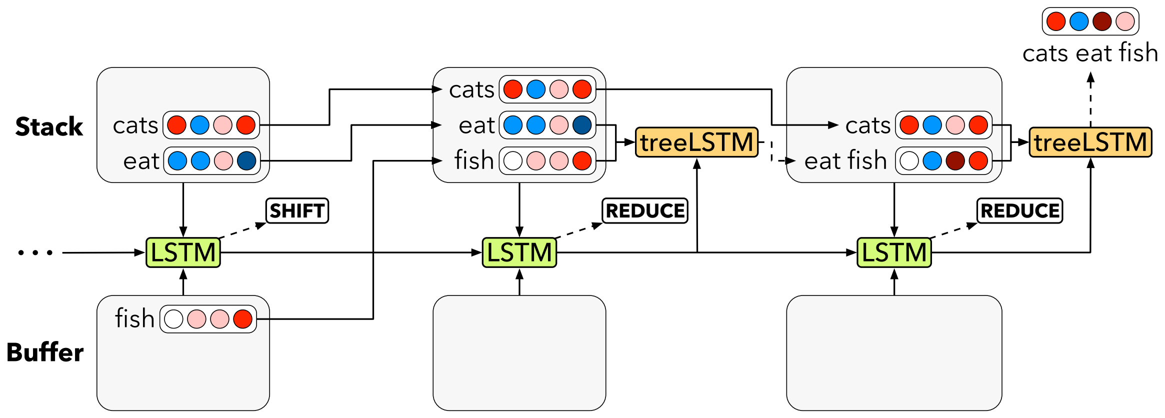


Figure 6.4: The SPINN model of Bowman et al. (2016), showing the final three transitions (SHIFT, REDUCE, REDUCE) for the sentence *cats eat fish*. The model here performs the composition according to the tree `( cats ( eat fish ) )`.

While SPINN does not require parsed data at inference time, it cannot be considered an LTL model: the parse trees it generates are chosen to emulate the choices of an automatic parser trained on gold-standard data, rather than being driven entirely by the downstream task.

Yogatama et al. (2017) were the first to propose a model which fully matches the LTL definition, as given in § 6.3 and Williams et al. (2018a). They achieved this by taking SPINN and training its transition classifier with REINFORCE (Williams, 1992), using performance on a downstream task as the reward. The model is thus no longer selecting trees that match those which might be assigned by trained linguists, but rather those which yield the best downstream performance. The authors evaluated their model on sentiment analysis, semantic relatedness, natural language inference, and sentence generation; and found that this approach outperforms sequential models such as LSTMs and biLSTMs, as well as models being explicitly provided with tree annotations.

6.3.3 Chart-based latent tree learning

The second LTL model, which will be discussed in full detail in Chapter 7, was developed by Maillard et al. (2017). It differs from the REINFORCE-trained variant of SPINN by Yogatama et al. (2017) in two main ways. First, rather than being based on a shift-reduce paradigm, it is based on a chart-based parsing method, inspired by the CKY algorithm (Cocke, 1969; Kasami, 1965; Younger, 1967). Second, it is fully differentiable, making it possible to train the whole model using off-the-shelf gradient descent.

The model was tested on a reverse dictionary task (Hill et al., 2016), showing improvements over several baselines, including a treeLSTM receiving supervision in the form of parse trees. It was also evaluated on a natural language inference task (Bowman et al., 2015), where it outperformed the approach of Yogatama et al. (2017) while also producing more interesting tree structures (Maillard et al., 2017; Maillard and Clark, 2018).

6.3.4 Easy-first latent tree learning

An alternative approach, proposed by Choi et al. (2018), is to perform latent tree learning using the easy-first parsing paradigm (Goldberg and Elhadad, 2010). Their model represents the sentence by composing, at every time step, two neighbouring nodes using a treeLSTM. At time step t, the model has $N - t$ nodes left (where N is the total number of words) and must choose which two adjacent nodes to merge. It attempts all possible compositions, leading to $N - t - 1$ candidate parent representations $\mathbf{h}_1, \ldots, \mathbf{h}_{N-t-1}$. It then calculates the validity score v_i of each candidate via a softmax function,

$$v_i = \frac{e^{\mathbf{q} \cdot \mathbf{h}_i}}{\sum_j e^{\mathbf{q} \cdot \mathbf{h}_j}}, \tag{6.3}$$

where $\mathbf{q}$ is a learned parameter known as the *composition query vector*, meant to measure the 'validity' of a representation. During training, the model samples the pair of nodes to be merged using the straight-through Gumbel-softmax estimator (Jang et al., 2017) – described in the next

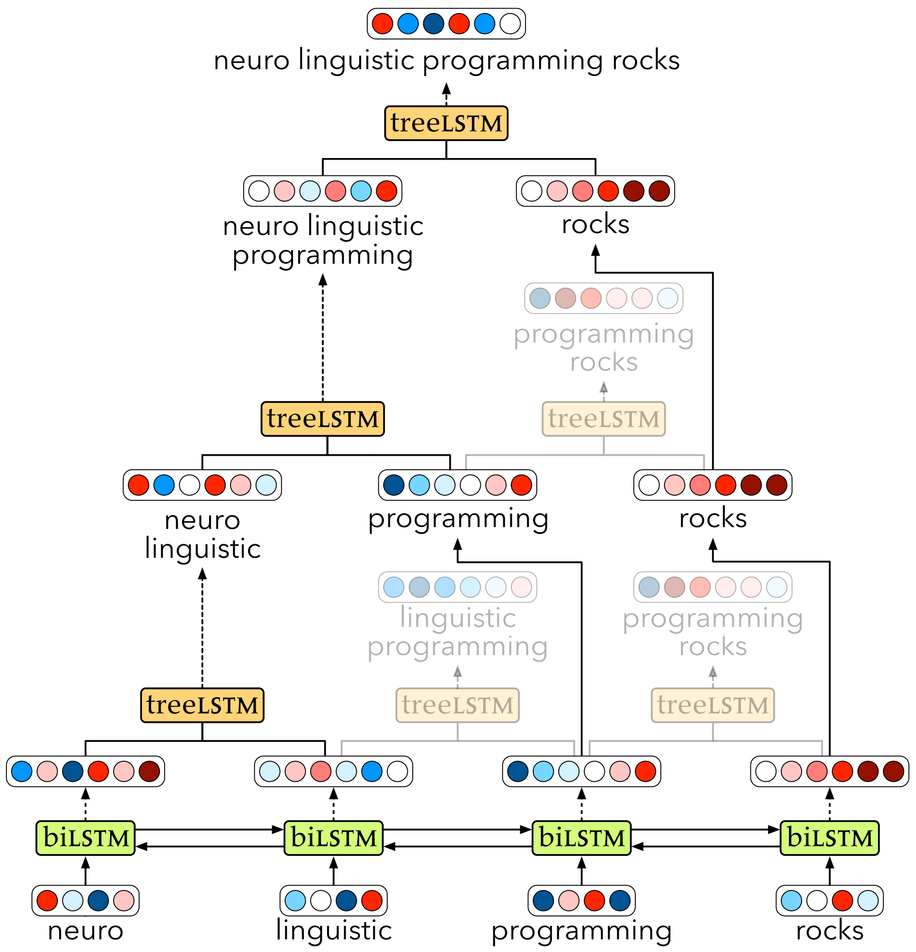


Figure 6.5: Latent tree learning model of Choi et al. (2018), based on a best-first parsing strategy. At each step, the parser attempts to compose every pair of neighbouring words, scores them, but only goes forward with the top ranking option. The procedure is repeated until the whole sentence is processed. The image illustrates an instance where the model picked the tree `( ( ( neuro linguistic ) programming ) rocks )`.

paragraphs – based on the validity scores v_i. During evaluation, the model simply selects the highest scoring candidate. This process is illustrated in Figure 6.5.

The Gumbel-softmax estimator is a continuous relaxation of the process of drawing samples from a categorial distribution. It is based on the Gumbel-max trick (Gumbel, 1954), and additionally employs the softmax function as a smooth, differentiable approximation to arg max. Given class probabilities π_i, it yields a k-dimensional sample vector $\mathbf{y}$ given by

$$y_i = \frac{e^{(\log \pi_i + g_i)/\tau}}{\sum_{j=1}^{k} e^{(\log \pi_j + g_j)/\tau}}, \tag{6.4}$$

where $g_1, \ldots, g_k$ are samples drawn from the Gumbel distribution, and τ is a hyperparameter known as the temperature. As $\tau \to 0$, samples drawn from Equation 6.4 become one-hot. The *straight-through Gumbel-softmax* estimator – the variant chosen by Choi et al. – is a discretisation of the Gumbel-softmax distribution: in the forward pass, it behaves exactly as Equation 6.4, while in the backward pass it replaces that expression with a one-hot discrete approximation,

$$y'_i = \begin{cases} 1 & i = \arg\max_j y_j, \\ 0 & \text{otherwise.} \end{cases}$$

Choi et al. tested their model on a natural language inference task, where it outperformed the CKY-based model and supervised-tree baselines; and a sentiment analysis task. Other than performing well on downstream tasks, this approach has the added advantage of being fast to train, as it requires only $\mathcal{O}(n^2)$ RNN evaluations for a sentence of n words, compared to the cubic complexity of the CKY-based approach.

6.3.5 Shift-reduce latent tree learning

Maillard and Clark (2018) have more recently proposed a new model which, like the SPINN variant described in § 6.3.2, is also based on shift-reduce parsing. The novelty of the approach is in the way in which the shift-reduce parsing component – which is based on intrinsically discrete actions – is made trainable via simple backpropagation and gradient descent. This is achieved by using a combination of beam search and a soft gating function to select between beam elements.

This model is much faster to train than the CKY-based approach, as it has a linear runtime of $\mathcal{O}(nb)$ in both the length of the sentence n and the size of the beam b. A full description of the model and of several experiments conducted with it will be given in Chapter 8.

6.4 Summary

Latent tree learning (LTL) is a recently developed family of sentence embedding models, demonstrating that it is possible to induce task-specific 'grammars' from a downstream task.

We discussed a number of LTL models that induce trees which, when used to drive treeRNN-based composition functions, outperform traditional trees in the style of the Penn Treebank (Marcus et al., 1994) on a particular set of tasks.

Central to these models is a parsing component, whose role is to induce the tree structures used to drive the composition process. I described four LTL models, based on a variety of parsing paradigms: two make use of shift-reduce parsing (Yogatama et al., 2017; Maillard and Clark, 2018), one uses chart parsing (Maillard et al., 2017), and one uses best-first parsing (Choi et al., 2018). In the next two chapters, I will describe in detail my two LTL models: in Chapter 7 I will present the CKY-inspired model already published in Maillard et al. (2017); while in Chapter 8 I will discuss the shift-reduce model recently published in Maillard and Clark (2018). I will further perform an analysis of the induced trees, using evaluation techniques first proposed by Williams et al. (2018a), in order to investigate whether they resemble traditional constituency parse trees; whether they contain any sort of recognisable structure; and to assess if different versions of the same model, differing only in their random initialisation, end up inducing similar-looking trees.

7 Chart parsing

Yogatama et al. (2017) were amongst the first to show that a treeLSTM architecture, trained to also induce optimal trees, is able to outperform a similar model which uses input from a parser to drive the compositional process. Their model, described in § 6.3.2, is based on a version of the SPINN architecture of Bowman et al. (2016) trained to jointly select trees and compose words. As this process makes the model non-differentiable, it needs to be trained with reinforcement learning, which can lead to high variance and slow convergence rates. In subsequent experiments, this approach was shown to have low performance compared to certain baselines, and the induced trees were shown to be mostly trivial (Williams et al., 2018a).

In this chapter, I describe an approach (already published in Maillard et al., 2017) to include a fully differentiable chart parser in a treeLSTM-based compositional model, inspired by the CKY chart parser (Cocke, 1969; Kasami, 1965; Younger, 1967). Due to the parser being made differentiable, the entire network can be trained end-to-end for a downstream task via backpropagation and gradient descent, which is easily available out-of-the-box in all the common deep learning frameworks. I will show how this model outperforms the approach of Yogatama et al., as well as supervised treeLSTM baselines, on a natural language inference task. Finally, I will also describe how this model and several baselines were tested on a reverse dictionary task, showing again how the proposed model compares favourably to the tested alternatives.

7.1 Model

While the treeLSTM composition function is very powerful, it requires as input not only the sentence, but also a parse tree structure defined over it. The extension proposed here optimises this step away, by including a basic CKY-style (Cocke, 1969; Kasami, 1965; Younger, 1967) chart parser in the model. The parser has the property of being fully differentiable, and can therefore be trained jointly with the treeLSTM for some downstream task.

The CKY parser relies on a *chart* data structure, which provides a convenient way of representing the possible binary parse trees of a sentence, according to some grammar. Here the chart is used as an efficient means to store all possible unlabelled binary-branching trees, effectively using a grammar with only a single non-terminal. This grammar can be described by two production rules, $X \rightarrow X\,X$ and $X \rightarrow \alpha$, where X is the non-terminal and α is any word in the vocabulary.

The chart is sketched in simplified form in Table 7.1 for the example input *neuro linguistic programming rocks*. It is drawn as a diagonal matrix, where the bottom row contains the individual

Table 7.1: Simplified chart for the sentence *neuro linguistic programming rocks*. In this illustration, parsing starts from the bottom where the cells correspond to individual words. Larger and larger constituents are progressively built up in the upper rows.

			neuro linguistic programming rocks
		neuro linguistic programming	linguistic programming rocks
	neuro linguistic	linguistic programming	programming rocks
neuro	linguistic	programming	rocks

words of the input sentence. The n^{th} row from the bottom contains all cells with branch nodes spanning n words (here, each cell is represented simply by the span – see Figure 7.1 below for a forest representation of the nodes in all possible trees). By combining nodes in this chart in various ways it is possible to efficiently represent every binary parse tree of the input sentence. For a more exhaustive treatment of CKY parsing and the core concepts behind it, see Jurafsky and Martin (2009; §13.4.1).

The CKY-based unsupervised treeLSTM uses an analogous chart to guide the order of composition. Instead of storing sets of non-terminals, however, as in a standard chart parser, here each cell is made up of a pair of vectors $(\mathbf{h}, \mathbf{c})$ representing the state of the treeLSTM recurrent neural network at that particular node in the tree. The process starts at the bottom row, where each cell is filled in by calculating the treeLSTM output as defined in Equation 6.2, with $\mathbf{w}$ set to the embedding of the corresponding word and $\mathbf{h}_{\ell/r}$ and $\mathbf{c}_{\ell/r}$ set to zero. These are the leaves of the parse tree. Then, the second row is computed by repeatedly calling the treeLSTM with the appropriate children, and the word embedding input set to zero. This row contains the nodes that are directly combining two leaves. They might not all be needed for the final parse tree: some leaves might connect directly to higher-level nodes, which have not yet been considered. However, they are all computed, as we cannot yet know whether there are better ways of connecting them to the tree. This decision is made at a later stage.

Starting from the third row, ambiguity arises since constituents can be built up in more than one way: for example, the constituent *neuro linguistic programming* in Table 7.1 can be made up either by combining the leaf *neuro* and the second-row node *linguistic programming*, or by combining the second-row node *neuro linguistic* and the leaf *programming*. In these cases, all possible compositions are performed, leading to a set of candidate constituents $(\mathbf{c}_1, \mathbf{h}_2), \dots, (\mathbf{c}_n, \mathbf{h}_n)$. Each is assigned an *energy*, given by

$$\varepsilon_i = \cos(\mathbf{q}, \mathbf{h}_i), \tag{7.1}$$

where $\cos(\cdot, \cdot)$ indicates the cosine similarity function and $\mathbf{q}$ is a (learned) vector of weights, playing an analogous role to the composition query vector in Equation 6.3 of § 6.3.4.[1] All energies

[1] I initially experimented with a more sophisticated approach, analogous to the tracking LSTM of Bowman et al. (2016). The much simpler approach using the $\mathbf{q}$ vector, however, proved to work just as well for these experiments.

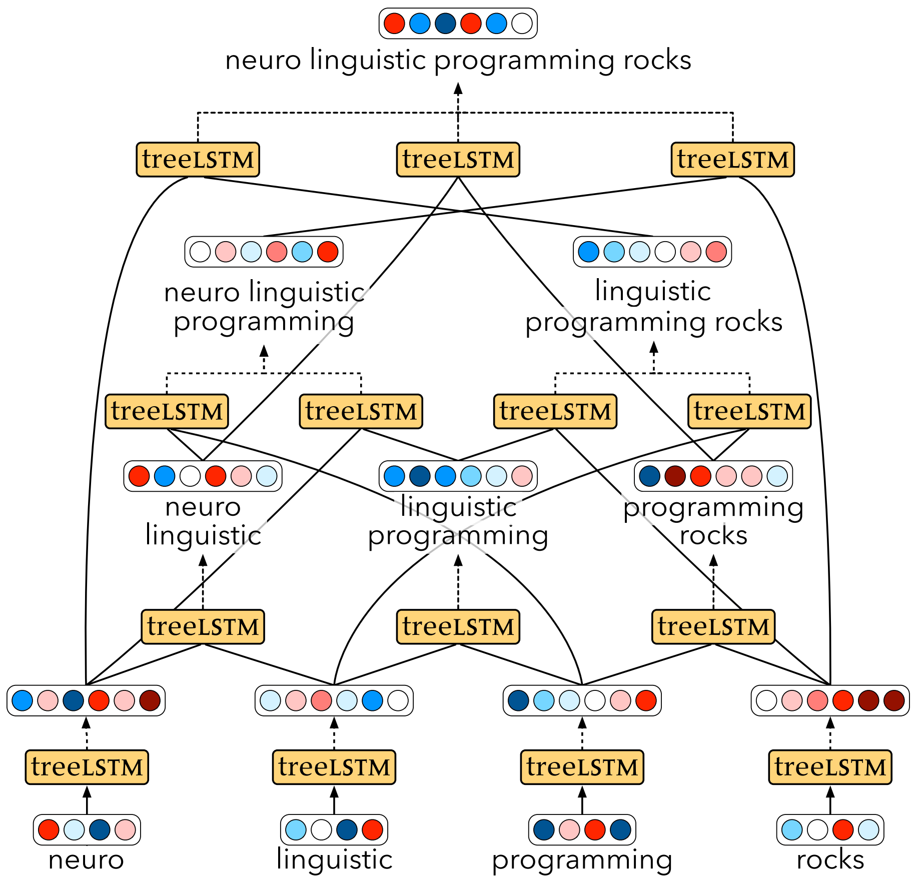


Figure 7.1: A latent tree learning sentence encoder based on the CKY parser.

are then passed through a softmax function to normalise them, and the cell representation is finally calculated as a weighted sum of all candidates using the softmax output:

$$s_i = \frac{e^{\varepsilon_i/\tau}}{\sum_j e^{\varepsilon_j/\tau}}, \tag{7.2}$$

$$\mathbf{c} = \sum_{i=1}^{n} s_i \mathbf{c}_i, \qquad \mathbf{h} = \sum_{i=1}^{n} s_i \mathbf{h}_i.$$

Analogously to Equation 6.4, the softmax here uses a temperature hyperparameter τ which, for small values, has the effect of making the distribution sparse by making the highest score tend to one. In all experiments the temperature is initialised as $\tau = 1$, and is smoothly decreased as $\tau = 1/2^f$, where $f \in \mathbb{Q}$ is the fraction of training epochs that have been completed. In the limit as $\tau \to 0^+$, this mechanism will only select the highest scoring option, and is equivalent to the argmax operation. The same procedure is repeated for all higher rows, and the final output is given by the $\mathbf{h}$-state of the top cell of the chart.

The whole process is sketched in Figure 7.1 for an example sentence. Note how, for instance, the final sentence representation can be obtained in three different ways, each represented by a treeLSTM cell with an outgoing dashed line. All are computed, and the final representation is a weighted sum of the three, represented by the merging of the three dashed lines. When the temperature τ in Equation 7.2 reaches very low values, this effectively reduces to the single 'best' tree, as selected by gradient descent.

7.1.1 Baselines

Bag-of-words The simplest baseline used in these experiments is a bag-of-words model. Given word embeddings $\boldsymbol{w}_1, \dots, \boldsymbol{w}_n$, it represents the corresponding sentence as

$$\mathbf{h} = \sum_{i=1}^{n} \tanh\left(\boldsymbol{W}\boldsymbol{w}_i + \mathbf{b}\right),$$

where $\boldsymbol{W}$ is a learned input projection square matrix, and $\mathbf{b}$ is a learned bias vector. Due to its reliance on addition, which is commutative, any information about the original word order is lost.

LSTM The next baseline is the popular Long Short-Term Memory network (Hochreiter and Schmidhuber, 1997), previously described and illustrated in § 1.3 and in the introduction of Chapter 6. Following the recommendation of Jozefowicz et al. (2015), I deviate slightly from the vanilla LSTM architecture described in Equation 6.1 by initialising the bias of the forget gate to one, which was found to improve performance.

TreeLSTM One of the differences between the proposed CKY-based model and the more common LSTM encoder is the composition function, which for the former is a treeLSTM. In order to determine whether any improvements achieved by the proposed model are only due to the different composition function, I use as baseline a *left-branching* treeLSTM which processes words left-to-right. This is identical to the LSTM baseline, other than the slightly more complex mathematical operation used to compose words. Due to English branching generally to the right, I also test a right-branching treeLSTM. Finally, the most complex baseline is termed *supervised* treeLSTM, and it composes words according to trees produced by the Stanford PCFG parser (Manning et al., 2014).

7.2 Experimental setup

All experiments in this chapter were implemented in Python 3.5.2 with the DyNet neural network library (Neubig et al., 2017) at commit `25be489`. The code for all following experiments is available at my personal website[2]. Performance on the development data was used to determine when to stop training.

Each model was trained three times, and the test set performance is reported for the model which performed best on the development set. The natural language inference model took

[2]`http://www.maillard.it/`

three days to converge on a 2.2 GHz Intel Xeon E5-2660 CPU, and the reverse dictionary model took five days on an Nvidia GeForce GTX Titan Black GPU.

7.2.1 Natural language inference

The first evaluation makes use of the Stanford Natural Language Inference dataset (Bowman et al., 2015), consisting of 570 k manually annotated pairs of sentences. Given two sentences, the aim is to predict whether the first *entails*, *contradicts*, or is *neutral* with respect to the second. For example, given *children smiling and waving at camera* and *there are children present*, the model would be expected to predict *entailment*.

The model uses 100-dimensional hidden states. It is given as input word embeddings which are initialised using 100-dimensional GloVe vectors (Pennington et al., 2014), with out-of-vocabulary words set to the average of all other vectors. This results in a 100×37 369 word embedding matrix, fine-tuned during training. The supervised treeLSTM model is also given as an additional input the parse trees which are included in the dataset. For training I chose the Adam optimisation algorithm (Kingma and Ba, 2014), with a batch size of 16.

Given a pair of sentences, one of the models is used to produce the embeddings $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{R}^{100}$. Following Yogatama et al. (2017) and Bowman et al. (2016), I then compute

$$\begin{aligned} \mathbf{u} &= (\mathbf{s}_1 - \mathbf{s}_2)^2, \\ \mathbf{v} &= \mathbf{s}_1 \odot \mathbf{s}_2, \\ \mathbf{q} &= \mathrm{ReLU}\left(\mathbf{A} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} + \mathbf{a} \right), \end{aligned} \tag{7.3}$$

where $\mathbf{A} \in \mathbb{R}^{200\times400}$ and $\mathbf{a} \in \mathbb{R}^{200}$ are trained parameters. Finally, the correct label is predicted by $p(\hat{y} = c \mid \mathbf{q}; \mathbf{B}, \mathbf{b}) \propto \exp(\mathbf{B}_c\mathbf{q} + \mathbf{b}_c)$, where $\mathbf{B} \in \mathbb{R}^{3\times200}$ and $\mathbf{b} \in \mathbb{R}^3$ are trained parameters.

Attention

Attention is a mechanism which allows a model to soft-search for relevant parts of a sentence. It has been shown to be effective in a variety of linguistic tasks, such as machine translation (Bahdanau et al., 2015; Vaswani et al., 2017), summarisation (Rush et al., 2015), and textual entailment (Shen et al., 2017).

In the spirit of Bahdanau et al. (2015), I modify the LSTM baseline such that it returns not just the output of the last time step, but rather the outputs for all steps. Thus, it no longer yields a single pair of vectors $\mathbf{s}_1, \mathbf{s}_2$ as in Equation 7.3, but rather two lists of vectors $\mathbf{s}_{1,1}, \dots, \mathbf{s}_{1,n_1}$ and

$\mathbf{s}_{2,1}, \dots, \mathbf{s}_{2,n_2}$. Then, in order to determine how the model should attend over all the outputs $\mathbf{s}_{1,i}$ of the first sentence, I compute the attention weights using a softmax operation:

$$w_{1,i} = \frac{\exp\left(f(\mathbf{s}_{1,i}, \mathbf{s}_{2,n_2})\right)}{\sum_{j=1}^{n_1} \exp\left(f(\mathbf{s}_{1,j}, \mathbf{s}_{2,n_2})\right)}, \qquad \text{with } f(\mathbf{x}, \mathbf{y}) \equiv \mathbf{a} \cdot \tanh\left(\mathbf{A}_i \mathbf{x} + \mathbf{A}_s \mathbf{y}\right),$$

where f is the *attention mechanism*, with vector parameter $\mathbf{a}$ and matrix parameters $\mathbf{A}_i, \mathbf{A}_s$. Finally I replace the overall sentence representation $\mathbf{s}_1$ in Equation 7.3 with the weighted sum

$$\mathbf{s}_1' = \sum_{i=1}^{n_1} w_{1,i} \mathbf{s}_{1,i},$$

where the weights $w_{1,i}$ are as defined above. This can be interpreted as attending over the first sentence, informed by the context of the second via the vector $\mathbf{s}_{2,n_2}$. Similarly, $\mathbf{s}_2$ is replaced by an analogously defined $\mathbf{s}_2'$, with separate attention parameters.

Further, I also extend the mechanism of Bahdanau et al. (2015) to the CKY-based treeLSTM. In this case, instead of attending over the list of outputs of an LSTM at different time steps, attention is over the whole chart structure described in Table 7.1. Thus, the model is no longer attending over all *words* in the source sentences, but rather over all their possible *subspans*.

7.2.2 Reverse dictionary

For the second evaluation, I use the reverse dictionary dataset of Hill et al. (2016), which consists of 852 k word-definition pairs. The aim is to retrieve the name of a concept from a list of words, given its definition. For example, when provided with the input *control consisting of a mechanical device for controlling fluid flow*, a model would be expected to rank the word *valve* above other confounders in a list. I use the three test sets provided by the authors: two sets involving word definitions, either seen during training or held out (respectively called the *seen* and *unseen* test sets); and one set involving concept descriptions instead of formal definitions (called the *concepts* test set). Performance is measured via three statistics: the *median rank* of the correct answer over a list of over 66 k words; and the proportion of cases in which the correct answer appears in the top 10 and 100 ranked words (*top 10 accuracy* and *top 100 accuracy*).

For this task, there are two sets of embeddings, which I shall call *output* and *input* embeddings. The input embeddings are the vectors used to represent the words making up the definitions, and they are fed to the model. The output embeddings are used to represent the words being defined (known as *headwords* in a dictionary). They live in the same space as the outputs of the model. While the two embedding spaces do not have to be separate, I choose to follow Hill et al. and use two distinct set of vectors.

As output embeddings, I use the 500-dimensional CBOW vectors (Mikolov et al., 2013a) provided by the authors. As input embeddings I use the same vectors, reduced to 256 dimensions with

PCA. As for the inference task, the model uses 100-dimensional hidden states in the treeLSTM. Given a training definition as a sequence of input embeddings $w_1, \ldots, w_n \in \mathbb{R}^{256}$, the model produces an embedding $\mathbf{s} \in \mathbb{R}^{256}$ which is then mapped to the output space via a trained projection matrix $\mathbf{W} \in \mathbb{R}^{500\times256}$. The training objective to be maximised is then the cosine similarity $\cos(\mathbf{W}\mathbf{s}, \mathbf{d})$ between the definition embedding and the output embedding $\mathbf{d}$ of the word being defined. For the supervised treeLSTM model, I additionally parsed the definitions with the Stanford PCFG parser to obtain syntax trees.

The model was trained using simple stochastic gradient descent, as that proved to work better on the development data compared to more complex optimisation algorithms, such as adaptive gradient methods. The first 128 batches were held out from the training set to be used as development data. The softmax temperature in Equation 7.2 was allowed to decrease as described in § 7.1 until it reached a value of $\tau = 0.005$, and then kept constant. This was found to have the best performance on the development set.

7.3 Results and discussion

Table 7.2 lists the SNLI test set accuracy for the CKY-based model, the baselines, as well as other sentence embedding models in the literature.[3] It also shows the number of model parameters, to give an idea of the complexity of each model. These figures are based on the original papers, when available, and the data from the SNLI website.[4] Table 7.3 shows results on the same task for the attention-augmented LSTM and CKY-based treeLSTM models. Finally, Table 7.4 shows the reverse dictionary task results for the proposed model and baselines, as well as the numbers for the cosine-based *w2v* models of Hill et al. (2016), taken directly from their paper.[5]

These results show a strong performance of the CKY-based treeLSTM against the baselines I implemented, as well as other similar methods in the literature with a comparable number of parameters. For the natural language inference task, the proposed model outperforms all baselines including the supervised treeLSTM, as well as some of the other sentence embedding models in the literature with a higher number of parameters. The use of attention, extended for the CKY-based model to be over all possible subspans, further improves performance.

In the reverse dictionary task, the poor performance of the treeLSTM can be explained by the unusual tokenisation used in the dataset of Hill et al. (2016): punctuation is simply stripped, turning e.g. *(archaic) a section of a poem* into *archaic a section of a poem*, or stripping away the

[3]Models specifically designed to solve this task, rather than being based on a more general-purpose sentence embedding architecture, are able to obtain higher performance. The current record on SNLI is held by Kim et al. (2018), with 90.1 test set accuracy.

[4]`https://nlp.stanford.edu/projects/snli/`

[5]My reimplementation of the *w2v cosine* models of Hill et al. (2016), using vectors provided by the authors, achieved lower performance than theirs. While I was unable to reproduce their results, I include their numbers for completeness. The baselines I implemented are architecturally different from theirs, but I found my variants to perform better on development data.

Table 7.2: Test set accuracy (higher is better) on the SNLI dataset, and number of parameters. Also reported is the number of intrinsic model parameters (excluding the number of word embedding parameters). Other models based on sentence embeddings are also reported.

Model	**Test accuracy**	**# Parameters**
100D bag-of-words	77.6 %	91 k
100D LSTM	82.2 %	161 k
100D left-branching treeLSTM	82.1 %	231 k
100D right-branching treeLSTM	82.5 %	231 k
100D supervised treeLSTM	82.5 %	231 k
100D CKY-based treeLSTM	**82.8** %	231 k
100D LSTM (Bowman et al., 2015)	77.6 %	220 k
300D SPINN (Bowman et al., 2016)	83.2 %	3.7 M
100D Yogatama et al. (2017)	80.5 %	500 k
100D Choi et al. (2018)	82.6 %	262 k
300D Choi et al. (2018)	85.6 %	2.9 M
300D DiSAN (Shen et al., 2017)	85.6 %	2.35 M

Table 7.3: Test set accuracy (higher is better) on the SNLI dataset for the two attentive models.

Model	**Test accuracy**
100D LSTM + attention	82.7 %
100D CKY-based treeLSTM + attention	**83.2** %

semicolons in long lists of synonyms, leading to many ungrammatical sentences. On the one hand, this might seem unfair on the supervised treeLSTM, which received suboptimal trees as input. On the other hand, it demonstrates the robustness of CKY-based method to noisy data: in a real-world setting, there is no guarantee that a trained parser would be available for the appropriate domain and language. The CKY-based model also performed well in comparison to the LSTM and the other treeLSTM baselines.

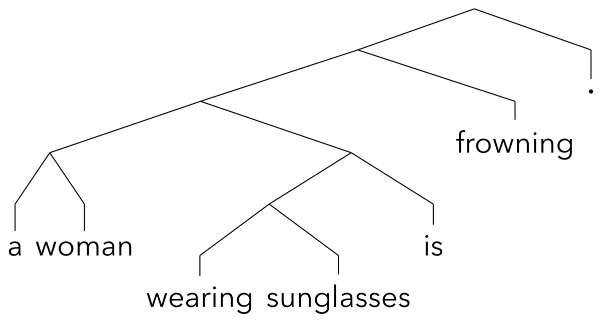
Following Yogatama et al. (2017), I also manually inspect the learned trees to see how closely they match conventional syntax trees, as would typically be assigned by trained linguists. I analyse the same four sentences they chose, taken from the SNLI corpus. The trees produced by the CKY-based model are shown in Figure 7.2. One notable feature is the fact that verbs are often joined with their subject noun phrases first, which differs from the standard verb phrase structure. This can be seen in Figure 7.2, and was also observed when manually inspecting trees from the development data. It should be noted that formalisms such as combinatory categorial grammar (Steedman, 2000), through type-raising and composition operators, do allow such constituents. The spans of prepositional phrases in Figures 7.2b to 7.2d are correctly identified at

Table 7.4: Median rank (lower is better) and accuracies (higher is better) at 10 and 100 on the three test sets for the reverse dictionary task: seen words (S), unseen words (U), and concept descriptions (C). Bold numbers indicate the top performance and exclude the greyed-out results of Hill et al. (2016), which I could not reproduce – see footnote 5 in this section for more information.

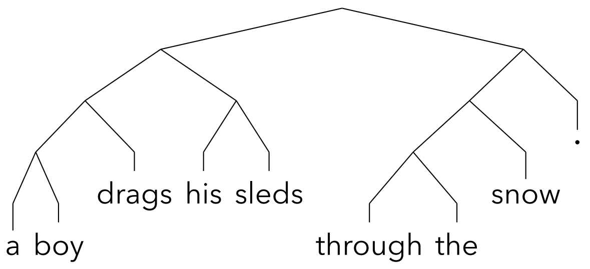
Model	**Median rank**			**Top 10 accuracy**			**Top 100 accuracy**		
	S	**U**	**C**	**S**	**U**	**C**	**S**	**U**	**C**
100D bag-of-words	75.0	66.0	70.5	30.3	29.9	25.8	53.7	55.2	56.6
100D LSTM	**57.5**	59.0	48.5	28.9	29.7	29.3	55.3	56.8	57.1
100D left-br. treeLSTM	78.0	64.0	48.0	28.9	28.3	28.8	52.7	54.8	61.1
100D right-br. treeLSTM	70.5	51.0	42.5	30.1	30.9	29.8	54.5	**58.0**	62.1
100D supervised treeLSTM	108.5	79.0	160.5	23.1	26.9	20.2	49.0	52.9	42.4
100D CKY-based treeLSTM	58.5	**40.0**	**40.0**	**30.9**	**33.4**	**30.3**	**56.1**	57.1	**62.6**
512D LSTM (Hill et al., 2016)	19	19	26	44	44	38	70	69	66
500D bag-of-words (Hill et al.)	15	14	28	46	46	36	71	71	66

the highest level; but only in Figure 7.2d does the structure of the subtree match convention. As could be expected, other features such as the attachment of the full stops or of some determiners do not appear to match human intuition.

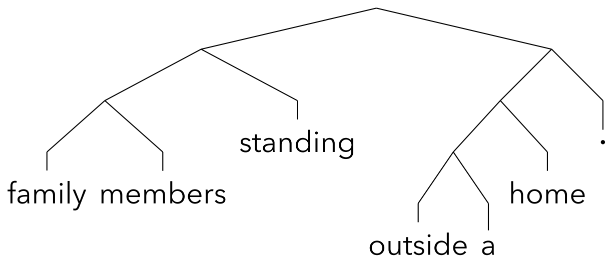
Further, I also analyse the trees induced by the model trained on the reverse dictionary task. The unusual tokenisation of this data, described earlier in the chapter, makes it hard to perform any kind of systematic comparison between the trees induced by the models trained on the two datasets. However, a manual inspection of the development set reveals some interesting regularities specific to the language constructs typical of dictionary definitions. Figure 7.3 shows definitions which refer the reader to other words, and how they were parsed. The referenced word, which is semantically closest to the word being defined, is almost always at the top of the tree, presumably making its effect stronger on the whole sentence representation, and aiding the model in performing its downstream task. Another notable regularity involved definitions of verbs (e.g. *trawling: to fish from a slow moving boat*, or *defer: to commit or entrust to another*) which were often very close to fully right-branching, putting the initial *to* and the subsequent infinitive very close to the top. A similar behaviour is observed for definitions of nouns starting with the indefinite article *a*, such as *fawn: a young deer, especially...* . None of these phenomena are observed for the model trained on natural language inference. Chapter 8 will include a more in-depth, quantitative analysis of the induced trees.



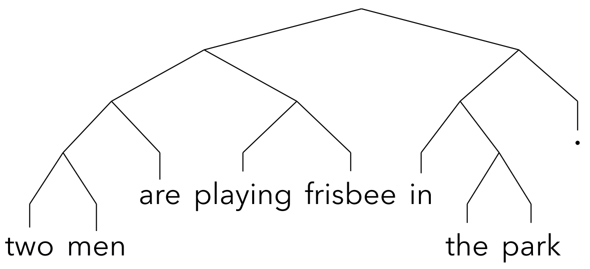
(a)



(b)

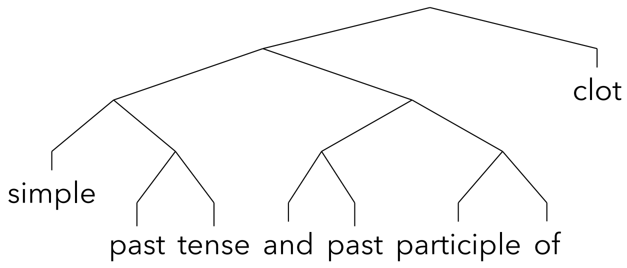


(c)

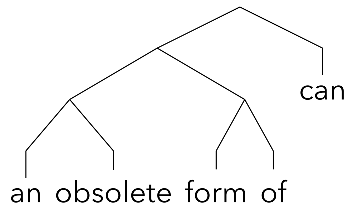


(d)

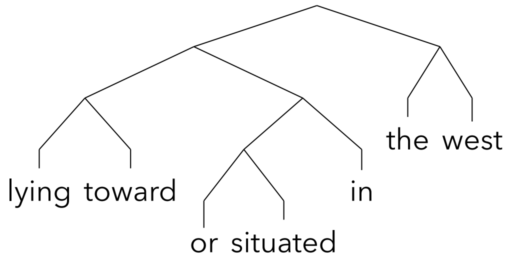
Figure 7.2: Binary parse trees of sentences induced by the CKY-based treeLSTM model trained on the SNLI dataset.



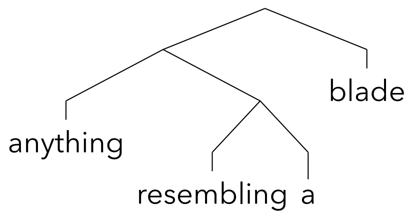
(a) Definition of *clotted*.



(b) Definition of *canes*.



(c) Definition of *western*.



(d) Definition of *blade*.

Figure 7.3: Binary parse trees of sentences induced by the CKY-based treeLSTM model trained on the dictionary task.

7.4 Summary

In this chapter I presented a fully differentiable model to jointly learn sentence embeddings and syntax, based on the treeLSTM composition function. Its benefits over standard treeLSTM encoders were demonstrated for a natural language inference task and a reverse dictionary task. Introducing an attention mechanism over the parse chart was shown to further improve performance. The model is a conceptually simple adaptation of the CKY parser, and it is easy to train via backpropagation and stochastic gradient descent with popular deep learning toolkits based on dynamic computation graphs, such as DyNet (Neubig et al., 2017) and PyTorch (Paszke et al., 2017).

The CKY-based treeLSTM I presented is relatively simple, but could be plausibly improved by combining it with aspects of other models. It should be noted in particular that Equation 7.1, the function assigning an energy to alternative ways of forming constituents, is extremely basic and does not rely on any global information on the sentence. Using a more complex function, perhaps relying on a mechanism such as the tracking LSTM in Bowman et al. (2016), might lead to improvements in performance. Techniques such as batch normalization (Ioffe and Szegedy, 2015) or layer normalization (Ba et al., 2016) might also lead to further improvements. In future work, it may be possible to obtain trees closer to human intuition by training models to perform well on multiple tasks instead of a single one, which is an important feature for intelligent agents to demonstrate (Legg and Hutter, 2007).

8 Shift-reduce parsing

In Chapter 7 I presented a latent tree learning model based on the CKY parser, which requires $\mathcal{O}\left(n^3\right)$ evaluation of its RNN cell per sentence, where n is the number of words. This leads to long training times. One of the more attractive aspects of the latent tree learning model of Yogatama et al. (2017), previously described in § 6.3.2, is its time complexity: a simple greedy shift-reduce parser, operating on an unlabelled binary grammar, will only require $\mathcal{O}(n)$ RNN evaluations. Unfortunately, as shown by Williams et al. (2018a) and as will be discussed later in this chapter, the trees induced by the model of Yogatama et al. are mostly trivial; and its downstream performance is lower compared to other models.

In this chapter, I will present a model (already published in Maillard and Clark, 2018) which, like Yogatama et al.'s, is based on shift-reduce parsing and therefore inherits its low time complexity; but with less trivial induced trees and better downstream performance on two natural language inference tasks. In order to avoid the complexities and potential drawbacks of reinforcement learning, the model exploits a trick to make its shift-reduce parsing component trainable via gradient descent. Using beam search, the final sentence representation is obtained using a soft-selection mechanism over representations corresponding to different beam elements. The selection mechanism makes use of the scores of the individual parsing actions, such that the training signal can reach the parsing component.

After having looked at the downstream performance, I will perform an analysis of the trees induced by the model, to investigate whether they are consistent with each other and across re-runs, and whether they resemble the trees produced by a standard parser. Further, I will compare these trees to those induced by the CKY-based model presented in the previous section, and to those induced by alternative models as reported by Williams et al. (2018a).

8.1 Models

For the experiments described in this chapter, I implement two models: the proposed beam search shift-reduce approach, and the CKY-based model, which I described in the previous chapter. The latter is reimplemented here with some slight differences compared to Chapter 7 to make comparisons to other latent tree learning models more fair, as will be described below in § 8.1.2.

8.1.1 Beam search shift-reduce treeLSTM

The proposed approach is based on shift-reduce parsing, and uses beam search to make the model differentiable. The CKY component of the model described in Chapter 7 is replaced here with a shift-reduce parser. It works with a queue, which holds the embeddings $\boldsymbol{w}_i \in \mathbb{R}^d$ for the nodes representing individual words which are still to be processed; and a stack, which holds the $\mathbf{c}$-states and $\mathbf{h}$-states ($\in \mathbb{R}^D$) of the nodes which have already been computed. The standard binary treeLSTM function, described in § 6.2, is used to compute the embeddings of nodes.

At the start, the queue contains embeddings for the nodes corresponding to single words. When a SHIFT action is performed, the topmost element of the queue is popped, passed through the treeLSTM (Equation 6.2), and pushed onto the stack. The resulting values at the top of the stack are thus

$$\mathbf{c}, \mathbf{h} = \text{treeLSTM}(\boldsymbol{w} = \boldsymbol{w}_{\text{word}}, \mathbf{c}_\ell = \mathbf{h}_\ell = \mathbf{c}_r = \mathbf{h}_r = 0),$$

where $\boldsymbol{w}_{\text{word}}$ is the word embedding that was popped off the queue. This is analogous to how leaf nodes are computed for the CKY-based model, as previously described in § 7.1.

When a REDUCE action is performed, the top two elements of the stack are popped. A new node is then computed as their parent, by passing the children's $\mathbf{c}$- and $\mathbf{h}$-states through the treeLSTM, with $\boldsymbol{w} = 0$. The resulting node is then pushed onto the stack. Details on how this component was implemented efficiently are available in § 8.2.1.

Parsing actions are scored with a simple multi-layer perceptron, which looks at the top two stack elements and the top queue element:

$$\mathbf{r} = \boldsymbol{W}_{s1} \cdot \mathbf{h}_{s1} + \boldsymbol{W}_{s2} \cdot \mathbf{h}_{s2} + \boldsymbol{W}_{q} \cdot \mathbf{h}_{q1}, \tag{8.1}$$

$$\mathbf{p} = \text{softmax}\left(\mathbf{a} + \boldsymbol{A} \cdot \tanh \mathbf{r}\right), \tag{8.2}$$

where $\mathbf{h}_{s1}, \mathbf{h}_{s2}, \mathbf{h}_{q1}$ are the $\mathbf{h}$-states of the top two elements of the stack and the top element of the queue, respectively. The three matrices $\boldsymbol{W} \in \mathbb{R}^{D \times D}$, the vector $\mathbf{a} \in \mathbb{R}^2$, and the matrix $\boldsymbol{A} \in \mathbb{R}^{2 \times D}$ are all learned. The final scores are given by $\log \mathbf{p}$, and the best action is greedily selected at every time step. The sentence representation is given by the $\mathbf{h}$-state of the top element of the stack after $2n - 1$ steps.

In order to make this model trainable with gradient descent, beam search is used to select the b best action sequences, where the score of a sequence of actions is given by the sum of the scores of the individual actions. The final sentence representation is then a weighted sum of the sentence representations from the elements of the beam. The weights are given by the respective scores of the action sequences, normalised by a softmax and passed through a straight-through estimator (Bengio et al., 2013). This is equivalent to having an argmax on the forward pass, which discretely selects the top-scoring beam element, and a softmax in

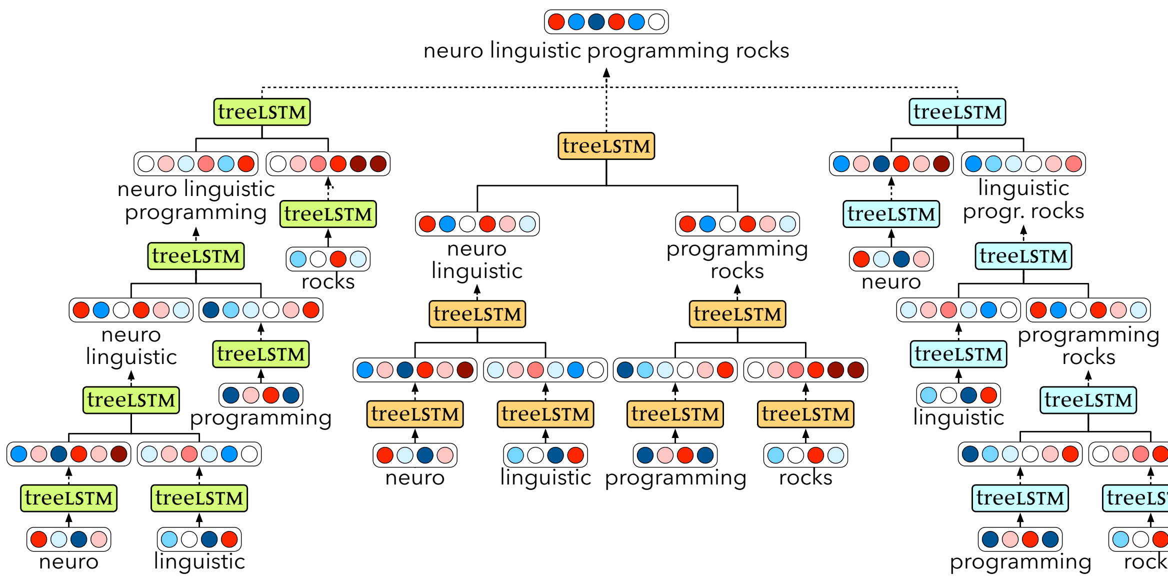


Figure 8.1: A latent tree learning sentence encoder based on a shift-reduce constituency parser, with beam search. In this example, the beam has size three. The treeLSTM composition functions are coloured differently to highlight the three beam elements. It should be noted, however, that they are the same function and share the same parameters.

the backward pass. The whole process for a beam of size $b = 3$ is illustrated in Figure 8.1 with an example sentence. The diagram shows the three different trees with the corresponding embeddings, and their weighted sum represented by the dashed lines merging into one.

8.1.2 CKY-based treeLSTM

When running preliminary experiments with the CKY-based model it became apparent that, despite the use of the temperature hyperparameter in Equation 7.2, the weighted sum still occasionally assigned non-trivial weight to more than one option. This was especially apparent with the longer sentences present in the MultiNLI dataset, as opposed to the generally shorter SNLI sentences used in the experiments of Chapter 7.

An example is given in Figure 8.2: the branch nodes show the weight assigned to each subtree, normalised over all possible other subtrees with the same span. For instance, out of the 16 possible choices the model could have made for the root node, the highest scoring one (showed in the figure) was assigned a normalised weight of 0.73. The remaining weight was spread over the other options.

The model was thus able to utilize multiple inferred trees, rather than a single one. This would have potentially given it an advantage over other tree-inducing models, making it not comparable to the other approaches. Hence, as the aim of this chapter's experiments is to analyse the (single) tree produced by each model for a given sentence, here I replace the temperature-weighting mechanism with a softmax followed by a straight-through estimator,

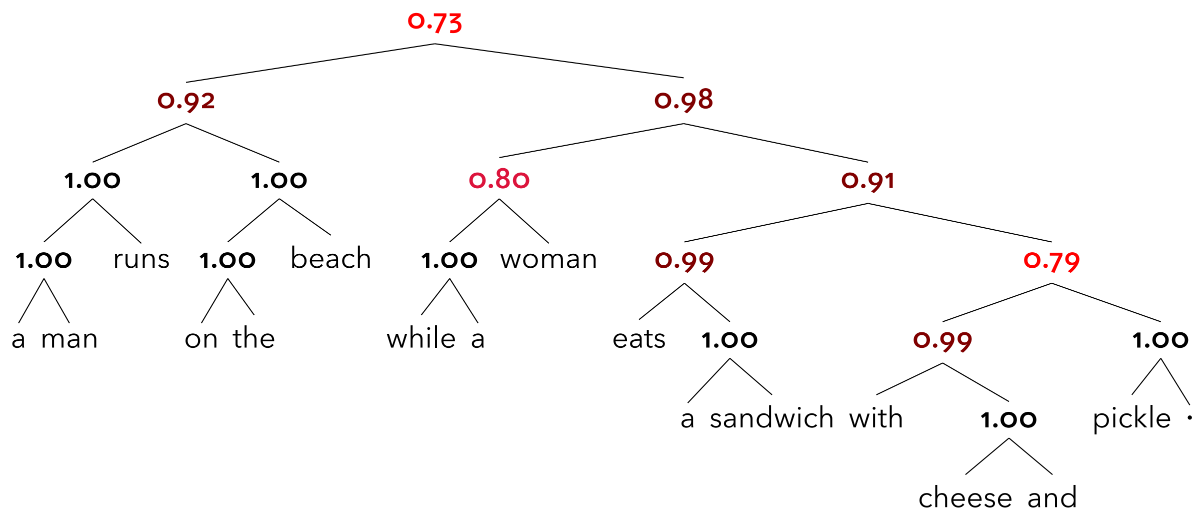


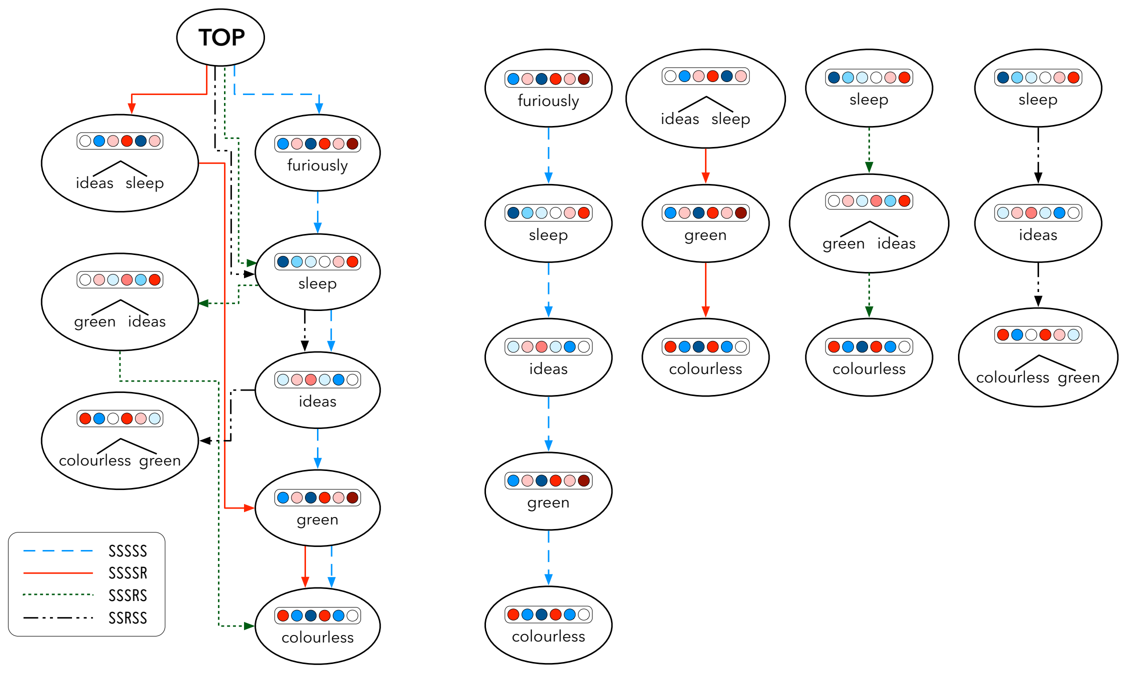
Figure 8.2: The tree induced for *a man runs on the beach while a woman eats a sandwich with pickle and cheese*. Branch nodes show the (normalised) weight that was assigned to a particular subtree for a given span, out of all possible other binary trees. Weights are coloured to highlight cases in which the model assigned non-zero weights to more than one option.

identical to the one used by the beam search model above. This change led to a slight decrease in downstream performance for the CKY-based model, when compared to the results of the experiments of Chapter 7. This was deemed acceptable, as the aim of this set of experiments is not to obtain the best possible downstream performance, but rather to perform a quantitative analysis of the induced trees.

8.2 Experimental setup

I initially attempted to run these experiments using the Tensorflow neural network library (Martın Abadi et al., 2015), at version 1.0. It quickly became apparent that this was not a viable approach. Tensorflow relies on a static computation graph, meaning that its computations are heavily optimised, but any changes to their structure will take a long time to re-optimise. Empirically, I found that building a new parse chart for the CKY-based model could take up to several minutes. As every batch of training data would have required a different sized parse chart, this would have made training prohibitively slow. The alternative approach of using padding was also highly inefficient, due to the time complexity of the algorithm. The experiments described in this chapter were thus run using the DyNet neural network library (Neubig et al., 2017) which, as the name suggests, is based on a dynamic computation graph.[1]

[1]As of April 2018, a form of dynamic computation is available in the stable version of Tensorflow under the name *eager execution*. While it does not support some of the advanced features of DyNet, such as automatic batching, it should now be possible to efficiently implement the experiments using Tensorflow.



(a) Pointer-based stack. (b) Naïve stack implementation.

Figure 8.3: The pointer-based (left) and naïve (right) stack implementations. The example shows the fifth time step of the parsing of *colourless green ideas sleep furiously*. Different styles of lines represent different beam elements, each with a corresponding sequence of parsing actions (bottom left). For the given example, using a pointer based stack reduces memory usage by a factor of 1.75, and avoids 12 repeated treeLSTM evaluations.

All experiments use natural language inference as the downstream task, evaluating on both the SNLI corpus (Bowman et al., 2015) and the MultiNLI corpus (Williams et al., 2018b) using the *matched* version of the development set. Following a common approach in the literature (Conneau et al., 2017; Williams et al., 2018a; inter alia), the MultiNLI corpus was augmented with SNLI training data.

As in Chapter 7, I used pre-trained 100-dimensional GloVe word embeddings (Pennington et al., 2014), fine-tuned during training, and Adam (Kingma and Ba, 2014) as the optimisation algorithm. For each combination of model and dataset, I trained five instances, each with a different random initialisation of the neural network parameters. Each instance was also fed the training data in a different random order. Thus, in total, I trained 2×2×5=20 different model instances, for a total training time of one and a half weeks.

8.2.1 Pointer-based stack

Due to the beam search used in the shift-reduce model, a naïve implementation of the algorithm would have led to multiple repeated computations. Let us consider, for example, the sentence *colourless green ideas sleep furiously*. Any beam element starting with the parsing actions

Table 8.1: SNLI and MultiNLI (matched) test set accuracy. Results marked with * are for the model variant without the leaf RNN transformation. Top results for 300-dimensional latent tree learning models are highlighted in bold; for 100-dimensional models, they are underlined.

Model	SNLI	MultiNLI
100D LSTM (Yogatama et al., 2017)	80.2	–
300D LSTM (Williams et al., 2018a)	82.6	69.1
100D treeLSTM (Yogatama et al., 2017)	78.5	–
300D SPINN (Bowman et al., 2016)	82.2	67.5
100D ST-Gumbel (Choi et al., 2018)	81.9	–
300D ST-Gumbel (Williams et al., 2018a)	83.3	**69.5**
300D ST-Gumbel* (Williams et al., 2018a)	**83.7**	67.5
100D RL-SPINN (Yogatama et al., 2017)	80.5	–
300D RL-SPINN* (Williams et al., 2018a)	82.3	67.4
100D CKY-based treeLSTM	82.2	<u>69.1</u>
100D shift-reduce treeLSTM	<u>83.0</u>	69.0

`shift,shift,reduce` would have needed to compute

$$\begin{aligned}
\mathbf{c}_1, \mathbf{h}_1 &= \text{treeLSTM}(\boldsymbol{w} = \boldsymbol{w}_{\text{colourless}}),\\
\mathbf{c}_2, \mathbf{h}_2 &= \text{treeLSTM}(\boldsymbol{w} = \boldsymbol{w}_{\text{green}}),\\
\mathbf{c}_3, \mathbf{h}_3 &= \text{treeLSTM}(\mathbf{c}_\ell = \mathbf{c}_1, \mathbf{h}_\ell = \mathbf{h}_1, \mathbf{c}_r = \mathbf{c}_2, \mathbf{h}_r = \mathbf{h}_2),
\end{aligned}$$

where $\boldsymbol{w}_{\text{word}}$ is the embedding for *word*, and the treeLSTM function is as described in Equation 6.2. As there are five possible parse trees starting with these actions, this would have led to potentially 12 repeated treeLSTM evaluations in the beam, just for the first three time steps.

In order to avoid these useless computations – and to make memory usage of the stack more efficient – I implemented the shift-reduce model using an approach vaguely inspired by the graph-structured stack of Tomita's GLR parsing algorithm (Tomita, 1984). The pointer-based approach, illustrated for an example sentence in Figure 8.3a, consists in representing the individual stacks of the beam elements as stacks of pointers, with the actual embeddings being memoised. As a result, different beam elements are effectively sharing memory. This should be compared to the naïve approach, shown in Figure 8.3b, which requires duplicating the stack several times. In the experiments described in this chapter, the pointer-based approach led to a reduction in memory usage of about 50% on average.

Table 8.2: Self and inter-model F1 scores for a number of models on the two datasets. Results marked with † are as reported in Williams et al. (2018a); and those marked with ∗ are for the model variant without the leaf RNN transformation.

(a) Self F1.

Model	**Self-F1**
MultiNLI	
300D SPINN†	71.5
300D ST-Gumbel†	49.9
300D ST-Gumbel*†	41.2
300D RL-SPINN*†	**98.5**
100D CKY-based	45.9
100D shift-reduce	46.6
Random Trees†	32.6
SNLI	
100D CKY-based	59.2
100D shift-reduce	**60.0**
Random Trees	35.9

(b) CKY-based and shift-reduce inter-model F1.

Dataset	**Inter-model F1**
SNLI	42.6
MultiNLI	55.0

8.3 Results and discussion

To ensure that models are learning useful sentence representations, I measure the downstream performance of the best CKY-based and shift-reduce models (as selected by development set performance) for both datasets. Table 8.1 shows accuracies of the two top-performing models and those of several other baselines, as well as the models of Yogatama et al. (2017) and Choi et al. (2018). These figures, along with the similar results from the previous studies in Tables 7.2 and 7.4, demonstrate that while the two models do not achieve the state of the art, they match or outperform other tree-inducing methods using 100-dimensional embeddings, as well as larger models using externally-provided parse trees.

In order to examine the consistency of trees induced by the CKY-based and shift-reduce models, I adapt the code of Williams et al. (2018a) to find the models' *self-F1*. This is defined as the unlabelled F1 between trees by two instances of the same model (given by different random initializations), averaged over all possible pairs. To make these figures more easily interpretable, I also report the self-F1 between randomly generated trees. Further, I measure the *inter-model F1*, which is defined as the unlabelled F1 between instances of the CKY-based and shift-reduce models trained on the same data, averaged over all possible pairs. These are reported in Tables 8.2a and 8.2b.

From the self-F1 results, it can be seen that both the CKY-based and the shift-reduce models are

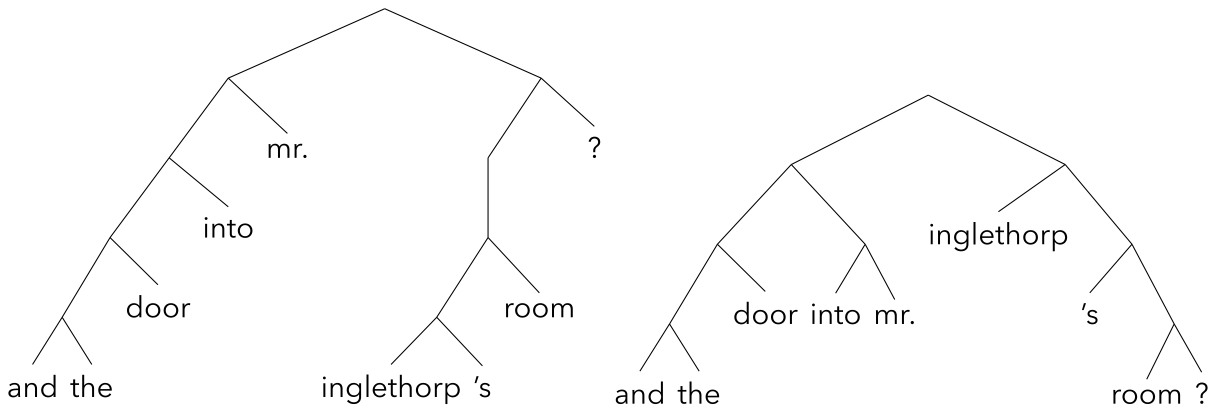
Table 8.3: Unlabelled F1 scores of the trees induced by various models against: other runs of the same model, fully left- and right-branching trees, and Stanford parser trees provided with the datasets. Results marked with † are as reported in Williams et al. (2018a); those marked with ‡ are from Yogatama et al. (2017); and those marked with ∗ are for the model variant without the leaf RNN transformation.

	F1 with respect to:					
	Left-branching		**Right-branching**		**Stanford parser**	
Model	μ (σ)	**max**	μ (σ)	**max**	μ (σ)	**max**
			MultiNLI			
300D SPINN†	19.3 (0.4)	19.8	36.9 (3.4)	**42.6**	**70.2** (3.6)	**74.5**
300D ST-Gumbel†	32.6 (2.0)	35.6	**37.5** (2.4)	40.3	23.7 (0.9)	25.2
300D ST-Gumbel*†	30.8 (1.2)	32.3	35.6 (3.3)	39.9	27.5 (1.0)	29.0
300D RL-SPINN*†	**99.1** (0.6)	**99.8**	10.7 (0.2)	11.1	18.1 (0.1)	18.2
100D CKY-based	32.9 (1.9)	35.1	31.5 (2.3)	35.1	23.7 (1.1)	25.0
100D shift-reduce	40.6 (6.5)	47.6	24.2 (6.0)	27.7	23.5 (1.8)	26.2
Random Trees†	27.9 (0.1)	27.9	28.0 (0.1)	28.1	27.0 (0.1)	27.1
			SNLI			
100D RL-SPINN‡	–	41.4	–	19.9	–	**41.7**
100D CKY-based	43.9 (2.2)	46.9	**33.7** (2.6)	**36.7**	30.3 (1.1)	32.1
100D shift-reduce	**48.8** (5.2)	**53.9**	26.5 (6.9)	34.0	**32.8** (3.5)	36.4
Random Trees	32.3 (0.1)	32.4	32.5 (0.1)	32.6	32.3 (0.1)	32.5

well above the baseline of random trees. Remarkably, the models trained on SNLI are noticeably more self-consistent, showing that the specific training data can play an important role, even when the downstream task is the same. A possible explanation is that the MultiNLI corpus has longer sentences, as well as multiple genres (including telephone conversations, which often do not constitute full sentences).

The inter-model F1 scores are not much lower than the self F1 scores. This shows that, given the same training data, the grammars learned by the two different models are not much more different than the grammars learned by two instances of the same model.

Finally, I investigate whether these models induce trees which are left-branching or right-branching, or similar to trees produceds by the Stanford parser. The rightmost column on Table 8.3 shows the unlabelled F1 between these and the trees from various models. While some models show a slight preference towards left-branching structures, it can be seen from the table that they do not learn anything resembling the trees from the Stanford parser. Figure 8.4 shows the trees induced by the CKY-based and shift-reduce models for a sentence sampled randomly from the development set. In this example, the shift-reduce model has a slight preference for



(a) Shift-reduce model. (b) CKY-based model.

Figure 8.4: Trees induced by the shift-reduce and CKY-based models for a sentence chosen at random from the MultiNLI development set: *And the door into Mr. Inglethorp's room?*

left-branching structures, confirming the numbers in Table 8.3.

8.4 Summary

In Chapter 7, we saw a latent tree learning model based on the CKY parsing. In this chapter, I went back to a shift-reduce parsing approach. I tried to see whether it would be possible to build an effective latent tree learning model using this parsing paradigm, without the drawbacks exhibited by the model of Yogatama et al. (2017).

I used beam search as a trick to enable training via gradient descent, even though shift-reduce parsing is based on a series of discrete actions. This is in contrast to the RL-SPINN model of Yogatama et al., which required reinforcement learning for training.

Even though the model is conceptually simpler than SPINN-based models, by replacing the tracking LSTM component (Figure 6.4) with a simple action scoring function (Equation 8.2), it was shown to outperform both 100- and 300-dimensional versions of RL-SPINN on the downstream tasks.

In the final section of this chapter I analysed the trees induced by the CKY-based and shift-reduce models, as well as two other latent tree learning models. The results confirm those of previous work on different models (Williams et al., 2018a), showing that the learned trees do not resemble Penn Treebank-style grammars. Interestingly, the two proposed models tend to induce trees which are not much more different than those learned by two instances of the same model.

There are several directions in which this work could be taken. Both the CKY-based and the shift-reduce models could be extended using a leaf transformation in the style of Choi et al. (2018), to give the model a more global overview of the sentence before the start of the parsing process. The shift-reduce model could be extended with a more sophisticated scoring function, either via a multi-layer perceptron that looks at more of the stack, or via a mechanism similar to the tracking LSTM of Bowman et al. (2016). Finally, it would be interesting to investigate using a shift-reduce dependency parser in the model, and allowing it to ignore words of low semantic content.

9 Conclusions

This thesis has compared two broad schools of thought regarding phrase and sentence embedding models. Both starting from the meanings of individual words as vectors, they differ in how they combine these to obtain the meaning of larger linguistic units. The first, widely adopted, mostly sees language processing as an engineering problem: recurrent neural networks, LSTM encoders, and *bag-of-words* models have little in common with the compositional process theorised in formal semantics, and are commonly used in other areas of machine learning. The second family, that of *compositional* models, is based on approaches which more closely follow the *principle of compositionality*, by imposing structural constraints or strong inductive biases consistent with the assumption of a syntax-semantics homomorphism (Dowty, 2007).

To summarise, after an introduction to compositional semantics in Part I, I presented several concrete proposals of compositional models. My contributions can be broadly grouped into two families: the tensor-based *categorial* methods, where the composition order is entirely driven (and therefore constrained) by an externally provided parse tree (Part II); and the *latent tree learning* methods, which are based on recurrent neural networks, and compose according to an automatically induced parse tree (Part III). Schematically, the areas covered by the core parts of this thesis can be visualised as follows:

Table 9.1: Schematic view of compositional models presented in this thesis.

		Composition function	
		Tensor-based	**Recurrent NN**
Parse tree	**Provided**	Part II: Categorial models	——
	Induced	——	Part III: Latent tree learning models

In Part II we saw some encouraging results for categorial models of simple adjective-noun compositionality (Chapter 5). However a similar technique, extended to the more complex case of relative clauses, failed to outperform a very simple bag-of-words baseline (Chapter 4). While there are reasons to believe that categorial models have more room for improvement than bag-of-words models (§ 4.5), the fact remains that these parameter-rich methods require a large amount of data to train. Furthermore, due to the nature of the approach, this data must be parsed, making it harder to create large training sets, especially for domains and languages that lack high-quality automatic parsers. Finally, it could be the case that modelling composition via element-element interactions, as is done by tensors, may not be the right approach after all – a potential mistake which would be costly in terms of model size, especially for complex grammatical structures requiring tensors of fourth and higher orders.

In order to obviate the need for automatic parsers, and to reduce model size, I began investigating a class of methods that would later become known as *latent tree learning* models. These are based on word embeddings and a variant of the popular LSTM architecture, leading to a dramatic reduction in the number of parameters. Furthermore, they are able to operate on unparsed inputs, thus widening the range of training data they can consume. We may summarise their differences with categorial models as follows:

Model class	**Number of parameters**	**Training data**
tensor-based categorial	$\sum_k \lvert V_k \rvert \cdot D^k$	parsed
treeLSTM-based latent tree learning	$\lvert V \rvert \cdot D + 5D^2$	raw

where $|V|$ is the total size of the vocabulary; $|V_k|$ is the size of the vocabulary of words with corresponding k-th order tensors, with $\cup_k V_k = V$; and D is the size of the semantic vector space – which, for simplicity, is here assumed to be equal for all atomic arguments, as well as for the hidden states of the RNN.[1] The $5D^2$ term in the second row derives from the treeLSTM composition function, with bias terms left out for simplicity.

The two latent tree learning models I put forward in Part III are based on chart parsing (Chapter 7) and shift-reduce parsing (Chapter 8). They showed promising results when evaluated on two natural language inference datasets and a reverse dictionary task, and an analysis of the induced trees revealed a certain level of consistency in the structures chosen by the models (§ 8.3). While these results, by themselves, are not enough to argue that these models should replace LSTMs in the standard NLP toolkit, there are reasons to be optimistic. The field of latent tree learning has seen some rapid development in the past few months. A success story that is worth mentioning is the very recently proposed model of Shen et al. (2018) – who, at the time of writing the paper, did not seem to be aware of any of the other literature on latent tree learning. Their model, based on convolutional neural networks and greedy parsing, achieves near state-of-the-art results on character- and word-based language modelling, and promising results on unsupervised constituency parsing. In a follow-up paper, Htut et al. (2018) show the viability of this approach for grammar induction. They demonstrate that the model of Shen et al. correctly brackets over

[1]This does not have to be true and, further, tensors could be decomposed to reduce the number of parameters (as previously discussed in § 3.1). In general, however, a tensor-based categorial model will have a much higher number of parameters than a treeLSTM-based latent tree learning model.

64% of the noun phrases in the Wall Street Journal corpus, and that it achieves over 70% F1 on a subset of the dataset with sentences of at most 10 words. Another novel approach is the ON-LSTM model (unpublished work by Shen et al., 2019), which imposes a strict hierarchy on constituents, but unlike a treeLSTM it does so without resorting to an explicit tree structure. This new generation of models, along with the rest of the results presented in this thesis, show that latent tree learning can achieve strong results in a variety of NLP tasks, while keeping model complexity and data annotation costs low.

With this thesis, I hope to have shown that models with strong inductive biases which are in line with the principle of compositionality – such as latent tree learning models – constitute a promising area for future research in compositional semantics. While we didn't completely succeed in finding a fully scalable, transparent, linguistically motivated approach, these models look promising in various ways, both in terms of downstream performance and interpretability.

The results presented in the previous chapters point out several potential future directions of investigation. Given the increase in availability of computational power since 2014, it would be interesting to repeat the experiments of Chapter 5 with a directly maximised likelihood. Similarly, for any further work on the relative clause models of Chapter 4, it would be worth investigating the use of nonlinearities to enable the modelling of higher-order interactions. For the latent tree learning approaches of Part III, the search space over possible model and task architectures is extremely vast, and only a small number of options were evaluated. Topics that are yet to be explored include: sampling via the Gumbel trick when using a straight-through estimator (§ 8.1.1), and more generally a thorough comparison of various gradient estimation techniques through discrete choices; testing different transition systems with the shift-reduce parser (to support, e.g., a dependency-based approach); and a comparison of the trees induced by models trained on a wider range of linguistic tasks. Finally, latent tree learning suggests new ways of performing regularisation. Currently, to reduce overfitting, neural models mainly rely on methods such as weight penalties, dropout, and architectural ablation. An interesting area of research would be to explore the use of soft constraints over types of trees as a regulariser.

Bibliography

E. Agirre, D. Cer, M. Diab, and A. Gonzalez-Agirre (2012). 'SemEval-2012 Task 6: A Pilot on Semantic Textual Similarity'. In: **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*. Montréal, Canada: Association for Computational Linguistics, pp. 385–393.

E. Agirre et al. (2015). 'SemEval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability'. In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, Colorado: Association for Computational Linguistics, pp. 252–263.

J. L. Ba, J. R. Kiros, and G. E. Hinton (2016). *Layer Normalization*. URL: `http://arxiv.org/abs/1607.06450`.

D. Bahdanau, K. Cho, and Y. Bengio (2015). 'Neural Machine Translation by Jointly Learning to Align and Translate'. In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*.

E. Balkır, D. Kartsaklis, and M. Sadrzadeh (2018). 'Sentence Entailment in Compositional Distributional Semantics'. In: *Annals of Mathematics and Artificial Intelligence* 82.4, pp. 189–218. ISSN: 1012-2443. DOI: `10.1007/s10472-017-9570-x`.

M. Baroni, R. Bernardi, and R. Zamparelli (2013). 'Frege in space: A program for compositional distributional semantics'. In: *Linguistic Issues in Language Technology* 9.

M. Baroni, G. Dinu, and G. Kruszewski (2014). '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 (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 238–247. DOI: `10.3115/v1/P14-1023`.

M. Baroni and R. 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*. Cambridge, MA: Association for Computational Linguistics, pp. 1183–1193.

R. W. Bemer (1980). 'Best of Interface Age, Volume 2: General Purpose Software'. In: Portland, OR, USA: Dilithium Press. Chap. Chapter 1: Inside ASCII, pp. 1–50. ISBN: 0-918398-37-1.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin (2003). 'A Neural Probabilistic Language Model'. In: *The Journal of Machine Learning Research* 3, pp. 1137–1155. ISSN: 1532-4435.

Y. Bengio, N. Léonard, and A. C. Courville (2013). 'Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation'. In:

P. Blackburn and J. Bos (2005). *Representation and Inference for Natural Language: a First Course in Computational Semantics*. Center for the Study of Language and Information. ISBN: 1575864967.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov (2017). 'Enriching Word Vectors with Subword Information'. In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. ISSN: 2307-387X.

G. Boleda and A. Herbelot (2016). 'Formal Distributional Semantics: Introduction to the Special Issue'. In: *Computational Linguistics* 42.4, pp. 619–635. DOI: 10.1162/COLI_a_00261.

G. Boleda, E. M. Vecchi, M. Cornudella, and L. McNally (2012). 'First Order vs. Higher Order Modification in Distributional Semantics'. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 1223–1233.

J. Bos (2005). 'Towards wide-coverage semantic interpretation'. In: *Proceedings of the Sixth International Workshop on Computational Semantics (IWCS-6)*, pp. 42–53.

J. Bos, S. Clark, M. Steedman, J. R. Curran, and J. Hockenmaier (2004). 'Wide-Coverage Semantic Representations from a CCG Parser'. In: *Proceedings of Coling 2004*. Geneva, Switzerland: COLING, pp. 1240–1246.

S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning (2015). 'A large annotated corpus for learning natural language inference'. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 632–642.

S. R. Bowman, J. Gauthier, A. Rastogi, R. Gupta, C. D. Manning, and C. Potts (2016). 'A Fast Unified Model for Parsing and Sentence Understanding'. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1466–1477.

T. Briscoe and J. Carroll (2002). 'Robust accurate statistical annotation of general text'. In: *Proceedings of the 3rd LREC Conference*. Las Palmas, Gran Canaria, pp. 1499–1504.

E. Bruni, N. K. Tran, and M. Baroni (2014). 'Multimodal Distributional Semantics'. In: *Journal of Artificial Intelligence Research* 49.1, pp. 1–47. ISSN: 1076-9757.

W. Buszkowski and K. Moroz (2008). 'Pregroup grammars and context-free grammars'. In: *Computational Algebraic Approaches to Natural Language*. Ed. by C. Casadio and J. Lambek. Polimetrica, pp. 1–21.

W. Buszkowski (2001). 'Lambek Grammars Based on Pregroups'. In: *Logical Aspects of Computational Linguistics*. Ed. by P. de Groote, G. Morrill, and C. Retoré. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 95–109. ISBN: 978-3-540-48199-7.

R. Cann (1993). *Formal Semantics: An Introduction*. Cambridge Textbooks in Linguistics. Cambridge University Press. DOI: 10.1017/CBO9781139166317.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). 'Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-*

guage Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734.

J. Choi, K. M. Yoo, and S.-g. Lee (2018). 'Learning to Compose Task-Specific Tree Structures'. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

N. Chomsky (1957). *Syntactic Structures*. Mouton de Gruyter. ISBN: 978-3-11-017279-9.

S. Clark (2015). 'Vector Space Models of Lexical Meaning'. In: *The Handbook of Contemporary Semantic Theory*. Ed. by S. Lappin and C. Fox. Wiley-Blackwell. DOI: 10.1002/9781118882139.ch16.

S. Clark, B. Coecke, and M. Sadrzadeh (2008). 'A compositional distributional model of meaning'. In: *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*. Oxford, United Kingdom.

S. Clark and J. R. Curran (2007). 'Wide-coverage Efficient Statistical Parsing with Ccg and Log-linear Models'. In: *Computational Linguistics* 33.4, pp. 493–552. ISSN: 0891-2017. DOI: 10.1162/coli.2007.33.4.493.

S. Clark, L. Rimell, T. Polajnar, and J. Maillard (2016). *The Categorial Framework for Compositional Distributional Semantics*. Tech. rep. University of Cambridge.

J. Cocke (1969). *Programming Languages and Their Compilers: Preliminary Notes*. Courant Institute of Mathematical Sciences, New York University. ISBN: B0007F4UOA.

B. Coecke and É. O. Paquette (2010). 'Categories for the Practising Physicist'. In: *New Structures for Physics*. Lecture Notes in Physics. Springer, Berlin, Heidelberg, pp. 173–286. ISBN: 978-3-642-12820-2 978-3-642-12821-9. DOI: 10.1007/978-3-642-12821-9_3.

B. Coecke, M. Sadrzadeh, and S. Clark (2011). 'Mathematical Foundations for a Compositional Distributed Model of Meaning'. In: *Linguistic Analysis* 36.1–4, pp. 133–140.

R. Collobert and J. Weston (2008). 'A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning'. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, pp. 160–167. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390177.

A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). 'Supervised Learning of Universal Sentence Representations from Natural Language Inference Data'. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 670–680.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman (1990). 'Indexing by latent semantic analysis'. In: *Journal of the American Society for Information Science* 41.6, pp. 391–407.

D. Dowty (2007). 'Compositionality as an empirical problem'. In: *Direct Compositionality*. Ed. by C. Barker and P. Jacobson. Oxford: Oxford University Press. Chap. 2.

D. R. Dowty, R. Wall, and S. Peters (1981). *Introduction to Montague Semantics*. Studies in Linguistics and Philosophy. Springer Netherlands. ISBN: 9789027711410.

M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). 'Retrofitting Word Vectors to Semantic Lexicons'. In: *Proceedings of the 2015 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies. Denver, Colorado: Association for Computational Linguistics, pp. 1606–1615.

M. Faruqui and C. Dyer (2015). 'Non-distributional Word Vector Representations'. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, pp. 464–469. DOI: `10.3115/v1/P15-2076`.

C. Fellbaum (1998). *WordNet: An Electronic Lexical Database*. MIT Press. ISBN: 9780262061971.

J. Firth (1957). *Papers in linguistics, 1934-1951*. Oxford University Press.

G. Frege (1980). 'Letter to Jourdain'. In: *Philosophical and Mathematical Correspondence*. Ed. by G. Gabriel, H. Hermes, F. Kambartel, C. Thiel, and A. Veraart. Trans. by H. Kaal. Chicago University Press. Original date ca. 1914.

D. Fried, T. Polajnar, and S. Clark (2015). 'Low-Rank Tensors for Verbs in Compositional Distributional Semantics'. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, pp. 731–736.

G. Gazdar (1996). 'Paradigm Merger in Natural Language Processing'. In: *Computing Tomorrow*. Ed. by I. Wand and R. Milner. New York, NY, USA: Cambridge University Press, pp. 88–109. ISBN: 0-521-46085-9.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin (2017). 'Convolutional Sequence to Sequence Learning'. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1243–1252.

A. Goldberg (2003). 'Constructions: A New Theoretical Approach to Language'. In: *Trends in Cognitive Sciences* 7.5, pp. 219–224.

A. Goldberg (2015). 'Compositionality'. In: *The Routledge Handbook of Semantics*. Routledge. Chap. 24, pp. 419–433.

Y. Goldberg and M. Elhadad (2010). 'An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing'. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California: Association for Computational Linguistics, pp. 742–750.

Y. Goldberg and O. Levy (2014). *word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method*. URL: `http://arxiv.org/abs/1402.3722`.

A. Graves and J. Schmidhuber (2005). 'Framewise phoneme classification with bidirectional LSTM and other neural network architectures'. In: *Neural Networks* 18.5–6, pp. 602–610.

E. Grefenstette, G. Dinu, Y. Zhang, M. Sadrzadeh, and M. Baroni (2013). 'Multi-Step Regression Learning for Compositional Distributional Semantics'. In: *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013) – Long Papers*. Potsdam, Germany: Association for Computational Linguistics, pp. 131–142.

E. Grefenstette (2013a). 'Category-Theoretic Quantitative Compositional Distributional Models of Natural Language Semantics'. PhD thesis. University of Oxford.

E. Grefenstette (2013b). 'Towards a Formal Distributional Semantics: Simulating Logical Calculi with Tensors'. In: *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*. Atlanta, Georgia, USA: Association for Computational Linguistics, pp. 1–10.

E. Grefenstette and M. Sadrzadeh (2011a). 'Experimental Support for a Categorical Compositional Distributional Model of Meaning'. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 1394–1404.

E. Grefenstette and M. Sadrzadeh (2011b). 'Experimenting with transitive verbs in a DisCoCat'. In: *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*. Edinburgh, UK: Association for Computational Linguistics, pp. 62–66.

E. Grefenstette and M. Sadrzadeh (2015). 'Concrete Models and Empirical Evaluations for the Categorical Compositional Distributional Model of Meaning'. In: *Computational Linguistics* 41.1, pp. 71–118. ISSN: 0891-2017. DOI: 10.1162/COLI_a_00209.

E. 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*. Uppsala, Sweden: Association for Computational Linguistics, pp. 33–37.

E. Gumbel (1954). *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*. U.S. Government Printing Office.

A. Gupta, J. Utt, and S. Padó (2015). 'Dissecting the practical lexical function model for compositional distributional semantics'. In: *Proceedings of the the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015)*. Denver, Colorado, pp. 153–158.

M. U. Gutmann and A. Hyvärinen (2012). 'Noise-contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics'. In: *The Journal of Machine Learning Research* 13.1, pp. 307–361. ISSN: 1532-4435.

K. M. Hermann and P. 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 (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 894–904.

K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom (2015). 'Teaching Machines to Read and Comprehend'. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., pp. 1693–1701.

F. Hill, K. Cho, A. Korhonen, and Y. Bengio (2016). 'Learning to Understand Phrases by Embedding the Dictionary'. In: *Transactions of the Association for Computational Linguistics* 4, pp. 17–30. ISSN: 2307-387X.

G. E. Hinton, J. L. McClelland, and D. E. Rumelhart (1986). 'Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1'. In: ed. by D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group. Cambridge, MA, USA: MIT Press. Chap. Distributed Representations, pp. 77–109. ISBN: 0-262-68053-X.

S. Hochreiter and J. Schmidhuber (1997). 'Long Short-Term Memory'. In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

J. Hockenmaier and M. Steedman (2007). 'CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank'. In: *Computational Linguistics* 33.3.

P. M. Htut, K. Cho, and S. Bowman (2018). 'Grammar Induction with Neural Language Models: An Unusual Replication'. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 4998–5003.

S. Ioffe and C. Szegedy (2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*. Vol. 37. Lille, France, pp. 448–456.

E. Jang, S. Gu, and B. Poole (2017). 'Categorical Reparameterization with Gumbel-Softmax'. In: *anternational Conference on Learning Representations*.

R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). *Exploring the Limits of Language Modeling*. URL: https://arxiv.org/abs/1602.02410.

R. Jozefowicz, W. Zaremba, and I. Sutskever (2015). 'An empirical exploration of recurrent network architectures'. In: *Journal of Machine Learning Research*.

D. Jurafsky and J. H. Martin (2009). *Speech and Language Processing*. 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 0131873210.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom (2014). 'A Convolutional Neural Network for Modelling Sentences'. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 655–665.

D. Kartsaklis, M. Sadrzadeh, and S. Pulman (2013). 'Separating Disambiguation from Composition in Distributional Semantics'. In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 114–123.

D. Kartsaklis (2014). 'Compositional Distributional Semantics with Compact Closed Categories and Frobenius Algebras'. PhD thesis. University of Oxford.

T. Kasami (1965). *An efficient recognition and syntax analysis algorithm for context-free languages*. Tech. rep. AFCRL-65-758. Bedford, MA: Air Force Cambridge Research Laboratory.

S. Kim, J. Hong, I. Kang, and N. Kwak (2018). *Semantic Sentence Matching with Densely-connected Recurrent and Co-attentive Information*. URL: http://arxiv.org/abs/1805.11360.

Y. Kim (2014). 'Convolutional Neural Networks for Sentence Classification'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1746–1751.

D. P. Kingma and J. Ba (2014). 'Adam: A Method for Stochastic Optimization'. In:

T. Kocmi and O. Bojar (2017). 'An Exploration of Word Embedding Initialization in Deep-Learning Tasks'. In: *Proceedings of the 14th International Conference on Natural Language Processing (ICON-2017)*. Kolkata, India: NLP Association of India, pp. 56–64.

J. Krishnamurthy and T. M. Mitchell (2014). 'Joint Syntactic and Semantic Parsing with Combinatory Categorial Grammar'. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 1188–1198.

J. 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'. In: *Psychological Review* 104(2), pp. 211–240.

F. W. Lawvere and S. H. Schanuel (2009). *Conceptual Mathematics: A First Introduction to Categories*. 2nd ed. Cambridge University Press. ISBN: 978-0-521-71916-2.

S. Legg and M. Hutter (2007). 'Universal Intelligence: A Definition of Machine Intelligence'. In: *Minds Mach.* 17.4, pp. 391–444. ISSN: 0924-6495. DOI: 10.1007/s11023-007-9079-x.

M. Lewis, K. Lee, and L. Zettlemoyer (2016). 'LSTM CCG Parsing'. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 221–231.

M. Lewis and M. Steedman (2013). 'Combined Distributional and Logical Semantics'. In: *Transactions of the Association for Computational Linguistics* 1, pp. 179–192.

S. Mac Lane (1978). *Categories for the Working Mathematician*. 2nd ed. Graduate Texts in Mathematics. New York: Springer-Verlag. ISBN: 978-0-387-98403-2.

J. Maillard and S. Clark (2015). 'Learning Adjective Meanings with a Tensor-Based Skip-Gram Model'. In: *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. Beijing, China: Association for Computational Linguistics, pp. 327–331.

J. Maillard and S. Clark (2018). 'Latent Tree Learning with Differentiable Parsers: Shift-Reduce Parsing and Chart Parsing'. In: *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*. Melbourne, Australia: Association for Computational Linguistics, pp. 13–18.

J. Maillard, S. Clark, and E. Grefenstette (2014). 'A Type-Driven Tensor-Based Semantics for CCG'. In: *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 46–54. DOI: 10.3115/v1/W14-1406.

J. Maillard, S. Clark, and D. Yogatama (2017). *Jointly Learning Sentence Embeddings and Syntax with Unsupervised Tree-LSTMs*. URL: http://arxiv.org/abs/1705.09189.

C. D. Manning (2017). *Representations for Language: From Word Embeddings to Sentence Meanings*. URL: https://simons.berkeley.edu/sites/default/files/docs/6449/christophermanning.pdf.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky (2014). 'The Stanford CoreNLP Natural Language Processing Toolkit'. In: *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60.

M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger (1994). 'The Penn Treebank: Annotating Predicate Argument Structure'. In: *Proceedings of the Workshop on Human Language Technology*. HLT '94. Plainsboro, NJ: Association for Computational Linguistics, pp. 114–119. ISBN: 1-55860-357-3. DOI: 10.3115/1075812.1075835.

M. Marelli, L. Bentivogli, M. Baroni, R. Bernardi, S. Menini, and R. Zamparelli (2014). 'SemEval-2014 Task 1: Evaluation of Compositional Distributional Semantic Models on Full Sentences through Semantic Relatedness and Textual Entailment'. In: *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Dublin, Ireland: Association for Computational Linguistics and Dublin City University, pp. 1–8.

M.-C. d. Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning (2014). 'Universal Stanford dependencies: A cross-linguistic typology'. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. Reykjavik, Iceland: European Language Resources Association (ELRA).

Martın Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*.

T. Mikolov, K. Chen, G. Corrado, and J. Dean (2013a). 'Efficient Estimation of Word Representations in Vector Space'. In: *International Conference on Learning Representations (ICLR) Workshop*.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013b). 'Distributed Representations of Words and Phrases and their Compositionality'. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., pp. 3111–3119.

J. Mitchell and M. Lapata (2008). 'Vector-based Models of Semantic Composition'. In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 236–244.

J. Mitchell and M. Lapata (2010). 'Composition in distributional models of semantics'. In: *Cognitive Science* 34.8, pp. 1388–1429.

R. Montague (1973). 'The Proper Treatment of Quantification in Ordinary English'. In: *Approaches to Natural Language*. Ed. by P. Suppes, J. Moravcsik, and J. Hintikka. Reidel, pp. 221–242.

L. Mou, R. Men, G. Li, Y. Xu, L. Zhang, R. Yan, and Z. Jin (2016). 'Natural Language Inference by Tree-Based Convolution and Heuristic Matching'. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 130–136.

M. Nadejde, S. Reddy, R. Sennrich, T. Dwojak, M. Junczys-Dowmunt, P. Koehn, and A. Birch (2017). 'Predicting Target Language CCG Supertags Improves Neural Machine Translation'. In: *Proceedings of the Second Conference on Machine Translation*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 68–79.

G. Neubig et al. (2017). 'DyNet: The Dynamic Neural Network Toolkit'. In:

J. Nivre (2003). 'An Efficient Algorithm for Projective Dependency Parsing'. In: *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pp. 149–160.

D. Paperno, N. T. Pham, and M. Baroni (2014). 'A practical and linguistically-motivated approach to compositional distributional semantics'. In: *Proceedings of the 52nd Annual Meeting* *of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 90–99. DOI: 10.3115/v1/P14-1009.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). 'Automatic differentiation in PyTorch'. In: *NIPS Autodiff Workshop*.

J. Pennington, R. Socher, and C. D. Manning (2014). 'Glove: Global Vectors for Word Representation'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543.

M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (2018). 'Deep Contextualized Word Representations'. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237.

N. Pham, R. Bernardi, Y. Z. Zhang, and M. Baroni (2013). 'Sentence paraphrase detection: When determiners and word order make the difference'. In: *Proceedings of IWCS 2013 Workshop Towards a Formal Distributional Semantics*. Potsdam, Germany: Association for Computational Linguistics, pp. 21–29.

T. Polajnar, L. Fagarasan, and S. Clark (2014a). 'Reducing Dimensions of Tensors in Type-Driven Distributional Semantics'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1036–1046.

T. Polajnar, L. Rimell, and S. Clark (2014b). 'Evaluation of Simple Distributional Compositional Operations on Longer Texts'. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Ed. by N. C. (Chair), K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, and S. Piperidis. Reykjavik, Iceland: European Language Resources Association (ELRA). ISBN: 978-2-9517408-8-4.

P. Rajpurkar, R. Jia, and P. Liang (2018). 'Know What You Don't Know: Unanswerable Questions for SQuAD'. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 784–789.

L. Rimell, J. Maillard, T. Polajnar, and S. Clark (2016). 'RELPRON: A Relative Clause Evaluation Data Set for Compositional Distributional Semantics'. In: *Computational Linguistics* 42.4, pp. 661–701. DOI: 10.1162/COLI_a_00263.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986a). 'Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1'. In: ed. by D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group. Cambridge, MA, USA: MIT Press. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986b). 'Learning representations by back-propagating errors'. In: *Nature* 323.6088, pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

A. M. Rush, S. Chopra, and J. Weston (2015). 'A Neural Attention Model for Abstractive Sentence Summarization'. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 379–389.

M. Sadrzadeh, S. Clark, and B. Coecke (2013). 'The Frobenius anatomy of word meanings I: subject and object relative pronouns'. In: *Journal of Logic and Computation* 23.6, pp. 1293–1317.

M. Sadrzadeh, S. Clark, and B. Coecke (2016). 'The Frobenius anatomy of word meanings II: possessive relative pronouns'. In: *Journal of Logic and Computation* 26.2, pp. 785–815.

G. Salton, A. Wong, and C. S. Yang (1975). 'A Vector Space Model for Automatic Indexing'. In: *Commun. ACM* 18.11, pp. 613–620. ISSN: 0001-0782. DOI: 10.1145/361219.361220.

M. Schuster and K. K. Paliwal (1997). 'Bidirectional recurrent neural networks'. In: *IEEE Transactions on Signal Processing* 45.11.

H. Schütze (1998). 'Automatic Word Sense Discrimination'. In: *Computational Linguistics Special-Issue-on-Word Sense Disambiguation* 24.1.

T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang (2017). 'DiSAN: Directional Self-Attention Network for RNN/CNN-free Language Understanding'. In:

Y. Shen, Z. Lin, C.-w. Huang, and A. Courville (2018). 'Neural Language Modeling by Jointly Learning Syntax and Lexicon'. In: *International Conference on Learning Representations*.

Y. Shen, S. Tan, A. Sordoni, and A. Courville (2019). 'Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks'. In: *International Conference on Learning Representations*.

S. M. Shieber (1983). 'Sentence Disambiguation by a Shift-Reduce Parsing Technique'. In: *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, pp. 113–118. DOI: 10.3115/981311.981334.

R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng (2013). 'Parsing with Compositional Vector Grammars'. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 455–465.

R. Socher, B. Huval, C. D. Manning, and A. Y. Ng (2012). 'Semantic Compositionality Through Recursive Matrix-vector Spaces'. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12. Jeju Island, Korea: Association for Computational Linguistics, pp. 1201–1211.

R. Socher, C. D. Manning, and A. Y. Ng (2010). 'Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks'. In: *Proceedings of the Deep Learning and Unsupervised Feature Learning Workshop of NIPS 2010*, pp. 1–9.

R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning (2011). 'Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions'. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 151–161.

M. Steedman (2000). *The Syntactic Process*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-19420-1.

S. Subramanian, A. Trischler, Y. Bengio, and C. J. Pal (2018). 'Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning'. In: *International Conference on Learning Representations*.

M. Sundermeyer, R. Schlüter, and H. Ney (2012). 'LSTM Neural Networks for Language Modeling'. In: *Proceedings of the 13th Annual Conference of the International Speech Communication Association (INTERSPEECH 2012)*. Portland, Oregon, USA, pp. 194–197.

S. Suster and W. Daelemans (2018). 'CliCR: a Dataset of Clinical Case Reports for Machine Reading Comprehension'. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1551–1563.

I. Sutskever, O. Vinyals, and Q. V. Le (2014). 'Sequence to Sequence Learning with Neural Networks'. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., pp. 3104–3112.

Z. G. Szabó (2001). *Problems of Compositionality*. Routledge. ISBN: 9780815337904.

K. S. Tai, R. Socher, and C. D. Manning (2015). 'Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks'. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1556–1566.

M. Tomita (1984). 'LR Parsers for Natural Languages'. In: *Proceedings of the 10th International Conference on Computational Linguistics and 22Nd Annual Meeting on Association for Computational Linguistics*. ACL '84/COLING '84. Stanford, California: Association for Computational Linguistics, pp. 354–357. DOI: 10.3115/980491.980564.

P. D. Turney and P. Pantel (2010). 'From Frequency to Meaning: Vector Space Models of Semantics'. In: *Journal of Artificial Intelligence Research* 37.1, pp. 141–188. ISSN: 1076-9757.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). 'Attention Is All You Need'. In:

E. M. Vecchi, M. Baroni, and R. Zamparelli (2011). '(Linear) Maps of the Impossible: Capturing Semantic Anomalies in Distributional Space'. In: *Proceedings of the Workshop on Distributional Semantics and Compositionality*. DiSCo '11. Portland, Oregon: Association for Computational Linguistics, pp. 1–9. ISBN: 9781937284022.

E. M. Vecchi, M. Marelli, R. Zamparelli, and M. Baroni (2017). 'Spicy Adjectives and Nominal Donkeys: Capturing Semantic Deviance Using Compositionality in Distributional Spaces'. In: *Cognitive Science* 41.1, pp. 102–136. DOI: 10.1111/cogs.12330.

D. Westerståhl (2002). 'On the compositionality of idioms: An abstract approach'. In: *Words, proofs, and diagrams*. Ed. by D. Barker-Plummer, D. Beaver, J. van Benthem, and P. Scotto di Luzio. CSLI press, pp. 241–271.

A. Williams, A. Drozdov, and S. R. Bowman (2018a). 'Do latent tree learning models identify meaningful structure in sentences?' In: *Transactions of the Association for Computational Linguistics* 6, pp. 253–267. ISSN: 2307-387X.

A. Williams, N. Nangia, and S. Bowman (2018b). 'A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference'. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1112–1122.

R. J. Williams (1992). 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. In: *Machine Learning*, pp. 229–256.

W. Xu (2016). 'LSTM Shift-Reduce CCG Parsing'. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 1754–1764.

W. Xu, S. Clark, and Y. Zhang (2014). 'Shift-Reduce CCG Parsing with a Dependency Model'. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 218–227.

D. Yogatama, P. Blunsom, C. Dyer, E. Grefenstette, and W. Ling (2017). 'Learning to compose words into sentences with reinforcement learning'. In: *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.

D. H. Younger (1967). 'Recognition and Parsing of Context-Free Languages in Time n^3'. In: *Information and Control* 10, pp. 189–208.

D. Yuan, J. Richardson, R. Doherty, C. Evans, and E. Altendorf (2016). 'Semi-supervised Word Sense Disambiguation with Neural Models'. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 1374–1385.

L. Zettlemoyer and M. Collins (2007). 'Online Learning of Relaxed CCG Grammars for Parsing to Logical Form'. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 678–687.

X. Zhu, P. Sobhani, and H. Guo (2015). 'Long Short-term Memory over Recursive Structures'. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, pp. 1604–1612.