Composition Models for the Representation and Semantic Interpretation of Nominal Compounds

D i s s e r t a t i o n
zur
Erlangung des akademischen Grades
Doktor der Philosophie
in der Philosophischen Fakultät

der Eberhard Karls Universität Tübingen

vorgelegt von

Gina-Corina Dima, geb. Vrînceanu

aus

Roman

2019

Gedruckt mit Genehmigung der Philosophischen Fakultät
der Eberhard Karls Universität Tübingen

Dekan: Prof. Dr. Jürgen Leonhardt

Hauptberichterstatter: Prof. Dr. Erhard Hinrichs
Mitberichterstatterin: Dr. Melanie Bell
Mitberichterstatter: Prof. Dr. Gerhard Jäger

Tag der mündlichen Prüfung: 7. Februar 2019

Universitätsbibliothek Tübingen (TOBIAS-lib), Tübingen

Acknowledgements

Looking back at the last eight years of my life, there is much to be grateful to God for. First, for giving me a curious mind and the opportunity to apply my curiosity to fun and exciting research problems. And then for meeting all the wonderful people that made it possible, through their contributions, to transform the theoretical possibility of pursuing a PhD degree into reality:

My advisor, Prof. Dr. Erhard Hinrichs from the University of Tübingen, Germany, who provided the scientific environment for my thesis to be developed. I am grateful to him for the trust and encouragement he has always showed me, for all his patience throughout my long PhD journey, and for the multitude of interesting comments and questions that helped steer my research efforts. Thank you, Prof. Hinrichs!

My second advisor, Dr. Melanie Bell from the Anglia Ruskin University in Cambridge, UK, who despite her very busy schedule made time to thoroughly comment on my paper drafts. I very much enjoyed our exchange of ideas whenever we met at conferences or in Tübingen. Thank you, Melanie!

The third reader of my thesis, Prof. Dr. Gerhard Jäger, for gracefully accepting to read my PhD thesis on short notice, and for his thoughtful review.

Prof. Dr. Tilman Berger and Prof. Dr. Fritz Hamm, who were also part of my defense committee, for taking the time to read my dissertation.

I benefited immensely from being part of one of Prof. Hinrichs' research projects - the A3 project of the SFB 833. This is where I first encountered compounds[1] - which would become my topic of interest - and also where I had ample opportunities to practice both my coding and my writing skills.

The A3 project also gave me the chance to work with wonderful colleagues - in chronological order: Jianqiang Ma, Dr. Yannick Versley, Christina Hoppermann, Dr. Verena Henrich, †Dr. Heike Telljohann, Daniil Sorokin and Patricia Fischer. It also gave me the opportunity to supervise young, talented students assistants: Nadine Balbach, Tabea Sanwald, Kathrin Adlung, Edo Collins, Neele Witte, Erik Schill, Sebastian Pütz and Tobias Pütz. Thank you, all! I've learned a lot from each and every one of you. Dr. Daniël de Kok, the co-PI of the A3 project, deserves a special mention - his enthusiasm for well-done research and for teaching are contagious, and I'm glad I was around him for enough time to catch this "virus". Thank you, Daniël!

[1] in the scientific sense, of course - I was already using compounds in my day-to-day life in Germany - avoiding them is not really possible.

My colleagues at the "Seminar für Sprachwissenschaft": Marie Hinrichs, Dr. Thorsten Trippel, Dr. Chris Culy, Dr. Scott Martens, Dr. Dörte de Kok, Wei Qiu, Reinhild Barkey, Jochen Saile, Dr. Çağrı Çöltekin, Yana Strakatova, Alexandr Chernov, Ben Campbell, Dr. Claus Zinn, Roberta Toscano, Lea Caspar. Thank you, all, for understanding and actively fueling my appetite for compounds with plenty of examples - particularly during our lunch discussions. And Cornelia Stoll, Beate Starke and Petra Burkhard, who helped me keep afloat in the administrative ocean.

My high-school teachers for computer science, Oana Butnărașu and Alina Scutaru, who taught me to "think in code". My math teacher, Dorel Luchian, who taught me that math is more about pausing and thinking than about diligently (but mindlessly) working through the exercises. And my physics teacher and class master who introduced me to science fiction by lending me books by Isaac Asimov and Frank Herbert. These books sparked in me an interest in robots that lead me to eventually study computer science in my bachelors and computational linguistics in my masters. And they have fueled my hope that someday humans will make computers understand human languages - a goal that I've also tried to bring a small contribution to through my PhD thesis.

My supervisor for my B.Sc. and M.Sc. theses, Prof. Dr. Dan Cristea from the Faculty of Computer Science in Iași, Romania, who encouraged me to follow an academic career, channeling my interests related to artificial intelligence and robots.

Părintele dr. Nicolae și doamna preoteasă Adina, care ne-au primit pe mine, Emanuel și Maria cu brațele deschise și m-au sprijinit, m-au încurajat și mi-au fost alături în faptă și în rugăciune.

Parohienii comunității ortodoxe române a parohiei Sf. Gheorghe, care au făcut din Tübingen o a doua casă pentru mine.

Părintele Vasile și doamna preoteasă Elena, care mi-au dăruit un soț minunat și o minunată familie extinsă care m-a susținut mereu - Irinel, Irina, Elisabeta, Gabriel, Sofian, părintele Nicolae, doamna preoteasă Cătălina, Ecaterina și Teodor.

Surorile mele, monahia Paisia și Cristina, care m-au răsfățat tot timpul ca sora mai mică, și famililei căpătate prin intermediul lor - obștea de la mănăstirea Cormaia și Cornel, Cezara și Matei.

Părinții mei, Gheorghe și Paraschiva, pentru nețărmuita lor dragoste și grijă.

Soțul meu, Emanuel, care mi-a fost ajutor, redactor, critic și comentator. Alături de tine greul e mai ușor. Ești punctul meu de sprijin în încercarea de a urni Pământul.

Fiica mea Maria, care mi-a umplut fiecare zi de bucurie și a așteptat cu multă răbdare ca mama să termine 'teza'. E gata teza, Maria, hai să ne jucăm!

The Free Resources During my PhD research I often found myself in need of knowing more about different subjects. And I was very lucky to come across open access learning materials made available by awesome instructors, who made it possible for me to learn or refresh my knowledge at my own pace. A big thanks to all of you!

- the Coursera *Machine Learning* course taught by Andrew Ng[2]
- the Coursera *Neural Networks for Machine Learning* course taught by Geoffrey Hinton
- the Khan Academy course on *Linear Algebra* taught by Salman Khan[3]
- the *Machine Learning* course taught by Nando de Freitas at UBC [4]
- the Coursera *Learning How to Learn* course taught by Barbara Oakley and Terrence Sejnowski [5]
- the machine learning blog posts of Chris Olah[6]

The code developed in this thesis used Torch[7], an open-source machine learning library maintained by Ronan Collobert, Clement Farabet, Koray Kavukcuoglu and Soumith Chintala, but sustained by a larger community of machine learning practitioners and enthusiasts. The Python machine learning ecosystem - including *scikit-learn*, *numpy*, *matplotlib* - was of great help for many scripts and utilities around the main code. The thesis itself was written in LaTeX, using the templates made available for free by the Department of Electrical Engineering at the University of Cambridge[8]. This thesis could not have its current form without the many contributions to all these open source projects - and I thank each of the developers for their time and effort.

[2] https://www.coursera.org/learn/machine-learning

[3] https://www.khanacademy.org/math/linear-algebra

[4] https://www.youtube.com/watch?v=w2OtwL5T1ow&list=PLE6Wd9FR--EdyJ5lbFl8UuGjecvVw66F6

[5] https://www.coursera.org/learn/learning-how-to-learn

[6] http://colah.github.io/

[7] http://torch.ch/

[8] http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/ThesisStyle/

Summary

The central topic of this thesis are composition models of distributional semantics and their application for representing the semantics of German and English nominal compounds. Composition models are mathematical transformations that, given a compound like *Apfelbaum* 'apple tree', can be applied to the vector representations of *Apfel* 'apple' and *Baum* 'tree' to obtain a vector representation for the compound *Apfelbaum* 'apple tree'. The new composed representation is deemed appropriate if it is similar to the representation of *Apfelbaum* that can be directly learned from large corpora using distributional methods.

The thesis is structured into eight chapters. The first four chapters introduce compounds from a linguistic perspective (Chapter 1), present a review of annotation schemes for nominal compounds and introduce a new hybrid annotation scheme (Chapter 2), introduce neural networks and how to represent words via numerical features (Chapter 3) and detail the distributional representation of words (Chapter 4).

Existing composition models of distributional semantics are reviewed and evaluated in Chapter 5. Chapter 5 also introduces three new composition models: `addmask`, `wmask` and `multimatrix`, that aim to improve over existing composition models either though a more efficient parametrization (`*mask`) or by promoting parameter reuse across different, but semantically similar words (`multimatrix`). The results show that composition models are able to construct meaningful composed representations for 81.8% of the German test compounds, and 78.03% of the English test compounds. In Chapter 6 composed representations are shown to be a useful indicator when investigating non-compositional (lexicalized) compounds. For example, when modeling a compound like *Tigerauge*, 'tiger eye', composition models will produce a composed representation that corresponds to the literal interpretation of the compound - the *eye of a tiger*. This vector, however, is dissimilar to the distributional vector learned directly from the corpus which captures the lexicalized meaning of *semi-precious stone*.

In Chapter 7 composed representations prove to be the best features for classifying compounds in terms of their semantic relations in setups where simplex words and compounds have representations of the same length. Further analyses show also that some of the modifier information is discarded during the composition process and that extrinsic evaluations tasks such as the semantic classification task are necessary for assessing and improving the quality of the composed representations.

Chapter 8 concludes by emphasizing the main contributions of the thesis and sketching directions for future contributions.

Table of contents

Chapter 1

Introduction

> *Some German words are so long that they have a perspective. Observe these examples:*
>
> *Freundschaftsbezeigungen.*
> *Dilletantenaufdringlichkeiten.*
> *Stadtverordnetenversammlungen.*
>
> *These are not words, they are alphabetical processions. And they are not rare; one can open a German newspaper at any time and see them marching majestically across the page, - and if he has any imagination he can see the banners and hear the music, too. They impart a great thrill to the meekest subject. I take a great interest in these curiosities.*
>
> - Mark Twain, *The Awful German Language*

Compound nouns, the 'curiosities' that Mark Twain took a great interest into in 1880, have continued to be of great interest to linguists and, more recently, computational linguists. And not just the German ones: Jackendoff (2016) refers to an example from Gleitman and Gleitman (1970), *an inflectional morphology instruction manual software programming course*[1], which could be understood by students in Introductory Linguistics as long as it was introduced piece by piece, each connection made explicit. *"But if it were presented as a whole to a naive class, few would get it."* (Jackendoff, 2016:17), cf. Gleitman and Gleitman (1970).

This thesis 'takes a great interest' in nominal compounds in an attempt to make them comprehensible for the modern day 'naive class' - the computer programs for natural language processing. Its aim is practical: to build vector-based, composed representations of compounds. Its motivation is also practical: natural language processing (NLP) tools face a deluge of compound words similar to the one described by the creator of Wortwarte (Lemnitzer, 2007) - see Section 1.2.

[1] The corresponding syntactic paraphrase is *a course in programming the software that accompanies manuals that teach inflectional morphology*, Jackendoff (2016).

Coverage can be seen as the 'Achilles' heel' of modern NLP. Recent research (Ballesteros et al., 2015; Ling et al., 2015) has shown that incorporating character-based information about words results in improved performance for tasks like parsing and part of speech tagging, particularly on out-of-vocabulary (OOV) words. If a word is out-of-vocabulary, i.e. is not part of the subset of words that a particular tool has a representation for, then none of the useful generalisations that the tool has captured during training can be applied.

New compounds are frequently OOV words, particularly in German. Computer algorithms need to be able to represent and interpret new compounds at the same rate that these are coined.

Composition models offer exactly this prospect - of being able to represent a compound in terms of its constituent parts, and the way they are combined. Their main advantage is that they require as input only the representations of the individual words that make up the compound. Consider for example the compound *Jubiläumskäse* 'jubilee cheese', which names a cheese variety manufactured for a special anniversary of a cheese factory - i.e. 50 years of production. This compound appears only once in the 10-billion-token German corpus described in Section 5.3.2. Due to its low frequency, *Jubiläumskäse* will be an OOV for many NLP tools. However, the words *Jubiläum* 'jubilee' and *Käse* 'cheese' are frequent enough to have representations, and can therefore be used to construct a representation for the compound as a whole.

Because they are just above the word level, compounds are an ideal test case for understanding how the representations of multiple words should be combined into a single unified representation before moving on to the representations of more complex phrases and, ultimately, entire sentences.

Noun-noun compounds in German and English were chosen as exemplary constructions for the analysis. The choice was motivated in part by the fact that they are the most frequent word construction pattern both in English (Plag, 2003:145) and German (Fleischer and Barz, 2012:117). Another rationale was that much of the work on the semantic interpretation of compounds described in Chapter 2 focuses on identifying the internal relation of noun-noun compounds. This meant that the quality of the composed representations could be evaluated through the lens of the semantic interpretation.

Much of the research on compound nouns (Levi, 1978; Lauer, 1995; Rosario and Hearst, 2001; Girju et al., 2005; Ó Séaghdha, 2008; Tratz, 2011) revolved around finding a set of labels that could be used to categorize compounds. Spärck Jones (1983) equates interpreting compound nouns with providing a meaning representation for them. This thesis combines both ideas: it develops an annotation inventory for noun-noun compounds in Chapter 2 and composition models for representing nominal compounds as vectors from distributional semantics in Chapter 5. The composed representations prove their worth when investigating lexicalized compounds in Chapter 6 and when used as features for classifying semantic relations in German and English compounds in Chapter 7. The next sections contain a chapter by chapter overview of the thesis, and a short introduction to what is a compound from a linguistic perspective.

1.1 Chapter Guide

The first part of **Chapter 2** is a literature survey covering the annotation of semantic relations in noun compounds both from a theoretical and from a computational perspective. The second part of the chapter introduces a newly hybrid annotation scheme that can be used to annotate compounds with a semantic relation (a property) and a preposition. The last section evaluates the proposed annotation scheme via an inter-annotator agreement study.

Chapter 3 covers the basics of neural networks and explains some of the characteristics that make them a good fit for modeling compound semantics. The theme of the second part of the chapter is creating feature representations for natural language processing. Several representation possibilities are described, with an emphasis on their generalization capabilities.

Chapter 4 introduces the ideas behind distributional semantics and surveys several approaches for building distributional representations for words.

Chapter 5 evaluates several existing composition models at the task of creating composed representations for compounds. The drawbacks of existing models are addressed by three new composition models. `multimatrix`, one of the newly introduced models, is shown to produce the best composed representations both for German and for English compounds.

Chapter 6 presents several investigations related to lexicalized German nouns and shows the potential of composed representation as a means for identifying lexicalized compounds.

Chapter 7 uses composed representations as features for identifying the semantic relations in German and English compounds. Composed representations are shown to be better than the original compound representations and than representations obtained via the addition of the constituent vectors. However, it also shows that some of the modifier information is discarded during the composition process, and that combining the modifier and the composed representation improves the classification.

Chapter 8 contains an overview of the contributions of the thesis and an outlook into future directions.

1.2 What is a Compound?

A **compound** is *"a lexical unit made up of two or more elements, each of which can function as a lexeme independent of the other(s) in other contexts"* (Bauer, 2001). *Chocolate cake* illustrates the definition: it is a new lexical unit, and its parts, *chocolate* and *cake* are frequently encountered separately. Adding more elements to a compound, as in *chocolate cheesecake recipe*, is a recursive process: the compound *cheesecake* is first modified by *chocolate* to form *chocolate cheesecake*, which in turn modifies *recipe* to obtain *chocolate cheesecake recipe*. Compounds have therefore a binary structure where the elements can in turn be compounds.

As the previous example illustrates, the left element will usually modify the right element (in English and German). Compounds have a **modifier-head structure** (Plag, 2003:135). The **head** of a compound is usually its right element, while the left element is called the **modifier**. The term **constituent** will be used throughout this thesis to refer to either of the elements of a compound. A compound typically inherits its semantic and syntactic characteristics from its head. Semantically, a *chocolate cake* is a type of *cake* - the compound will usually denote a subset from the set denoted by the head. Syntactically, *cake* is a noun and the resulting compound - *chocolate cake* - is also a noun. Both in English and German compounds inherit their gender and plural form from the head noun - *die Schauspielerin* 'the actress' is a feminine noun, and so is *die Filmschauspielerin* 'the film actress'.

Compounding, the productive word formation process that gives rise to compounds, places little restrictions on the syntactic category of the elements that combine: noun-verb (*brainwash*), noun-adjective (*knee-deep*), verb-noun (*pickpocket*), verb-verb (*stir-fry*), adjective-noun (*greenhouse*), adjective-verb (*blackmail*), adjective-adjective (*light-green*) and preposition-noun (*afterbirth*) combinations[2] are all possible combinations in English, although not all are equally productive (Plag, 2003).

The investigations in this thesis focus on *nominal compounds*, i.e. compounds where the head is a noun and the modifier can be either a noun, a verb or an adjective, with a particular emphasis on noun-noun compounds.

Noun-noun compounds result from the combination of two nouns. If the nouns that are being combined are simple, i.e. they are not derived from any verb, then the compound bears the name **root compound**. In contrast, compounds where the head noun is derived from a verb are called **synthetic compounds** in English (Plag, 2003) and **Rektionskomposita** 'valence compounds' in German (Eisenberg, 2013). *Apfelbaum* 'apple tree' and *Apfelpflücker* 'apple picker' illustrate the two classes: 'apple tree' is constructed from the nouns 'apple' and 'tree' and is a root compound. In the case of 'apple picker', the head 'picker' is derived from the verb 'to pick' with the suffix '-er'. Moreover, the noun 'apple' is the accusative object of the verb 'pick', making 'apple picker' a synthetic compound.

When compounds name a subset of the things denoted by the head noun, like in *chocolate cake*, they are called **endocentric compounds** in English and **Determi-**

[2]All examples are from (Plag, 2003:144).

nativkomposita 'determinative compounds' in German (Klos, 2011). Plag (2003) explains that the semantic head of such compounds is 'inside', as suggested by the neoclassical element *endo* 'inside' of the term endocentric.

An *Angsthase* lit. 'fear rabbit', however, is no *Hase* 'rabbit': it is a frightened person, the type described by the English *scaredy-pants* - which, as it happens, is also not a type of pants. Such compounds are called **exocentric/possessive compounds** in English, **Possessivkomposita** in German and **bahúvrīhi**[3] compounds according to the Sanskrit terminology (Bauer, 1983). Here the semantic head refer to something outside of the compound - it is usually a person or an animal that possesses the characteristic named by the modifier. As Plag (2003) notes, however, exocentric compounds still derive their syntactic characteristics from their rightmost component, like the endocentric compounds. It is only the semantic head that deviates, in many cases by means of a metaphor or a synecdoche.

Musikerfreund 'musician friend' is both a friend and a musician. When both modifier and head contribute equally to the meaning, the compound is called a **copulative compound** in English, a **Kopulativkompositum** in German and a **dvandva**[4] **compound** in Sanskrit.

Noun-noun compounds are "a salient feature of Sanskrit and Germanic word formation, whereas in other Indo-European language families like Latin and the Romance languages, Celtic or Slavic, they are marginal and replaced by syntactic phrases" (Kastovsky, 2009). Often encountered are patterns of the form noun-preposition-noun, e.g. the Romanian *mașină de spălat*, 'washing machine' lit. 'machine for washing' or denominal adjective-noun, e.g. *comerciant stradal* 'street merchant' which is formed though the combination of the noun *comerciant* 'merchant' and the denominal adjective *stradal* derived from *stradă* 'street'.

Compounds can contain **linking elements**, which are called **Fugenmorpheme** or **Fugenelemente** in German. The English list is rather thin, with only *-s* appearing as a linking element in compounds like *sport**s**man*. In German compounds, however, linking elements are far more pervasive: **n** (*Blume**n**vase* 'flower pot'), **s** (*Zweifel**s**fall* 'case of doubt'), **ns** (*Glaube**ns**frage* 'question of faith'), **e** (*Pferd**e**wagen* 'horse carriage'), **er** (*Kind**er**garten* 'kindergarten'), **en** (*Held**en**mut* 'bravery'), **es** (*Sieg**es**wille* 'desire to win') and **ens** (*Schmerz**ens**schrei* 'cry of pain') are all possible linking elements according to (Eisenberg, 2013:226). The elision of the final vowel in compounds like *Erdöl* - **Erd**e**öl* is also considered to be a type of linking element (Fleischer and Barz, 2012).

Meyer (1993) gives as example four compounds with the head *Käfig* 'cage': *Schaf**s**käfig* 'sheep cage' , *Löwe**n**käfing* 'lion cage', *Alligatorkäfig* 'alligator cage' and *Krokodil**s**käfig* 'crocodile cage'. All four examples refer to a cage intended for the animal denoted

[3]The term *bahúvrīhi* itself is a *bahúvrīhi* compound, formed from the Sanskrit words *bahú* 'much' and *vrīhi* 'rice'. The meaning relates neither to *much* nor to *rice* - but to a person that has a lot of rice, i.e. a rich person (Bauer, 2009)§x.

[4]*dvandva* is Sanskrit for 'pair' and is a reduplication of the Sanskrit word *dva* 'two'. Source: `https://www.merriam-webster.com/dictionary/dvandva`

by the modifier. The inclusion of a linking element in each of the cases seems rather arbitrary: in *Schaf**s**käfig* the modifier has the linking element *-s* - a genitive-marking morpheme, *Löw**en**käfig* has the linking element *-n* which corresponds to the plural-marking morpheme for the noun *Löwe*, *Alligatorkäfig* has no linking element, while the almost synonym form *Krokodilskäfig* has again the genitive marking *-s* as a linking element. The linking element seems to be semantically justified only in cases where the modifier denotes a collection of objects, i.e. *Studententreffen* 'students meeting'. However, as Meyer (1993) points out, the fact that semantically the modifier is a collection does not necessarily result in the plural linking element being used in every case: e.g. a *Kartoffelverkauf* 'potato sale' presupposes the setting of several potatoes, not just one, but the form **Kartoffelnverkauf* 'potatoes sale' is not used. This is why linking elements are generally considered to be 'semantically empty' (Fleischer and Barz, 2012:186).

The meaning of compounds evolves over time. All compounds start off compositionally - i.e. when first uttered, the speaker has to make sure that the constituents and the context supply enough information for the correct interpretation of the compound. However language - and the world it describes - evolves over time. What used to be considered common knowledge from a synchronic perspective gradually becomes marginal from a diachronic perspective. Compounds like *Blümchenkaffee* 'very thin coffee', lit. 'little flower coffee' evoke an 18th century reality where a coffee was served in expensive Meissener porcelain cups. The compound captures the stark contrast between the very expensive porcelain and the avarice of using only a scant amount of coffee - which made it possible to see the small floral ornament on the bottom of the cup. The present day speaker, however, needs more explanations to understand the semantics of such compounds.

Bauer (1983) describes the different stages that a compound goes through from the moment it is first coined as follows. First, the compound is a **nonce formation**. It is created in order to name something, and the same name might be given independently by several people. Characteristic for this stage is the high potential ambiguity of the semantics of the compound: if the first constituent has four meanings and the second one has two meanings, the new compound could have any of the eight meanings obtained by combining the constituent meanings. Illustrative in this respect are the range of interpretations reported by Ryder (1994) for the new compounds she had presented her study participants (e.g. a *llama-pen* was paraphrased as 'a pen where llamas are kept', 'a fountain pen made out of llama fur', 'a pen with a llama on it', 'furry fountain pen', 'a pen shaped as a llama', etc.).

A second stage in a compound's formation is the **institutionalization** stage, in which the nonce formation starts being adopted by the wider community as a known lexical item. The potential ambiguity of compounds in this stage is restricted - the interpretation of the compound is fixed to a subset of the possible meanings, in many cases only one. A *sandwich box* is usually interpreted as a 'box to store/carry sandwiches', although it might also mean 'a box shaped like a sandwich', 'a box that snugly holds the content like a sandwich', 'a box with multiple layers of cardboard and

foam', etc. However, none of these other interpretations will usually come to mind if the context does not warrant a reinterpretation of the institutionalized meaning.

In a third stage, a compound takes on a form or meaning that is not consistent with the application of productive rules. Compounds in this stage are **lexicalized**. Between the different types of lexicalization described by Bauer (1983), the **semantic lexicalization** is in particular applicable in the case of compounds. In German for example many names of plants and animals are lexicalized compounds, with a history that is sometimes hard to track: e.g. *Meerkatze* 'guenon', lit. 'sea cat' or *Geißfuß* 'ground elder', lit. 'goat's foot'. Lexicalization is, however, not restricted to names of plants and animals. For compounds like *Schneebesen* 'whisk', lit. 'snow broom' and *Windbeutel* 'profiterole', lit. 'wind bag' it is unlikely that a speaker that does not know what they mean will come up with the correct interpretation.

It should be noted that a subset of the compounds in the nonce formation stage are never meant to make it past the first stage. Downing (1977) mentions the compound *apple-juice seat* and explains its meaning as 'the seat in front of which a glass of apple-juice had been placed.' Lemnitzer (2007) refers to *Bierdeckelsteuer* 'beer mat tax', a word that brings a vision of a *tax return* form that is so compact it fits on a coaster. Such **deictic compounds** can be used to name things in a particular context. However, they cannot be interpreted without the context that has called for their use, and the relationship they express is too infrequent or too specific to warrant the further use of the compound by a larger community of users.

Lemnitzer (2007) refers to compounding as the (by-far) most frequent method for creating new words in German. His ∼15 daily additions to the *Wortwarte*[5] are chosen from a pool of 1000-2000 words that are not part of the already stored words. The words come from a selection of online newspapers and websites. He estimates that around 10% of these are spelling errors, and then there are some names, and some deictic compounds - but the remainder are newly coined words. The *Wortwarte* is a very telling account of how productive this composition process it, and how many new words one can expect a natural language processing system to have to deal with each new piece of written text.

[5] `http://www.wortwarte.de/`

Chapter 2

The Semantics of Nominal Compounds

Buying a box of breakfast cereal in a supermarket is not an easy task. There are a multitude of characteristics that need to be taken into account in order to make an informed choice. Should it be oats, wheat or maybe barley? Should it be thickly rolled, flaked, powder or in shapes? Should it contain dried fruit or nuts? Or maybe chocolate and honey? There is a multitude of choices on the market, and the range of products is constantly expanding[1].

Selecting an annotation scheme has much in common with choosing a box of breakfast cereal: there are many annotation schemes that have been proposed over the years, that come in various shapes and sizes, and were build with different goals in mind. Some advocate for the existence of 'an infinite number of potential compounding relations' (Downing, 1977), others list a restricted set predicates that 'are recoverably deleted in the process of complex nominal formation' (Levi, 1978). Some are developed in a bottom-up fashion, based on compounds that occur in corpora (Warren, 1978), while others use a top-down approach, where the semantic relations are first defined and then a set of compounds is annotated with them (Nastase and Szpakowicz, 2003; Girju et al., 2005).

This chapter starts off, in Section 2.1, with a **literature survey** covering the annotation of semantic relations in English and German noun compounds, both from a theoretical and from a computational perspective. However, in all fairness, this chapter only scratches the surface of the literature on the semantics of compounds, and it is impossible to do justice to all the proposals from the vast literature on the subject. Most of the work presented in what follows is reflected in one way or another in **the new annotation scheme** developed as part of the thesis, to be presented in Section 2.2. The survey is dotted with remarks that connect the main points drawn from the existing literature to the new annotation scheme and the corresponding annotated dataset, presented in Section 7.1.1.

[1] A bit of trivia: in America, the number of types of cereal has seen a substantial growth: from 160 in 1970, to 340 in 1988 and to 4,945 in 2012 (Aichner and Coletti, 2013).

The decision to develop a new 'annotation brand', despite the multitude of choices available, was influenced by several factors. The goal of this thesis is to learn how to represent compounds with the help of computers. The idea is to supplement the directly observable information a computer receives about compounds, i.e. the compound constituents, with as much information as possible about the non-apparent semantics of the compound. To maximize the supplied amount of semantic information, a new, hybrid annotation scheme was developed. It uses both semantic relations and prepositional paraphrases for rendering the compound-internal semantics. Creating a new annotation scheme made it possible to adjust its granularity to fit the task at hand: computers work better with rule-like distinctions, thus a fine-grained annotation scheme is a better choice for capturing such distinctions. Ultimately, the work on the annotation scheme meant a close interaction with a variety of compounds, and building an intuition about the different nuances of compounding as a linguistic process. A linguistic process that children grasp when they are between 2 and 6 years old (Nicoladis, 2006), and continue using throughout their whole life. Can a computer learn to interpret compounds like a preschool child does?

2.1 Analyzing Compound Semantics: Challenges and Existing Approaches

Downing (1977) investigates the creation and use of English noun compounds in an experimental setting. Human subjects are asked either: (i) to create new compounds, by naming specific objects from a drawing, (ii) to interpret novel noun-noun compounds or (iii) to judge the appropriateness of a given set of descriptions for a novel compound.

The discussion is focused on compounds that are generally usable and interpretable, as opposed to deictic compounds which can only be interpreted in context. The compound *apple-juice seat* is used to illustrate deictic compounds, where the meaning of the compound, i.e. *the seat in front of which an apple-juice has been placed*, makes sense in the context of use, but would not be generally interpretable. At the same time, (Downing, 1977:820) makes a point about the difficulties in interpreting existing nominal compounds:

> 'A compound may be highly transparent semantically when it is coined; but once it has been accepted by the community as a conventionalized name, it may come to be as arbitrary as any monomorphemic name.'

This observation justifies her decision to study the semantic relations in **novel** nominal compounds, as opposed to identifying the relation in **institutionalized**[2] compounds, as in previous studies. According to Downing (1977), identifying the semantic relation in such institutionalized compounds is more challenging because of historical and cultural processes that might have altered the initial meaning of the compound.

[2]The term **institutionalized** is used here in the sense defined by Bauer (1983) to mean 'compounds that have been accepted by the community as a conventionalized name for a particular concept'.

Downing (1977) draws several conclusions after analyzing the compounds and the compound interpretations produced by the human subjects. A first conclusion is that a necessary condition for creating a new compound is the existence of a relationship between the compound constituents. Compounds cannot be formed because of the lack of a relation that connects the constituents: e.g. *cousin-chair* cannot mean *a chair reserved for non-cousins*. This argument also leads to the observation that entities with no typical interaction patterns (e.g. *bird* and *door*) are unlikely to form a compound (e.g. **bird door*).

A second take-away point is that the range of possible new compounds produced by the human subjects and the diversity of their semantic relations points to the existence of an **infinite number of potential compounding relations**. However, Downing (1977) agrees that one could make a list of the most common relations, which should include: WHOLE-PART (e.g. *duck foot*), PART-WHOLE (e.g. *pendulum clock*), HALF-HALF(e.g. *giraffe-cow*), COMPOSITION (e.g. *stone furniture*), COMPARISON (e.g. *pumpkin bus*), TIME (e.g. *summer dust*), PLACE (e.g. *Eastern Oregon meal*), SOURCE (e.g. *vulture shit*), PRODUCT (e.g. *honey glands*), USER (e.g. *flea wheelbarrow*), PURPOSE (e.g. *hedge hatchet*) and OCCUPATION (e.g. *coffee man*)[3].

As a third point, Downing (1977) lists several factors that can influence the suitability of a particular relationship in a specific context:

- **the semantic class of the head noun**. Particular semantic relations are preferred by particular classes of head nouns: e.g. OCCUPATION for humans; APPEARANCE and HABITAT for animals and plants; COMPOSITION, ORIGIN and LOCATION for natural objects; PURPOSE for synthetic objects.
- **the predictability of the relationship between the two constituents**. Compounds where the set of entities denoted by the modifier is the same as the one denoted by the head, like **book-novel*, are deemed implausible; similarly implausible are compounds where the modifier denotes a subset of the entities denoted by the head (e.g. *truck-vehicle*)[4], or where the head implies the modifier (e.g. **head-hat*).
- **the permanence of the relationship**. The majority of novel compounds coined by the human participants in Downing (1977)'s studies referred to a relationship that was habitual or generic in nature. This shows that while it is possible to create a compound based on a conjunctural relation, like *apple-juice seat*, most compounds are based on habitual or generic relations.

Downing (1977) justifies the existence of such constraints by referring to the information exchange taking place between the speaker and the hearer. The speaker must make use of the available informational resources and select a modifier that adds to the information provided by the head, otherwise using the head alone should suffice

[3] examples from Downing (1977).

[4] although for heads denoting plants or animals this type of relation can be used - e.g. *salmon fish*, *pine tree*, *banana plant*.

(e.g. *head-hat* does not bring anything new to the hearer, a *hat* is typically worn on the *head*). Additionally, the speaker must guarantee the interpretability of the compound, by making it possible to derive the meaning of the novel compound from the meaning of its constituents and the context of use. The use of habitual relations as a basis for compounding would then facilitate the interpretation of the compound, as habitual relations are likely to be part of the hearer's mental model for the compound constituents. In contrast, compounds based on a temporary relation depend much more on contextually-supplied information and tend to become uninterpretable if taken out of context. As a last desideratum, the speaker should strive to use a novel compound for denoting a relevant category. Given the frequent use of compounds as a naming device, the speaker should try to denote a category that, if not yet conventionalized, should at least be considered a worthy candidate for being conventionalized.

Downing (1977)'s conclusion is that even if any relationship could be used as a basis for forming a compound (given the appropriate context), compounds generally tend to be based on relationships that are perceived as permanent or habitual.

The new annotation scheme proposed in Section 2.2 starts off from very similar premises to those put forth by Downing (1977). First, it does not claim to include all the possible compound-internal semantic relations, but only those that were identified when annotating the chosen subset of data. It includes, however, most of the semantic relations labeled as common by Downing (1977), thus confirming her analysis. The factors that influence the suitability of a particular relationship will be more closely analyzed using the dataset of annotated German compounds in Section 7.1.1. The analysis will focus on finding out if there are strong preferences for particular semantic relations given a particular head noun or a particular modifier.

Levi (1978) proposes a study of English complex nominals. She identifies complex nominals as consisting of three subgroups: nominal compounds (e.g. *apple cake*), nominalizations (e.g. *film producer*; the noun *producer* is derived from the verb *to produce*) and noun phrases with non-predicating adjectives (e.g. *electrical engineer*; the adjective is considered non-predicating because the construction cannot be paraphrased as **an engineer who is electrical*). Levi (1978) justifies including the last category because of the parallelism between compounds and noun phrases including non-predicating adjectives (e.g. *language difficulties* and *linguistic difficulties*). Her claim is that non-predicating adjectives are derived from underlying nouns, and therefore behave like a noun when they are part of complex nominals.

Levi (1978) argues that complex nominals should not be treated as strictly idiosyncratic constructions that need to be individually listed in the lexicon. Rather, complex nominals are constructions that show clear syntactic and semantic regularities, which make possible the interpretation of newly coined constructions. She acknowledges, however, the existence of a certain level of ambiguity regarding the exact interpretation that a construction should receive. This interpretative ambiguity can make even compounds with an institutionalized interpretation to be analyzed differently when the context licenses the alternate interpretation.

2.1 Analyzing Compound Semantics: Challenges and Existing Approaches

Complex nominals are characterized by Levi (1978) as a "very successful means of compressing syntactic and semantic information into a highly compact form". Complex nominals are seen as "naming devices that pick out the relevant categories of the speaker's experience". The "naming device" perspective explains why complex nominals are such a productive category and also why each such construction must necessarily start off being semantically transparent and may become institutionalized or even completely opaque along its history of use. Furthermore, Levi (1978) argues that from a syntactic perspective complex nominals should be considered nouns and not noun phrases.

Levi (1978) approaches the semantics of complex nominals stating that

> '... the larger part of the semantic relationships that may be associated grammatically with the surface structures of complex nominals (CNs) can be expressed by a small set of specifiable predicates that are recoverably deleted in the process of CN formation. This set is made up of nine predicates: CAUSE, HAVE, MAKE, USE, BE, IN, FOR, FROM and ABOUT. These predicates, and only these predicates, may be deleted in the process of transforming an underlying relative clause construction into the typically ambiguous surface configuration of the CN.'

and goes on to specify more details about each of the proposed predicates:

- USE (e.g. *steam iron*) refers to **use** in an **instrumental** sense (*iron* using *steam* to function), not in an agentive sense (e.g. *professor* using *books*).
- IN (e.g. *field mouse*, *morning prayers*) has a **locative** meaning and can refer both to spatial and to temporal locations; the naming of the predicate does not imply that it applies only to constructions paraphrased by the preposition *in* - the name is a symbolic convention; the predicate refers to constructions that can be paraphrased via *in* or other location-pointing prepositions, i.e. *on* or *at*.
- CAUSE (e.g. *tear gas*, *tears* caused by *gas* - CAUSE_1; also *birth pains*, *pains* caused by *birth* - CAUSE_2); the predicate is used to label complex nominals expressing **causative** relationships; has two instantiations, CAUSE_1 and CAUSE_2, depending on the constituent expressing the cause; CAUSE_1 labels only a small set of compounds.
- HAVE (e.g. *apple cake*, *cake* that has *apples* - HAVE_1; *lemon peel* - *lemon* has *peel* - HAVE_2); expresses a **possessive** relation; HAVE_2 is very frequent, while HAVE_1 labels only a handful of constructions;
- MAKE. Several subgroups can be identified: (i) constructions like *honeybee*, where **make** stands for "physically producing, causing to come into existence"; (ii) *daisy chains*, where the modifier denotes a unit that makes up the group named by the head; (iii) complex nominals like *stone wall*, where the head describes an artifact and the modifier describes the material the artifact is made of; (iv) constructions like *student committee*, where the head noun names a human collectivity and the modifier names its members; just like CAUSE_1 and HAVE_1, there are far fewer instances labeled with MAKE_1 than with its counterpart, MAKE_2.

- BE. Has three subgroups: (i) where the modifier names the material of the head (e.g. *water drop*), (ii) where the modifier and the head are in a genus-species relationship (e.g. *pine tree*) and (iii) where there is a metaphorical link between the head and the modifier (e.g. *queen bee*).
- FOR (e.g. *nose drops*). Specifies a relation of **intent** or **purpose** between the constituents. Complex nominals labeled with this predicate can be paraphrased as "X for Y". More precise paraphrases using a verb are generally possible (e.g. *nose drops* - *drops* for *decongesting* the *nose*; *headache pills* - *pills* for *reducing/relieving headache*; *clothing shop* - *sell*; *picture album* - *store*). Levi (1978) identifies the lexicon as the place where the extra information about the verb used to paraphrase each particular construction should be stored.
- FROM. The modifier denotes the **source** of the head; stands out as one of the categories with the most homogeneous set of modifiers: they are either natural sources which are used to extract/harvest the head (e.g. *olive oil*, *alligator leader*) or "places of origin" for the head (e.g. *country butter*, *sea breeze*).
- ABOUT. Has a large number of equivalent expressions in English: *concerned with*, *dealing with*, *pertaining to*, *on the subject of*, *on*, *about*, *over*; the head is usually an abstract noun/nominalization (e.g. *tax law*, *history conference*, *price war*); modifiers can be very diverse - anything that a *law*, *conference*, *war*, etc. might be about.

Levi (1978) takes the potential interpretation ambiguity to be something that people can generally cope with: for example if, in using the construction *musical talent* the speaker means '*talent* in *music*' (IN deletion) whereas the listener means '*talent* for *music*' (FOR deletion), the communication will not suffer from this slight misinterpretation. It is only when the difference in meaning diverges in a significant way that the communication has to suffer. She states that her theory is not meant to associate every construction with exactly one of the proposed predicates, but rather to specify the range of interpretations that a particular construction might receive. I.e., *chocolate bunny* might be analyzed both as BE and as MAKE_2 - *bunny* is *chocolate*; *bunny* is made of *chocolate*, and there are reasons to defend any of these analyses. However, natural language allows describing the construction in both ways. This suggests that complex nominals display an "indeterminacy of analysis" - for a subset of complex nominals there is not clear, principled way to choose one predicate over the other.

Levi (1978) reserves a special treatment for complex nominals containing nouns derived from verbs - i.e. nominalizations. As Ó Séaghdha (2008) observes, according to Levi (1978)'s annotation principles *history professor* and *history teacher* are assigned different categories because *teacher* is derived from the verb *to teach*. These syntactically motivated categories lead to similar concepts being sometimes assigned different categories, as illustrated by the previous example.

Although the predicates proposed by Levi (1978) are too general to provide the type of fine-grained distinctions needed for the computational processing, the new

annotation scheme does not claim to be able to remove all ambiguity just because it is more detailed. The type of ambiguity mentioned by Levi (1978) with regard to the compound *chocolate bunny*, where the same compound can be labeled in different ways, without affecting the interpretation assigned to the compound, is unlikely to be solved using the new annotation scheme. This genuine ambiguity derives from the fact that there are multiple ways to conceptualize the same compound, and depending on the conceptualization one can choose one label over the other. The discussion of the inter-annotator agreement study in subsection 2.2.8 will come back to this issue.

Another aspect from Levi (1978)'s annotation proposal that was incorporated into the new annotation scheme is the marking of the relations' **directionality**, depending on how the two constituents fill the relational slots (e.g. see above the difference between the compounds labeled with CAUSE$_1$ and CAUSE$_2$). What is meant through the directionality of a semantic relation in the new annotation scheme will be described in detail in Section 2.2.

Warren (1978)'s empirical investigations aim at discovering whether there is a limited number of semantic relations between the constituents of noun-noun compounds. Her analysis is based entirely on naturally-occurring compounds from corpora, an impressive undertaking for the time the thesis was written. She analyzes a dataset of 4566 compound types, extracted from a selection of texts in the Brown corpus[5]. Warren (1978)'s study is **comprehensive** in that she compiles the dataset by extracting all the compounds from the selected texts. Moreover, the compound analysis is **context-dependent**: the interpretations are chosen after analyzing the compound in its context of use. An analysis of the syntactic structure revealed that 58% of the dataset consisted of two-part compounds, and that for compounds with three or more constituents the left-branching structure is the most common.

The morphological analysis performed by Warren (1978) showed that 118 of the compounds in the dataset have the first noun inflected: in some cases, the inflected form is recognized as a plural (*sports center*), while in other cases the *-s* is closer to the German notion of *linking element*, with no clear rules as to when the inflection should appear (compare *statesman* and *fireman*). In some cases, the trailing *-s* is a way to avoid a sense ambiguity, like in *plains people*.

Warren (1978) uses the term **incomplete compounds** to refer to combinations where a member (in most cases the middle component) is left unexpressed but its presence is assumed by the semantic interpretation, e.g. *air(plane) strip*. Here the semantic relation connecting the constituents can only be specified by referring to the missing component: e.g. an *air strip* is a *strip* for *airplanes* to land on, and not a *strip* made of *air*. Such elliptical constructions generally appear for convenience reasons, and the resulting compound becomes a conventional name for the referent. They can also become patterns for constructing other compounds where the same component

[5]Corpus of American English, created at the Brown University in 1964, under the direction of W. Nelson Francis.

is deleted, e.g. *airport*, *aircrew*. For such compounds the initial meaning cannot be recovered without additional knowledge regarding the deleted material.

Compounds are analyzed by Warren (1978) using a **comment-topic** structure: the head represents the **topic** of the compound, while the modifier is used to express the **comment**. The comment's purpose is to further specify the topic by restricting its scope. This can be done either by classifying the topic or by identifying the particular topic that is considered in a given context. **Classifying comments** usually delimit a class, subgroup or type of the topic. The distinguishing feature can be either directly named by the comment (e.g. *stone wall*), or can be only indirectly implied (e.g. *bilge water*[6]). It is interesting to note that when the modifier is used as an indirect specification of a particular feature, its literal sense is replaced by the implied sense - e.g. *room* in *room temperature* resolves to the normal temperature of a room, which is around 18°-23°C.

Warren (1978) identifies six major types of semantic relations that compounds can express, and uses a set of **participant roles** to name them. In most cases, the participant roles may be reversed, leading to the same semantic relation being applied in the opposite direction.

1. A is something that wholly constitutes B, or vice-versa: SOURCE-RESULT (e.g. *leather shoe*), RESULT-SOURCE (e.g. *paste wax*), COPULA (e.g. *girl friend*).

2. A is something of which B is a part or feature or vice-versa: WHOLE-PART (e.g. *eggshell*), PART-WHOLE (e.g. *wheelchair*), SIZE-WHOLE (e.g. *22-inch board*).

3. A is the location or origin of B in time or space: PLACE-OBJ (e.g. *city lights*), TIME-OBJ (e.g. *morning train*), ORIGIN-OBJ (e.g. *seafood*).

4. A indicates the purpose of B: PURPOSE (e.g. *coffee cup*), GOAL-OBJ (e.g. *moon rocket*).

5. A indicates the activity or interest which B is habitually concerned with: ACTIVITY-ACTOR (e.g. *sportsman*).

6. A indicates something that B resembles: COMPARANT-COMPARED (e.g. *clubfoot*).

Each category is further subdivided into subcategories. Warren (1978) assigns features to the two constituents of the compound in order to make clear the delimitations between the different subcategories. For example, in distinguishing MATERIAL-ARTIFACT compounds like *clay bird*, the first noun is required to have the +Material feature, while the result should have the +Man-made and +Concrete features. Other features that are used in the classification are +Abstract, +Material, +Artifact, +Shape, +Group, +Animate, +Inanimate, +Human, +Body Part, +Animal, +Building, +Plant, +Area, +Time, +Place, +Natural, +Event, +Phenomenon, +Organization, etc.

[6]dirty water that collects inside the lowest internal portion of a ship's hull.

Warren (1978) also notes that **prepositional paraphrases** are particularly suitable for making apparent the compound-internal semantic relation. She lists the prepositional paraphrases that are usually associated with the proposed participant roles, e.g. SOURCE-RESULT - *of*: *student group - group of students*; PART-WHOLE - *of, with*: *clay soil - soil with clay*; PLACE-OBJ and TIME- OBJ: *in, at, on*; ORIGIN-OBJ - *from*; GOAL-INSTRUMENT - *for*; COMPARANT-COMPARED - *like*.

Warren (1978)'s study served as an inspiration for many features that are included in the new annotation scheme (see Section 2.2): the annotation of **elliptical modifiers** when the relation refers to an implied modifier rather than to the actual modifier; the annotation of relation **directionality** (also present in Levi (1978)'s work); the use of features like +Animal, +Human to delimit the semantic class of the constituents which can be annotated using a particular semantic relation. Warren (1978) is also one of the first to point at the regularities in the assignment of semantic relations and prepositional paraphrases, and to list the prepositions that are typically associated with particular semantic relations. The combination of the semantic relation and prepositional paraphrase annotation is one of the main features of the new annotation scheme.

Finin (1980) proposes a rule-based system for automatically deriving semantic interpretations for nominal compounds. In his system, words are represented as **frames** which contain **slots**, and each slot can in turn have multiple **facets**. The frames are organized into an abstraction hierarchy and are connected via the `ako` and the `instance` relations. The `ako` ('a kind of') relation points to the superconcept of a concept, while its inverse, the `instance` relation, links concepts to their sub-concepts or instantiations. For example, the frame for the concept `to-fly` includes the slots `ako`, `instance`, `agent`, `object`, `instrument`, `source` and `destination`. In addition to these local slots, the concept `to-fly` will also **inherit** slots from all its direct and indirect ancestor concepts. The facets of a slot are a method to describe the semantics of the slot by specifying, for instance, requirements (`$require` facet), preferred values (`$prefer` facet), default values (`$default` facet), typical values (`$typical` facet), the importance of the role for the concept as a whole (`$salience` facet) and whether the role is obligatory, optional, prohibited or dependent (`$modality` facet). For example, the `source` and `destination` slots of the concept `to-fly` could require that the matching element is a `city` or an `airport` (using the `$require` facet).

In his discussion about **conceptual modification** Finin (1980) compares two fundamentally different ways of identifying the compound-internal semantic relation intended by the speaker: a **concept independent** mode, where the relation is chosen from a fixed set of potential relations that stays the same for every compound and the **concept dependent** mode, where the set of relations is determined by the identity of the concepts and can therefore be different from one compound to the other. Finin (1980)'s own work takes the selection of the relation set to be concept dependent.

Finin (1980)'s system takes as input complete concept representations containing all the necessary frames, slots and corresponding facets (in his system these are manually

defined). The output is a list of possible semantic interpretations and their associated scores, with the highest scoring interpretation being considered the most probable. The system assigns interpretations based on a series of rules. Each rule defines the pre-conditions that have to be met for the rule to apply and the interpretations that are generated when the rule is applied. Finin (1980) defines several classes of rules:

- idiomatic rules, where the rule applies only if the constituents exactly match the ones specified by the rule. Idiomatic rules are defined for exocentric compounds and for compounds where the meaning cannot be easily recovered based on the interaction patterns of the constituents.
- productive rules, which define a general pattern with many possible instantiations.
- structural rules, which create a structural relationship between the modifying and the modified concepts and are particularly useful for analyzing compounds containing nominalized verbs.

The biggest emphasis is placed on the structural rules, for which the author identifies eight possible subtypes:

1&2. N_1 fills one of N_2's slots or N_2 fills one of N_1's slots: e.g. *magnesium wheel*, where the concept `magnesium` is considered to fill the `raw-material` slot of the concept `wheel`; *January flight*, where `January` is taken to fill the `time` slot of the concept `to-fly`

3&4. Thing + Role Nominal or Role Nominal + Thing: when the modified concept is a noun that refers to a slot/role in of an underlying concept. E.g. *cat food* is taken to be the `object` of a `to-eat` event where the `agent` is a `cat`. In the reverse direction, a *food bowl* is an `instrument` of `to-eat` with the `object` being `food`.

5&6. Specific + Generic or Generic + Specific, where one concept is the subconcept of the other, like *F4 plane* or *building NE43*.

7. N_1 be N_2, where the compound is at the same time an N_1 and an N_2, as in *woman doctor*.

8. Attribute transfer, when an attribute or property of the modifier is transfered to the modified concept, as in *iron will* or *elephant legs*.

Finin (1980)'s system is remarkable in that it underlines the need for a more comprehensive representation of concepts. Although he does not address the problem of mapping the surface form of a compound into the corresponding concept frames, he advocates for the need to develop good representations for polysemous words and for sense-disambiguating the constituents before the compound interpretation process. The use of underlying concepts allows the rules to access not only the semantics of the constituents themselves, but also the semantics of associated concepts, like it is

the case with *pilot* and *to fly*, while the attribute transfer rule presupposes access to a variety of prototypical concept attributes: *iron* is *strong*, *elephants* are *large*, etc. The saliency of a slot creates a ranking of the concept's slots, with the more salient slots having a higher probability of becoming the basis for compound formation. Another noteworthy aspect is the treatment of idiomatic compounds: they have dedicated rules that match the surface form directly to the interpretation, an approach which closely matches the treatment of idiomatic compounds proposed in this thesis, which will be presented in Chapter 6.

The annotation guidelines of the new annotation scheme contain restrictions or specifications of the type of constituents that can take part in a given relation. Even if the restrictions are not defined in a logical model, they do serve as a pre-filtering mechanism, designed to simplify the annotation process, given the large number of semantic properties proposed by the new annotation scheme. Consider for example the annotation of compounds with a head that denotes an animal: most of their compound-internal semantic relation will probably be of type HABITAT, or APPEARANCE. It is very unlikely for compounds with such a head to have a modifier connected via a TOPIC relation. This means that even if the annotation scheme contains many different relations, the semantic class of the head and/or modifier could be used to pre-filter the most likely semantic relations.

Ryder (1994) proposes a model for compound interpretation based on Langacker's **cognitive grammar** (Langacker, 1987) and ideas inspired by the **schema theory** as proposed in Rumelhart et al. (1986b). Her analysis of compounds is relevant to our explorations because she considers *"[...]the problem of compound interpretation from the point of view of a real person faced with a real noun-noun compound"*. Even as her goal is to make predictions about what types of patterns are used by human speakers and listeners when creating and interpreting compounds, the insights of her analysis can be readily transfered and applied to the computer-based interpretation of compounds.

Ryder (1994) distinguishes between three possible compound types that were also previously identified by Downing (1977): deictic, novel and institutionalized compounds. Ryder emphasizes the importance of having an appropriate context when learning the meaning of institutionalized compounds for the first time. She gives the example of a person, unfamiliar with the compound *milkman*, that first encounters the compound in the context *'I saw the milkman'*. Even if the person is aware of the individual meanings of the words *milk* and *man*, given only this impoverished context it is impossible to correctly guess what the semantic relation connecting *milk* to *man* is. This observation prefigures the importance of choosing the right context when building distributed word representations for the compound and its constituents, a matter that will be the topic of Chapter 4.

The grammar used by Ryder (1994) is defined to be *"a structured inventory of conventional **units**"* (Langacker, 1987:57). There are three possible types of units: **phonological units** (segments, syllables, words, familiar phrases), **semantic units**

(concepts) and **symbolic units**, which form through the association of a phonological unit with a semantic unit. The units are entries in a **semantic network**, and the separate pieces of information associated with a unit can be accessed independently, some more frequently than others. The units are connected via three types of relations: (i) the relation that connects a phonological and a semantic unit, giving rise to a symbolic unit; (ii) the **categorization** relation, where a unit can be seen as an instantiation of a more abstract unit: e.g. the semantic unit [CAT] is an instantiation of the unit [ANIMAL] which in turn instantiates the semantic unit [LIVING THING]; (iii) the **integration** relation, where multiple units of the same type combine to create a larger unit of the same type, e.g. the symbolic units [[CAT]/[kæt]] and [[FOOD]/[fud]] combine into a new symbolic unit with composite structure, [[CAT]/[kæt]]-[[FOOD]/[fud]]. A crucial aspect that is underlined by Langacker (1987) with respect to the resulting composite structures is that generally they have qualities that cannot be directly extrapolated starting from the original elements that were combined. The term **accommodation** is used to refer to the process of altering the meaning of one or both components in order to form a meaning for the composite structure (e.g. in the case of the compound *doghouse*, the semantics of the word *dog* remain unaltered while the meaning of the word *house* undergoes a heavy accommodation process).

These cognitive grammar concepts are operationalized by Ryder (1994) using **schemata**. Schemata are *"[...]basic data structures for representing the generic concepts stored in memory"* (Rumelhart et al., 1986b:18). They can model different concept types including objects, situations, events, sequences of events, actions or sequences of actions. Schemata have **variables** with default values. The variables correspond to possible instantiations of the abstract concepts represented by the schemata. Ryder (1994) illustrates the idea of a variable inside a schemata using the schemata for the *party* event: there is a variable for the `purpose` of the *party*, and one for the `present` that is brought to the *party*. If `purpose` is filled with *given in the honor of the host's birthday*, then the `present` variable should be filled with *present for the person who celebrates birthday*.

Schemata can **embed**: the schema corresponding to the event *patient visit to the doctor* includes separate subschemata for the *patient* and for the *doctor*. Moreover, the schemata can inherit from each other: the schema for *doctor* inherits from the more general schema for *person*. In general, schemata represent knowledge at all levels. They are active processes, in that their variables and defaults vary in response to new situations and change with use. They are recognition devices for particular situations - i.e., there must be ways to measure how well a schemata fits a given situation.

Based on ideas from cognitive grammar and schemata, Ryder (1994)'s investigations focus on analyzing the strategies used by the speaker and the hearer in the creation and interpretation of compounds. She takes the interpretation to be guided, in the absence of context, by known **linguistic patterns** or **semantic information schemas**. The linguistic patterns are derived from all the expressions known to the person:

- known compounds that share a modifier (e.g. *sea* + Obj. meaning a metaphorical extension of Obj. that lives in the *sea*: *sea cucumber*, *seahorse*) or head (e.g. the pattern Obj. + *box* where the *box* is meant as a container for Obj.: *pencil box*, *wine box*)
- frequently encountered combinations (e.g. Location + Animal, where the animal originates from the specified location: *Angora goat, Labrador retriever*, Animal_1+Animal_2, where Animal_2 resembles Animal_1 in some way: *tiger cowrie, zebra spider*)
- combinations that are based on matching features in the schemas of their components: e.g. in *baby spoon*, the notion of bringing food to a person's mouth is central to the schema for *spoon*, while the *baby* fill the person role
- groups of compounds where the individual components are similar (e.g. Contained-Container: *flour sack*, *water glass*, *flowerpot*)
- compounds formed by analogy to other compounds (*Angora goat* lead to *Angora rabbit*, meaning *rabbit with long, silky fur*)

Ryder (1994) notes that compounding forms *"... a continuum from smaller and less productive patterns to the larger, extremely productive ones"*. She devises two experiments where she elicits judgments about the interpretation of novel compounds, built either using a specific pattern or at random. The conclusion of her experiments is that if the compound is based on a specific pattern, the responses of the participants will be very homogeneous and will produce approximately the same meaning interpretation that was used to create the compound in the first place. Furthermore, she found a remarkable uniformity in the participant's responses when interpreting random compounds[7], suggesting that different individuals tend to use similar linguistic templates, even when confronted with a seemingly odd construction.

Another interesting observation Ryder (1994) makes is that if accommodation is required to arrive at a meaning for the compound, the head is accommodated 4.6 times more frequently than the modifier. An example she gives is the compound *school-shark*, where 62.5% of the subjects chose to accommodate the head *shark* to one of its metaphorical senses[8], even as the noun *school* has a sense related to marine life[9].

She concludes that the use of compounding is predictable, but that each compound can be assigned a range of possible meanings. The range is very restricted in some cases, leading to a rule-like interpretation, while in other cases the range covers a larger number of interpretations. Ryder (1994)'s probabilistic perspective on the interpretation of compounds, justified by the methodology used in her experiments, suggests that a model for the automatic interpretation of compounds should be designed to allow for the same range of interpretations that was observed in the human responses.

[7] compounds produced by randomly pairing two concrete nouns. E.g. from Ryder (1994): *dish-stick*, *ear-stone*, *elephant-web*.

[8] 1. a person who is ruthless and greedy and dishonest; 2. a person who is unusually skilled in certain ways; definitions from WordNet 3.1 (Fellbaum, 1998).

[9] a large group of fish; definition from WordNet 3.1.

Lauer (1995) covers two aspects of noun compound interpretation: parsing compound noun sequences of length greater than two, and the semantic analysis of noun compounds. The parsing component is intended to distinguish between left-branching and right-branching interpretations: e.g. the compound *company tax policy*[10] can be either interpreted in a left-branching way, as a *policy* regarding *company tax* or in a right-branching way, as *tax policy* that applies to *companies*.

Lauer (1995) proposes a model that captures dependency relations between the compound constituents. The intuition behind his model is that if the first two nouns form a more acceptable sequence (e.g. *company tax*), then the construction should have a left-branching interpretation. If, however, the first and the third noun are a more acceptable sequence (e.g. *company policy*), then the construction is deemed to be right-branching.

Lauer (1995)'s parsing system requires three 'ingredients': a corpus, a part-of-speech lexicon and a thesaurus. He uses *The New Grolier Multimedia Encyclopedia* (Grolier Inc., 1992), comprising about 8 million words, as a corpus. The part-of-speech lexicon is used to determine which words are listed as always being nouns. This information is used in extracting compound sequences from the raw text corpus, while trying to minimize the erroneous extraction of sequences where words with multiple possible parts-of-speech are used in their non-nominal sense. The third ingredient, the thesaurus, is used to group together the constituent nouns into a set of categories. Lauer (1995) uses Roget's thesaurus containing 20,445 nouns grouped into 1043 categories, with an average of 34 distinct words per category (e.g. *goldfish* and *trout* are in the same category).

Lauer (1995)'s training set consists of 24,251 two-noun compounds extracted from the Grolier corpus, where each constituent appeared in Roget's thesaurus. The model is based on the concept of **affinity**, i.e. how likely it is for the concept represented by the first noun to act as a modifier for the concept represented by the second noun. The affinity between concepts is estimated using the training set compounds, and is then used to predict the correct interpretation of the compounds in the test set. The test set has 244 three-noun compounds which were manually annotated for being either left or right branching. The model computes a ratio between the probability of the sequence being left-branching and it being right-branching. One interesting aspect is that the formula used by Lauer (1995) factors in the possibility of a word being **polysemous** and thus 'shared' between multiple categories - consider *suit (law)* and *suit (clothing)*. The model is able to predict the correct branching for 77.5% of the test compounds, while the most frequent interpretation baseline is at 66.8%. The most frequent interpretation baseline is the left-branching one, thus concurring with the empirical observation concerning the most common type of structure made earlier by Warren (1978).

Lauer (1995) approaches the semantic analysis of noun compounds using **prepositional paraphrasing**. He states the problem as follows: given a noun-noun compound, specify its semantic interpretation by predicting the preposition involved in the com-

[10] example from Lauer (1995).

pound's preferred paraphrase. The choice is between eight possible prepositions: *of*, *for*, *in*, *about*, *with*, *from*, *on* and *at*. The intuition here is that a compound like *reptile heaven* and its prepositional paraphrase *heaven for reptiles* are interchangeable and will be used with approximately the same frequency, so one can use the prepositional paraphrase estimates to model the compound interpretation. The test set consists of a random sample of 400 noun-noun compounds from the Grolier corpus, which he manually annotates, talking the actual sentential context into account.

The model makes again use of the categories in Roget's thesaurus to abstract above the lexical level. For each (concept, preposition) tuple the model estimates two probabilities, depending on whether the concept is the attachment head or the object of the preposition. The model makes a strong independence assumption, namely that the modifier and the head concept contribute their preferences for a particular prepositional paraphrase independently, and the best interpretation is the one that is the most likely given the independent preferences of the modifier and of the head.

The model analyses each compound with respect to each preposition, and provides a score for every choice. The prediction of the model is taken to be the preposition with the highest score among the eight possible ones. The model achieves an accuracy of 47%, where the most frequent preposition baseline (*of*) is at 33%. In the discussion section Lauer (1995) points to the fact that the concept association was not the best modeling choice because preposition have lexical collocation preferences, and the same concept might in reality require different prepositions depending on the actual noun. Also, the assumption that the head and the modifier have completely independent preferences does not always hold - e.g. in *welfare agencies* one has to take both nouns into account to rule out the fact that *agencies* cannot be the beneficiary of *welfare*.

Lauer (1995)'s study is one of the first to attempt a statistical modeling of the noun compounds, by leveraging information obtained from large amounts of unannotated text. This makes his system one of the first ones to depart from hand-crafted knowledge-based models, which generally had good results on the subset of modeled compounds but poor generalization capabilities. However, the relatively modest results obtained by Lauer (1995)'s system point to the challenges in building an automated system for the semantic analysis of noun-noun compounds, and underline the importance of choosing compound representations that are compatible with the proposed task.

Barker and Szpakowicz (1998) propose an inventory of 20 semantic relations for specifying the semantics of noun compounds. For each semantic relation they provide a paraphrase to aid in the interpretation task (e.g. *winter semester*:TIME:'modifier is the time of compound'; *paper tray*:CONTENT:'modifier is contained in compound'). They propose an interactive, semi-automatic system for assigning semantic relations to compounds. It keeps a history of the semantic relations previously assigned to compounds with the same head or with the same modifier. For each new compound to be interpreted, it proposes a list of likely semantic relations to choose from. The list includes the paraphrase for each semantic relation, with the head/modifier/compound slots filled in by the modifier/head/compound that is currently being analyzed. Barker

and Szpakowicz (1998) report that the correct semantic relation was among the ones proposed by the system for 69% of the 886 compounds analyzed in this way. Their system relies on the intuition that compounds that share the modifier or the head are likely to also share the subset of semantic relations that render their semantics.

The new annotation scheme to be presented in Section 2.2 will follow similar principles, advocating for a head-centric annotation of compounds. In this setup, the annotation task presupposes that one annotates all the compounds with the same head at one time, thus encouraging the emergence of annotation patterns: e.g. the annotation of compounds with the head *cup* will likely lead to the identification of compound clusters that are labeled with a MATERIAL relation (e.g. *plastic cup*, *porcelain cup*) and with a CONTENT relation (e.g. *tea cup*, *coffee cup*). The same patterns might emerge later when annotating a similar head like *glass* - e.g. *plastic glass*, *beer glass* etc.

Rosario and Hearst (2001) study the semantic relations in nominal compounds in a domain-specific setting. They focus on noun compounds extracted from biomedical text and consequently their relation inventory is the result of applying the general techniques previously described in the literature to the particularities of the biomedical domain. It is interesting to note that some of the relations in their inventory are used in the same way as in general-purpose classification schemes (e.g. MATERIAL for *latex glove*, TOPIC for *headache questionnaire*, INSTRUMENT for *laser irradiation*), whereas others are overloaded with meanings specific to the biomedical domain (e.g. LOCATION is used both in for physical locations as in *hospital beds* and for parts of the human body, as in *brain artery*, *liver cell*). In addition, their inventory also contains relations that cater to the particular needs of the biomedical subdomain they are modeling (e.g. DEFECT for *gene mutation*, or PROCEDURE for *brain biopsy*). Some of the relations also specify the directionality of the relation (e.g. CAUSE(1-2) for *food infection*, where the *food* causes the *infection* vs. CAUSE(2-1) for *flu virus*, where the *flu* is caused by the *virus*).

Rosario and Hearst (2001) identify 38 semantic relations, but use for the automatic classification only a subset of 18 semantic relations which label a minimum of 25 of examples in their dataset. The classification experiments are based on two types of representations: first, a *lexical representation*, where each word is represented as a one-hot vector of the dimensionality of the vocabulary (1184 words, in their case); second, a *knowledge-based representation* using MeSH (Medical Subject Headings), a domain-specific lexical hierarchy. Compound representations are formed by concatenating the representations of their two constituents.

A neural network with one hidden layer and a `tanh` nonlinearity is used to perform the classification. The network outputs a number in (0,1) for each of the 18 possible semantic relations, representing the probability that the compound under examination is based on a particular semantic class. This architecture makes it possible to model compounds where two or more relations are deemed as possible (e.g. *bladder dysfunction*

can be labeled both LOCATION and DEFECT), and render unnecessary strict guidelines for labeling a compound with one relation or the other.

Rosario and Hearst (2001) obtain almost the same performance using the lexical features (62% accuracy) and the MeSH hierarchy features (61% accuracy). However, they prove that the model based on MeSH hierarchy features has better generalization capabilities than the model based on lexical features. They simulate more realistic scenarios in which the system has to predict the relations of compounds whose heads, modifiers or both constituents were not seen during the training procedure. In such cases, the hierarchy-based model is still able to make sensible predictions, whereas the lexical model is reduced to random guessing.

Rosario et al. (2002) show that the compound-internal semantic relation can be identified with an accuracy of approximately 90% by using the semantic categories that the head and the modifier belong to. The semantic categories are identified by traversing in a top to bottom fashion the same domain-specific lexical hierarchy, MeSH, that was used by Rosario and Hearst (2001). The examples they give are compounds like *leg paresis*, *skin numbness* and *hip pain* where in each case the modifier is a *Body Region* (MeSH category A01) and the head is a *Nervous System Disease* (MeSH category C10). They consider the relation that holds for these compounds to be LOCATED IN. This semantic relation is considered then to be a general pattern for the compounds with modifiers and heads belonging to these two particular categories.

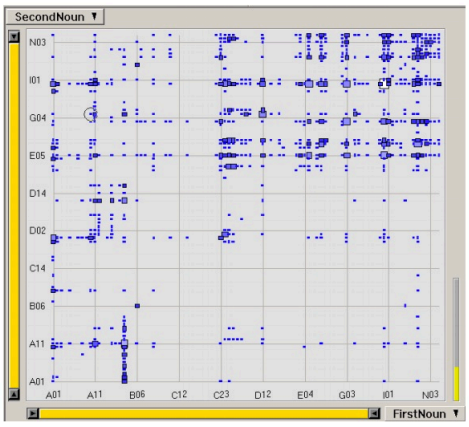


Figure 1: Distribution of Level 0 Category Pairs. Mark size indicates the number of unique NCs that fall under the CP. Only those for which > 50 NCs occur are shown.

Fig. 2.1 Clustering of compounds with respect to the MeSH categories of their constituents; figure reproduced from Rosario et al. (2002).

Figure 2.1, reproduced from Rosario et al. (2002), illustrates the clustering of compounds with respect to the MeSH categories. The 15 level 0 categories of the MeSH hierarchy are plotted on the horizontal axis - for the modifier - and on the vertical axis - for the head. The distribution of compounds is shown to be non-uniform, meaning

that for a compound with the head/modifier from a specific category there is only a subset of semantic relations that could potentially apply. Intuitively, their study shows that categorical information regarding the semantics of the modifier and the head can be used to pre-filter the semantic relations that are possible given the observed categories of the constituents, an idea also put forth in previous studies (Warren, 1978; Finin, 1980; Ryder, 1994).

Nastase and Szpakowicz (2003) propose an annotation scheme with 50 relations for annotating the semantics of modifier-noun pairs. The modifiers can be nouns, adjectives or adverbs. They use the relations to annotate a dataset of 600 modifier-noun pairs, and end up using only 30 of the 50 proposed relations, because the other 20 relations do not occur in their dataset. Examples of compounds annotated using their annotation scheme are *flu virus*, annotated with the semantic relation CAUSE, *concert hall* annotated as PURPOSE, *printer tray* annotated as PART and *daisy chain* annotated as WHOLE. Each relation has an associated paraphrase that can be used for testing if the relation applies to a specific compound: e.g. CAUSE is paraphrased as *H makes M occur or exist*. The constituents of the annotated compounds are also disambiguated with respect to the word senses in Roget's thesaurus, using WordNet (Fellbaum, 1998) as additional information. In their experiments they use Roget's and WordNet-based features to build semantic relation classifiers, with an expressed aim of obtaining an interpretable algorithm rather than a perfect score. The best performance was obtained using a rule induction system.

Lapata and Keller (2004) showed that the prepositional paraphrases for noun compounds can be predicted using co-occurrence frequencies obtained from the web. They show that large amounts of unannotated web data can provide more information about the typical preposition than the statistics obtained from a balanced corpus like the BNC, or by the use of concepts instead of nouns in the interpretation process like in Lauer (1995). Lapata and Keller (2004) evaluate their co-occurrence models on Lauer (1995)'s test dataset and obtain 55.71% accuracy at predicting the correct prepositional paraphrase using a trigram model, $f(n_1, p, n_2)$. For comparison, Lauer (1995)'s best result was 47% accuracy.

Kim and Baldwin (2005) use the inventory of 20 semantic relations proposed by Barker and Szpakowicz (1998) to annotate a dataset of 2,169 binary noun compounds. They note that particular pairs of semantic relations seem to be the cause for systematic disagreement between the annotators: SOURCE and CAUSE, PURPOSE and TOPIC, OBJECT and TOPIC. Their approach to the automatic interpretation is based on four WordNet similarity measures, namely `wup` (finds the depth of the least common subsumer (LCS) of two words, normalized using the combined depth of the concepts), `lch` (length of the path between the two concepts, scaled by the maximum length path), `jcn` (subtracts the information content on the LCS from the sum) and `lin` (scales the information content of the LCS relative to the sum).

Compounds in the test set are compared (pairwise) to all the compounds in the training set. Each test compound is then assigned the semantic relation of the most similar compound from the training set. When comparing two compounds, each similarity measure is applied to the modifier pair (m_i, m_j) and to the head pair (h_i, h_j). The modifier and head similarities are then integrated into a compound-level similarity either using a linear combination with a weighting parameter α or using the F-score formula. The best accuracy, 53.3%, is obtained using `wup` similarity and α=0.5.

Although they are using a WordNet-based approach, Kim and Baldwin (2005) argue that explicit disambiguation of the constituents' senses is not required for the interpretation of noun compounds. Their position is that the pairwise comparison with other compounds in enough to filter out compounds containing different senses of a word: e.g. *loan rate* is judged to be more similar to *bank interest* than to *personal interest* because $sim(bank, loan) > sim(personal, loan)$.

Turney and Littman (2005) use the 600 noun-modifier pairs labeled by Nastase and Szpakowicz (2003) in a series of experiments. Their approach stands out from others because they try to explicitly model the relation and quantify **relational similarity**, rather than modeling the constituents and thus model the relation between the constituents only indirectly. They construct relation representations by considering a series of 64 patterns such as *X of Y*, *X for Y*, *X onto Y* and their inverses. For each pattern, they record the frequency of occurrence (X and Y are replaced by the modifier and the noun, respectively). The logarithm of the frequencies is used to build a 128-dimensional feature vector. The similarity of two relations is computed by measuring the cosine of the angle between their log-frequency vectors. Turney and Littman (2005) obtain an F_1 score of 26.5 at predicting one of the 30 semantic relations for the 600 noun-modifier pairs, using nearest neighbor classification and leave-one-out cross-validation.

Turney (2006) reports on another set of experiments using the same dataset labeled by Nastase and Szpakowicz (2003). Turney (2006) makes explicit the distinction between attributional and relational similarity. The **attributional similarity** of two concepts *A* and *B "depends on the degree of correspondence between the properties of A and B"* (Turney, 2006:381). Concepts with a high degree of attributional similarity are for example *dog* and *wolf*. **Relational similarity** is measured between two pairs of words *A*:*B* and *C*:*D*, and *"depends on the degree of correspondence between the relations between A and B and the relations between C and D"* (Turney, 2006:382). Good analogies, like *mason:stone::carpenter:wood* have both high relational similarity between the pairs *mason:stone* and *carpenter:wood* as well as high attributional similarities between the individual components of the pairs - *mason-carpenter* and *stone-wood*. Turney (2006) models the relational similarity of noun-modifier pairs using a technique he calls Latent Relational Analysis (LRA). Using the same setup like in Turney and Littman (2005) and the LRA-based features, Turney (2006) obtains an F_1 score of

36.6 at predicting the correct semantic relation for the 600 noun-modifier pairs in the dataset.

Girju et al. (2005) annotate noun compounds using the list of 8 prepositional paraphrases proposed by Lauer (1995) and their own list of 35 semantic relations. Girju et al. (2005) annotate both two part compounds as well as compounds that consist of three nouns. The constituents of the compounds are disambiguated with respect to WordNet senses. It is interesting to note that the authors devise an inventory with 35 semantic relations before they start the annotation task. At the end of the annotation (approx. 4500 compounds), they discover that only 21 out of the 35 proposed relations were actually used by the annotators. Most frequent in their case are the relations PART-WHOLE(*girl mouth*), ATTRIBUTE-HOLDER(*quality sound*), PURPOSE(*migraine drug*), LOCATION(*Texas university*), TOPIC(*art museum*) and THEME(*car salesman*). Their experiments focus on the automatic identification of the best semantic relation for a given compound. The best results are obtained using a binary Support Vector Machine (SVM) classifier for each pair of relations and a majority voting scheme for selecting the winning semantic relation label. The results for the semantic relation classification task show a marked increase in accuracy (from 43.53% to 66.78%) if the "gold" preposition is added as an additional feature, suggesting that there is a close interaction between the semantic relation and the corresponding preposition.

Girju (2006) approaches the compound interpretation problem from a multilingual perspective, and studies compounds in English along with their translations in four Romance languages: French, Italian, Spanish and Romanian. The annotation inventory contains in this case 22 semantic relations and the corresponding prepositional paraphrases. Each compound is translated into the four Romance languages. A dataset is formed by taking only those noun-noun compounds, 'N N', whose translations in all four Romance languages follow the pattern noun-preposition-noun, 'N P N'. The automatic classification experiments in Girju (2006) show that: (i) adding the English prepositional paraphrase improves the accuracy of semantic relation identification (from 56.03% to 58.02%); (ii) accuracy is further improved by adding the prepositions from translations in different languages as features (66.1% using the French prepositions; 68.5% with French and Italian; 69.3% with French, Italian and Spanish and 71.2% with the prepositions used for translating the compound into all four Romance languages).

Ó Séaghdha (2008) develops a relational annotation scheme for compounds, following Levi (1978)'s predicate-based proposal. His inventory and the associated annotation guidelines are aimed at minimizing the ambiguity in assigning an unique annotation label to each compound. The inventory includes six categories for capturing coherent semantic relations (BE, HAVE, IN, ACTOR, INST, ABOUT) and three categories for other compounds, whose semantics is not covered by the previous six categories (REL, LEX, UNKNOWN). The annotation also considers the directionality of the semantic relation: subscript 1 is used to indicate that the order of the constituents matches

the order specified by the relation's definition, while subscript 2 is used to mark the reverse order of the constituents w.r.t. the definition of the relation. Table 2.1 presents the inventory of semantic relations proposed by Ó Séaghdha (2008), together with illustrative examples.

Relation	Example	Definition
BE		*N_2 which is (a) N_1; N_2 like N_1:*
	soya bean	appositive compounds
	ice crystal	material-form compounds
	cat burglar	resemblance compounds
HAVE		compounds expressing:
	reader mood	mental or physical conditions
	grass scent	and properties;
	computer clock	part-whole relations
	star cluster	group membership
IN	*pig pen*	spatial location
	evening edition	temporal location
ACTOR	*student demonstration*	participants are *sentient* (includes organizations)
INST	*production line*	participants are not sentient
ABOUT	*history book*	an item that is *about* something
	exam practice	mental state or activity
	house price	prices and charges
REL	*lithium hydroxide*	used as a fall-back category when none
	diamond jubilee	of the six categories above can be used;
LEX	*monkey business*	for lexicalized or idiomatic compounds;
UNKNOWN		non-interpretable compounds

Table 2.1 Semantic relations in Ó Séaghdha (2008)'s inventory - excerpt from Table 3.1 in Ó Séaghdha (2008).

Ó Séaghdha (2008) evaluates the annotation scheme in an inter-annotator agreement study. He reports 66.2% raw agreement and a Kappa score of 0.62 for the semantic relation annotation on a sample of 500 compounds. The compounds are automatically extracted from the British National Corpus (Burnard, 1995), a text collection containing 90 million words. The interpretation of the compound is always performed in context: the annotator is presented with the compound and the corpus sentence where it occurs, and is asked to provide a relation label and a directionality judgment Another important aspect of this study is that many of the chosen compounds are infrequent - 97 out of the 500 compounds in the IAA study occur only once in the BNC.

1443 compounds annotated with the six semantic relations are subsequently used in machine learning experiments. The six semantic classes label roughly similar proportions of data: the most frequent relation, IN, labels 21.3% of the data; the least frequent, BE, labels 13.2%.

Two types of information are used to capture the interaction between the compound constituents: lexical information - information about the compound constituents as individual words and relational information - information about the typical interactions between the entities designated by the pair of constituents. Based on the lexical and relational information one can distinguish between different types of similarity for a particular concept. **Lexical similarity** considers pairwise similarities between constituents: *plastic knife* and *metal spoon* are similar because the pairs (*plastic*, *metal*) and (*knife*, *spoon*) are similar. **Relational similarity** models the typical interactions between compound constituents using the contexts where they appear together as independent words. The underlying intuition is that word pairs appearing in similar contexts will have similar semantic relations. Consider the contexts: "The *knife* was made of cheap *plastic*."; "*Spoons* are typically made of *metal*.". Given the similarity of the contexts, the relation between *knife* and *plastic* is assumed to be similar to the relation between *spoon* and *metal*.

The distinction between lexical and relational similarity is similar to the one proposed by Turney and Littman (2005) and Turney (2006), who refer to lexical similarity as attributional similarity.

Ó Séaghdha (2008) uses distributional information to model the semantics of the compound constituents. A first set of features, which is based on conjunctions, is extracted from the tagged, lemmatized and parsed BNC corpus. In effect, the noun N_i is modeled using the information provided by the counts of the relation $conj(N_i, N_j)$, were N_j is part of the target vocabulary. The target vocabulary contains the 10,000 nouns that occur most frequently in a conjunction relation in the corpus. Each co-occurrence vector is normalized to either L_1 or L_2 norm, depending on the kernel it will be used with.

Another set of features is based on the Web 1T Google 5-Gram Corpus (Brants and Franz, 2006), and builds distributional representations in a similar way to the conjunction-based features. However, parsing cannot be used in this case, as the corpus consists only of word ngrams. Therefore, the pattern used to extract co-occurrence information from the ngrams is based on the "joining terms" used previously by Turney and Littman (2005)[11]. The target vocabulary is constructed independently for each joining term and consists of the 10,000 most frequent co-occurrences with that term. The extraction pattern allows for non-nominal material after the joining term: $N_i\ J\ (\neg N)^*\ N_j\ \neg N$. In this extraction pattern N_i is the noun to be modeled, J is the joining term, $\neg N$ is a non-nominal and N_j is a noun from the target vocabulary.

The Web 1T corpus is also the source of verbal co-occurrence information, extracted using the patterns $N_i\ that|which|who\ V\ \neg N$ and $N_i\ that|which|who\ V\ (\neg N)^*\ N_j\ \neg N$,

[11]The joining terms used by Ó Séaghdha (2008) are: *and*, *or*, *about*, *at*, *by*, *for*, *from*, *in*, *is*, *of*, *to*, *with* and *like*.

where N_i and N_j are the nouns to be modeled and V is the verb. The feature takes into account if the noun occurs before the verb (subject reading) or after the verb (object reading).

The classification experiments use SVMs with an extension for multi-class classification. The setup uses 5-fold cross-validation. The experiments compare the performance of different kernel types, and the impact of the different features on the classification task. The best classifier based on lexical features is the JSD RBF kernel using 5-gram features based on the joining term *and*, which obtains 61.0% accuracy and 58.8% F_1-score. The contribution of the distributional representation of the head noun is proven to be more important for the classification than the contribution of the modifier noun: a classifier using head-only features reaches an accuracy of 51%, while the modifier-only classifier obtains only 40.1% accuracy.

While a classifier based only on lexical similarity obtains good results, Ó Séaghdha (2008) further shows that the combination of lexical and relational features can improve the classification results. Relational features are based on contexts where the two constituents of the compound occur together as separate words and are modeled via string kernels. Relevant contexts for each compound are extracted from the BNC and from the Gigaword corpus. The combination of the best classifier using lexical features and the best JSD set kernel using relational features resulted in the best classification performance overall: 62.7% accuracy, 61.2% F_1-score.

The best results on the dataset proposed by Ó Séaghdha (2008) were reported in **Ó Séaghdha and Copestake (2013)**. The setup follows closely the one in Ó Séaghdha (2008), with the difference that now the feature extraction relies heavily on the grammatical relations between the constituent words, extracted in the form of dependency tuples. The lexical information is gathered from corpus sentences containing the compound constituents. The sentences are automatically parsed and all the tuples of the form (dependency relation, dependent word) which have a constituent as a head are considered as features. Ó Séaghdha and Copestake (2013) consider as relational information only the sentences where a pair of constituents appears as two separate words and are no more than ten words apart. As an additional restriction, the contexts that extend past five words to the left of the leftmost constituent or five words past the right of the rightmost constituents are discarded.

Again, classifiers using lexical information performed better than the ones using relational information: the best result using lexical information was 63.0% accuracy and 61.0 F_1-score, while the best result obtained using relational information was 52.7% accuracy and 50.7 F_1-score. The best overall result was obtained by a combination of the individual lexical and relational classifiers, which obtained an accuracy of 65.4% and 64.0 F_1-score.

Tratz and Hovy (2010) propose a fine-grained taxonomy with 43 semantic relations for annotating the semantics of noun compounds, and an annotated dataset containing 17509 compounds. The taxonomy consists of semantic categories that resemble but are not identical to the ones previously proposed by Barker and Szpakowicz (1998)

and Girju et al. (2005). Tratz and Hovy (2010) motivate their new inventory by the necessity to achieve more reliable inter-annotator agreement than was obtained for these earlier inventories.

Word-based features
- {synonyms, hypernyms} for all NN and VB entries for each word
- intersection of the words' hypernyms
- all terms from the 'gloss' of each word
- intersection of the words' 'gloss' terms
- lexicographer file names for each word's NN and VB entries (e.g. n_1:substance)
- logical AND of lexicographer file names for the two words
- lists of all link types (e.g. meronym links) associated with each word
- logical AND of the link types (e.g. n_1:hasMeronym(s)$\wedge$$n_2$:hasHolonym(s))
- PoS indicators for the existance of VB, ADJ and ADV entries for each noun
- logical AND of the PoS indicators for the two words
- 'lexicalized' indicator for the existence of an entry for the compound as a single term
- indicators if either word is a hypernym of the other
- indicators if either word is in the definition of the other
Roget's Thesaurus-based features
- Roget's division for all noun (and verb) entries for each word
- Roget's divisions shared by the two words
Surface-level features
- indicators for the suffix types (e.g. deadjectival, de-nominal [non]agentive, deverbal [non]agentive)
- indicators for degree, number, order, locative prefixes (e.g. ultra-, poly-, post- and inter-, respectively)
- indicators for whether or not a preposition occurs within either term
- the last {two, three} letters of each word
Web 1T ngram features
- patterns over trigrams and 4-grams containing both constituents

Table 2.2 Features used for automatically classifying semantic relations in noun compounds by Tratz and Hovy (2010).

The taxonomy was developed gradually and the process involved both refining existing categories and removing problematic categories. The taxonomy's development process was guided by the analysis of the disagreements between the annotations of different annotators, gathered using Amazon's Mechanical Turk service. The taxonomy and the definitions that were provided with it are intended to provide a way to uniquely label any given compound using a single relation label. The refinement steps

Category name	Dataset percentage	Example
Objective		
OBJECTIVE	17.1%	leaf blower
Doer-Cause-Means		
SUBJECT	3.5%	police abuse
CREATOR-PROVIDER-CAUSE__OF	1.5%	ad revenue
JUSTIFICATION	0.3%	murder arrest
MEANS	1.5%	faith healer
Purpose/Activity Group		
PERFORM&ENGAGE__IN	11.5%	cooking pot
CREATE-PROVIDE-GENERATE-SELL	4.8%	nicotine patch
OBTAIN&ACCESS&SEEK	0.9%	shrimp boat
MITIGATE&OPPOSE	0.8%	flak jacket
ORGANIZE&SUPERVISE&AUTHORITY	1.6%	ethics authority
PURPOSE	1.9%	chicken spit
Ownership, Experience, Employment, Use		
OWNER-USER	2.1%	family estate
EXPERIENCER-OF-EXPERIENCE	0.5%	family greed
EMPLOYER	2.3%	team doctor
USER__RECIPIENT	1.0%	voter pamphlet
Temporal Group		
TIME-OF1	2.2%	night work
TIME-OF2	0.5%	birth date
Location and Whole+Part/Member of		
LOCATION	5.2%	hillside home
WHOLE+PART__OR__MEMBER__OF	1.7%	robot arm
Composition and Containment Group		
CONTAIN	1.2%	shoe box
SUBSTANCE-MATERIAL-INGREDIENT	2.6%	plastic bag
PART&MEMBER__OF__COLLECTION&CONFIG&SERIES	1.8%	truck convoy
VARIETY&GENUS__OF	0.1%	plant species
AMOUNT-OF	0.9%	traffic volume
Topic Group		
TOPIC	7.0%	travel story
TOPIC__OF__COGNITION&EMOTION	0.3%	auto fanatic
TOPIC__OF__EXPERT	0.7%	policy expert
Other Complements Group		
RELATIONAL-NOUN-COMPLEMENT	5.6%	eye shape
WHOLE+ATTRIBUTE&FEATURE &QUALITY__VALUE__IS__CHARACTERISTIC__OF	0.3%	earth tone
Attributive and Equative		
EQUATIVE	5.4%	fighter plane
ADJ-LIKE__NOUN	1.3%	core activity
PARTIAL__ATTRIBUTE__TRANSFER	0.3%	skeleton crew
MEASURE	4.2%	hour meeting
Other		
LEXICALIZED	0.8%	pig iron
OTHER	5.4%	contact lense
Personal*		
PERSONAL__NAME	0.5%	Ronald Reagan
PERSONAL__TITLE	0.5%	Gen. Eisenhower

Table 2.3 Semantic relation inventory used by the Tratz dataset - abbreviated version of Table 4.5 from Tratz (2011). Note that some relations have a slightly different name in the actual dataset than the aforementioned table; this table lists the relation names as found in the dataset.

described by the authors clearly target relations that were systematically confused by the annotators, with the aim of modifying the taxonomy in order to minimize such sources of confusion.

The categories are defined using sentences: e.g. the category SUBSTANCE has the definition n_1 *is one of the primary physical substances/materials/ingredients that* n_2 *is made/composed out of/from.* The name of the relation is usually a two part description of the expected categories of the two constituents separated by the + sign: e.g. SUBSTANCE/MATERIAL/ INGREDIENT + WHOLE. The individual categories form taxonomic groupings, e.g. the **temporal group**, the **causal group**, the **purpose group**, etc.

Tratz and Hovy (2010) report on automatic classification experiments on their newly proposed dataset using a Maximum Entropy classifier. They extract a wide variety of features, relying both on knowledge-bases like WordNet and Roget's thesaurus, on ngram data from the Web 1T corpus as well as on surface-level features. Table 2.2 presents a list of the features used by Tratz and Hovy (2010). The features model both the semantics of the individual constituents of the compound, as well as their pairwise interactions (especially via relations in the WordNet hierarchy). One of the features explicitly targets lexicalization, and marks compounds that have dedicated entries in WordNet. The experimental setup used 10-fold cross-validation. The feature ablation study shows that the two features with the strongest influence were the hypernyms feature and the WordNet gloss terms feature, thus underlining the importance including type-level information into the feature set. Tratz and Hovy (2010)'s classifier obtains an accuracy of 79.3% on their dataset, and 63.6% on Ó Séaghdha (2008)'s dataset.

Tratz (2011) proposes a revised noun compound relation inventory with only 37 semantic relations which allows for a better mapping between prepositional paraphrases and noun compound relations. The compound classification experiments described in Tratz and Hovy (2010) were, however, not re-run on the revised dataset. Since only the Tratz (2011) dataset is publicly available as part of the semantically-enriched parser built by Tratz (2011)[12], this dataset is used in the semantic relation classification experiments in Chapter 7. The Tratz (2011) dataset is the largest publicly-available annotated noun compound dataset for English, containing 19158 compounds annotated with 37 semantic relations. Table 2.3, which is an abbreviated version of Table 4.5 in Tratz (2011), illustrates these relations by characteristic examples and indicates the relative frequency of each relation within the dataset as a whole.

Klos (2011) analyses the process of compounding in German, focusing in particular on the impact of composition and compositionality in the interpretation of nominal compounds. The analysis includes a series of studies whose goal is to uncover the mechanisms that people make use of when interpreting compounds. Two of these studies are of particular interest for this thesis: the first one, which studies the context-free interpretation of compounds, and the third one, which complements the first study by looking at how the interpretation is affected by context.

[12]The dataset is available for download at `http://www.isi.edu/publications/licensed-sw/fanseparser/`

Study 1: Context-free interpretation of compounds. Although from a data-driven perspective Klos (2011)'s corpus of 20 compounds in the first study seems exceedingly small, it becomes clear that the added value here are the range of interpretations offered for these compounds by the 117 participants in the online study. The compounds chosen for the study fall into four categories, ranging from idiomatic to newly coined: lexicalized compounds (e.g. *Geisterfahrer* 'ghost rider'), institutionalized compounds (e.g. *Raucherkneipe* 'smoker's bar'), deictic compounds (e.g. *Autofieber* 'automobile fever') and randomly constructed compounds (e.g. *Tassentier* 'cup animal'). Klos (2011) draws several conclusions from this study:

- people are generally aware of the fact that a compound can have multiple senses, and will, when the compound is unknown to them, propose several likely interpretations;
- the interpretation is usually based on a concrete semantic relation involving the two constituents of the compound; an interpretation based on an underspecified relation is in most cases to vague to be considered a satisfactory meaning interpretation;
- each compound is compositional to a different degree; this makes it possible to derive the generally accepted interpretation for a subset of the unknown compounds; however, the compositional interpretation will not correspond to the generally accepted interpretation in all cases;
- the moment a person knows the object denoted by a compound, she can have the illusion of compositionality: *"once one knows that a houseboat is a boat that one can sleep and cook in, the contributions of its morphemes to the meaning seem clear"* (Libben, 2006:11).
- when the analysis of a compound using the normal ordering (modifier first, head second) does not lead to a plausible interpretation, the grammatical rules will be overridden and the interpretation will use the constituents in the reverse order;
- compounds can be analyzed in isolation, as long as the person can think of a suitable interpretative context for their analysis; the context is generally determined by the semantics of the head word.
- people make use of known word associations when interpreting unknown compounds: the randomly created compound *Hausleger*, lit. 'house layer' is interpreted as *persons that lays houses like eggs, i.e. that builds many houses* because the verb *legen*, from which the noun *Leger* is derived, collocates strongly with the noun *Eier* 'eggs' in the expression *Eier legen* 'to lay eggs'.

Study 3: Compound interpretation in and out of context. Klos (2011) looks at the differences in compound interpretation with and without context. A person is first asked to interpret a compound without context, and is subsequently asked to reconsider the interpretation while looking at the compound's actual context of occurrence. The goal of the study is to find out if the initial out-of-context interpretation is revised,

modified or confirmed in the presence of context. The study focuses on five compounds identified in the columns of a magazine for women. Thirty participants are asked to interpret the five compounds, first in isolation and then given the original context of the article where the compound was extracted from. The results of the study show that while context can help in the interpretation of compounds, its presence does not necessarily lead to a clear, unique and unambiguous interpretation of the compound. The compositional interpretation plays, also in this case, the leading role, while the immediate context serves as a way to identify the textual referents named by the compound.

Summary. The approaches to noun-noun compound interpretation surveyed in this section present an interesting discrepancy: early theoretical models make use of a very rich set of semantic features and an accompanying format for arranging this knowledge and try to come as close as possible to describing the compound interpretation process as done by humans. In contrast, the modern computational approaches aim at modeling compounds using a 'flat' representation of the constituents, obtained using various counts and statistics derived from corpora. Even if the modern annotation schemes make use of semantic distinctions similar to the features proposed early by Warren (1978), Finin (1980) or Ryder (1994), the computational methods focus directly on the task of classifying the compounds according to some annotation scheme. The aim of this thesis is to analyze the semantics of compounds with the purpose of understanding how to devise a method for creating computational representations for nominal compounds. These representations should, ideally, be able to capture both the semantics of the individual constituents as well as the semantic relation that connects them.

The next section introduces a new hybrid annotation scheme. The dataset annotated using this annotation inventory, presented in Section 7.1.1, is used in computational experiments targeting the automatic identification of compound-internal semantic relations described in Chapter 7.

2.2 Head-Centric Compound Analysis with a Hybrid Annotation Scheme

Acknowledgement. The annotation scheme described in this section is the result of a team effort - see the Chapter 8.1, Contributions section for details.

This section introduces a new, **hybrid annotation scheme** which uses both semantic relations (**properties**) and prepositional paraphrases (**prepositions**) to render the semantics of noun-noun compounds. The motivation for combining semantic relations with prepositional paraphrases can be illustrated for German by considering the set of compounds involving the concrete noun *Haus* 'house' presented in Table 2.4. They illustrate the range of modifiers the head *Haus* can combine with and the diverse set of semantic relations and prepositional paraphrases that need to be assigned.

MATERIAL, USER, GOODS and LOCATION are only a subset of the properties that compounds with the head *Haus* can have[13].

Compound	Property/Preposition
Holz*haus* 'wooden house'	MATERIAL/*aus* 'of'
Schnee*haus* 'snow house'	MATERIAL/*aus* 'of'
Stein*haus* 'stone house'	MATERIAL/*aus* 'of'
Gäste*haus* 'guest house'	USER/*für* 'for'
Jugend*haus* 'youth house'	USER/*für* 'for'
Auto*haus* 'car dealership'	GOODS/*für* 'for'
Möbel*haus* 'furniture store'	GOODS/*für* 'for'
Baum*haus* 'tree house'	LOCATION/*in* 'in'
Eck*haus* 'corner house'	LOCATION/*an* 'on'

Table 2.4 Compounds with the head *Haus* 'house', annotated both with a semantic relation (*property*) and a prepositional paraphrase (*preposition*).

A straightforward way of conveying the meaning of a compound is to annotate it with the **preposition** that best paraphrases the interaction between the modifier and the head. While such prepositional paraphrases have been proposed in the literature as being are natural and intuitive (Lauer, 1995), this type of annotation assigns in some cases the same preposition to compounds where the modifier serves different functions with respect to the same head.

For example, the compound *Gästehaus* 'guest house' refers to a building whose intended users are the *guests*. In contrast, the compound *Autohaus* 'car dealership' does not refer to a building whose intended users are *cars*, but rather to a building that is used for displaying and selling *cars*. Still, both compounds are paraphrased using the same preposition *für* 'for'. These differences in interpretation are reflected in the new hybrid annotation scheme by the annotation with different **properties**, in this case USER for *Gästehaus* and GOODS for *Autohaus*. Examples like these justify the hybrid nature of the new annotation scheme: semantic properties that name the relation between the modifier and the head are required to further specify the meaning of the prepositions. Thus, the correlation between prepositions and properties, which was also noticed in previous analyses (Warren, 1978; Girju et al., 2005; Girju, 2006), results in a better disambiguation of both meaning aspects.

The dual annotation also yields more consistent annotations. As seen in Table 2.4, compounds annotated with the same property will typically be associated with the same preposition or the same subset of prepositions. For example, the properties MATERIAL and USER are each associated with one preposition in the examples in

[13]In the annotated dataset which will be presented in Section 7.1.1, *Haus* is the head of 161 compounds, which are annotated with 19 different semantic relations.

Table 2.4 (*aus* 'of' and *für* 'for', respectively). The property LOCATION is associated with two different prepositions that further specify the spatial arrangement: *Baumhaus* 'tree house' refers to a house that is located *in* 'in' a tree, whereas *Eckhaus* 'corner house' signifies a house that is located *an* 'on' the corner. In cases where the same preposition is associated with more than one property, the property serves to further disambiguate the meaning of the preposition. The preposition *für* 'for', associated with the properties USER and GOODS in the examples in Table 2.4, is a case in point.

This multi-way mapping between the properties and the prepositions in the annotation scheme can be explained by the fact that the set of prepositions outnumbers the set of properties: Table 2.5 presents an overview of the inventory of properties and prepositions used by the new annotation scheme. The properties are used to label the relation between the immediate constituents of a compound and are assumed to be language-independent. The prepositions, on the other hand, are language-specific and therefore need to be specified each time the annotation scheme is applied to a new language. The annotation scheme, in its current instantiation for German, uses 57 properties and 19 prepositions. 26 properties label the compound-internal relation in its **default direction**, by specifying the way the first constituent, the modifier, relates to the second constituent, the head. Examples of such relations are MATERIAL, HABITAT, HYPONYM, etc. E.g. in *Lederschuh* 'leather shoe' the modifier *Leder* 'leather' names the MATERIAL that is used for making the *Schuh* 'shoe'. These relations have only appeared in the default direction in the dataset. However, given the data-driven methodology used for creating the annotation inventory, the possibility that some of them might have inverses should not be excluded. A point in case is the compound *Geigenholz* 'violin wood', which is not part of the current dataset. Here the head *Holz* 'wood' names the MATERIAL used for crafting the modifier *Geige* 'violin', so a suitable property would be MATERIAL*, where * is used to indicate the inverse direction of a relation. However, because this compound is not part of the dataset, the inverse relation corresponding to MATERIAL is not part of the current inventory.

The inventory includes 14 bi-directional properties, i.e. where the modifier and the head can swap the roles they play in the semantic relation: e.g. in *Kinderchor* 'children chorus', the modifier *Kinder* 'children' denotes the members of the group denoted by the head *Chor* 'chorus', relation annotated as MEMBER. In the case of *Marinesoldat* 'marine soldier', the situation is reversed: the head *Soldat* 'soldier' is a member of the denotatum of the modifier, *Marine* 'marine', relation annotated as MEMBER*.

Althogh far fewer, there are also 3 properties that appear only in the inverse sense - these are ACCESS*, STORAGE* and RELATION*. E.g. in the case of *Bücherregal* 'bookshelf', the head *Regal* 'shelf' is used as STORAGE for the modifier *Bücher* 'books'. They are considered inverse relations because they specify how the head connects to the modifier.

Identifying the directionality of the relation has also been undertaken in previous studies, like Levi (1978) and Ó Séaghdha (2008). If the relation and its inverse are collapsed into a single unit, the inventory then contains 43 distinct categories and is comparable in terms of size to the inventories proposed by Warren (1978), Rosario and

Hearst (2001), Nastase and Szpakowicz (2003) or Tratz and Hovy (2010). As a side note, Ó Séaghdha (2008) also collapses the direct and inverse of the predicates in his inventory (containing 6 actual semantic categories). The decision to collapse the two directions into one label, was, in Ó Séaghdha (2008) case, motivated by the relatively small size of his compound dataset, and Ó Séaghdha and Copestake (2013) also report the (more modest) results for the uncollapsed categories. Collapsing the two directions and identifying the semantic relation as being the same in the two cases makes sense from a semantic point of view. As illustrated in the example above, the membership of one constituent to the group denoted by the other constituent can be easily established, irrespective of the role played by each constituent in the context of the compound. The experiments in Chapter 7 will report on the results of the automatic classification using both collapsed and uncollapsed categories.

	Property Labeling the Semantic Relation	**Associated Prepositions in German**	**Dataset %**	**Example**
	Composition Group		**17.05%**	
1	COMPONENT	mit	0.65%	*Chlorwasser* 'chlorine water'(with)
2	INGREDIENT	aus,mit	2.50%	*Nudelsalat* 'pasta salad'(with)
3	INGREDIENT*	für	0.16%	*Kaffeemilch* 'coffee creamer'(for)
4	MEMBER	aus	0.66%	*Kinderchor* 'children chorus'(of)
5	MEMBER*	von	0.79%	*Marinesoldat* 'marine soldier'(of)
6	PART	mit,aus	5.61%	*Pendeluhr* 'pendulum clock'(with)
7	PART*	von	6.68%	*Pferderücken* 'horseback'(of)
	Descriptive Group		**15.61%**	
8	APPEARANCE	mit,wie	3.40%	*Babypuppe* 'baby doll'(like)
9	APPEARANCE*	wie	0.86%	*Mauerring* 'ring wall'(like)
10	ATTRIBUTE	als,an,für,in,mit, keine_prep,von	4.16%	*Berufssoldat* 'professional soldier'
11	CONSTRUCTION METHOD	in,mit,wie	0.27%	*Etagenbett* 'bunk bed'(in)
12	CONSISTENCY	mit,wie	0.46%	*Fleischtomate* 'beef tomato'(like)
13	MATERIAL	aus	5.89%	*Lederschuh* 'leather shoe'(of)
14	MEASURE	an,für,in,mit	0.15%	*Literflasche* 'litre bottle'(of)
15	MEASURE*	von,zwischen	0.42%	*Körpertemperatur* 'body temperature'(of)
	Locative Group		**10.04%**	
16	HABITAT	an, auf,bei,in,unter	2.34%	*Wasserfrosch* 'water frog'(in)
17	LOCATION	an,auf,bei,durch, in,nach,über,um, unter,zwischen	5.10%	*Kirchenkonzert* 'church concert'(in)
18	LOCATION*	auf,durch,in,um	0.95%	*Schlossberg* 'castle hill'(on)
19	PROTOTYPICAL PLACE OF USE	an,auf,in,unter	1.65%	*Gartenstuhl* 'garden chair'(in)
	Origin Group		**3.08%**	
20	EPONYM	aus,nach,von	1.21%	*Sachertorte* 'Sacher cake'(of)
21	ORIGIN	aus,von	1.87%	*Angorakatze* 'Angora cat'(from)
	Temporal Group		**3.09%**	
22	OCCASION	an,bei,für,in,zu	1.47%	*Abendkleid* 'evening dress'(for)
23	TIME	an,aus,bei,für, in,um,von,zu	1.62%	*Abendblatt* 'evening newspaper'(in)
	Usage Group		**30.70%**	
24	ACTIVE USE	für	4.68%	*Ballettschuh* 'ballet shoe'(for)
25	CONTAINER	aus,für,in	0.32%	*Dosensuppe* 'canned soup'(in)
26	CONTAINER*	für	2.66%	*Brotkorb* 'bread basket'(for)
27	CONTENT	aus,mit,von	2.14%	*Adressbuch* 'address book'(with)
28	FUNCTION	als	2.71%	*Hausboot* 'house boat'(as)
29	FUNCTION*	als	0.26%	*Landbrücke* 'land bridge'(as)
30	MANNER OF FUNCTIONING	durch,für,in,mit, über,wie,keine_prep	2.77%	*Kohleherd* 'coal stove'(with)

	Property Labeling the Semantic Relation	Associated Prepositions in German	Dataset %	Example
31	PRODUCTION METHOD	an,aus,durch,in, mit,nach	0.61%	*Pfannkuchen* 'pancake'(in)
32	PROTOTYPICAL HOLDER	an,auf,in,um	1.30%	*Haarschleife* 'hair ribbon'(in)
33	PROTOTYPICAL HOLDER*	an,für	0.22%	*Kleiderpuppe* 'mannequin', lit. 'clothes doll'(for)
34	PURPOSE OF USE	für,gegen	3.51%	*Brotmesser* 'bread knife'(for)
35	RESULT OF USE	für	0.60%	*Schneekanone* 'snow cannon'(for)
36	STORAGE*	für	0.85%	*Bücherregal* 'bookshelf'(for)
37	USER	für	4.47%	*Babybett* 'baby bed'(for)
38	USAGE	für	3.60%	*Konzerthaus* 'concert house'(for)
	Other Properties		**18.61%**	
39	ACCESS*	zu	0.22%	*Kellertreppe* 'basement stairs'(to)
40	CAUSE	aus,durch	0.19%	*Vulkaninsel* 'volcanic island', lit. 'volcano island'(by)
41	CAUSE*	durch	0.10%	*Regenwolke* 'rain cloud'(by)
42	COMPARISON	wie	0.42%	*Satellitenstadt* 'satellite city'(like)
43	COMPARISON*	wie	0.06%	*Karriereleiter* 'career ladder'(like)
44	DIET	keine_prep	0.37%	*Ameisenbär* 'anteater', lit. 'ant bear'
45	DIET*	für	0.11%	*Gänsedistel* 'boar thistle', lit. 'geese thistle'(for)
46	GOODS	für,mit	0.66%	*Blumenladen* 'flower shop'(with)
47	HYPONYM	keine_prep	1.64%	*Eichenbaum* 'oak tree'
48	OWNER	von	2.50%	*Metzgerladen* 'butcher's shop'(of)
49	OWNER*	mit,von	0.21%	*Grundeigentümer* 'landowner'(of)
50	PRODUCT	für,mit	1.70%	*Apfelbaum* 'apple tree'(with)
51	PRODUCT*	von	0.60%	*Hühnerei* 'chicken egg'(from)
52	RAW PRODUCT	aus,von	2.29%	*Lavandelöl* 'lavender oil'(from)
53	RELATION*	von	0.85%	*Chorleiter* 'choir leader(of)'
54	SPECIALIZATION	für	3.77%	*Augenarzt* 'eye doctor'(for)
55	TOPIC	für,über	2.92%	*Klavierschule* 'piano school'(for)
56	NO PROPERTY	keine_prep	**0.72%**	*Eselsbrücke* 'mnemonic', lit. 'donkey bridge'
57	OTHER PROPERTY	keine_prep	**1.06%**	*Pfandflasche* 'deposit bottle'

Table 2.5 Semantic relation inventory. The percentages and examples listed here come from the annotated German dataset presented in more detail in Section 7.1.1.

The properties and prepositions in the new annotation scheme were derived in a data-driven manner, starting from naturally-occurring examples of compounds. The compound dataset was obtained by extracting compounds headed by concrete nouns from the German wordnet GermaNet (Hamp and Feldweg, 1997; Henrich and Hinrichs, 2010). Annotating compounds from GermaNet had several advantages: the compounds in GermaNet have already been split into their immediate constituents (Henrich and Hinrichs, 2010). In addition, each constituent is labeled with its part of speech and its role - whether it is the head or the modifier of the compound.

The particular choice of head nouns was based on a list by Melinger et al. (2006)[14]. The list is organized into several categories including animals, plants, professions, food, furniture, vehicles, buildings, clothing, tools, etc. Additional head nouns were extracted from GermaNet from similar semantic categories. The motivation for starting with

[14] `http://www.psycholing.es.uni-tuebingen.de/nag/index.php`, last accessed April 27, 2017.

the class of concrete head nouns is that their associated properties are relatively easy to identify and therefore also to annotate. In some cases, the chosen heads had both concrete and abstract senses. An example of this type is *Zug*, meaning 'train' in a concrete sense but which has an assortment of abstract meanings like 'trait', 'feature', etc. In such cases the annotation focused only on those compounds with the head in a concrete sense. The chosen heads also include ambiguous nouns where both senses refer to a concrete object, like *Schloss*, which can mean either 'castle' or 'lock'. In this case compounds relating to both senses were annotated.

The decision to annotate only concrete nouns brings with it an important implication: the annotation scheme, in its current form, can only be used to label concrete compounds, similar to the already annotated ones. No assumptions can be made that using the annotation scheme presented here one will be able to annotate all the potential compounds of a language. The goal of the annotation effort was to analyze the patterns that appeared in the naturally-occurring compounds that were investigated, and advance the understanding of the mechanisms at work for this subset of concrete compounds.

The annotation was performed on a per head basis: all compounds with the same head from the data were analyzed and grouped semantically as illustrated for the head *Haus* 'house' in Table 2.4. This allowed the annotators to identify patterns in the annotation of similar heads: for example a head that denotes an animal, like *Maus* 'mouse' is likely to form compounds that will identify its HABITAT (*Feldmaus* 'field mouse') or its APPEARANCE (*Streifenmaus* 'striped mouse'), categories that will also be encountered when annotating other heads denoting an animal: e.g. *Eisfuchs* 'polar fox', *Wasserspinne* 'water spider' - HABITAT, *Silberfuchs* 'silver fox', *Kreuzspinne* 'cross spider' - APPEARANCE.

The annotation scheme was designed for annotating noun-noun compounds. In some cases, the compounds extracted from GermaNet were assigned two modifiers, a noun and a verb. An example of this type is the compound *Badeanzug* 'bathing suit', where both the noun *Bad* 'bath' and the verb *baden* 'to bathe' are listed as modifiers. The annotation in this case considers only the noun to be the modifier, and assigns the (property, preposition) label that characterizes the relation between the nominal modifier and the head. Nominalized verbs are allowed both as heads (e.g. *Arbeiter* 'worker' from *arbeiten* 'to work' in *Bauarbeiter* 'construction worker') and as modifiers (e.g. *Benutzer* 'user' from *benutzen* 'to use' in *Benutzerkonto* 'user account'). The annotation is in accord with the observation made by Ó Séaghdha (2008), that compounds having nominalizations as constituents and root compounds should receive compatible analyses. This is in contrast to previous proposals like Levi (1978) and Lauer (1995) that considered nominalizations a separate group, leading to different analyses for similar compounds like *history professor* and *history teacher*.

The annotation inventory listed in Table 2.5 is the result of a long process of refinement of the annotation scheme. The initial version of this annotation scheme was presented in Dima et al. (2014), and contained 37 properties (6 of which with direct/inverse directions) and 17 prepositions. The inventory and the associated

annotation guidelines were further extended and improved for the new annotation scheme version. In some cases, the name of the category was kept, but the definition and the distinguishing properties of the category were changed to better fit the patterns observed in the data. The refining step also took into account the lessons learned from a set of machine learning experiments presented in Sorokin et al. (2015), that used a dataset of compounds annotated with the initial version of the annotation scheme. Such refinement steps are common when designing new annotation schemes (Girju et al., 2005; Tratz and Hovy, 2010).

The annotation scheme design criteria identified by Ó Séaghdha (2008) were instrumental in choosing the properties to be included in the inventory. In particular, the proposed inventory contains a coherent set of properties and tries to establish clear boundaries between the different properties. Another desideratum is to reduce to a minimum the cases where two properties might seem appropriate. The annotation guidelines contain several tests to help distinguish among two competing properties. An example of this type is the test distinguishing the property APPEARANCE* from the property MATERIAL. APPEARANCE* labels compounds where the modifier denotes a material, and the head evokes (the shape of) the object made from that material (like in *Schokoladenhase* 'chocolate bunny'). In contrast, the property MATERIAL labels the compounds where the head not only evokes a shape but refers to the actual object, while the modifier names the material it is made of (like in *Holzlöffel* 'wooden spoon'). The annotation principles, as well as the criteria used for identifying and distinguishing between properties are presented in a comprehensive set of annotation guidelines (Telljohann et al., 2017).

In some of the cases, the constituents of the compound or the compound itself allow for multiple interpretations. For the new annotation scheme, if the annotators were aware of multiple interpretations of the same compound, they had to choose one of the senses as the main sense and annotate it. The properties and prepositions associated with the additional interpretations would then be added to a separate list. An example of this type is the compound *Apfelring* 'apple ring', which in its primary sense means 'thin, dried pieces of apple, cut lengthwise from the cored fruit' and is annotated as RAW PRODUCT/*aus* 'from'. In a secondary sense, *Apfelring* refers to a 'sour candy with apple flavor, similar to gummy bears'. This second sense is annotated as INGREDIENT/*aus* 'from'.

Lexicalization. Another particular aspect of the annotation scheme is the annotation of **idiomatic compounds** such as *Löwenzahn* 'dandelion', literally 'lion tooth'. Similar to Ó Séaghdha (2008), these compounds are annotated according to the relation between the components in the absence of an idiomatic meaning, e.g. *Löwenzahn* is annotated as PART*/*von* 'of'. In addition, however, such compounds receive a lexicalization annotation. The lexicalization annotation is meant to capture the different ways in which the meaning of a compound deviates from its compositional meaning.

Compound	**Property/Preposition**	**Lexicalization pattern**
Pfeffer*kuchen* 'gingerbread', lit. 'black pepper cake'	ZUTAT/*mit* 'with'	`lex_M`
Finger*beere* 'fingerpad', lit. 'finger berry'	PART*/*von* 'of'	`lex_H`
Bull*auge* 'porthole', lit. 'bull's eye'	PART*/*von* 'of'	`lex_M`,`lex_H`, `lex_R`
Schwert*fisch* 'swordfish'	PART/*mit* 'with'	`lex_MS`
Kirsch*wasser* 'cherry brandy', lit. 'cherry water'	RAW PRODUCT/*aus* 'from'	`lex_HS`

Table 2.6 Lexicalization annotation for partially- or fully non-compositional compounds.

Table 2.6 illustrates the different lexicalization patterns identified by the annotation scheme. The label `lex_M` marks cases when the modifier is used quasi-metaphorically, as in the compound *Pfefferkuchen*: *Pfeffer* does refer to black pepper, but to a spice mix used to flavor the cake (which does not contain black pepper). Similarly, `lex_H` labels cases where the head of the compound undergoes a similar process: the *Beere* in *Fingerbeere* does not refer to a 'berry', but to the soft pad of flesh at the top of a finger.

The compound *Bullauge*[15] illustrates the case when, at a first glance, the compound seems perfectly compositional, but the actual referent has no visible connection to the meaning of the constituents (in this case, 'bull' and 'eye'). The lexicalization annotation in Table 2.6 marks in this case both the atypical use of the constituents via `lex_M` and `lex_H`, as well as the atypical nature of the relation between them, via the label `lex_R`. An seen in this example, the lexicalization labels can be freely combined to annotate the lexicalization pattern of any compound.

Another two labels that can be used are `lex_MS` and `lex_HS`, which label an atypical sense of either the modifier or the head, but where there is a clear resemblance relation between the canonical meaning of the constituent and the one used in the compound. Examples of this type are *Schwertfisch* 'sword fish', where the modifier 'sword' names the long, pointy snout of the fish which resembles a sword, and *Kirschwasser* 'cherry brandy', where the head *Wasser* 'water' is used to denote the similarity between this type of brandy and water (both are colorless).

In Chapter 6 this lexicalization annotation is used to compare the automatic 'compositionality' score assigned using a composition model to the judgements made by the human annotators.

[15]Small, round window in the hull of a ship/air plane; name for the door of a washing machine.

No preposition. The annotation scheme presumes that every compound is annotated with at least one property and one preposition. There are, however, cases when the compound does not license a preposition, like in the case of the compound *Lachsfisch* 'salmon fish', annotated with the property HYPONYM in Table 2.5. In general, compounds annotated with the semantic relation HYPONYM do not allow for a prepositional paraphrase, as also observed by Warren (1978) and Lauer (1995). Rather than not providing a preposition label for a subset of the compounds, the annotation uses the label *keine_Präposition* 'no preposition' to mark such compounds. The explicit annotation with the *no preposition* label is very convenient from an automatic classification perspective, giving the classifier the possibility to identify types of compounds that do not take a preposition. As Table 2.2 shows, the *keine_prep* 'no preposition' label is typical for compounds with the property HYPONYM and DIET and for a part of the compounds annotated as ATTRIBUTE and MANNER OF FUNCTIONING.

No/other property. In a similar vein, the label NO PROPERTY is used to annotate compounds where the semantic relation between the two constituents has lost so much of its meaning that it is impossible to recover, and its exact origin is uncertain. This is the case with strongly lexicalized compounds like *Meerkatze* 'guenon monkey', lit. 'sea cat'.

Another special-status property label is OTHER PROPERTY. It is used to label compounds whose relation is known, but does not conform to any of the previously identified patterns, like *Pfandflasche* 'deposit bottle'. Most previous annotation schemes (Girju et al., 2005; Ó Séaghdha, 2008; Tratz and Hovy, 2010) include an OTHER relation that can be used to annotate the compounds that do not fall under any of the other categories defined by the inventory. However, since the new annotation scheme does not claim to be exhaustive, the use of the label OTHER PROPERTY is reserved for those compounds where the semantic relation is only applicable to a single constituent pair, or to a very restricted set of compounds which usually share the modifier. OTHER PROPERTY is thus used to label **non-productive** patterns. If the annotation scheme is used to annotate new compounds, the annotators are encouraged to create new properties for the productive semantic relations that they encounter and preserve this property for such 'idiosyncratic constructions' - as (Warren, 1978:241) refers to them.

Elliptical annotation. In compounds like *Buchstabensuppe* 'alphabet soup', lit. 'letters soup', the semantic relation does not directly connect the two constituents of the compounds. Instead, it relates the head to an implied modifier. In the example above, the modifier *Buchstabe* 'letter' is a shorthand for *Buchstabennudel* 'letter-shaped pasta', which is the actual ingredient of the soup. In such cases, the annotation involves reinserting the missing argument, by specifying the elliptical modifier, and assigning the property/preposition label that corresponds to the compound with the reinserted, correct, modifier. Approx. 6% of the compounds in the annotated dataset require the annotation of an elliptical modifier.

2.2.1 The Composition Group

The composition group includes four properties that connect two constituents involved in a composition relation: PART/*, MEMBER/*, INGREDIENT/* and COMPONENT. The properties marked with an /* appear in the inventory of the new annotation scheme both as a direct and as an inverse relation.

The PART relation labels compounds where one of the constituents is a part of the other: *Pendeluhr* 'pendulum clock', annotated as PART/*mit* 'with', where the modifier 'pendulum' is a part of the head 'clock' and *Pferderücken* 'horseback', annotated as PART*/*von* 'of', where the modifier *Pferd* 'horse' has as part the head *Rücken* 'back'. The two compound constituents can belong to a very diverse set of semantic categories: they can be artifacts, animals or body parts but also plants, buildings or clothing. Together, the PART relation and its inverse are the most frequent category in the annotated dataset, labeling approx. 12% of the data (see Section 7.1.1, describing the annotated dataset, for details).

The MEMBER relation relates two constituents that are not physically part of one another. It indicates that the entity denoted by one of the constituents is, conceptually, a part of the entity or group denoted by the other constituent. Examples are *Kinderchor* 'children chorus', annotated as MEMBER/*aus* 'of' and *Marinesoldat* 'marine soldier', annotated as MEMBER*/*von* 'of'. As seen from the examples, this property is more restrictive in terms of the semantic categories allowed for the constituents: one constituent must have the +Group feature for this property to apply.

INGREDIENT labels the relation between a food item and an ingredient (another food item) that is used in its making: e.g. *Nudelsalat* 'pasta salad' annotated as INGREDIENT/*mit* 'with'; *Kaffeemilch* 'coffee creamer', lit. 'coffee milk', annotated as INGREDIENT*/*für* 'for'. The constituents can refer either to solid food or to drinks. The INGREDIENT property imposes stronger restrictions on the semantic class of the constituents: both are expected to have an +Edible feature for the label to apply.

The COMPONENT relation describes the case when the modifier is a part of the head but cannot be seen with the naked eye: e.g. *Chlorwasser* 'chlorine water', *Vanilinzucker* 'vanillin sugar'.

2.2.2 The Descriptive Group

The descriptive group contains six properties: MATERIAL, APPEARANCE/*, ATTRIBUTE, CONSTRUCTION METHOD, CONSISTENCY and MEASURE/*. The most frequent property from this group in the annotated dataset is MATERIAL, which labels close to 6% of the compounds. The modifier can be either generally recognized as a material, like in *Lederschuh* 'leader shoe', or it can be used to denote the material only in the compound's context: e.g. *Daunenkissen* 'down pillow', *Drahtbügel* 'wire coat hanger'.

The APPEARANCE property is used to label compounds where the modifier describes the outer appearance of the head, e.g. *Babypuppe* 'baby doll'. The property can also be used in its inverse form, where the second constituent provides a description for

the first constituent: *Mauerring* 'ring wall', lit. 'wall ring' denotes a wall shaped as a ring. APPEARANCE* is also used to label compounds like *Schneemann* 'snowman' and *Schokoladenhase* 'chocolate bunny', where the first constituent names a material that is formed to resemble the entity denoted by the second constituent.

Another property where the modifier is related via analogy with the head is CONSISTENCY, which labels compounds like *Fleischtomato* 'beef tomato', lit. 'meat tomato'. In this case the modifier refers to the consistency of the head rather than to its outer appearance as in the case of APPEARANCE.

The ATTRIBUTE property connects a descriptive modifier to the head, e.g. *Berufssoldat* 'professional soldier'. The modifiers of the compounds labeled by this property will, in many cases, be translated into English via adjectives. Warren (1978) calls such modifiers *adjective-like comment-nouns*, and shows that the sense of these nouns in compounds can be readily described via adjectives. Examples she gives for English are *key* (*key fact*), *record* (*record time*), *root* (*root issue*). Similar examples from the German dataset are: *Traum* 'dream' in *Traumfrau, Traummann* 'ideal woman/man', *Traumauto* 'ideal car'; *Schutz* 'protection' in *Schutzmauer* 'defensive wall', *Schutzleiter* 'wire with ground protection' lit. 'protective conductor'. Another defining trait of such modifiers is that they preserve the sense in which they are used in most compounds they are part of.

CONSTRUCTION METHOD labels compounds where the modifier describes a particular aspect in the construction of the head: e.g. *Etagenbett* 'bunk bed', lit. 'story bed'.

The MEASURE property connects two constituents where the first constituent names a measure for the second. It can refer to standardized measurements (e.g. *Literflasche* 'liter bottle') or ad-hoc ones (e.g. *Kniestrumpf* 'knee sock'). The inverse property, MEASURE*, labels cases when the head denotes a measurement for the modifier: e.g. *Körpertemperatur* 'body temperature'.

2.2.3 The Locative Group

The locative group contains three properties: LOCATION/*, HABITAT and PROTOTYPICAL PLACE OF USE. They relate the modifier and the head in terms of their spatial properties.

In the case of LOCATION the modifier is the location of the head's denotatum. The head can be a building (e.g. *Berghütte* 'mountain hut'), an object (e.g. *Seitentür* 'side door') or an event (e.g. *Kirchenkonzert* 'church concert'). The property is also used to locate body-related compounds such as *Rückenhaar* 'back hair' or *Kopfhaar* 'head hair'. In its inverse sense the property labels compounds where the head specifies the location of the modifier: e.g. *Schlossberg* 'castle hill'.

LOCATION refers to fixed locations, from which the entities denoted by the constituents are rarely, if ever, removed. By contrast PROTOTYPICAL PLACE OF USE labels compounds where the modifier names the typical place of use for the head, e.g. *Gartenstuhl* 'garden chair'. The prototypical aspect of this relation comes into play

as the compound is called with the same name even when removed from the location named by the modifier: a *garden chair* remains a *garden chair* even if it happens to be on the balcony. The name does not change with this change of location. Moreover, the typical location of use has a big impact on the appearance and characteristics of the object: a *garden chair* will have different characteristics from an *office chair*.

The HABITAT property connects the constituents of a compound where the head is an animal or a plant, and the modifier represents its natural habitat. The modifier can name either a typical location, e.g. *Alpenpflanze* 'Alpine plant', a type of environment, e.g. *Wasserfrosch* 'water frog' or a plant, e.g. *Himbeerkäfer* 'raspberry beetle'. In the last case, the plant usually serves both as the habitat and as the main food source of the entity denoted by the head.

2.2.4 The Origin Group

The origin group contains two properties: ORIGIN and EPONYM. Both label compounds where the modifier describes the origin of the head. The ORIGIN can refer to the place (e.g. *Angorakatze* 'Angora[16] cat') or time period (e.g. *Barockfassade* 'Baroque façade') something originates from. EPONYM labels compounds where the head is named after its inventor, e.g. *Sachertorte* 'Sacher cake' is a cake named after its inventor, Austrian Franz Sacher.

2.2.5 The Temporal Group

The temporal group encompasses two properties: TIME and OCCASION. The property TIME is used to label compounds where the modifier specifies the time point (e.g. *Abendblatt* 'evening newspaper') or time interval (e.g. *Sommerreifen* 'summer tires') when the head occurs, is produced or is used. Due to the temporal reference, the set of modifiers is restricted to time-denoting nouns: names of days, time of days, seasons, time periods, religious holidays etc.

The property OCCASION labels compounds where the modifier specifies the (festive) occasion for which the head is created (e.g. *Geburtstagskuchen* 'birthday cake'), used (e.g. *Adventskerze* 'Advent candle'), worn (e.g. *Abendkleid* 'evening dress') or given (e.g *Hochzeitsgeschenk* 'wedding gift'). In addition to the set of time-denoting nouns used also by the TIME property, the modifiers labeled by the property OCCASION also include names of special events in a person's life (e.g. birthday, wedding, vacation, beginning/ending of school/university etc.).

[16] *Angora* is the historic name of Ankara, the capital of Turkey. The initial compound with this modifier was 'Angora goat', a goat with long, silky fur. The modifier *Angora* has taken on a second sense, meaning 'with long fur', both in the original compounds, and also in compounds using the original compound as a pattern, e.g. 'Angora ferret'.

2.2.6 The Usage Group

The usage group is by far the largest group of properties of the new annotation scheme, encompassing 12 properties: ACTIVE USE, CONTAINER/*, CONTENT, FUNCTION/*, MANNER OF FUNCTIONING, PRODUCTION METHOD, PROTOTYPICAL HOLDER/*, PURPOSE OF USE, RESULT OF USE, STORAGE*, USER and USAGE. All these properties are connected by the fact that they describe a facet of usage. This group is a clear illustration of the fact that compounding serves as a naming device, and that things are generally defined in terms of their intended use.

The property ACTIVE USE connects a modifier expressing an activity to the head denoting the object used for that activity. A typical example is the compound *Ballettschuh*, 'ballet shoe', where the head is an artifact. The head can also be a physical location, e.g. *Arbeitszimmer* 'study', lit. 'work room'.

CONTAINER is a bidirectional property expressing the relation between the two constituents where one is a container and the other is its content. In its direct sense, CONTAINER labels compounds where the content is packaged in the container: e.g. *Dosensuppe* 'canned soup', lit. 'can soup'. The inverse relation, CONTAINER*, labels containers that are manufactured with a particular content in mind, e.g. *Brotkorb* 'bread basket', *Bierflasche* 'beer bottle', etc.

In the case of the CONTENT relation, the modifier refers to the typical content of the head: e.g. *Adressbuch* 'address book'. In this case the head itself is not a type of container, but rather a visual or textual artifact where the content denoted by the modifier is depicted or stored.

The FUNCTION/* property refers to a group of compounds where one constituent specifies the functions that the other constituent takes on, in addition to the ones it inherently has. For compounds labeled with FUNCTION, the modifier is the source of the additional attributes taken on by the head: i.e. a *Hausboot* 'house boat' is a boat were one can cook, sleep, or work - in other words - live in, like in a house. In the case of compounds labeled by FUNCTION* like *Landbrücke* 'land bridge', it is the modifier that takes on additional functional attributes entailed by the head - a *land bridge* is a strip of land that serves as a bridge between two larger land areas separated by water.

MANNER OF FUNCTIONING connects a modifier denoting a power source to a head denoting, in most cases, a man-made artifact. The modifier can be a natural combustible - *Kohleherd* 'coal stove', an animal - *Hundeschlitten* 'dog sledge', or denote the payment necessary for the head to function, e.g. *Münztelefon* 'pay phone', lit. 'coin phone'. Similarly, PRODUCTION METHOD labels compounds where the modifier denotes the method through which the head was created: e.g. *Pfannkuchen* 'pancake'.

The property PROTOTYPICAL HOLDER/* links constituents denoting artifacts to the place they were designed to adorn. The holder can be either the modifier, e.g. *Haarschleife* 'hair ribbon' or the head, e.g. *Kleiderpuppe* 'mannequin', lit. 'clothes doll'.

PURPOSE OF USE and RESULT OF USE describe the compound-internal relationship where the modifier is the purpose (e.g. *Brotmesser* 'bread knife') or the result (e.g.

Schneekanone 'snow cannon') of using the head. In general, the same head will be connected to different modifiers with the same semantic class, resulting in a typology of the head - see, for example, the additional compounds with the head *Messer* 'knife': *Fischmesser* 'fish knife', *Käsemesser* 'cheese knife', *Fleischmesser* 'meat knife', where all modifiers name things that can be cut using the knife.

In the case of the property STORAGE*, the head specifies a place where the things denoted by the modifier are typically stored, e.g. *Bücherregal* 'bookshelf'. The property relates semantically to the property CONTAINER*, however, the heads are not typically regarded as containers. They assume a container functionality only when they are part of this type of compounds.

The property USER connects a modifier denoting a person, group of persons or animal to a head denoting some building or artifact that it uses, e.g. *Babybett* 'baby bed'. In many cases the head undergoes an accommodation process, and the meaning of the head in the resulting compound can drift quite far from the initial meaning of the head, like it is the case for *Hundehütte* 'dog house', literally 'dog hut' (a *Hundehütte* will be much smaller in size than a typical *Hütte* 'hut').

The property USAGE labels compounds where the modifier names the designated usage for the head, e.g. *Konzerthaus* 'concert house'. In contrast to the property PURPOSE OF USE, in this case the head is not meant for performing some action on the modifier. The head is in most cases a place/building where the modifier is performed/performs its action. The property is also used to connect something that is being transported to its transport route, e.g. *Autostraße* 'car road'.

2.2.7 Other properties

Twelve other properties were considered too specific to be assigned to any of the above mentioned groups, and are going to be individually introduced in this section.

The property ACCESS* labels compounds where the head is the means for reaching or accessing the place denoted by the modifier. The modifier can be a closed or open space, e.g. *Kellertreppe* 'basement stairs', *Gartentür* 'garden gate', lit. 'garden door'. The head can be a path leading to a place, like *Straße* 'street ' in *Klosterstraße* 'monastery street'.

The CAUSE/* property connects a constituent naming (usually) a phenomenon to the head that is regarded as its cause. In *Vulkaninsel* 'volcanic island' lit. 'volcano island', the activity of the volcano leads to the creation of the island. Conversely, in *Regenwolke* 'rain cloud', the cloud is regarded to be the cause of the rain.

COMPARISON/* labels compounds where one of the constituents is compared to the other. In contrast to APPEARANCE/*, this property does not refer to the outer appearance of the constituents, but relates them in terms of more abstract characteristics. The comparison can be related to their function, e.g. *Satellitenstadt* 'satellite city', but also to more abstract traits, e.g. *Rabentochter* 'cruel daughter', lit. 'raven daughter'. In a small number of cases the inverse relation, COMPARISON*, also appears, e.g. *Karriereleiter* 'career ladder'.

The DIET/* property connects an animal or insect with a type of food (typically another animal, insect or plant). Cases where the head denotatum has genetically evolved to eat the modifier denotatum are annotated with DIET, e.g. *Ameisenbär* 'ant eater', lit. 'ant bear'. Cases where the modifier denotatum is particularly fond of eating the head denotatum (although it is not its main food source), are annotated as DIET*, e.g. *Gänsedistel* 'boar thistle', lit. 'geese thistle'.

The property GOODS labels compounds where the modifier is the merchandise of the place named by the head, e.g. *Blumenladen* 'flower shop'. The head might not be conventionally regarded as a selling location but it acquires this additional meaning through the combination with the modifier, e.g. *Bierzelt* 'beer tent'.

The HYPONYM relation connects modifiers that denote an individual from a family to a head denoting its family (in the biological sense), e.g. *Eichenbaum* 'oak tree'. In most cases, however, the modifier alone will have the same meaning as the compound - *Eiche* 'oak' and *Eichenbaum* 'oak tree' are synonyms.

OWNER/* connects a constituent specifying an owner to the thing it owns. In the case of OWNER, the modifier can also name a profession exercised (historically) by the owner of the place, e.g. *Metzgerladen* 'butcher's shop'. The reverse direction, OWNER*, labels compounds where the modifier names the thing that is owned, which becomes a categorization factor for the entity denoted by the head, e.g. *Grundeigentümer* 'landowner'.

The category PRODUCT/* connects a producer to the thing it produces. The method of production can be natural - like a tree producing fruit, e.g. *Apfelbaum* 'apple tree', or industrial - like a factory producing artifacts, e.g. *Brotfabrik* 'bread factory'. The inverse relation, PRODUCT* displays the same patterns, this time with the modifier naming the producer: e.g. *Hühnerei* 'chicken egg', *Vereinszeitung* 'association's newspaper'.

RAW PRODUCT labels compounds where the head is obtained from the modifier, usually through an industrial process, e.g. *Lavandelöl* 'lavender oil', *Grapefruitsaft* 'grapefruit juice', *Maismehl* 'corn flour'.

RELATION* connects a modifier denoting a group to a head denoting a special function for that group, e.g. *Chorleiter* 'choir leader'. In many cases only one person can assume the specified function at a given time.

SPECIALIZATION connects a head denoting a person or animal to an activity that it specializes in. The activity can be specified either directly, e.g. *Bauarbeiter* 'construction worker', *Jagdhund* 'hunting dog' or via the main item of interest, e.g. *Augenarzt* 'eye doctor'.

TOPIC connects a modifier that denotes the thematic content of the head, e.g. *Klavierschule* 'piano school'. The subset of heads that can participate in a TOPIC relation are knowledge-containing artifacts like books, newspapers, magazines or knowledge-spreading locations, like schools, universities, academies etc.

2.2.8 Inter-annotator Agreement Study

To verify the consistency of new annotation scheme, an inter-annotator agreement (IAA) study was conducted. 44 head nouns and the corresponding 481 compounds were selected for the study. To make the evaluation more indicative, the selection was limited to previously unannotated heads. The modifier set, however, which consists of 387 unique modifiers, overlaps in proportion of 79.84% (309 modifiers) with the modifier set of previously annotated compounds.

The IAA heads were selected from GermaNet, using similar criteria to the ones used for the previously annotated heads (see description at the beginning of Section 2.2). In order to be selected, the head should have at least a concrete sense. Only the compounds corresponding to the concrete sense(s) of the head were selected for annotation.

The main reasons for choosing the IAA compounds in this way - and not use a random sample of the GermaNet compounds - are two-fold: first, the annotation scheme was designed for concrete nouns, and the selection had to contain exclusively compounds denoting concrete objects. Second, the annotation guidelines specify that for a more consistent annotation the compounds with the same head should be annotated together. This meant first picking the heads and then selecting all the non-abstract compounds from GermaNet formed with those heads.

The compounds were annotated by two annotators, native speakers of German. One had extensive experience with using the current version of the annotation scheme, while the other was familiar with the annotation setup, and had annotated compounds before, but not using the current inventory. An initial familiarization step was required for the second annotator. It involved annotating 100 of the already annotated gold compounds, and discussing the annotation with the experienced annotator. This step was necessary as the annotation inventory is relatively large, and annotating some training compounds gives the starting-out annotator a better grasp on what are the distinctions that need to be made.

The annotators then proceeded to independently annotate the 481 compounds. Again, the compounds were annotated per-head: the annotator was aware of the whole cluster of compounds with the same head while performing the annotation. The annotation for each compound consisted of the property, the preposition(s), and, where necessary, the elliptical modifier, the lexicalization pattern(s) and the annotation of the extra sense(s).

The agreement among the annotators was measured using Cohen's Kappa coefficient (Cohen, 1960), defined as

$$\kappa = \frac{A_0 - A_E}{1 - A_E} \tag{2.1}$$

In Eq. 2.1, A_0 represents the proportion of annotations the annotators agree on, while A_E is the proportion of annotations for which the agreement is expected by chance. In other words, the κ coefficient computes the proportion of agreement after the agreement due to chance is removed.

If the annotation guidelines are very clear about when to use a particular property and preposition label, then A_0 should be well above chance, and result in a κ coefficient that is close to the theoretical maximum, which is 1.

Separate κ values were computed for the property and the preposition annotation. The property annotation result is $\kappa = 0.59$, which is very close to a *good* agreement according to the classification of Kappa coefficients proposed by Landis and Koch (1977)[17]. The preposition annotation resulted in a *good* agreement with $\kappa = 0.65$, for the subset of 467 compounds which were assigned exactly one preposition.

Although the theoretical upper limit of κ is 1, in practice this is only achieved if the two annotators assign the categories with the same probability distribution. In practice, however, annotators might exhibit biases towards different categories. The actual maximum Kappa, κ_M, can be computed for two particular annotators, using the formula in Eq. 2.2,

$$\kappa_M = \frac{A_M - A_E}{1 - A_E} \tag{2.2}$$

A_M is obtained by pairing the marginals for each annotated category, taking the minimum of each pair and summing the minimum values. In this study, the values for the maximum agreement are $\kappa_M = 0.80$ for the property annotation and $\kappa_M = 0.92$ for the preposition annotation.

Table 2.7 and Table 2.8 present a detailed account of the IAA agreement, by looking individually at each of the categories.

Property IAA. In the case of the property annotation, one immediate observation is that only 47 out of the 57 properties in the inventory were used for annotating the IAA subset of compounds. Overall, a couple of trends can be observed:

- the most frequently annotated property is ATTRIBUTE, which was annotated 81 times; the most frequently annotated properties tend to have a below average agreement, around 0.4-0.5.

- the categories with the highest agreement (and with a relatively high frequency of annotation) are categories with a more restricted definition, where the expected features are clearly defined: e.g. MATERIAL (modifier is generally a material), TIME (modifier is a time expression), INGREDIENT (both constituents are edible), CONTENT (modifier denotes the content, head is restricted to visual and textual artifacts where the content is depicted or stored), PART* (the modifier is the whole and the head is the part), OCCASION (the modifier is an event), USER (the modifier is a living being, the intended user of the head).

- the annotators systematically confused particular properties: e.g. TOPIC vs. SPECIALIZATION, where one of the annotators consistently annotated one relation

[17]The *good* range is from 0.6 to 0.8.

	Property	Per property agreement	Total annotations	Nr. agreements	Ann1 only	Ann2 only
1	CONTAINER	1.000	4	2	0	0
2	COMPARISON	1.000	2	1	0	0
3	MEMBER*	1.000	8	4	0	0
4	RELATION*	1.000	2	1	0	0
5	HABITAT	1.000	4	2	0	0
6	MATERIAL	0.972	37	18	1	0
7	TIME	0.915	24	11	1	1
8	INGREDIENT	0.866	23	10	2	1
9	CONTENT	0.865	55	24	3	4
10	PART*	0.860	53	23	3	4
11	OCCASION	0.820	17	7	1	2
12	USER	0.806	32	13	4	2
13	PROTOTYPICAL PLACE OF USE	0.777	23	9	2	3
14	SPECIALIZATION	0.756	52	20	2	10
15	APPEARANCE*	0.748	8	3	2	0
16	INGREDIENT*	0.724	11	4	1	2
17	OWNER	0.724	11	4	2	1
18	PRODUCT	0.694	20	7	2	4
19	ACTIVE USE	0.684	26	9	8	0
20	ORIGIN	0.674	35	12	2	9
21	CONSISTENCY	0.666	3	1	1	0
22	MANNER OF FUNCTIONING	0.664	9	3	2	1
23	APPEARANCE	0.568	7	2	1	2
24	PRODUCTION METHOD	0.568	7	2	2	1
25	CONTAINER*	0.562	38	11	0	16
26	RAW PRODUCT	0.560	58	17	9	15
27	PART	0.548	25	7	5	6
28	ATTRIBUTE	0.528	81	23	31	4
29	MEASURE*	0.517	19	5	3	6
30	PROTOTYPICAL HOLDER	0.498	4	1	1	1
31	HYPONYM	0.487	24	6	5	7
32	LOCATION	0.452	42	10	6	16
33	CAUSE	0.397	5	1	0	3
34	PURPOSE OF USE	0.371	45	9	26	1
35	FUNCTION	0.325	12	2	1	7
36	USAGE	0.203	42	5	9	23
37	NO PROPERTY	0.172	11	1	1	8
38	OTHER PROPERTY	0.145	24	2	10	10
39	COMPONENT	0.142	13	1	5	6
40	PROTOTYPICAL HOLDER*	-0.001	1	0	0	1
41	CONSTRUCTION METHOD	-0.001	1	0	0	1
42	LOCATION*	-0.001	1	0	0	1
43	EPONYM	-0.004	4	0	4	0
44	GOODS	-0.004	4	0	4	0
45	RESULT OF USE	-0.006	6	0	4	2
46	TOPIC	-0.007	7	0	7	0
47	PRODUCT*	-0.023	22	0	15	7

Table 2.7 Agreement per property for the 481 compounds in the IAA study. Total annotations = Nr. agreements * 2 + Ann1 only + Ann2 only.

instead of the other; this inconsistency was due to the annotator disregarding one of the restrictions of the SPECIALIZATION class, namely that the head must be a living being, and using the property for heads denoting buildings; also, the existence of compounds with the same modifier annotated as SPECIALIZATION contributed to this confusion, e.g. *Augenklinik* 'eye clinic' was annotated similarly to *Augenarzt* 'eye doctor'.

- another type of systematic confusion occurred between the properties GOODS vs. CONTAINER*: for compounds like *Zigarettenautomat* 'cigarette vending machine', there is a potential ambiguity between the 'vending machine' acting as a container for the 'cigarettes' and it being the selling place of the 'cigarettes'. The annotation guidelines did not mention this type of ambiguity. It was, however, implied - for the creators of the annotation scheme - that a more specific semantic relation like the one expressed by the GOODS property would take precedence over the more abstract containment relation expressed by CONTAINER*. A similar confusion occurred between PURPOSE OF USE and CONTAINER*, for compounds like *Pfeffermühle* 'pepper mill'.

- the properties RAW PRODUCT and PRODUCT* were the subject of another systematic confusion. This time, however, it was the new head, *Holz*, that was the source of the ambiguity. In compounds like *Eichenholz* 'oak wood' the 'wood' can be seen either as being a natural product of the tree - a PRODUCT* annotation, or as something that is obtained through an industrial process - a RAW PRODUCT annotation. After discussion, the decision was to annotate compounds of this type with the property RAW PRODUCT. However, this type of confusion between the properties links back to the "indeterminacy of analysis" observed by Levi (1978), which noted that for a subset of compounds there is not a clear, principled way of choosing one predicate over another. The PRODUCT* property is also the property with the lowest agreement, the negative value indicating that the agreement is less than expected by chance.

- the property OTHER RELATION appears to be problematic from another perspective: as described in Section 2.2, it targets compounds where the semantic relation is very specific. However, it gives the annotator the freedom to asses the specificity of the relation and decide if it should be annotated with OTHER RELATION or if it is just a borderline case of a more general relation. An example of this type is the compound *Zielfoto* lit. 'target photo', which refers to 'a snapshot of the participants crossing of the finish line in a sports competition'. One annotator chose to label the semantic relation using the property OTHER PROPERTY, while the other annotated an elliptical modifier, *Zieleinlauf* 'finish' and the property CONTENT.

Preposition IAA. The per preposition agreement shows a better correlation between the frequency with which a particular preposition is annotated and the agreement. The most often annotated preposition, *für* 'for', also reaches the maximum agreement score,

	Preposition	Per Preposition agreement	Total annotations	Nr. agreements	Ann1 only	Ann2 only
1	für	0.792	294	126	26	16
2	gegen	0.724	11	4	2	1
3	wie	0.693	20	7	4	2
4	aus	0.690	118	43	13	19
5	durch	0.662	12	4	1	3
6	an	0.661	15	5	2	3
7	mit	0.654	93	32	14	15
8	von	0.627	162	56	29	21
9	in	0.554	65	19	10	17
10	keine_Präposition	0.546	107	32	21	22
11	als	0.298	13	2	2	7
12	auf	0.215	9	1	2	5
13	um	-0.001	1	0	0	1
14	zu	-0.002	2	0	0	2
15	nach	-0.003	3	0	3	0
16	unter	-0.003	3	0	2	1
17	bei	-0.006	6	0	5	1

Table 2.8 Agreement per preposition for the 481 compounds in the IAA study. Total annotations = Nr. agreements * 2 + Ann1 only + Ann2 only.

0.792. It is interesting to compare in this regard the maximum values in Table 2.7 and Table 2.8: the agreement per property reaches close to perfect agreement for categories like MATERIAL and TIME - 0.972 and 0.915, respectively, but is much lower for the most consistently annotated preposition - 0.792. However, overall, the per preposition κ score is better than the property one. Another observation is that in 85% of the cases if the annotators disagree about the preposition annotation, they also disagree about the property label. Table 2.9 presents a comparative disagreement analysis. In most disagreement cases, the annotators disagreed both on the property and the preposition (so on the combined label). Cases where the disagreement affected only the preposition usually refer to a genuine ambiguity: a *Schlaftablette* 'sleeping pill' could be *für* 'for' or *gegen* 'against' sleeping. The cases where the annotators disagreed on the property but agreed on the preposition are generally cases when the same preposition is associated with different properties: e.g. for *Hautklinik* 'skin clinic', both annotators chose the same preposition *für* 'for', but two different properties, TOPIC and SPECIALIZATION. However, both properties allow for a prepositional paraphrase using *für* 'for', so in effect these annotations should also count as cases where they disagreed on the combined label annotation.

Disagreement in	**# of instances**
Both property and preposition	116
Property only	69
Preposition only	20
Total # of property disagreements	185
Total # of preposition disagreements	136

Table 2.9 Disagreement analysis.

The insights of the IAA study were used to improve the annotation guidelines, by including further specifications of the expected annotations where necessary. The

obtained κ scores are similar to the ones reported by Girju et al. (2005) for English compounds ($\kappa = 0.58$ for the annotation with 35 semantic relations, $\kappa = 0.80$ for the annotation with 8 prepositions), but are lower than the scores reported for the initial version of the annotation scheme ($\kappa = 0.74$ for property annotation and $\kappa = 0.75$ for preposition annotation). The decrease in the κ score between the different versions of the annotation scheme is most likely a consequence of increased complexity of the annotation scheme, due to the addition of new properties (from 37 to 57).

The 481 compounds in the IAA study will be further used in Chapter 7 to assess the capabilities of the machine learning model for automatic interpretation of noun compounds. This subset of compounds should be particularly challenging for the computer-based analysis, given that the heads of these compounds will not be part of the training data.

All the inter-annotator agreement statistics described in this section were obtained using DKPro Agreement (Meyer et al., 2014), a module of DKPro Statistics for computing inter-annotator agreement measures. DKPro Statistics is a collection of open-licensed statistical tools written in Java.

General remarks. The annotation scheme presented in this section illustrates the wide range of semantic relations that nominal compounds can be based on. As stated before, the main goal in creating the new annotation scheme was to be able to render the semantics of nominal compounds as precisely as possible, using a combined (property, preposition) label. The annotation effort has confirmed the thesis put forth by Ryder (1994), that compounding forms *"[...] a continuum from smaller and less productive patterns to the larger, extremely productive ones"*.

From a computational perspective, the challenge that lies ahead is to understand and model the mechanisms behind compounding in order to provide suitable representations for nominal compounds. The next two chapters describe the machinery used to build compound representations, while Chapters 5, 6 and 7 present the empirical results.

Chapter 3

Neural Networks for Natural Language Processing

Deep learning is the technology behind the recent progress in speech processing (Dahl et al., 2010; Hinton et al., 2012) and image recognition (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014). The use of deep, multi-layer neural networks has lead to large error reductions (∼30%) on landmark datasets in the respective fields. The use of deep learning has also lead to advances in natural language processing: starting with Bengio et al. (2003)'s neural language model and the introduction of word embeddings in Collobert and Weston (2008), Turian et al. (2010), Collobert et al. (2011b), the field has seen a shift in the choice of machine learning techniques to use, and features to represent language data. The combination of neural network architectures and distributed representations has brought about advances in the state-of-the-art of several NLP tasks and applications like POS tagging (Santos and Zadrozny, 2014), parsing (Chen and Manning, 2014), named entity recognition (Collobert et al., 2011b), machine translation (Sutskever et al., 2014; Bahdanau et al., 2014) etc.

The combination of neural networks and distributed word representations has thus far shown good generalization behaviour. Given that compounding is a productive linguistic process, it is important that the computational models used to model it have the ability to generalize to unseen examples. This makes the combination of neural networks and distributed representations a natural setting for exploring the semantics of nominal compounds.

The first part of this chapter will cover the basics of neural networks and explain some of the characteristics that make them a 'good fit' for modeling compound semantics. The presentation will focus in particular on the building blocks of a neural network, and the algorithms that are used for training it - i.e. for making a network pick up patterns in the data, and generalize from examples that it sees in order to predict particular traits for new examples of the same type. The theme of the second part of the chapter is creating feature representations for natural language processing. The discussion will cover several types of representations and focus in particular on

their generalization capabilities. Appendix A provides an overview of the mathematical notation used throughout this and the upcoming chapters.

3.1 What are Neural Networks?

The exposition in this and the following two sections is based on materials from MacKay (2003) and Nielsen (2015).

Neural networks are a biologically-inspired class of machine learning algorithms. A neural network consists of a multitude of individual **units**, organized in several **layers** and connected via **weights** of varying strength. Artificial neural networks were designed by analogy to the human brain, which can be seen as a network made of tens of billions of neurons (i.e. the units) connected via each neuron's dendrites and axon (i.e. the weights).

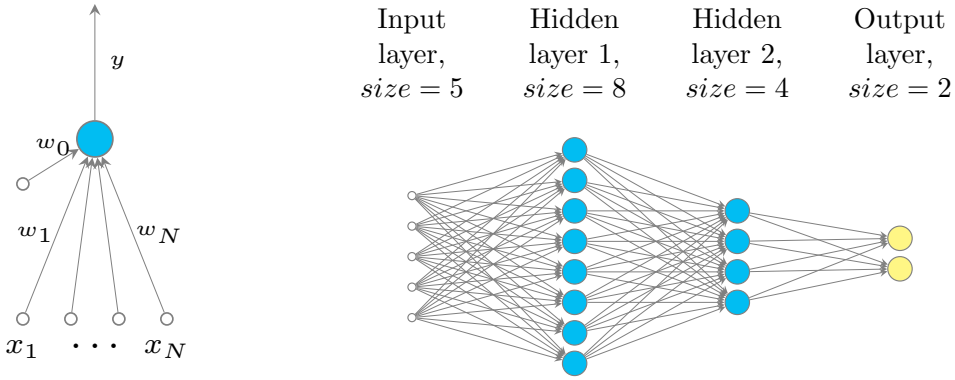
Figure 3.1a illustrates a neural network unit, with **inputs** x_1 to x_N and weights w_1 to w_N. An additional parameter called the **bias** is usually added as w_0. The input of the bias is always taken to be $x_0 = 1$. Both the inputs and the weights are real-valued numbers. The **activation** of a unit, a, can then be computed using the formula in Eq. 3.1:

$$a = \sum_{i=0}^{N} w_i x_i = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_N x_N \tag{3.1}$$

The **output** of the unit, y, can then be computed using Eq. 3.2:

$$y = f(a) \tag{3.2}$$

where f is an **activation function**, applied to the activation of the unit, a, to produce y.



(a) A single neuron.

(b) A feed-forward neural network.

Fig. 3.1 Neural network illustrations.

A **feed-forward neural network**, like the one illustrated in Figure 3.1b, is organized into multiple layers: the **input** layer, whose purpose is to provide the initial inputs to the network, one or multiple **hidden** layers, and the **output** layer. The number of layers and the number of units per layer define the **architecture** of a network: the example network in Figure 3.1b has 5 input units, two hidden layers with 8 and 4 units respectively, and two output units. Networks with two or more hidden layers are usually called **deep neural networks**. The architecture of the neural network will have an impact on its efficiency, and is, in practice, chosen empirically depending on the type of problem to be solved.

Activation Functions. The activation function f is, as mentioned above, the function that is applied to the activation of a unit to obtain its output value. A set of activation functions widely used in the literature is illustrated in Figure 3.2:

- the identity function, $f(x) = x$, $f : \mathbb{R} \to \mathbb{R}$
- the logistic function, $f(x) = \frac{1}{1+e^{-x}}$, $f : \mathbb{R} \to [0, 1]$
- the hyperbolic tangent function (`tanh`), $f(x) = \tanh(x)$, $f : \mathbb{R} \to [-1, 1]$
- the rectified linear unit function (`ReLU`), $f(x) = \max(0, x)$, $f : \mathbb{R} \to [0, \infty]$
- the parametric rectified linear unit function (`PReLU`), $f(x) = \max(0, x) + a \min(0, x)$, where a is a learnable parameter.

The essential property of a good activation function is that a small change in its input value should result in a small change in its output value. This makes it possible for small adjustments in the weights of the network to result in small improvements of the network's capability to produce the correct output. Different activation functions transform their input differently: e.g. the output of the logistic function can be directly interpreted as a probability, because its codomain is [0,1]. In cases where a probabilistic interpretation is not required, the rectifiers `ReLU` (Glorot et al., 2011) and its parametric variant `PReLU` (He et al., 2015) have proven to be some of the best choices for an activation function, especially when the architecture of the neural network involves multiple layers of hidden units.

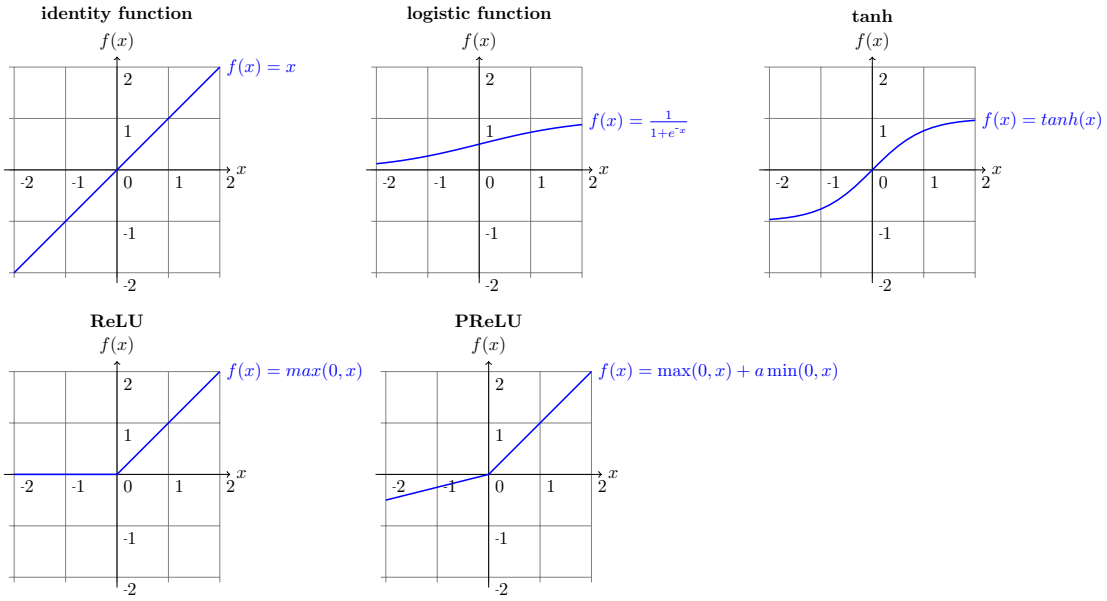


Fig. 3.2 Activation functions.

3.2 Supervised Learning with Neural Networks

Neural networks are appealing because they offer a way to **predict** an output given an input or set of inputs. For example, the network can learn to predict the selling price of a house (the **target** value, $\mathbf{t}$), given some of its characteristics like number of rooms (x_1), age (x_2), location (x_3), size of the plot it is built on (x_4), etc. This type of setup, where the desired target is a real-valued number, is a **regression** problem. If the target is instead a discrete set of items, such as the property labels in inventory of the annotation scheme previously introduced in Section 2.2, then it is a **classification** problem.

Intuitively, the main idea behind supervised learning with neural networks is the following: given a set of N examples $(\mathbf{x}_i, \mathbf{t}_i)$, find a set of weights $\mathbf{w}$ and biases $\mathbf{b}$ for the units of the neural network that capture the correlation between each of the input vectors $\mathbf{x}_i$ and its target $\mathbf{t}_i$. The performance of the weights at capturing this correlation can be assesed by comparing $\mathbf{y}_i$, the output of the network for the input $\mathbf{x}_i$, to the target vector, $\mathbf{t}_i$.

$\mathbf{x}_i$ is the **input vector** for the example i. $\mathbf{x}_i = (x_1, x_2, \ldots, x_k)^\top$ is also called the **feature representation** of example i, where k is the number of features in the input. In the example above, the features are all the characteristics of the house that are considered informative for the task of predicting its price, such as its number of rooms, age or location.

$\mathbf{t}_i$ is the **target representation** of the example i. For a regression problem, the target value is, as mentioned in the example above, a real-valued number. For a classification problem, the target vector $\mathbf{t}_i$ is defined as a **one-hot vector**, i.e. a vector where all components are 0 save for the component corresponding to the index of the

current label. E.g. when classifying trees in terms of their size, the possible labels can be defined to be {`small`, `medium`, `large`}. The target vector for `small` can then be represented as $(1,0,0)^\top$, `medium` as $(0,1,0)^\top$ and `large` as $(0,0,1)^\top$.

$\mathbf{y}_i$ is the **output** of the neural network for the example i. This is the prediction that the network makes based on the input $\mathbf{x}_i$. If the network is good at capturing the correlations in the data, $\mathbf{y}_i$ is expected to be close to the target vector $\mathbf{t}_i$. If the network has not learned the correlations, $\mathbf{y}_i$ and $\mathbf{t}_i$ are expected to be far apart.

Cost function. The function that is used to quantify how much has the network 'learned' is called a **cost function**, the **loss** or the **objective function**. The cost function can have different definitions, depending on the problem at hand.

One of the most commonly used cost functions is the mean squared error (MSE) function from Eq. 3.3:

$$J(\mathbf{w},\mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}||\mathbf{y}_i - \mathbf{t}_i||^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{j=1}^{K}\frac{1}{2}(y_{ij} - t_{ij})^2 \tag{3.3}$$

where $\mathbf{w},\mathbf{b}$ are the weights and biases of the neural network to be estimated, N is the number of training examples, K is the number of features for each example, $\mathbf{y}_i$ is the output of the network for the input $\mathbf{x}_i$ and $\mathbf{t}_i$ is the correct target vector associated with the input $\mathbf{x}_i$. In training a neural network with a MSE cost function, the goal is to **minimize** the error between the network's output and the real target vector. The overall performance of the network can be judged based on this 'mean of errors' computed by the MSE function: if most errors are close to 0, then the mean is also going to be close to 0; conversely, if most errors are large, the mean of the errors will also be a large number. The MSE cost function can be used both for regression and for classification problems.

Another choice is the **cross-entropy cost function**, presented in Eq. 3.4, where again N stands for the number of training examples, $\mathbf{y}_i$ is the output of the network for the input $\mathbf{x}_i$ and $\mathbf{t}_i$ is the target value for the example $\mathbf{x}_i$. ln is the natural logarithm, i.e. the logarithm with base e. For numbers between (0,1), ln takes strictly negative values, therefore the components of the sum in Eq. 3.4 are all negative. The initial minus sign makes the cost function positive overall.

$$J(\mathbf{w},\mathbf{b}) = -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{t}_i \ln \mathbf{y}_i + (1-\mathbf{t}_i)\ln(1-\mathbf{y}_i)] \tag{3.4}$$

The cross-entropy cost function is particularly well suited for classification problems, where the possible target/output values are 0 and 1. For example, if the target value $\mathbf{t}_i$ is 0, and the output of the network, $\mathbf{y}_i$, is close to 0, then the first term of the sum cancels, and the second term is $\approx \ln 1$, which is ≈ 0. For the other case, when the target value is 1, if the output of the network is also close to 1, then the second term of the sum cancels, and the first term is $\ln(\approx 1)$, so, again ≈ 0. This means that if our network makes correct predictions, the cross-entropy cost function will be 0.

The cross-entropy cost function can also be used for regression problems, provided that the output(s) of the network are between 0 and 1. This can be obtained, for example, by using the logistic function as the activation function of the last layer.

If a probabilistic interpretation of the outputs of the network is desired, a cost function called the **categorical cross-entropy** or **negative log likelihood** can be used. The first step in using such a cost function is to equip the network with a custom output layer, called a **softmax layer**, that computes the output of each unit of the output layer[1] according to the formula in Eq. 3.5:

$$\mathbf{y}_j^L = \frac{e^{a_j^L}}{\sum_{k=1}^{m} e^{a_k^L}} \tag{3.5}$$

where k ranges over the m possible classes that an output might belong to in a given classification scheme. The softmax function is a normalized exponential - the denominator sums over the outputs for all m classes, and thus guarantees that the outputs of the network sum to 1 (see Eq. 3.6). Intuitively, the softmax layer allows the network to express its certainty about the class a particular example belongs to. Probabilities close to 1 mean that the network is quite certain about the class it has chosen, whereas having approximately equal probabilities for each of the classes means that the network is uncertain about the class assignment for that particular example[2].

$$\sum_j \mathbf{y}_j^L = \frac{\sum_j e^{a_j^L}}{\sum_{k=1}^{m} e^{a_k^L}} = 1 \tag{3.6}$$

If there is only one correct class assignment, then the negative log likelihood cost function is defined by Eq. 3.7:

$$J(\mathbf{w}, \mathbf{b}) = -\frac{1}{N} \sum_x \ln \mathbf{y}_k^L \tag{3.7}$$

where k is the correct class for example $\mathbf{x}$. Remember that in a multi-class classification problem, $\mathbf{y}_k^L$ is a one-hot vector of size m, the number of classes, where the non-zero element is the component with index k. This cost function minimizes the case when k is not the winning class. The softmax layer ensures that if the winning class is maximized, i.e. if the output for the correct class is pushed towards 1, the output of the remaining classes is necessarily driven towards 0.

Cost functions are parametrized by the weights and biases of the neural network. The cost function is applied to the input vectors, which are generally provided from an external source and can be considered fixed from the perspective of the cost function. This means that changes in the output value of the cost function can only be caused by changes in the values of its parameters. Minimizing the cost function means, in effect, finding a set of parameters, i.e. of weights and biases, that will minimize the

[1] Activations and outputs of the units in the last layer are marked with a superscript L: a^L, y^L.

[2] This interpretation is only valid for cases when there is only one correct class assignment.

difference between the output predicted by the network and the real target value. A good set of parameters that minimize the cost function can be discovered by **training** the neural network.

3.3 Training a Neural Network

In a supervised learning setup, the data that needs to be modeled is split into three disjoint sets: a **training set**, a **validation (dev) set** and a **test set**. The idea behind this type of setup is to use the examples in the training set to discover a good way to set the weights of the network such that the **generalization error**, i.e. the error on the test set, is as small as possible. The splitting in disjoint datasets is important because during training the networks tend to **overfit**, i.e. to model the training set down to its last details, even if this turns out to be just noise in the data.

In such a setup, the parameters of the network ($\mathbf{w}$, $\mathbf{b}$) are determined based on the network's performance on the validation set. The rationale to using the validation set as an extra split of data is that the network should not be 'biased' in any way towards the test data. If during training the performance of the network were measured directly against the test set, its parameters would, in effect, be 'tuned' directly to the test set. This would lead to a non-generalizing network, which might behave differently given a new set of data from the same underlying distribution.

The best practice recommendation is to use the test set just once at the end of the training cycle, to report on the results of the network trained on the training set using the set of parameters that obtained the best performance on the validation set.

What does it mean to train the neural network? Let's consider a network having the architecture depicted in Figure 3.3. First a bit of notation: for weights, the superscript denotes the layer number and the subscript denotes the units that a weight connects: so w^1_{12} is a unit on layer 1, which connects unit 1 on the current layer to unit 2 on the previous layer. For the biases only the index of the layer and the index of the unit in the layer are specified - so b^1_2 is the bias weight corresponding to the second unit of layer 1.

For the units the superscript identifies the layer while the subscript denotes the index of the unit on the layer: h^2_1 is hidden unit number 1 on layer 2.

Consider a training example, $\mathbf{x} = (x_1, x_2)^\top$ with the target value $\mathbf{t} = (t_1)$. The output of the network in Figure 3.3 for this training example can be obtained by applying Eq. 3.1 and Eq. 3.2:

- first, compute the activations of the hidden units on layer 1:

  $a^1_1 = w^1_{11}x_1 + w^1_{12}x_2 + b^1_1$, for the first hidden unit on the first layer, h^1_1, using the dashed red weights

  $a^1_2 = w^1_{21}x_1 + w^1_{22}x_2 + b^1_2$, for the second hidden unit on the first layer, h^1_2, using the solid blue weights

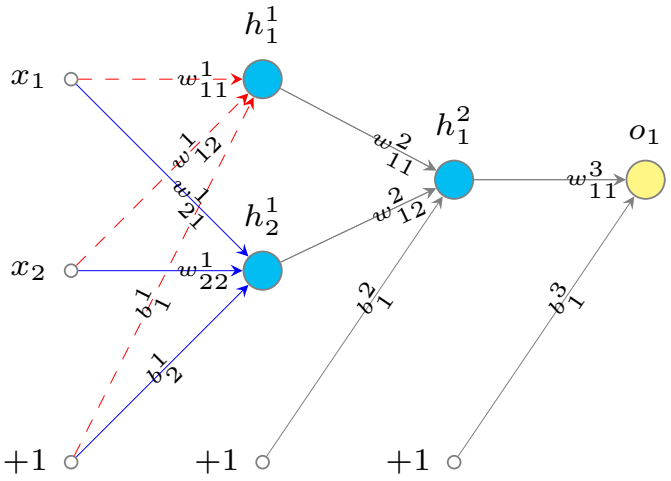


Fig. 3.3 Forward pass through a neural network.

- then compute the output of the hidden units on layer 1, using Eq. 3.2:

  $y_1^1 = f(a_1^1)$; $y_2^1 = f(a_2^1)$

- the outputs of the units in the first hidden layer become the inputs of the units on the second layer, so the activation and the output of the hidden unit in the second hidden layer, h_1^2 can now be computed:

  $a_1^2 = w_{11}^2 y_1^1 + w_{12}^2 y_2^1 + b_1^2$; $y_1^2 = f(a_1^2)$

- the final output of the network is completed in a similar step, using the output of the second hidden layer as the input:

  $a_1^3 = w_{11}^3 y_1^2 + b_1^3$; $y_1^3 = f(a_1^3)$

The stepwise description above can be generalized to the following equations for computing the activation of a unit in a neural network:

$$a_j^l = \sum_k w_{jk}^l y_k^{l-1} + b_j^l \tag{3.8}$$

and the output of a unit in a neural network:

$$y_j^l = f(a_j^l) = f(\sum_k w_{jk}^l y_k^{l-1} + b_j^l) \tag{3.9}$$

The procedure described so far is called the **forward pass**: it starts with the input values and computes all the activations and the outputs of the units, layer by layer, until the last layer of the network. The result of the forward pass is $\mathbf{y} = (y_1^3)$, the prediction of the network for the input $\mathbf{x} = (x_1, x_2)^\top$, which should correspond to the real target value $\mathbf{t} = t_1$.

The weights of a neural network are usually initialized using small, random values, e.g. by generating numbers from a Gaussian distribution of mean 0 and standard deviation ϵ. If the example above was the first training example shown to the neural network, there's a big chance that y_1^3, the predicted output, is far away from the target output $\mathbf{t} = t_1$. The error made by the network can be quantified using a cost function, e.g. the MSE cost function from Eq. 3.3. Ideally, this error should be as small as possible, for the network to be considered a good predictor. Since the inputs are given by the training set, the only possibility of improving the predictions of the network comes from improving the weights of the network.

3.3.1 Optimization with Gradient Descent

Gradient descent is a method for determining what changes should be made to the weights in order to improve the cost of the network. Imagine a simplified setup where the cost function J depends only on two weights, w_1 and w_2. If ΔJ denotes the change in the cost function, the notion of a derivative from calculus can be used to write out ΔJ with respect to the weights w_1 and w_2, as in Eq. 3.10:

$$\Delta J \approx \frac{\partial J}{\partial w_1}\Delta w_1 + \frac{\partial J}{\partial w_2}\Delta w_2 \tag{3.10}$$

The **derivative** of a function is a way to describe the change in a function's output as a result of the change in the function's input values. The goal of the optimization process is to figure out how to chose Δw_1 and Δw_2, i.e. how to change the current weights w_1 and w_2 such that ΔJ is negative, meaning that the associated error becomes smaller. Eq. 3.10 can be rewritten in a vectorial form, where $\Delta w \equiv (\Delta w_1, \Delta w_2)^\top$ is the vector of changes to the parameters, while $\nabla J \equiv (\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2})^\top$ is the **gradient** of the function J, which holds all the **partial derivatives** $\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}$ of the function J with respect to all its parameters w_1 and w_2.

Eq. 3.10 now becomes

$$\Delta J \approx \nabla J \cdot \Delta w \tag{3.11}$$

By taking

$$\Delta w = -\eta \nabla J \tag{3.12}$$

where η is a small, positive number called the **learning rate** and rewriting Eq. 3.11 using Eq. 3.12, the change in the cost function, ΔJ, becomes

$$\Delta J \approx -\eta \nabla J \cdot \nabla J = -\eta ||\nabla J||^2 \tag{3.13}$$

Using the ΔJ in Eq. 3.13 will minimize the cost function: $||\nabla J||^2$ is strictly positive, meaning that overall ΔJ is a negative and updating J using ΔJ will lower the cost function. Eq. 3.14 gives the formula for updating the weights. Here, $\mathbf{w}$ is the vector

of parameters at timestep t, and $\mathbf{w}'$ is the vector of parameters at the subsequent timestep:

$$\mathbf{w}' = \mathbf{w} - \eta \nabla J \tag{3.14}$$

Gradient descent is an iterative procedure: the update rule in Eq. 3.14 has to be applied over and over to minimize J. The learning rate η plays an important role in the success of the optimization process: it defines the proportion of the gradient that is taken into consideration at each iteration. The learning rate is usually chosen empirically for each problem: if it is too small, the minimization process will take many steps to complete. If the learning rate is too large, an update might lead to a 'jump' over the minimum of the cost function, which will start then to increase instead of decreasing. In practice, the learning rate is often chosen to be a function of time, i.e. of the number of time steps that the gradient descent procedure has taken. In such a setup the learning rate starts out large at the beginning of the descent, and becomes gradually smaller with each time step.

Gradient descent is thus a method of updating the weights of the network in the direction that improves the cost function. Typical state-of-the-art neural networks, however, can be made of millions of weights. In such a setup, it is important not only to know how to update the weights, but also to be able to perform these updates in a computationally-efficient manner. The cost function J from Eq. 3.3 involves a sum over all the training examples. In order to learn, the gradient has to be computed with respect to each of the training examples. However, this could mean that the minimization algorithm takes a long time to reach a good solution.

In practice, an approximation of the real gradient is usually computed. The approximation uses only a subset of the training examples, called a **mini-batch**. The updates are then performed using this approximation of the gradient. The mini-batch variant of the algorithm is called **stochastic gradient descent (SGD)** (Bottou, 2012). In such a setup, the training set is partitioned into mini-batches, where each training example is randomly assigned to a single mini-batch. A system is said to have performed an **epoch** of training if it has done a number of gradient steps equal to the total number of mini-batches in the training set.

The number of epochs together with the mini-batch size and the learning rate are called **hyperparameters** of the network. In contrast, the values $\mathbf{w}$ and $\mathbf{b}$ that the cost function J depends on are the **parameters** of the network. Using algorithms like gradient descent one can discover good values for the parameters of the neural network. The hyperparameters, on the other side, are in general chosen empirically, and the particular choices will have a large impact on the performance of the network.

Stochastic gradient descent gives good results, but might, however, take a long time to converge - i.e. to find a solution that is close to the minimum value. Several improvements have been proposed in the literature, offering substantial speedups compared to the classic algorithm. A variant called AdaGrad (Duchi et al., 2011) will be used in the experiments described in Chapters 5, 6 and 7.

3.3.2 Backpropagation

Training a neural network involves computing the gradient of the cost function J with respect to its parameters, $\mathbf{w}$ and $\mathbf{b}$. However, as mentioned before, a neural network can have millions of parameters. This section describes a method called **backpropagation** (Rumelhart et al., 1986a; LeCun et al., 2012), that can efficiently compute the gradient of such complex functions.

Looking back at the expressions of the activations from the forward pass for the network in Figure 3.1b, the following patterns can be observed: first, the output is a function of the activation of the last layer, $y_1^3 = f(a_1^3)$. a_1^3, the activation of the last layer, is a function of the output of the previous layer, $a_1^3 = w_{11}^3 y_1^2 + b_1^3$.

The output of the network could, in consequence, be rewritten as a function of the activation of the second layer, a_1^2, resulting in the equation $y_1^3 = f(w_{11}^3 f(a_1^2) + b_1^3)$. Another backward step, and the output can be written as $y_1^3 = f(w_{11}^3 f(w_{11}^2 f(a_1^1) + w_{12}^2 f(a_2^1) + b_1^2) + b_1^3)$. So in effect, the computation performed by a neural network is a composition of functions.

The derivative of a composition of functions can be computed using the **chain rule**: the derivative of a composed function, $h(x) = f(g(x))$, is given by the formula in Eq. 3.15:

$$h'(x) = f'(g(x)) \cdot g'(x) \tag{3.15}$$

With backpropagation, the architecture of the network is used to compute the **error signal**, δ, for each unit of the network. The computation begins from the outputs. For the output units, the error signal can be computed using the formula

$$\delta_j^L = \frac{\partial J}{\partial y_j^L} f'(a_j^L) \tag{3.16}$$

where f is the activation function of the last layer, indexed with ^{L}. If the cost function is MSE, from Eq. 3.3, its derivative is $(y_j^L - t_j)$, so the error signal becomes

$$\delta_j^L = (y_j^L - t_j) f'(a_j^L) \tag{3.17}$$

The activation and output of the last layer, a_j^L and y_j^L, are known - they were computed in the forward pass. Similarly, the derivative of the activation function, f', is known, so it is easy to compute the value of the derivative for the input a_j^L. Equation 3.17 can be written in a vectorized form, using the Hadamard product, $\odot$:

$$\delta^L = (y^L - t) \odot f'(a^L) \tag{3.18}$$

Once δ^L is computed, the error signal for the units in the previous layer can be computed using the formula

$$\delta_j^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(a^l) \tag{3.19}$$

Here, $(w^{l+1})^T$ is the matrix of weights on the next layer, and δ^{l+1} is the error signal that was just computed in the previous step. The left side of the Hadamard product in Eq. 3.19 effectively propagates backward through the network the error signal from the previous layer, while the right hand side is again the derivative of the activation function applied to the activation of the units in the current layer.

This process of propagating the errors back one layer at a time allows the recursive computation of the error signals for each unit in our network. Once the values for all δ_{ij}^l are computed, the corresponding weight updates can be performed using the learning rate η, as described in Eq. 3.14.

What makes backpropagation a computationally-efficient algorithm? In a network with one million weights, a direct computation of the partial derivatives with respect to each weight would mean computing the cost function, i.e. doing a forward pass through the network, **for each partial derivative and for each example**. This would mean computing the cost function a million times for each example. With backpropagation **all** the partial derivatives are computed at the same time, using only one forward pass and one backward pass, thus making the optimization process substantially more efficient.

3.3.3 Regularization

Neural networks generally have a large number of parameters, and, as mentioned at the beginning of Section 3.3, they will generally **overfit** the training data. Overfitting comes with the unwanted side-effect of reducing the network capability to generalize, i.e. to make good predictions on previously unseen data.

One way to reduce overfitting is to increase the amount of training data while keeping the number of parameters unchanged. Since there are more data points to be modeled, the network will be less likely to learn insignificant details particular to a training sample. The cost function is only going to be low if the network makes good predictions for most of the training data, so the focus has to be on capturing trends in the data that describe the majority of the training material.

However, getting new training data is, in most supervised learning scenarios, a costly enterprise, usually involving the effort of trained human annotators. This section presents other approaches to avoiding overfitting, i.e. other **regularization methods**: weight decay, early stopping and dropout.

Weight decay or L_2 regularization. The intuition behind L_2 regularization is that the weights of the neural network should remain small. To achieve this goal, the cost function $J(\mathbf{w}, \mathbf{b})$ presented in Section 3.2 will be updated to incorporate a penalty for the magnitude of the weights, as shown in Eq. 3.20.

$$J_{reg}(\mathbf{w}, \mathbf{b}) = J(\mathbf{w}, \mathbf{b}) + \frac{\lambda}{2N} \sum_w w^2 \tag{3.20}$$

In effect, the two sides of Eq. 3.20 have conflicting goals: the left hand side tries to improve the unregularized cost function, which could lead to an increase in the magnitude of the weights. In contrast, the right hand side term seeks to decrease the weights, and would be 0 when all the weights are 0. The regularization term $\frac{\lambda}{2N}\sum_w w^2$ favors the solutions to the initial cost function $J(\mathbf{w}, \mathbf{b})$ where the parameters have small values. The parameter λ controls how aggressive the regularization is: a large value for λ corresponds to very small weights and possibly larger errors for the initial cost function. A small value for lambda gives more importance to minimizing the unregularized cost function.

Early stopping (Prechelt, 1998) is a technique that involves stopping the training procedure before overfitting has a chance to occur. The idea is to monitor the generalization error, i.e. the value of the cost function on the validation set. Typically, the generalization error will be large at the beginning of the training process and will progressively decrease as the network learns to make better predictions. In keeping track of the generalization error, one can pinpoint the moment when the generalization error starts to increase again, i.e. the moment when the network starts overfitting the training data.

The process is complicated a bit by the fact that the generalization error is not guaranteed to be strictly decreasing over time: a plot of the generalization error might show small increases, even when the trend is a generally decreasing one. This is why in practice the early stopping procedure uses a parameter s. s is the number of successive epochs that the error is allowed to increase. If the network has an increase/decrease fluctuation that is smaller than s epochs, than the generalization error is still improving, so the network should be trained on. If, however, the generalization error increases steadily over this period of s epochs, this is a signal that overfitting is likely to have begun. A trade-off that has to be made when choosing s concerns the training time versus the generalization error: choosing a higher value for s leads to a smaller error, at the cost of increased training time. Conversely, for a small s the network will train faster, at the cost of a slightly larger generalization error.

Dropout is a regularization technique introduced by Hinton et al. (2012). The intuition behind dropout is that the generalization performance of a neural network can be improved by preventing the co-adaptation of the units. The technique involves randomly 'dropping' hidden units from the network, with a probability of 0.5, each time a new training example is presented for training. The result is that only 50% of the hidden units in the network will play a role in the prediction made by the network for that particular example. The random dropping of the units means, in effect, that each unit becomes more robust and is able to make better predictions on its own, since it cannot rely on particular units in the network being active at the same time.

The effect of dropout is comparable to a setup where a prediction is obtained by averaging predictions from a large number of neural networks, trained individually on the same problem. The averaged prediction is obtained, however, at a fraction of the

cost. The dropping of units happens only during training. At test time, all hidden units are kept but the weights are halved, to compensate for the fact that all of them are now active.

Srivastava et al. (2014) extend the idea of Hinton et al. (2012) by parametrizing dropout. p represents the probability that a unit in the network is kept. The probability is independent for each individual unit. Good values for p are between 0.5 and 0.8 for hidden units, and between 0.8 and 1 for input units. If dropout with the probability p is used, the activation and the output of the neural network are computed using the equations in listing 3.21. At test time, the weights of the network are scaled by the factor p.

$$\begin{aligned} r_j^l &\approx \mathsf{Bernoulli}(p) \\ \tilde{y}_j^l &= y_j^l r_j^l \\ a_j^l &= \sum_k w_{jk}^l \tilde{y}_k^{l-1} + b_j^l \\ y_j^l &= f(a_j^l) = f(\sum_k w_{jk}^l \tilde{y}_k^{l-1} + b_j^l) \end{aligned} \tag{3.21}$$

Dropout has proven to be a very effective regularization method for many datasets from different domains: image recognition (experiments on MNIST, CIFAR-10 and CIFAR-100, ImageNet), speech processing (TIMIT), text categorization (Reuters), etc. - for details see Hinton et al. (2012); Srivastava et al. (2014).

3.4 Neural Networks instead of Linear Models

The machine learning and, through extension, the NLP landscape before the introduction of deep learning models revolved mainly around linear models like linear and logistic regression, or models like support vector machines (Boser et al., 1992; Cortes and Vapnik, 1995) that can produce non-linear decision boundaries using transformations of the input space.

The output predicted by a linear regression model is defined by Eq. 3.22, where $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional input vector, $\mathbf{w} \in \mathbb{R}^n$ is an n-dimensional weight vector, and $y \in \mathbb{R}$ is the prediction of the model for the input $\mathbf{x}$.

$$y = \mathbf{w}^\top \cdot \mathbf{x} = \sum_{i=0}^{n} w_i x_i \tag{3.22}$$

The intuition behind Eq. 3.22 is that every component of the feature vector, x_i, is going to contribute to the final prediction y by an amount moderated by the weight w_i. The prediction is thus a linear combination of the initial features and the weights. It is interesting to note the striking similarity between the formulation of linear regression in

Eq. 3.22 and the activation of a neural network unit in Eq. 3.1. A neural network with a single unit and an identity activation function will, in fact, perform linear regression.

Logistic regression can be used for classification. If the output can only belong to one of two possible classes, $y \in \{0, 1\}$, a function of the form

$$p(y = 1|\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\top \cdot \mathbf{x}) = \frac{1}{1 + e^{(-\mathbf{w}^\top \cdot \mathbf{x})}} \tag{3.23}$$

can be used to predict the output. Note that the decision boundary of logistic regression is a line (in 2D) or a hyperplane (in three dimensions and higher). The logistic function, $\frac{1}{1+e^{-x}}$, only helps us map the output of the linear combination of the input features and the weights onto a probability. If the examples that need to be classified cannot be well separated using a line/plane, linear models will not be good predictors for the data. If the data is not linearly separable, the problem can be sidestepped using transformations of the initial inputs, e.g. by using the polynomial of degree n in Eq. 3.24:

$$y = \mathbf{w}^\top \cdot \mathbf{x} = \sum_{i=0}^{n} w_i x_i^n \tag{3.24}$$

Notice that the polynomial is still a linear combination in terms of the weights, but uses the transformed input features. The drawback of this approach is that the designer of the machine learning system needs to empirically decide which feature transformations should be added to the model in order to improve its prediction capability.

With linear models, it is simple to understand the effect of particular input features on the performance of the machine learning model. By contrast, the composition of functions typical to a deep neural network is associated with a lack of transparency with regard to the patterns that the network uses to classify new examples. This means that even if the predictions made by the system are correct, its more difficult to gain intuitions about the patterns used by the system used to make the predictions. The patterns learned by a neural network are not easy to tease out once the training is complete.

At the same time, it is this very trait of neural networks that makes them such powerful classifiers: with each new layer of hidden units, the network computes a new, more abstract version of the initial representation that was given as input. For example, in the image recognition domain, the visualization of the features learned by different layers of a (convolutional) neural network shows that the initial layers capture features that are local to the input - e.g. lines, edges, while subsequent layers build upon previous layers and capture more global representations - e.g. eyes, ears, shapes of different animals for the ImageNet data (Zeiler and Fergus, 2014).

This abstraction power of the neural networks gives the machine learning practitioner the possibility to provide the network with a simple feature representation as input and to let the network do most of the heavy-lifting in terms of choosing the most

appropriate feature combinations that would lead to optimal decision boundaries. When using linear models, the discovery of useful feature combinations was one of the most time-intensive aspects of improving a machine learning system. The combinations of features to be added were carefully chosen by inspecting the errors of existing models. Defining feature combinations that would capture the right aspects of the data and make the relevant information available to the machine learning model used to be the task of the practitioner. By contrast, the neural network-based models start off with all the available features and have the ability to discover on their own the feature combinations that are useful in the prediction process.

The choice of using neural networks for modeling compound semantics brings with it the implications typical for such models: if the neural network model is able to predict the correct semantic relation labels, it is not straightforward to figure out what sort of patterns it captures and how do these correspond to human intuitions. However, the aim of this thesis is not the full understanding of compound semantics. The declared goal is to create computational representations for nominal compounds, and neural networks seem to be a promising tool for this task.

The results presented in Chapters 5, 6 and 7 are all obtained using neural network models. The implementations are based on Torch7 (Collobert et al., 2011a), a scientific computing framework for Lua with an underlying C/CUDA implementation. Torch7 comes with a modular approach to neural networks, and provides a flexible framework for creating architectures with multiple inputs and outputs. Furthermore, it supports GPU computation, meaning that matrix multiplication, the most frequent operation when training neural networks, can be massively parallelized and executed in only a fraction of the time required when using traditional CPU-based computation.

The only piece of the puzzle that is still missing is how can words be represented in such a way that neural networks can be of use? The next section will introduce the basics of feature representation for natural language processing.

3.5 Feature Representation for NLP

Feature representation was illustrated at the beginning of Section 3.2 as using the characteristics of a house - like its number of rooms, or age - to predict the price of the house. In this case, the feature representation is a vector. For example the vector $(4, 17.2)^\top$ could serve as a representation for a house with 4 rooms that is a bit over 17 years old.

Words in natural language are written using letters. The written form is a convention, a system of symbols that serve as pointers to the meaning of words. Machine learning algorithms like neural networks require numerical representations to work with. Feature representation for NLP is the task of finding a mapping between natural language symbols, i.e. words in the case of written language, and real-valued numbers or vectors. This mapping should not be made at random: if two words have strong semantic interactions - i.e. they are synonyms (e.g. 'bush' and 'shrub'), they are similar (e.g.

'cat' and 'dog') or related (e.g. 'lock' and 'key') - the corresponding vectors should reflect these patterns of interaction. The remainder of this section will introduce two types of feature representations used in natural language processing, and discuss their advantages and disadvantages.

3.5.1 One-hot representations

Imagine the task of representing a **vocabulary** V, containing $|V|$ distinct words. The **one-hot representation** or **one-hot encoding** of the word $w_i \in V$ is a $|V|$-dimensional vector, where all the components are 0 except for the component with index i, which is equal to 1. Such representations are usually high-dimensional, i.e. the dimension of the vector is equal to the size of the vocabulary V, and grows linearly with the number of words in the vocabulary to be represented. One-hot representations are also **sparse**, meaning that most of the entries of the vector are zeros (here, in effect, there is only one entry per word with a non-zero value).

The high dimensionality of such representations can be a problem when using neural networks, because it leads to an increase in the number of parameters of the network. Imagine for example a 100,000-dimensional vocabulary, and the words represented using a one-hot encoding. If the hidden layer has, for example, 1000 hidden neurons, then the weight matrix corresponding to this layer has 10^8 weights and 1000 biases, which all have to be estimated during the training process. This increases the complexity of the network and leads to increases both in the time needed to train it, and in the number of training examples required to obtain good estimations for all the parameters.

Although problematic, the high dimensionality is not the most challenging issue when using one-hot representations for words. The biggest shortcoming of such representations is their failure to **generalize above the individual word level**. The one-hot representations of the words *bush* and *shrub* offer no clue about their similarity. The vectors of the two synonyms are just as arbitrary as the vectors of any other two words that have no semantic interaction, e.g. *shrub* and *cake*. The similarity between the two vectors can be computed using the **dot product**, defined algebraically as:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n \tag{3.25}$$

Intuitively, Eq. 3.25 captures how similar the components of the vector are: every individual component contributes an item to the sum. If the elements of the two vectors for a particular component are approx. equal, then they will make a high contribution to the sum. If, in contrast, the product of the elements of the two vectors is 0 or negative, it will have a negative impact on the overall sum. The dot product is highest when the two vectors are the same on every component.

Figure 3.4 illustrates the one hot representations for the words *bush*, *shrub* and *cake*. The dot products between each vector pair - (*bush*, *shrub*), (*bush*, *cake*) and (*shrub*, *cake*) can then be computed using the formula in Eq. 3.25. The result is disappointing:

the dot product of the representations of all word pairs equals 0, since each of the vectors has exactly one non-zero component, and the index of the non-zero component is different for each of the vectors. This means that one-hot representations cannot translate the semantic similarities of the words into the vector space: the dot product of the pair (*bush*, *shrub*) is not larger than the one for the pair (*shrub*, *cake*), even if the words in the first pair are more similar. In other words, when using one-hot representations, each word is an independent entity, and a machine learning algorithm will not learn anything about how to analyze the word *shrub* based on training examples involving a synonym word like *bush*.

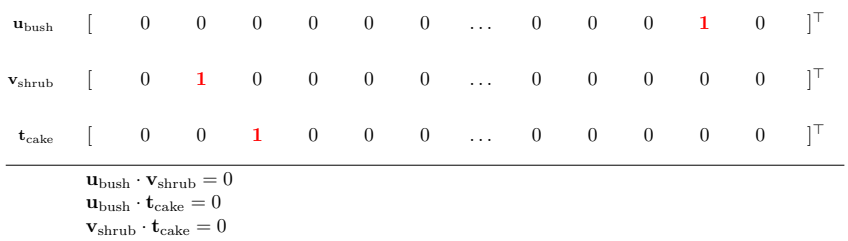


Fig. 3.4 One-hot representations for words. Every word is represented by 1 at a particular position. The representations of *bush* and *shrub* give no clue about the similarity between the two words.

In the context of statistical language modeling, Bengio et al. (2003) used the phrase **curse of dimensionality** to refer to this conundrum: *"a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training."*

Learning is hopeless if there is no generalization, i.e. if what the model has learned from a particular training example cannot be applied to a test example that is similar, but not identical to the previously encountered training example.

Truly effective word representations should not be merely convenient vectorial representations. They should strive to make available to the learning algorithm all the different facets of similarity that humans perceive when using words. For example, what do the words *pond*, *vase*, *lake* and *bathtub* have in common? They have a strong semantic affinity for the word *water*, which they generally containers for. Moreover, *ponds* and *lakes* form a semantic subcluster: both can hold a comparatively larger volume of water, and are cavities on the surface of the earth. *Vase* and *bathtub* can also be seen as another subcluster: they are man-made artifacts and can be found in a typical home. A class of representations that has the potential to give rise to the versatile representations needed for natural language symbols are **distributed representations**, which will be introduced in the next subsection.

3.5.2 Distributed representations

The one-hot representation presented above is a **local representation**. A separate index in the vector is required to represent every word in the vocabulary V. That index takes a non-zero value only for a single word in the vocabulary, and plays a minimal, passive role in the representations of all the other words. So each index is used exactly once in the representation of the vocabulary.

Distributed representations (Hinton et al., 1986) are an alternative representation paradigm, where each word is represented by a vector $\mathbf{x}$ with n real-valued components, $(x_1, x_2 \ldots x_n)^\top$, $x_i \in \mathbb{R}$. The 'meaning' is given by the configuration of values in the vector, and is therefore **distributed** across vector, instead of being localized in only one of the vector's components. In this case, each index of the representation gets reused multiple times and can play a role in the representation of every one of the words in the vocabulary.

As a result, the typical size of distributed representations is much smaller, since the size of the representation is no longer tied to the size of the vocabulary. In practice, distributed representations will typically use vectors having between 50 and 1000 components to represent vocabularies of hundred thousands or millions of words (Collobert and Weston, 2008; Mikolov et al., 2013a; Pennington et al., 2014). This is very advantageous when working with neural networks, as it means that the number of parameters that have to be estimated in order to train a neural network using distributed representations will be much smaller than when using one-hot representations.

Another advantage of distributed representations becomes evident when the need arises to represent a new word, which was so far not part of the vocabulary. In a local representation, each index corresponded to exactly one word. Representing a new word involves increasing the representation size for all the words, and expanding each vector with a new component, which will be 0 for all but the newly added word. In the case of a distributed representation, however, adding a new word means finding an appropriate configuration using the existing vector components. The adjustments that need to be made do not affect the size of the vectorial representations, but rather the patterns of meaning that emerge though the combination of specific values at specific positions in the representation.

An application that uses distributed representations is described in Rumelhart et al. (1986b), who model schemata using a constraint satisfaction network. They give an example of predicting the characteristics and objects in a room using a set of 40 descriptors or **microfeatures** (*ceiling*, *walls*, *door*, *very-small*, *very-large*, *stove*, *sink*, *computer*, *carpet*, *desk*, *telephone*, etc.). They collect 80 room descriptions from human participants, and form a network where the weights between the descriptors are set in a way that mimics the patterns seen in the collected data: if the descriptors appear together, their corresponding units will be connected via large positive weights; if two units generally take on different values, they are connected by large negative weights, and if the values of the units are independent of each other, they are connected via zero weights. Experiments start with a small set of active descriptors (e.g. *oven* and

ceiling), and the network proceeds to update its weights in order to maximize its cost function: first it adds *coffee-cup*, then *sink* and *refrigerator*, the fact that the rooms is *small*, and reaches its maximum fit with all units corresponding to the descriptors that are typical for a *kitchen* being active. The cost function is defined in terms of the set of descriptors for 'prototypical' rooms: *kitchen*, *bedroom*, *bathroom*, *living room* and *office*. In such a representation the 'meaning' of *kitchen* is distributed across the microfeatures, and the activation patterns across microfeatures determine the type of the room.

An apparent disadvantage of distributed representation is that they tend to blur the understanding of the emergent meaning patterns. In a local representation, the meaning attached to a particular index is known, because it has been a priori chosen by deciding on the word to be represented using that index. Distributed representations, on the other hand, make it more difficult to label the activation patterns that correspond to particular meanings. Chapter 4 will go over some interesting 'patterns of meaning' that were observed in the case of distributed word representations created using *distributional methods*. For such representations, applying the transformation *king* - *man* + *woman*[3] results in a vector close to the vectorial representation of the word *queen*.

Using distributed representations to model compound semantics is attractive for several reasons. As seen in the discussion of annotation schemes for compounds in Chapter 2, compounds with similar constituents tend to be interpreted in a similar way. For example, the compounds *metal cup* and *plastic glass* should be interpreted in the same way because of the similarity of their components: *metal* and *plastic* are both materials, while *cup* and *glass* both have a container sense. If distributional representations can capture this type of 'features' that humans intuitively seem to use as a base for compounding (e.g. word refers to a material, a time expression, an event, a living being, to something edible, etc.) than machine learning algorithms using these representations should be able to infer that the compound-internal relation in *plastic glass* is similar to the internal relation found in *metal cup*.

Furthermore, the fact that distributed representations make possible the representation of new concepts without the need for increasing the size of the representation make them a good fit for modeling compounding. Compounding is a productive process in language, meaning that there will always be newly coined compounds, in need of a representation that is compatible with the representations of already existing words (and compounds). Think for example of compounds with the head *knife*: they appeared in the language as the original utensil, the *knife*, was specialized for particular food items that were routinely cut: e.g. *bread knife*, *cheese knife*, *vegetable knife*, etc. Each of these compounds is still a *knife*, and its representation should be able to capture this marked similarity between the head and the resulting compound. The emerging meaning patterns have to express both the similarity to the head noun and the shift in meaning due to the semantic interaction with the modifier noun.

Ideally, the resulting compound representation should be just an adjustment of the initial representation of the head word, moderated using the representation of the

[3]using the vectorial representations of the words *king*, *man* and *woman*.

modifier word. In reality, however, language changes over time, and words that have appeared via the typical compounding process might at some point become overloaded with different, generally more abstract senses. This diachronic view of compounding brings in the possibility that the meaning of the compound has 'drifted' relatively far away from the meanings of its constituents, and as a result the representations of such 'drifted' compounds might, as a result, be quite different from their 'expected' representation. This issue will be discussed in more detail in Chapter 5, which is about creating compositional representations for compounds and in Chapter 6, which is dedicated entirely to the analysis and representation of such 'drifted', non-compositional compounds. But first Chapter 4 will introduce a series of methods for creating such distributed representations for words, called distributional word representations.

Chapter 4

Distributional Word Representations

"What is that" asked Seldon.

They were standing before a small tray filled with little spheres, each about two centimeters in diameter.[...] He lifted the sphere cautiously [...]

"What are they"

"Dainties. Raw dainties. For the outside market they're flavored in different ways, but here in Mycogen we eat them unflavored - the only way."

She put one in her mouth and said,"I never have enough" [...] It was slightly sweet and, for that matter, had an even fainter bitter aftertaste.

- Isaac Asimov, *Prelude to Foundation*

4.1 Introduction to Distributional Semantics

When reading a book, does a reader always recognize the meaning of every word used by the author? Or does she always keep a dictionary nearby? Most readers would answer 'No' to both of the previous questions. Not knowing the meaning of a word while reading a piece of text is just a normal 'fact of life'.

In fact, the reader might have been in this position just now, when going over the introductory quote of the chapter, if the meaning of the word *dainties*, used by Isaac Asimov in his *Prelude to Foundation*, was unknown. However, the whole paragraph is meant to give an intuition about the meaning of the word *dainties*: they are served on a *tray*; can be *flavored* or *unflavored*; one can *eat* them; they have a *slightly sweet* taste, and a *fainter bitter aftertaste*. Based on all this evidence, one can conclude that *dainties* are small sphere-shaped things that can be eaten (and presumably taste really nice)[1].

[1]Truth be told, the reason so many 'clues' can be 'picked up' from this particular text is the use of a technique called *coreference resolution*. This means that the understanding process is not restricted

This idea that one can identify the meaning of a word by looking at the words it co-occurs with was introduced by Harris (1954), who considered meaning to be a function of distribution. Harris (1954) takes the **distribution** of an element A to be represented by the sum of all its environments. An **environment** of the element A is taken to be the array of all the other elements that co-occur with the element A. These co-occurring elements, each in a particular position with respect to the target element A, will, together with A, give rise to a meaningful utterance. The elements that co-occur with A in a particular position are called the **selection** of the element A for that position.

Harris[2] further fleshes out the intuition behind the distributional meaning of a word, in saying that knowing the **words** that a target word co-occurs with is not enough to get to the **meaning of the word**. But, "if we know the *meanings of the words with which the word occurs*, we can guess very very closely what the *meaning of the word* is." (from Harris, Bampton Lectures). Harris illustrates this point using the dictionary as a example. In a dictionary, a word is defined in terms of other words, and in many cases an actual context of use of the target word is provided. At the same time, the target word will be part of the explanation of the meaning of other words, and this dependency between words goes on in a seemingly circular way. Looking up the meaning of a word in a dictionary is only useful if the meanings of the words that are used to explain it are already known.

In other words, according to Harris (1954)'s **distributional hypothesis**, the meaning of a word can be found by investigating its usage patterns. Similar intuitions were also presented by Firth (1957), who argued that the meaning of a word can be understood by looking at its surrounding words, or 'the company it keeps' (Firth, 1957). These proposals make sense because of the way humans use language. The words are not just used at random, and sentences are not made of uniformly sampled words from the language's vocabulary. Language is instead made of word patterns, meaningful sequences of words, which affect the likelihood of a word following another, already uttered, word.

In natural language processing, the process of estimating the probability of a word given an already uttered sequence of words is called **probabilistic language modeling**. Probabilistic language modeling is essentially a matter of estimating the joint probability of a word sequence S, made of n words $w_1 w_2 \dots w_n$, i.e. of computing

$$P(S) = P(w_1 w_2 \dots w_n) \tag{4.1}$$

The probability of a word sequence S can be broken down into individual word probabilities, using the **chain rule for probability**:

to using only the sentences where the word *dainties* itself appears. The information in the whole paragraph is used, including words like *little spheres*, *the sphere*, *they're*, *them*, *one*, *it*, all of which refer to *dainties*. The intuition about the meaning of the word *dainties* is built up by combining all the information at hand, both direct and indirect.

[2]In the Bampton Lectures, Columbia University, 1986, lecture 3.2, `http://zelligharris.org`.

$$P(S) = P(w_1 w_2 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \dots P(w_n|w_1 w_2 \dots w_{n-1}) \quad (4.2)$$

The formula for the probability of a word sequence illustrates exactly the point from the paragraph above: an already uttered word will affect the likelihood of the next word in the sequence. For example, if a sequence starts with the word *Humpty*, there's a very strong likelihood that the following word is *Dumpty*, given the typical usage of this sequence. In fact, Brown et al. (1992) estimated, with the help of corpus data, that the pair *Humpty Dumpty* occurs together roughly 6,000,000 times more than one would expect given the individual frequencies of the words *Humpty* and *Dumpty*.

The meaning patterns that are known to the users of a language will impact the new word sequences they produce. The same intuition applies in the case of compounding: new compounds will not magically appear in the language, out of thin air. Instead, the choices of language users will be guided by other compounds they have already encountered, and which use the same or similar constituent words or describe similar patterns of interaction between the constituents. Therefore, it makes sense to investigate such distributional representations as a representational method.

This chapter explores different ways of using the distributional hypothesis to construct useful meaning representations for words. All the approaches presented in the next sections have a common core: first, the meaning of words is captured in the form of mathematical **vectors**. Second, the meaning of all words is represented into using the same 'frame of reference', i.e. in the same **vector space**. This means that the meaning of one word can be easily compared to the meaning of another word, using vector similarity measures. Next, the 'environments' of a word - or its **contexts**, as they will be called in the next sections - are modeled using written text.

The next sections introduce the notion of context and a way to characterize the meaning of words in terms of their co-occurrences. The discussion continues with a discussion about sparse and dense word representations, their properties and methods for transforming sparse into dense representations.

The last subsections are dedicated to introducing several methods for creating dense word representations, either via explicit dimensionality reduction (SVD, PCA), by using neural network models (like Collobert and Weston (2008), `skip-gram`, `CBOW`) or by combining ideas from the explicit dimensionality reduction models with the neural network ones (`H-PCA`, `GloVe`).

The models introduced in these sections will play an important role in the upcoming chapters. These word representations will serve as the starting point for modeling compound semantics. Before modeling compounds, syntactic units which are just above the word level, it's good to have a solid understanding regarding the creation of **individual word representations**. The aim of this chapter is to present the intuitions behind the different families of models and to describe the interactions between them.

There is a vast body of work concerned with creating distributional word representations. Early influential techniques like Brown clustering (Brown et al., 1992), Latent Dirichlet Allocation (LDA) (Blei et al., 2003), ideas on using syntax to define contexts (Hindle, 1990; Lin, 1998), have all shaped the current distributional semantics landscape, and it would be impossible to cover in detail all the proposals made in the literature. This chapter will focus mainly on the techniques that will be used in the rest of the thesis. For a more detailed overview, the interested reader is referred to the surveys in Turney and Pantel (2010); Clark (2012); Erk (2012); Goldberg (2017) and the reference materials mentioned below.

The exposition in this chapter is, unless otherwise stated, based on the chapters *Vector Semantics* and *Semantics with Dense Vectors* from Jurafsky and Martin (2017)[3] and on materials from the Stanford lectures in the course *CS224d: Deep Learning for Natural Language Processing*, winter 2017, taught by Chris Manning and Richard Socher[4].

4.1.1 Setting the Context

Consider, for example, the task of representing the meanings of the words *cucumber*, *tomato*, *chair* and *table*. A first step towards building their vector representations is to collect sentences from a corpus where these words occur, like the ones in Listing 1 below, which were extracted from the `encow14ax` corpus (Schäfer, 2015).

(1) a. A cool and delicious marinated **cucumber**, onion and **tomato** salad.

b. Also on the menu was home-made **tomato** soup made from tomatoes grown in the school's poly-tunnel.

c. Dill also takes a starring role in this Bulgarian cold **cucumber** soup recipe.

d. A "Life" magazine happened to be there on a big wooden **chair**.

e. There was a plain wooden study **table** opposite the bench and he sat on the **chair** next to it .

f. The cook set a bottle, two glasses, and a pie on the **table**.

The next step is to choose some **contexts** to represent the meaning of each of the target words. For simplicity this example will use a two-word **symmetric context** - four words in total - for each target word. If the target word is labeled as w, the four context words considered are w_{-2}, w_{-1}, w_{+1}, and w_{+2}. A negative index, w_{-k}, indicates that the word is the k-th word before the target word, while a positive index, w_{+k}, is used to denote the k-th word after the target word. In sentence 1a, for the target word *cucumber*, w_{-2} is *delicious*, w_{-1} is *marinated*, w_{+1} is *onion*, and w_{+2} is the word *and*. The boxes in Listing 1 indicate the selected contexts for the target words

[3] *Speech and Language Processing* (3rd ed. draft) by Dan Jurafsky and James H. Martin is available online at `https://web.stanford.edu/~jurafsky/slp3/`.

[4] CS224d video lectures available online, see `http://cs224d.stanford.edu/` for details.

cucumber, *tomato*, *chair* and *table*. The context boxes are color-coded with respect to their target word.

Note that the chosen contexts do not cross sentence boundaries: the word *tomato* in sentence 1a has a single word as its right context, and the word *table* in sentence 1f has no right context. Two words can share a context: in example 1a the words *onion and* appear as right context for *cucumber* and as a left context for *tomato*. Punctuation marks are generally not considered as context candidates. For some applications contexts can be extracted for word lemmas instead of word forms. In this example, however, different word forms with a shared lemma have different contexts: the context for *tomato* does not include the words around the form *tomatoes* in sentence 1b.

The choice between using lemmas or word forms when creating distributional representations depends on the particular application. Syntax-oriented tasks like part-of-speech tagging might benefit from representing the word forms directly, whereas more semantic tasks could take advantage of the abstraction level offered by lemmata. In Chapter 5 the implications of this choice will be taken into consideration when building representations for German and English noun compounds.

Once the contexts are chosen, the next step in finding a representation for the target word is to take a look at the distribution of its contexts. Section 4.1.2 describes a way to summarize this distributional information using the word-context co-occurrence matrix.

4.1.2 The Word-Context Co-occurrence Matrix

The contexts selected in the previous step can be used to compute the **word-context co-occurrence matrix**. This matrix captures the co-occurrence patterns of each **target word** with the set of **context words**. The columns of the co-occurrence matrix correspond to each of the target words to be represented (four in the example above: *cucumber*, *tomato*, *chair* and *table*). The number of rows is equal to the cardinality of the set of context words (20 in the example above).

Figure 4.1 displays the co-occurrence matrix for the target words *cucumber*, *tomato*, *chair* and *table*, using the mini-corpus in Listing 1 and a two-word symmetric context window. The co-occurrence matrix is populated by counting how many times each word in the set of context words appears in the context of a target word. For example the word *delicious* appears 1 time in the context of the target word *cucumber*, but 0 times in the context of the target word *chair*. Similarly, the word *wooden* appears 1 time in the contexts of *chair* and *table*, but never in the contexts of *cucumber* and *tomato*. Another observation is that words with a similar meaning will tend to have the same words in their context: *cucumber* and *tomato* share three context words: *onion*, *and* and *soup*. A dashed box indicates the common context words for two target words in Figure 4.1. Conversely, words with little semantic overlap will have few or no common context words (e.g. *tomato* and *chair* share no context words).

The choice of context size will have a large impact on the co-occurrences that will be gathered. The restricted definition of context in the example above meant that

some words that could have been relevant for capturing the meaning of the target word have actually been discarded. An example are the words *grown* and *poly-tunnel* in sentence 1b, which would have made good context words for the word *tomato*. A definition of context with a different take on lemmatization would have included the verb *grown*, which appears next to *tomatoes*. A larger context window, of size 5, would have also included the word *poly-tunnel*. The choice of context size should generally be guided by the application foreseen for the word representations. Levy and Goldberg (2014a) observe that a window size of 5 (symmetrical, using in effect 10 words as context) is commonly used to capture "broad topical content", while smaller windows correspond to "more focused information about the target word".

target words

context words

	cucumber	tomato	chair	table
delicious	1	0	0	0
marinated	1	0	0	0
onion	1	1	0	0
and	1	1	0	0
salad	0	1	0	0
was	0	1	0	0
home – made	0	1	0	0
soup	1	1	0	0
made	0	1	0	0
Bulgarian	1	0	0	0
cold	1	0	0	0
recipe	1	0	0	0
big	0	0	1	0
wooden	0	0	1	1
study	0	0	0	1
opposite	0	0	0	1
the	0	0	1	2
on	0	0	1	1
next	0	0	1	0
to	0	0	1	0

Fig. 4.1 Co-occurrence matrix for the target words *cucumber*, *tomato*, *chair* and *table*, using the mini-corpus in Listing 1 and a two-word symmetric context window. Dashed boxes indicate common context words for two target words.

Another apparent problem is that a good percentage of the initial words in the sentences seem to be discarded. Remember, however, that a meaning of the word stands in this circular relationship with the meanings of the other words in its context. This means that if all the words in a sentence are modelled, and not just the chosen few, the illustration of context for sentence 1a would look something like Figure 4.2, and the word at each position would be used as a context for the other words between k and $2k$ times, where k is the size of the context window. This analysis suggests that the most cost-effective way to make use of a given piece of text is to represent

most (or all) of the words it contains. The neural word representations presented in Section 4.3.2 use the training text exactly in this way.

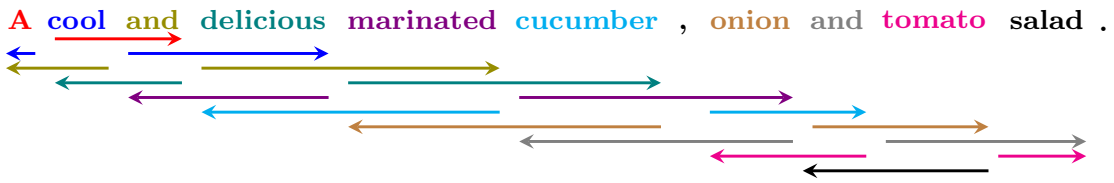


Fig. 4.2 Illustrating a two-word context window for a full sentence. The arrows show the span of the left and right context for each target word, and are color coded with respect to the target word. Note that the first word of the sentence only has a right context, while the last word only has a left context.

4.2 Sparse Vector Representations for Words

Once the word-context co-occurrence matrix is constructed, each target word can be represented via its corresponding column vector (e.g. the representation for the word *chair* is the column vector highlighted in gray in Figure 4.1). The vector representation of a word w is a vector $\mathbf{u} \in \mathbb{R}^n$, where n is the cardinality of the set of context words. Real-world context sizes range from thousands to millions of context words, so this type of word vectors are high-dimensional. Additionally, as each target word usually co-occurs only with a small subset of the context words, the vectors are sparse, i.e. most of their entries are 0.

As mentioned in Chapter 3, Section 3.5.1, the advantage of representing words as vectors is that vector similarity metrics can be used to assess word similarity. A popular similarity metric for word vector spaces is the **cosine similarity**, a metric that computes the normalized dot product between two vectors[5]. The cosine similarity between two n-dimensional vectors $\mathbf{u}$ and $\mathbf{v}$ can be computed using the formula in Equation 4.3.

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}} \tag{4.3}$$

Equation 4.3 can be used to compute the similarities between target words:

[5]The dot product formulation was presented in Chapter 3, Eq. 3.25.

$$\cos(\mathbf{u}_{cucumber}, \mathbf{u}_{tomato}) = \frac{1 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + \ldots + 0 \cdot 0 + 0 \cdot 0}{\sqrt{1^2 + 1^2 + \ldots + 0^2 + 0^2}\sqrt{0^2 + 0^2 + \ldots + 0^2 + 0^2}} = \frac{3}{\sqrt{8}\sqrt{7}} = 0.4008$$

$$\cos(\mathbf{u}_{chair}, \mathbf{u}_{table}) = \frac{0 \cdot 0 + 0 \cdot 0 + \ldots + 1 \cdot 0 + 1 \cdot 0}{\sqrt{0^2 + 0^2 + \ldots + 1^2 + 1^2}\sqrt{0^2 + 0^2 + \ldots + 0^2 + 0^2}} = \frac{4}{\sqrt{6}\sqrt{8}} = 0.5773$$

$$\cos(\mathbf{u}_{tomato}, \mathbf{u}_{chair}) = \frac{0 \cdot 0 + 0 \cdot 0 + \ldots + 0 \cdot 1 + 0 \cdot 1}{\sqrt{0^2 + 0^2 + \ldots + 0^2 + 0^2}\sqrt{0^2 + 0^2 + \ldots + 1^2 + 1^2}} = \frac{0}{\sqrt{7}\sqrt{6}} = 0$$

The cosine similarity is 1 when the two vectors are identical, 0 when the vectors are orthogonal (at a right angle) and -1 when the vectors are opposite[6]. The cosine similarity between two word vectors can be seen as a measure of context overlap: the cosine similarity grows with the number of common contexts. High cosine similarity values correspond to words which occur only with a small number of context words and share most of the contexts, like it is the case for *table* and *chair*, each co-occurring with 5 and 6 words respectively, out of which 3 occurrences are in common. Low cosine similarity values are associated with words that share little or no contexts (e.g. *tomato* and *chair*).

Co-occurrence counts provide a simple and intuitive way to construct vector representations for words. Their simplicity, however, must be balanced against their shortcomings. One such shortcoming is that function words such as *the*, *and*, *on* are frequent co-occurrence partners for most target words. However, the regular co-occurrence of a target word with a function word will provide less information about the meaning of the target word than a comparatively less frequent co-occurrence with a non-function word. Knowing that *tomato* appears frequently with *the* and *and* is not very useful for representing the meaning of *tomato*, because so do the majority of the other target words. However, the fact that *tomato* co-occurs (albeit with fewer counts) with *salad* and *soup* is more informative, as *salad* and *soup* only appear in the contexts of a small subset of target words.

A measure that is able to overcome this shortcoming is **pointwise mutual information (PMI)**, proposed by Fano (1961) and adapted to the the use in natural language processing by Church and Hanks (1990). Church and Hanks (1990) define the mutual information of two words x and y with the probabilities $P(x)$ and $P(y)$ as:

$$I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} \tag{4.4}$$

The intuition behind Equation 4.4 is that one can learn more about the co-occurrence patterns of the words x and y by comparing the number of times x and y appear together in a fixed-size context - their joint probability $P(x, y)$ - to the occurrence probabilities of x and y - $P(x)$ and $P(y)$. If the words x and y have a higher probability of occurring together than separately ($P(x, y) > P(x)P(y)$), their mutual information

[6]When using count-based co-occurrence matrices, the cosine similarity will only take values between 0 and 1; weighted co-occurrence matrices (introduced in the next sections) allow for negative association scores between words, and the cosine similarity of such vectors will therefore span the full codomain of the cosine function, $[-1, 1]$.

is positive. If x and y appear together as expected by chance ($P(x, y) \approx P(x)P(y)$), the mutual information is close to 0. The mutual information can also be negative, when the two words occur together less than expected by chance.

The individual probabilities of the words x and y are estimated using large text corpora. The probability of a word x in a corpus with N individual tokens is given by the number of occurrences of x in the corpus, divided by the size of the corpus N. The joint probability $P(x, y)$ can be estimated by counting the number of times the word x appears together with the word y in a context of fixed size, k, and dividing this count by the size of the corpus, N.

Coming back to the goal of building word representations, the co-occurrence matrix in Figure 4.1 can be transformed into a PMI matrix. Following the notation proposed by Levy et al. (2015a), the co-occurrence matrix can be represented in terms of a set of target words $w \in V_W$, a set of context words $c \in V_C$ and a set of observed word-context pairs $(w, c) \in D$. $\#(w, c)$ is used to denote the number of times the pair (w, c) occurs in D. The total number of occurrences of a target word w in any of the contexts is represented as $\#(w) = \sum_{c' \in V_C} \#(w, c')$, while $\#(c) = \sum_{w' \in V_W} \#(w', c)$ represents the number of times the context c is seen in conjunction with any target word w'. The PMI can then be computed as in Eq. 4.5:

$$PMI(w, c) = log_2 \frac{\hat{P}(w, c)}{\hat{P}(w)\hat{P}(c)} = \log_2 \frac{\#(w, c) \cdot \sum_{c \in V_c} \#(c)}{\#(w) \cdot \#(c)} = log_2 \frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \quad (4.5)$$

Negative estimates can be misleading unless the corpora are extremely large, so in practice a variant of pointwise mutual information is usually used, **positive pointwise mutual information (PPMI)**, where all negative PMI values are replaced with 0 (Church and Hanks, 1990; Niwa and Nitta, 1994) - see Eq. 4.6. Applying the PPMI weighting scheme to the co-occurrence matrix in Figure 4.1 results in the co-occurrence matrix displayed in Figure 4.3.

$$PPMI(w, c) = \max(PMI(w, c), 0) \quad (4.6)$$

The word interactions are now painted in a different light using the PPMI weighting: the one occurrence of *delicious* and *cucumber* is assigned a large weight, 1.7548, while the co-occurrence of *the* with *chair* is assigned 0.5849, thus conveying the weaker association to the function word *the*, which was, in the example corpus, also a frequent collocate of *table*.

The cosine similarity can also be recomputed based on the new PPMI-based word representations, $\vec{u_p}$:

$$\cos(\vec{u_{p_{cucumber}}}, \vec{u_{p_{tomato}}}) = 0.1227$$
$$\cos(\vec{u_{p_{chair}}}, \vec{u_{p_{table}}}) = 0.2306$$
$$\cos(\vec{u_{p_{tomato}}}, \vec{u_{p_{chair}}}) = 0$$

The PPMI weighting scheme tends to favor rare contexts: the largest values associated with each target word in Figure 4.3 (maximum value per column) are the ones where the context word is associated with only one target word: *delicious*, *marinated*, *Bulgarian*, *cold* and *recipe* for *cucumber*; *salad*, *was*, *home-made* and *made* for *tomato* etc. This property makes PPMI assign a disproportionately high weight to such infrequent contexts, in the detriment of contexts that occur more frequently with the target word on the basis of a meaningful semantic association. Levy et al. (2015a) proposed context distribution smoothing as a solution: all the context counts are raised to the power α before the PMI computation, leading to the new PMI formula listed in Eq. 4.7.

target words

context words	cucumber	tomato	chair	table
delicious	1.7548	0	0	0
marinated	1.7548	0	0	0
onion	0.7548	0.9474	0	0
and	0.7548	0.9474	0	0
salad	0	1.9475	0	0
was	0	1.9475	0	0
home – made	0	1.9475	0	0
soup	0.7548	0.9474	0	0
made	0	1.9475	0	0
Bulgarian	1.7548	0	0	0
cold	1.7548	0	0	0
recipe	1.7548	0	0	0
big	0	0	2.1699	0
wooden	0	0	1.1699	1.1699
study	0	0	0	2.1699
opposite	0	0	0	2.1699
the	0	0	0.5849	1.5849
on	0	0	1.1699	1.1699
next	0	0	2.1699	0
to	0	0	2.1699	0

Fig. 4.3 PPMI-weighted co-occurrence matrix for the target words *cucumber*, *tomato*, *chair* and *table*. Maximum PPMI values for every target word (for every column) are highlighted in red.

$$PMI_\alpha(w,c) = \log_2 \frac{P(w,c)}{P(w)P_\alpha(c)} = \log_2 \frac{\#(w,c) \cdot \sum_{c \in V_c} \#(c)^\alpha}{\#(w) \cdot \#(c)^\alpha} \tag{4.7}$$

Setting α to 0.75 was proven beneficial both by Levy et al. (2015a) and Mikolov et al. (2013b). In fact, Levy et al. (2015a) judge this type of context smoothing to be one of the most consistent ways to improve the resulting word representations.

This section introduced the idea of sparse vector representations for words, as well as the basic mechanisms for building such representation using contexts extracted

from large corpora. Although sparse representations are a simple and intuitive way to create meaning representations for words, they do have some shortcomings. One is their high-dimensionality, which, as mentioned in Section 3.5.1, can be problematic when using these representations in a particular learning task, because they lead to an increase in the number of required parameters.

Another problem associated with sparse vector representations is that they cannot 'see' much beyond the actual set of contexts they are trained on. For example, if a woman describes her latest *holiday* using the words *photo*, *card*, *beach*, and her husband uses instead the words *picture*, *postcard* and *seaside* to describe the same *holiday*, it might seem that the couple is not providing a meaning for the same word. If $holiday_1$ is the representation obtained using the woman's description of the *holiday*, and $holiday_2$ as the representation obtained using the husband's description, the two representations have no words in common, although there seems to be a high similarity between the descriptions in terms of word pairs: (*photo*, *picture*), (*card*, *postcard*), (*beach*, *seaside*).

The sparse vector representation cannot take advantage of the semantic similarity between these word pairs. The sparse representation of *holiday*, obtained using both descriptions, will make use of all the six words above. Other words will be judged to be very similar to *holiday* only if they co-occur, again, with all the six words. However, this example shows that a good meaning representation should be able to abstract away from the lexical level, and consider the words that co-occur only with a subset of the words from a particular theme also to be similar. A word like *vacation* should be judged to be similar to *holiday* even if it only co-occurs with the subset of words, e.g. *photo*, *postcard* and *beach*.

The next section introduces several ways of enhancing the representational capabilities of sparse vector representations, either directly, via dimensionality reduction, or indirectly, by using neural networks to process the information provided by word co-occurrences.

4.3 Dense Word Representations

Dense vector representations, or **word embeddings**, are the result of embedding high-dimensional, sparse word representations into a low-dimensional vector space. The size of a sparse vector representations is given by $|V_C|$, the size of the context vocabulary, and can, as detailed in Section 4.2, range from thousands to millions of words. In contrast, dense vector representations reside in a much smaller vector space, typically ranging from 50 to 1600 dimensions.

4.3.1 Explicit Dimensionality Reduction

The idea of obtaining more compact word representations by applying dimensionality reduction was initially used in the context of information retrieval by Deerwester et al. (1990), who entitled their technique **Latent Semantic Analysis (LSA)**. LSA

involves constructing a **term-document matrix**, where the **term** is a word from a search query and the **documents** are the text files to be searched.

Translating these terms into the terminology introduced in the previous section, the term would correspond to the target word to be modelled, whereas a document represents a chosen context. The entries of the matrix are filled in just as in the case of the word-context matrix. The term-document matrix is then reduced to a lower rank approximation using **singular value decomposition (SVD)**.

SVD is a linear algebra operation that can be applied to any rectangular matrix X of size $m \times n$ and rank r. Such a matrix X can be factorized into a product of three matrices: an orthogonal matrix U, a diagonal matrix Σ and (the transpose of) another orthogonal matrix, $V^\top$, as shown in Eq. 4.8:

$$\underset{m\times n}{X} = \underset{m\times r}{U}\ \underset{r\times r}{\Sigma}\ \underset{r\times n}{V^\top} \tag{4.8}$$

The matrix Σ contains the **singular values** of the initial matrix X, from the largest to the smallest, while the matrices U and $V^\top$ contain X's left- and right-singular vectors. In effect, Σ, U and $V^\top$ break down the initial information of the matrix X into **linearly independent** components. This means that the information captured by each of these components corresponds to an unique perspective over X's content. The **dimensionality reduction** is obtained by discarding all but the top k elements of Σ ($k \leqslant r$). This corresponds to keeping from the original matrix X only the top k dimensions, which account for most of the variation. The top k elements of Σ are sufficient to approximate X as $\hat{X}_k$, the matrix of rank k which is the closest to the original matrix X in the least squares sense, i.e. where $||X - \hat{X}_k||_2$ is minimal. The form of $\hat{X}_k$ is given in Eq. 4.9.

$$\underset{m\times n}{\hat{X}_k} = \underset{m\times k}{U_k}\ \underset{k\times k}{\Sigma_k}\ \underset{k\times n}{V_k^\top} \tag{4.9}$$

Figure 4.4 illustrates the SVD factorization operation, as well as the process of approximating X using $\hat{X}_k$. The value of k is generally heuristically defined: it should be large enough to allow the representations to capture most of the interesting aspects from the initial co-occurrence data. At the same time, it should be smaller than the initial rank r of the original X matrix - to allow discarding minor variations that act as noise, and to obtain a good dimensionality reduction.

The same dimensionality reduction technique can also be applied on the word-context matrix, and the literature contains several proposals along this line. Schütze (1992) takes context to be a window of 1000 characters, with the intuition that a few long words are as good as more short words for representing a particular target word. The 1000-dimensional space is reduced using SVD to a 97-dimensional space.

An interesting observation is that Schütze (1992) shows that the representations obtained via SVD are **distributed**. As discussed in Section 3.5.2, distributed representations have many advantages over local representations - in particular their robustness at dealing with partial inputs. Schütze (1992) experiments with a classification task

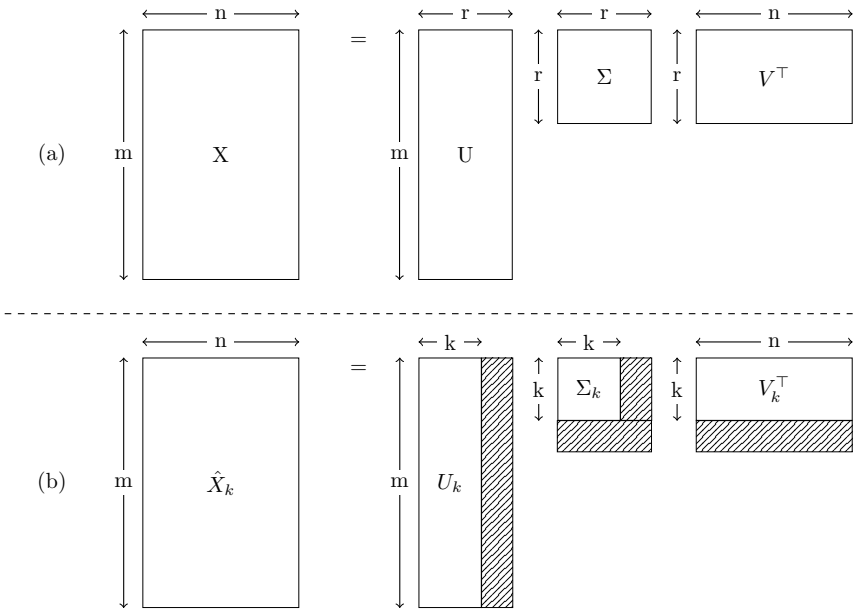


Fig. 4.4 Dimensionality reduction using Singular Value Decomposition (SVD); (a) initial factorization of the matrix X; (b) $\hat{X}_k$ is the approximation of X obtained by zeroing out all but the top k elements of Σ and discarding the corresponding columns of U and the corresponding rows of $V^\top$.

where he uses only parts of the SVD-induced representation, and shows that discarding parts of the representation does not lead to a marked increase in the error rate. His experiments suggest that the vector representations obtained via SVD are redundant enough to allow for such discarding of elements, without this leading to a total loss of meaning in the remaining representation.

The dimensions resulting after applying SVD no longer have a well-defined meaning. This was previously not the case for sparse vector representations, where the dimensions corresponded to the chosen context words. In contrast, the dimensions of dense representations capture some more abstract aspects of meaning, and there is no general method for discovering what exactly they capture. The new dimensions abstract common traits from the initial vector space, and do not rely anymore on particular words. These new representations are thus able to overcome the representational problem associated with sparse vector representations: they capture enough similarities in the data in order to judge words that have only partially overlapping contexts as being similar.

The dense word representations obtained by applying SVD are, at the same time, both **distributed** and **distributional**. **Distributed**, because the representation is spread across the whole vector and does not reside in any particular component of the vector. **Distributional**, because the representations are modeled according to the distributional hypothesis - by computing word usage metrics over large text corpora.

When using SVD for creating word representations, it is common to represent the words and the contexts using the formula in Eq. 4.10 (Dinu et al., 2013a). Each word in the vocabulary V_W is represented by a k-dimensional row vector from W, while each context is represented by a k-dimensional column of the matrix C.

$$W = U_k \; \Sigma_k \qquad C = V_k \tag{4.10}$$

Levy et al. (2015a) showed that weighting the values in the singular value matrix Σ_k can improve the quality of the embeddings, and suggest using factorizations like the one in Eq. 4.11, with $(\Sigma_k)^0$, or like the one in Eq. 4.12, using $(\Sigma_k)^{\frac{1}{2}}$.

$$W = U_k \qquad C = V_k \tag{4.11}$$

$$W = U_k \; \sqrt{\Sigma_k} \qquad C = V_k \sqrt{\Sigma_k} \tag{4.12}$$

Explicit dimensionality reduction is one way of obtaining dense vector representations. The literature on distributional semantics also contains proposals that model word meaning directly as dense, real-valued vectors. A selection of these proposals will be covered in the next section.

4.3.2 Implicit Dimensionality Reduction - Neural Word Embeddings

The idea of directly creating dense, real-valued distributed word representations using large collections of texts was introduced by Bengio et al. (2003) in the context of constructing probabilistic language models. Bengio et al. (2003) proposed a model where each word (from a fixed vocabulary) is associated with a distributed word feature vector $\in \mathbb{R}^n$. The joint probability function of word sequences is then expressed in terms of these distributed word feature vectors. The goal of the model is to learn, at the same time, both the word feature vectors and the parameters of the probability function. Bengio et al. (2003) use, in their experiments, a vocabulary of 17,000 words, and feature vectors of size 30, 60 and 100, and are able to prove the superiority of the new models when compared to smoothed trigram methods.

The intuition behind the model proposed by Bengio et al. (2003) is that similar words - (*cat*, *dog*), (*a*, *the*), (*room*, *bedroom*), (*running*, *walking*) should be assigned similar representations. Training the model on a word sequence like *The cat is walking in the bedroom* will inform the model not only about this particular sentence, but also about the neighboring 'cloud' of similar sentences, like the one in Listing 2, and thus lead to much better generalization capabilities.

(2) a. The cat is walking in the bedroom
b. A dog was running in a room
c. The cat is running in a room
d. A dog is running in a bedroom

Bengio et al. (2003) trained the probabilistic language model using a feed-forward neural network. The word representations are modeled by a linear projection layer (a lookup table, see below), and the interactions between the words are captured via a non-linear hidden layer. The network uses a softmax[7] output layer, which computes the probability of the next word given an input sequence, over the whole vocabulary. This large softmax layer and the accompanying log-likelihood criterion incurred a very high computational cost, and limited the size of the language models that could be trained effectively using this approach.

A next step in building neural word embeddings was made by Collobert and Weston (2008), who used a much cheaper ranking-type criterion to train their word representations[8]. It is exactly this ranking-type criterion that made a difference in practice between the proposal made by Bengio et al. (2003) and the one made by Collobert and Weston (2008). The criterion involved only a pairwise comparison between a correct word sequence and its corrupt counterpart. This gave the model the ability to scale to a much larger vocabulary - 30,000 words - almost double the vocabulary size used by Bengio et al. (2003).

Collobert and Weston (2008) train a language model using as input a text window of fixed size and a neural network. The network has to learn to discriminate between s, the original windows of text that were obtained from the data and s^w, corrupted versions of these windows of text, where the original middle word from s has been replaced by another word, w, chosen randomly from the vocabulary. E.g. for an original window of size 5 with the words *The cat is walking in*, a possible corrupted window is *The cat sky walking in*.

The network is then trained to minimize the ranking-type cost presented in Eq. 4.13, where $f(s)$ is the score assigned by the network described using the function f to the correct sequence s, and $f(s^w)$ is the score assigned by the same network to s^w, the corrupted version of s, obtained by replacing the middle word with w, a random word from the dictionary.

$$\sum_{s \in S} \sum_{w \in V_C} \max(0, 1 - f(s) + f(s^w)) \tag{4.13}$$

The idea behind this cost function is that the neural network f should learn to assign higher scores to the original sequence s than to the corrupted version s^w. It is easier to see this if Eq. 4.13 is rewritten as in Eq. 4.14:

$$\sum_{s \in S} \sum_{w \in V_C} \max(0, 1 - (f(s) - f(s^w))) \tag{4.14}$$

The function in Eq. 4.14 reaches its minimum when the term $f(s) - f(s^w)$ is 1, meaning that the score for the correct sequence, s, is higher than the score of the

[7] see Eq. 3.6 and the surrounding discussion for an explanation of softmax.

[8] However, it's worth pointing out that even with this cheaper ranking criterion training the word embeddings took approximately two months - see Collobert et al. (2011b) for details.

incorrect one, s^w, by a *margin* of 1. If the difference $f(s) - f(s^w)$ is smaller than the margin 1, say 0.7, then the remaining 0.3 increases the error associated with the function f. The most unfavorable case, where $f(s) - f(s^w)$ is negative - meaning that the f is ranking s lower than s^w - leads to a penalty term that is larger than 1.

A key ingredient in Collobert and Weston (2008)'s architecture is the **lookup table**, LT_W, which allows for every word i from a vocabulary V_W to be embedded into a d-dimensional space. The lookup table is illustrated in Fig. 4.5, and its mode of operation is given by Eq. 4.15,

$$LT_W(i) = W_i \tag{4.15}$$

where $W \in \mathbb{R}^{d \times |V_W|}$ is a matrix of parameters containing the vector representations of all the words from V_W that need to be learned and $W_i \in \mathbb{R}^d$, the i^{th} column of the matrix W, is the representation of the word i from the vocabulary V_W. All the representations are of size d, a hyperparameter of the model that is chosen once by the user at the beginning of the training process.

A sequence of n words, $\{w_1, w_2, \ldots, w_n\}$ is transformed into a sequence of vectors $\{W_{w_1}, W_{w_2}, \ldots, W_{w_n}\}$ by applying the lookup table operation for each of the individual words in the sequence.

Fig. 4.5 A lookup table for a vocabulary V_W with word embeddings of size d. Highlighted in red is W_3, the vector of the 3^{rd} word of the vocabulary. The real-valued components of the word representations are presented in this illustration as filled circles.

The architecture proposed by Collobert and Weston (2008) also features **convolutional layers**. Convolutional layers allow the network to work with sentences of varying length. A parameter of the convolution is the *kernel size* (ksz), which defines the width of the convolution.

The convolution itself is a linear operation parametrized by a weight matrix $L \in \mathbb{R}^{d \times ksz}$ and a bias vector $\mathbf{b} \in \mathbb{R}^d$. It can capture the most relevant aspects of a sentence by considering all the sequences of kzs words in the sentence.[9] The same convolution matrix is then applied to each of the word sequences. Using a convolutional architecture the network can make predictions that are based not only on a limited window of words, but on the information of the whole sentence.

An important contribution of Collobert and Weston (2008) was to show that one could leverage the power of unlabeled data to train meaningful word representations. The word representations could then be further fine-tuned for other supervised NLP tasks, such as POS tagging, chunking, named entity recognition, semantic role labeling

[9]Considering the most simple case, where the step of the convolution equals 1.

and for predicting if two words are semantically related (i.e. if they are synonyms, holonyms, hypernyms, etc.). It is important to note that in such a setup the word embeddings are mostly trained using the unlabeled data. The annotated data - the gold standards for all the previously mentioned tasks - is usually several orders of magnitude smaller than the unlabeled data available and only serves to make the representations better fit a particular task.

This point was further developed in Collobert et al. (2011b), in their paper entitled *'Natural Language Processing (almost) from Scratch'*. Collobert et al. (2011b) make a strong point about the effectiveness of the word representation trained on large amounts of unannotated text on a host of NLP tasks, both syntactic and semantic in nature. They show that a network trained in a completely supervised manner, using gold data for the semantic role labeling task is not able to induce a coherent set of vector representations, where similar words are neighbors of each other. By contrast, a network trained in a semi-supervised fashion, using large amounts of unlabeled data and a ranking-type criterion like the one in Eq. 4.13 can give rise to meaningful word representations, where (some of) the similarities observed in the word space are preserved in the vector space[10].

The word embeddings induced in this way generalize well across tasks, and lead to results comparable to the state-of-the-art on the respective tasks. However, instead of relying on the traditional way of engineering features independently for each of the tasks, the approach using word embeddings achieved results comparable to the state-of-the-art *for all the tasks*, using just *one common set of features* - the word representations learned in a semi-supervised manner from unlabeled text.

The work done by Collobert and Weston (2008); Collobert et al. (2011b) helped in switching the focus from building probabilistic language models that learn to predict the next word to models whose main goal is just to learn good word representations.

The large scale adoption of word embeddings as features for various NLP tasks came with the public release of `word2vec`[11], a publicly available implementation for training distributed word representations using very large collections of text. `word2vec` was introduced in Mikolov et al. (2013a). It describes two algorithms for estimating word representations using neural networks: the **skip-gram model** (`skip-gram`) and the **continuous bag-of-words model** (`CBOW`).

The two models can be thought of as reflections of each other, since they differ mostly in what they consider to be the input and what they try to predict as the output of the network. The `skip-gram` model predicts context words given the target word. The `CBOW` model takes the opposite view and tries to predict the target word using the surrounding bag-of-words context as input. Figure 4.6 illustrates the input and the prediction of the two models for the same window of text.

The objective of the `skip-gram` model is shown in Eq. 4.16. The idea is that given a sequence of training words w_1, w_2, w_3, ..., w_T, the goal is to learn to predict the m

[10] Observe, in particular, the difference between the neighbors of the words in Table 5 and Table 6 from Collobert et al. (2011b).

[11] `word2vec` archive, `https://code.google.com/archive/p/word2vec/`.

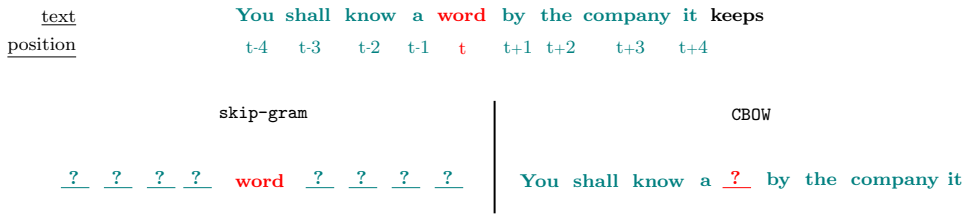


Fig. 4.6 Illustration of the input and output of the `skip-gram` and `CBOW` models, using a context window of size 4. The target word is highlighted in red, the context words are colored in teal, and the words that are not part of the current context are black. The slots marked with a question mark are supposed to be predicted by each of the models. The example sentence is Firth (1957)'s famous quote, capturing the essence of distributional semantics.

context words that surround the target word, w_t. The context words can be placed both to its left ($j < 0$) and to its right ($j > 0$).

$$J(\Theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leqslant j\leqslant m\\ j\neq 0}} \log p(w_{t+j}|w_t) \tag{4.16}$$

In the basic `skip-gram` formulation $p(w_{t+j}|w_t)$ is defined using a `softmax`, like in Eq. 4.17,

$$p(w_O|w_I) = \frac{\exp(u_{w_O}^\top v_{w_I})}{\sum_{w=1}^{|V_C|}\exp(u_w^\top v_{w_I})} \tag{4.17}$$

where w_I is the input word (the target word) and w_O is the output word to be predicted (each one of w_I's context words in turn). u and v are two vectorial representations associated to any $w \in V_C$[12]: u_w is the vector representation of the word w when w is a context word and v_w is w's representation as a target word.

Goldberg and Levy (2014) provide an intuition as to why this dual representation of w with the two vectors u_w and v_w makes sense: if one takes any word, e.g. *dog*, it is very unlikely that the word *dog* itself will appear in its own context. When using a single vector, x_{dog}, the model would assign a low probability to $p(dog|dog)$, leading to the dot product $x_{dog}^\top x_{dog}$ to be low, which is not possible (a vector will always be similar to itself). This problem is solved by having the word be represented by two separate vectors, one representing the word as the target and the other representing the word as a context word.

It follows that the model parameters that have to be learned reside in two matrices, $W_{target} \in \mathbb{R}^{d\times|V_C|}$ and $W_{ctx} \in \mathbb{R}^{d\times|V_C|}$, containing the target and the context vectors of dimensionality d for every word in the vocabulary V_C. The elements of these two

[12]Note that in this model V_W and V_C coincide, so any of them can be used in the definitions.

matrices represent in fact Θ, the parameters of the model, whose optimal value have to be discovered using the objective function in Eq. 4.16.

An important observation is that in the `skip-gram` formulation, the position of the content word with respect to the target word does not matter. The only thing that matters is the identity of the word itself. In the example sentence in Fig. 4.6, given the target word *word*, the model just has to predict the identity of the context words *You*, *shall*, *know*, *a*, *by*, *the*, *company* and *it*. The context vectors always come from the same matrix W_{ctx}. If, say, the word *company* would occur in another context window at position $t-1$ (instead of position $t+3$, like in Fig. 4.6), it would still be represented by the same context vector from W_{ctx}.

Another noteworthy detail is that after the training procedure is completed, W_{ctx}, the matrix of context representations for every $w \in V_C$, is discarded, and the words are represented using only the representations in W_{target}.

Why does this work? Why can such an objective lead to the model learning good word representations? The intuition here is that the only way to minimize the objective function is to push the parameters of the network towards configurations where the words that frequently appear in each other's context have similar vectors. Moreover, because the representations are distributed and continuous, the update of one word vector will in effect have an impact onto the neighbors of the word. Making a particular word better fit its context makes the whole 'cloud of words' around it better fit that context.

Mikolov et al. (2013c) introduce a test set for assessing the quality of different sets of word representations. The test set contains analogy questions of the form 'a is to b as c is to ?'. The relation between the elements of the analogy can be a syntactic one, i.e. the singular/plural of common words - e.g. *year*:*years law*:*?*, the base/superlative/comparative forms of adjectives - e.g. *good*:*better rough*:*?*), etc. In addition, the word representations are also tested using a set of semantic relations, thus probing the semantic information present in the word representations. Examples of such analogies are further detailed in Mikolov et al. (2013a). They include relations like the one between a capital city and its country (e.g. *Athens*:*Greece Oslo*:*?*), between a country and its currency, or between the masculine and feminine form of particular concepts (e.g. *man*:*woman king*:*?*). In the case of the semantic relations the system has to guess the second term of the second pair (i.e. *Norway* and *queen*), just like in the case of the syntactic analogies.

Mikolov et al. (2013c) and Mikolov et al. (2013a) show that the information captured by the word representations allows this type of analogies to be solved using a simple vector offset method. The vectors are first normalized to unit norm, and then the answer vector, y, is computed using the formula in Eq. 4.18.

$$y = x_b - x_a + x_c, \text{where the analogy is expressed as } a{:}b\ c{:}?(d) \tag{4.18}$$

The answer vector y is then compared using the cosine similarity measure to all the vectors of the other words in the vocabulary, and the nearest neighbor $x_{\hat{d}}$ is returned

as an answer. The analogy test is considered to be correctly resolved if $\hat{d}$ turns out to be exactly the fourth element of the analogy, the word d. On a dataset containing ∼20,000 analogy questions, the `skip-gram` model was able to find the correct answer in ∼60% of the cases (Mikolov et al., 2013a).

The implications of this result is that some of the relations captured by the word representations can be expressed in terms of linear vector differences. If one could separate out these differences, and figure out a general method for picking the base vectors to apply them to, a concept could then be represented using several 'characteristics', i.e. gender (*man-woman*; *woman-man*), social status (*king-man*; *man-king*), etc. However, such methods for automatically discovering meaningful characteristics of distributional word representations are yet to be discovered.

Mikolov et al. (2013b) extend the `word2vec` formulation and replace the rather impractical softmax over the whole vocabulary with alternative formulations. The first one, the **hierarchical softmax**, had already been described in Mikolov et al. (2013a). The second one, the **negative sampling** formulation, is introduced in Mikolov et al. (2013b) as a more simple alternative to the hierarchical softmax. The intuition behind negative sampling is that one does not need to update the parameters of all the incorrect context words at every evaluation step (since this would mean updating most of the weights in the context matrix). Rather, a fixed number of negative samples are chosen from a unigram distribution at each evaluation step, such that more frequent words will be updated more frequently than less frequent words. This has the effect that each update will impact only a small percentage of the weights in the context matrix, and thus the training will proceed faster.

Mikolov et al. (2013b) also make an attempt at modeling phrases using the same approach. Their approach involves automatically identifying phrases in text by scoring each pair of words using the formula in Eq. 4.19, where δ is a coefficient that prevents too many phrases made of words with infrequent counts to be created:

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)} \tag{4.19}$$

The bigrams in the training corpus would be scored multiple times using this formula, and if above a certain threshold they would be considered to be a phrase and would be added to the dictionary. The authors mention that using 2-4 passes over the data with a decreasing threshold they can also pick up longer phrases in the data. This procedure increases the size of the vocabulary to 3 million items (both single-token words and phrases). While many of the phrases are names of entities (e.g. *Grand_Hotel_Timeo*, *Kaiser_Aluminum_Corp*, etc.), some of the phrases in this over-sized vocabulary turn out to be nominal compounds (e.g. *glass_carafe*, *potato_patties*, *pear_cider*, etc.).

4.3.3 To Count, or to Predict, that is the Question

The effectiveness of the word representations trained using a neural network model (Collobert and Weston, 2008; Collobert et al., 2011b; Mikolov et al., 2013a,b) made it look like by predicting the identity of a word given its context (or of the context given a word) one can capture similarities of a completely different nature than the similarities captured using only co-occurrence counts.

An initial claim, made by Baroni et al. (2014), was that the predict vectors were superior to the count vectors. In their study, Baroni et al. (2014) built count vectors using the DISSECT toolkit (Dinu et al., 2013a) and experimented with different weighting schemes (PPMI, Local Mutual Information) and different factorization algorithms (SVD, Non-negative Matrix Factorization). Their predict vectors were trained using the `CBOW` method of `word2vec`. In their evaluation, the predict vectors emerge as clear winners across a wide range of semantic tasks.

This claim was, however, disputed in several other papers (Levy et al., 2014; Lebret and Collobert, 2015a; Levy et al., 2015a), which showed that if the hyperparameters are carefully tuned for both types of models, the word representations induced using count methods can be as competitive as the ones obtained by the predictive methods. Moreover, Levy and Goldberg (2014b) showed that the `skip-gram` model using k negative samples performs an implicit factorization of the matrix M, as shown in Eq. 4.20. The entries of M_{ij} correspond to the dot product between the target representation of the word i and the context representation of the word j:

$$M_{ij} = W^i_{target} \cdot W^j_{ctx} = \vec{v_i} \cdot \vec{u_j} = PMI(\vec{v_i}, \vec{u_j}) - \log k \tag{4.20}$$

The next sections introduce two alternatives to the prediction-based models, that rely, however, on some of the intuitions gained from the prediction-based modeling: the H-PCA model introduced by Lebret and Collobert (2014) and the `GloVe` model introduced by Pennington et al. (2014).

4.3.4 Hellinger PCA (H-PCA) - PCA-based Dimensionality Reduction

Lebret and Collobert (2014) propose a method for creating word representation starting from the word counts in co-occurrence matrix. However, their context definition is quite different from the contexts proposed before: they work with an asymmetric context of 1 word. In other words, they consider as context only the word that comes immediately after the target word.

This restricted definition of context is used to fill in the positions of a co-occurrence matrix that contains all the words in the training corpus whose frequency is above a specified threshold. In their experiments the threshold is set at 100, and results in a vocabulary V_W of 178,080 words. In contrast to the `word2vec` models, the context is modeled using only a subset of the dictionary words, V_C, made of the 10,000 most frequent words.

The word co-occurrence matrix $X \in \mathbb{R}^{|V_W| \times |V_C|}$ is approximated as the matrix $\hat{X} \in \mathbb{R}^{|V_W| \times d}$, where $d \ll |V_C|$ is the dimensionality of the word representations, as chosen by the user. $\hat{X}$ is the result of a **principal component analysis (PCA)** of the initial matrix X.

The idea behind principal component analysis is that the most useful characteristic of the data is the one that accounts for the most **variation**. For the data points illustrated in Figure 4.7, the direction where the data exhibits most variation is the one corresponding to the red arrow. A second direction of variation that captures most of the remaining variation in the data is the one corresponding to the green arrow. These main directions of variation are called the **principal components** of the data, and they are orthogonal to each other.

For n dimensional data there will be n principal components. In general, however, the data will be re-represented using only the top d principal components, where $d \ll n$. The new representation of the data is thus much more compact, but captures a great deal of the information that was initially available in the data.

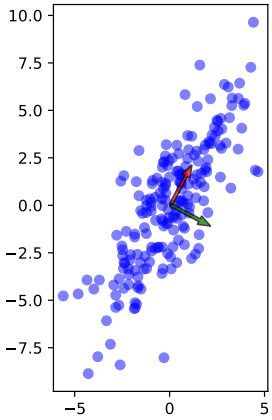


Fig. 4.7 In illustration of principal component analysis (PCA). The red arrow shows the direction where the data exhibits the most variation. The green arrow points the second direction of variation for the given data points.

Just like in the SVD case, the goal of PCA is to find an $\hat{X}$ such that the difference to X is minimal. However, Lebret and Collobert (2014) choose to use the Hellinger distance instead of the Euclidean distance to quantify the error between X and its approximation $\hat{X}$. The Hellinger distance of two discrete probability distributions $P = (p_1, \ldots, p_k)$ and $Q = (q_1, \ldots, q_k)$ is defined by Eq. 4.21

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2} \tag{4.21}$$

It follows that $\sqrt{X}$ should be used instead of X to perform PCA. In the PCA eigenvalue decomposition formulation[13], a symmetric positive semi-definite matrix $R = \sqrt{X^\top}\sqrt{X}$, $R \in \mathbb{R}^{|V_W| \times |V_W|}$ is defined. R can be rewritten as $R = ADA^\top$, where D is the diagonal matrix of eigenvalues of R and A is the orthogonal matrix of eigenvectors of R. It follows that the matrix containing the word embeddings, $\hat{X}$, can be obtained by projecting the transformed word distributions from $\sqrt{X}$, on the first d principal components of the matrix R:

$$\hat{X} = \sqrt{X}A_d \tag{4.22}$$

where $A_d \in \mathbb{R}^{|V_W| \times d}$ are the first eigenvectors of the matrix R (Lebret, 2016).

Lebret and Collobert (2014) show that the embeddings obtained via `H-PCA` are useful as features for downstream semantic task such as named entity recognition and sentiment analysis, despite being trained only on one word of context. In a subsequent set of experiments Lebret and Collobert (2015a) show that count methods like the `H-PCA` can produce competitive word embeddings also for solving word analogy problems, provided the context size is increased to five or ten words of context, as in a typical `word2vec` setup. The software package for training `H-PCA` word representations is publicly available[14].

4.3.5 Global Vectors (GloVe) - Combining Global Matrix Factorization and Local Context Window methods

Creating word representations using the SVD/PCA methods presented in the previous sections have significant advantages. The co-occurrence information can be gathered efficiently in one single step, and the co-occurrence counts can then be re-weighted and transformed into suitable word representations. However, if the size of the matrix increases, the matrix factorization operation becomes very expensive, and the factorization has to be redone for every new word that needs a representation.

The neural network-based models - `skip-gram`, `CBOW` (Mikolov et al., 2013a) and the Collobert and Weston (2008) models - learn word representations using a local context window. While they sidestep the need for a costly matrix factorization operation, such models require going through every word window in the training data. The word co-occurrence information is spread out across the corpus, and amount of time needed to train the word vectors grows with the size of the training corpus.

Pennington et al. (2014) proposed a model that combines the strengths of these two approaches. The *Gobal Vectors* (`GloVe`) model trains on the global word-word co-occurrence matrix, just like the SVD/PCA models, thus making efficient use of the statistics. However, the optimization procedure applies to one cell of this matrix at a time, and does not require factorizing a high-dimensional matrix.

[13] PCA can be performed using multiple approaches: using eigenvalue decomposition, using SVD, or using stochastic low-rank approximations. For the other formulations see details in Lebret (2016).

[14] `H-PCA` - `https://github.com/rlebret/hpca`.

Just like in the `skip-gram` model, a word w is represented in `GloVe` using two separate vectors, one as a context representation ($\mathbf{u}_w$) and one as a target representation ($\mathbf{v}_w$). All the word representations are again kept in two matrices $W_{ctx}, W_{target} \in \mathbb{R}^{d \times V_C}$. The vocabulary V_C is made of the most frequent $|V_C|$ words in the training corpus. The context size is chosen by the user, and can be either symmetric or asymmetric. The co-occurrence counts are computed once for all the words in the vocabulary. The matrices W_{ctx} and W_{target} are initialized to small values from a uniform distribution, and the training proceeds to minimize the cost function in Eq. 4.23.

$$J(\Theta) = \frac{1}{2} \sum_{i,j=1}^{V_C} f(P_{ij})(\mathbf{u}_i^\top \cdot \mathbf{v}_j - \log P_{ij})^2 \tag{4.23}$$

In effect, the training goes over all pairs of words that actually co-occur (it trains only on the non-zero entries of the co-occurrence matrix). For each word pair it minimizes the squared distance between the dot product of the word vectors and the log of the co-occurrence probability of the two words. The function reaches its minimum when the dot product $\mathbf{u}_i^\top \mathbf{v}_j$ is equal to the logarithm of the co-occurrence probability of the two words. The function $f(P_{ij})$ is a weighting function, whose form is given by Eq. 4.24, and whose main purpose is to down-weight the very frequent counts. This step is needed because very frequent words do not inform the model as much as less frequent words do. E.g. the fact that a word like *cat* co-occurs with *the* and *in* is less informative than the fact that *cat* co-occurs with a comparatively less frequent word like *running*. $\alpha = 3/4$ was found to be a beneficial setting for smoothing the counts, just like in the case of the `word2vec` model (Mikolov et al., 2013a).

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if} x < x_{max} \\ 1 & \text{otherwise.} \end{cases} \tag{4.24}$$

`GloVe` differs from the `word2vec` models in that instead of updating each co-occurrence count independently, it captures all the co-occurrences once, at the beginning of the training procedure, and then proceeds to perform local updates on the global counts. Updates are performed once for all occurrences of a pair of words, thus making much better use of the statistics of the corpora. The cost function in Eq. 4.23 is optimized using AdaGrad (Duchi et al., 2011). At the end of the training procedure the matrices W_{ctx} and W_{target} are summed up. The vectorial representation of the word w is the component-wise sum of its context and target vectors, $\mathbf{u}_w + \mathbf{v}_w$.

The vectors produced by the `GloVe` model exhibit the same type of meaningful substructure that is beneficial for solving analogy questions (e.g. the *king-man+woman* $\approx$ *woman* analogy). Evaluation on different datasets, both in terms of word analogy and word similarity showed that the `GloVe` vectors perform better across the board than vectors obtained using any of the previously proposed methods, and that the `GloVe` vectors are particularly suited for semantic tasks (Pennington et al., 2014).

The GloVe[15] word representations will be used extensively in the machine learning experiments presented in Chapter 5, 6 and 7. Their effectiveness on semantic tasks makes them a particularly good candidate for capturing the compound-internal semantics.

[15]The GloVe software package for training new word representations, as well as pre-trained word embeddings for English can be downloaded from the project's website, https://nlp.stanford.edu/projects/glove/.

Chapter 5

The Mathematics of Composition

Generalstaatsverordnetenversammlungen.
Altertumswissenschaften.
Kinderbewahrungsanstalten.
Unabhängigkeitserklärungen.
Wiederherstellungsbestrebungen.
Waffenstillstandsverhandlungen.

[...] when one of these grand mountain ranges goes stretching across the printed page, it adorns and ennobles that literary landscape - but at the same time it is a great distress to the new student, for it blocks up his way; he cannot crawl under it, or climb over it, or tunnel through it. So he resorts to the dictionary for help, but there is no help there. The dictionary must draw the line somewhere - so it leaves this sort of words out. And it is right, because these long things are hardly legitimate words, but are rather combinations of words [...]

- Mark Twain, *The Awful German Language*

5.1 A Case for Composition

Vector space models of language like the ones presented in Chapter 4 create good representations for the **simplex words** of a language. Each word from a fixed-sized dictionary is assigned an n-dimensional, real-valued vector which encodes a mixture of its syntactic and semantic characteristics. They can provide vectors for representing individual words like *computer*, *software*, *programmer*, *developer*, etc.

But how should a compound like *software developer* be represented? From a semantic perspective, a *software developer* is similar to a *programmer*, a *developer* and a *coder*. The ideal representation is one that makes possible the comparison between compounds and simplex words with similar meaning.

Another argument for using compatible representations for simplex words and compounds is that many English compounds exhibit regular differences in spelling. The

same 'unit of meaning' can be written as two separate words (*dress code*), contiguously (*dresscode*) or with a dash (*dress-code*). The spelling of the word, should, however, have minimal impact on its semantic representation. The idea introduced in this chapter is that in the context of vector space models of language, a compound should be represented by an n-dimensional, real-valued vector, just like a simplex word.

A natural choice for building such n-dimensional compound representations is to use the exact same mechanisms that are used for creating representations for simplex words: compile a list with all the compounds to be represented, append it to the dictionary of the language model containing only simplex words and train a new, compound-aware language model. This approach was proposed by Mikolov et al. (2013b), and resulted in good representations being trained both for individual words and for phrases, including some noun-noun compounds. The problem, however, is that adding word combinations quickly expands the size of the dictionary. A large vocabulary increases the time, the computational resources and amount of text needed for training language models.

Moreover, due to the productivity of the compounding process, such models will have data sparsity problems. For example, Baroni et al. (2002) analyzed the 28 million words German APA news corpus and discovered that compounds account for 47% of the word types but only 7% of the overall token count, with 83% of compounds having a corpus frequency of 5 or lower. Language models are at a loss when trying to produce meaningful representations for such infrequent words.

A different way of tackling the problem of creating vector representations for compounds is to find ways to combine the representations of the constituent words. The meaning of a compound often involves combining some aspect or aspects of the meanings of its constituents. From the computational perspective, if a compound is infrequent then it would be difficult to directly create a good representation for it. However, there is a good chance that the compound's constituents are frequent enough to have meaningful vector representations. The idea introduced in this chapter is to use the representations of the compound's constituents to build a meaningful representation for the compound itself.

This alternative way of building compound representations involves finding methods for **composing** the representations of the compound's constituents. Example of possible **composition functions** are addition and multiplication: in both cases, if $\mathbf{u}$ is the vector representation of *software* and $\mathbf{v}$ is the vector representation of *developer*, the meaning of *software developer* can be represented as the vector $\mathbf{p}$, obtained using component-wise vector addition or multiplication from $\mathbf{u}$ and $\mathbf{v}$, as illustrated in Figure 5.1.

This representation technique assumes that compounds are **compositional**, i.e. that the meaning of a compound can be derived from the meaning of its constituent words. This is an obvious simplification of the compounding process as it takes place in natural language. The discussion in Chapter 2 provided numerous examples of compound-internal semantic relation which connect the two constituents of a compound. In contrast, composition models like addition or multiplication take a much simplified view. The composition function operates only on the constituents and does not

$$\mathbf{u}_{\text{software}} = [\ 0.5\ \ 0.3\ \ 0.1\ \ 0.02\ \ 0.08\ ]^\top$$
$$\mathbf{v}_{\text{developer}} = [\ 0.01\ \ 0.4\ \ 0.3\ \ 0.09\ \ 0.2\ ]^\top$$

$$\mathbf{p}_{\text{software developer}} = f_{\text{addition}}(\mathbf{u}_{\text{software}}, \mathbf{v}_{\text{developer}}) = \mathbf{u}_{\text{software}} + \mathbf{v}_{\text{developer}} = [\ 0.51\ \ 0.7\ \ 0.4\ \ 0.11\ \ 0.28\ ]^\top$$
$$\mathbf{p}_{\text{software developer}} = f_{\text{multiplication}}(\mathbf{u}_{\text{software}}, \mathbf{v}_{\text{developer}}) = \mathbf{u}_{\text{software}} \odot \mathbf{v}_{\text{developer}} = [\ 0.005\ \ 0.12\ \ 0.03\ \ 0.0018\ \ 0.016\ ]^\top$$

Fig. 5.1 Addition and multiplication as composition functions for vector representations.

model the implicit compound-internal semantic relation. Despite these shortcomings, **composition models** are a tractable way to create meaningful representations for all the compounds that have been or will be coined by the speakers of a language. Libben (2006) gives an illustrative example of the potential generative capabilities of the compounding process: if the dictionary of a speaker contains $N = 100$ simplex words, with no restrictions on how the words should be combined, and taking into account differences in semantics that stem from the differences in word order (e.g. *houseboat* is different from *boathouse*), then the speaker is able to create $\frac{N!}{(N-2)!} = (N-1) * N = 9900$ new compounds from the 100 initial simplex words. Even if in reality not all possible word combinations are used as compounds, the compositional process seems like the most promising approach for dealing with such a rapid increase in vocabulary size.

It would then be convenient to equip the vector space model with a composition function able to construct a **composed representation** for *software developer* from the representations of *software* and *developer*. The composed representation should ideally be indistinguishable from its **observed representation**, i.e. the representation learned directly by the language model if the compound were part of the dictionary.

There are, however, compounds for which the compositional approach is unsuited. A language will also have 'icebergs', compounds whose meaning started up compositionally but 'froze' as the language evolved. A good example of this kind is *Eisbergsalat* 'iceberg salad', a particular type of salad whose name stems apparently from it being shipped by train across America inside large ice blocks at the beginning of the 20$^{\text{th}}$ century[1]. From a meaning perspective such compounds are closer to a simplex word than to a compound: the English speaker of the 21$^{\text{st}}$ century will most likely be unaware of the original meaning, but will easily identify the particular head of salad this compound refers to. Such **non-compositional compounds** will be studied in more detail in Chapter 6.

This chapter introduces, in Section 5.2, several composition models that have been proposed in the literature. Section 5.3 describes the process of creating evaluation datasets for German. Section 5.4 presents the evaluation criteria and the results for German and introduces two new composition models. Section 5.5 presents the results of the composition models on the English compounds composition dataset. The German and English results are compared to those in other compositionality studies in Section 5.6, and a summary of the main results is presented in Section 5.7.

[1] `https://de.wikipedia.org/wiki/Eisbergsalat`, last accessed 19 June 2018.

5.2 Introducing Composition Models

A common view of natural language regards it as being inherently compositional. Words are combined to obtain phrases, which in turn can be combined to create sentences. The composition continues to the paragraph, section and document levels. It is this defining trait of human language, its **compositionality**, that enables humans to produce and to understand the potentially infinite number of utterances of a human language.

Gottlob Frege (1848-1925) is credited with phrasing this intuition in the form of a principle, known as the Principle of Compositionality: *"The meaning of the whole is a function of the meaning of the parts and their mode of combination."* (Dowty et al., 1981:8)

The adoption of the distributional vectors (described in Chapter 4) as a proxy for the meaning of individual words - in other words, having a *"meaning of the parts"* - encouraged researchers to focus their attention on finding **composition models** which could act as the *"mode of combination"*.

For compounds, the idea of looking for a *"mode of combination"* translates to finding a composition function f which takes as input two n-dimensional distributional representations, one for each of the two constituents. The constituent representations, $\mathbf{u}^{corpus}, \mathbf{v}^{corpus} \in \mathbb{R}^n$, are learned using a language model and a support corpus. The result of applying f to the constituent representations is another n-dimensional representation for the compound, $\mathbf{p}^{composed} \in \mathbb{R}^n$. The idea can be written more formally as in Eq. 5.1:

$$\mathbf{p}^{composed} = f(\mathbf{u}^{corpus}, \mathbf{v}^{corpus}) \tag{5.1}$$

An additional vector to be considered is $\mathbf{p}^{corpus} \in \mathbb{R}^n$, the observed representation of the compound that was learned from the support corpus. $\mathbf{p}^{corpus}$ can be thought of as a 'gold standard' for the composed representation $\mathbf{p}^{composed}$. Ideally, the application of the composition function f to the constituent vectors $\mathbf{u}^{corpus}$ and $\mathbf{v}^{corpus}$ should result in a composed representation, $\mathbf{p}^{composed}$, that encodes the syntactic and semantic characteristics captured by the corpus-induced representation of that compound, $\mathbf{p}^{corpus}$.

Modeling compound composition can be seen as one of the more simple composition types: it requires only modeling the modification of the head by the modifier. Even in the case of multi-token compounds, a representation can be obtained by recursively combining the immediate constituents using the same composition function. The only prerequisite is that the internal syntactic structure of the compound has to be available - Henrich and Hinrichs (2010) present several ways to automatically derive the necessary splitting information.

The remainder of this section will introduce several composition functions proposed in the literature, which will be evaluated on the task of creating representations for German and English compounds in Sections 5.4 and 5.5, respectively.

5.2.1 Composition Models in the Literature

Some of the first comprehensive studies of composition using distributional models of semantics were proposed by Mitchell and Lapata (2008, 2010), who focus on the family of composition functions described by Eq. 5.2:

$$\mathbf{p} = f(\mathbf{u}, \mathbf{v}, R, K) \tag{5.2}$$

Mitchell and Lapata (2008, 2010) take the meaning of a two-component phrase $\mathbf{p}$ to be the result of applying a composition function f, which has yet to be defined, to the components of the phrase represented by the vectors $\mathbf{u}$ and $\mathbf{v}$. In its most general form, the composition function must also consider the impact of the syntactic relation R between $\mathbf{u}$ and $\mathbf{v}$, as well as that of the background knowledge K.

Mitchell and Lapata (2008, 2010) enlist the WordNet (Fellbaum, 1998) as a possible source for the background knowledge K, but focus on studying the mechanism of composition without the influence of background knowledge.

In the case of noun compound composition a useful window into the background knowledge would be the types of semantic relations that the two components are most likely to enter, of the type described in Chapter 2. The lack of extensive datasets annotated with such information makes it impractical to integrate this type of information in the composition models presented in this thesis, leading to the background knowledge component K of Eq. 5.2 being discarded. However, as such annotations become available, they should provide valuable input signals for composition models.

Compounds are nominal phrases, so the syntactic relation R is in this case fixed. The order in which the constituents are considered is also fixed: the vector $\mathbf{u}$ always refers to the first constituent - the modifier - and the vector $\mathbf{v}$ always refers to the second constituent - the head. It follows that the syntactic relation R in Eq. 5.2 can also be discarded, leading to the more simple formula in Eq. 5.3:

$$\mathbf{p} = f(\mathbf{u}, \mathbf{v}) \tag{5.3}$$

Another restriction imposed on the function f is that $\mathbf{p}$, the composed representation, should be of the same dimensionality as the input vectors $\mathbf{u}$ and $\mathbf{v}$. This ensures that the composed representations of compounds can be compared to the representations of simplex words. The results presented in this chapter focus exclusively on the class of composition functions that takes as inputs two vectors of dimensionality n and composes them into another vector of dimensionality n.

5.2.2 Linear Composition Functions

A first class of composition functions introduced in Mitchell and Lapata (2008, 2010) are **linear composition functions**. The general form of a linear composition function is given in Eq. 5.4: the composed representation $\mathbf{p}$ is the result of adding the vectors $\mathbf{u}$

and $\mathbf{v}$, where the contribution of each vector is controlled by the matrices $\mathbf{A}$ and $\mathbf{B}$. Several composition functions can be derived from Eq. 5.4.

$$\mathbf{p} = \mathbf{A}\mathbf{u} + \mathbf{B}\mathbf{v} \tag{5.4}$$

Head Equating $\mathbf{A}$ with the zero matrix $\mathbf{0}$ results in a composition function where the head constituent contributes all its information while the modifier brings no information (Eq. 5.5). This type of composition will henceforth be referred to as `head` composition.

$$\mathbf{p} = \mathbf{v} \tag{5.5}$$

Modifier Equating $\mathbf{B}$ with $\mathbf{0}$ results in a composed representation that is determined in its entirety by the representation of the modifier (Eq. 5.6). This model will henceforth be called `modifier` composition.

$$\mathbf{p} = \mathbf{u} \tag{5.6}$$

The two models in Eq. 5.5 and 5.6 admittedly discard a substantial amount of information, since they only consider one of the two constituents of the compound. This same property, however, makes them interesting **baselines** that other models can be compared to. Models that underperform the `head` or the `modifier` composition can hardly be considered good composition models.

Addition Replacing $\mathbf{A}$ and $\mathbf{B}$ with the identity matrix $\mathbf{I}$, results in the formula for `component-wise vector addition` (Eq. 5.7). The vector addition function has the drawback that both vectors contribute equally to the composed representation, making the composition process symmetric. A symmetric composition function is unable to take into account the differences in meaning that stem from differences in word order: the compounds *car factory* and *factory car* will be given the same vector representation, even though one is a type of *factory* and the other a type of *car*.

$$\mathbf{p} = \mathbf{u} + \mathbf{v}; \quad \mathbf{p}_i = \mathbf{u}_i + \mathbf{v}_i, i \in 1, \ldots, n \tag{5.7}$$

Multiplication Another symmetric operation is `multiplication`. In this case each component of the vector $\mathbf{u}$ is multiplied with the corresponding component of the vector $\mathbf{v}$. The expression of this composition function is given in Eq. 5.8.

$$\mathbf{p} = \mathbf{u} \odot \mathbf{v}; \quad \mathbf{p}_i = \mathbf{u}_i \cdot \mathbf{v}_i, i \in 1, \ldots, n \tag{5.8}$$

Weighted Addition One way to introduce asymmetry in the addition function is to make $\mathbf{A}$ and $\mathbf{B}$ 1×1 matrices and then consider only their scalar counterpart. The composition function where two scalars $\lambda, \beta \in \mathbb{R}$ adjust the influence of $\mathbf{u}$ and $\mathbf{v}$ is

called `weighted addition` (Eq. 5.9). `weighted addition` is the first example of a **parametric** composition function: the optimal values for λ and β can be learned using a set of training examples. All previous functions - `head` composition, `modifier` composition, `addition` and `multiplication` - are **non-parametric**, i.e. they cannot be improved via training. The next sections will provide evidence to the fact that even if parametric models are usually more complex, they generally perform better as composition functions than the non-parametric models.

$$\mathbf{p} = \lambda\mathbf{u} + \beta\mathbf{v} \tag{5.9}$$

The linear composition functions presented so far were all introduced in Mitchell and Lapata (2008, 2010). Other examples of linear composition functions in the literature are the `lexical function` model and the `full additive` model, which will be introduced next.

Lexical Function was introduced by Baroni and Zamparelli (2010)[2] as a composition function for adjective-noun phrases. In a phrase like *red car*, the adjective *red* modifies the noun *car* to produce another *car* which has the attribute 'color' set to *red*. Baroni and Zamparelli (2010) modeled this type of interaction by representing the noun as a vector $\mathbf{v} \in \mathbb{R}^n$ and the adjective as a matrix $\mathbf{A_u} \in \mathbb{R}^{n \times n}$. The full form of the composition function is given in Eq. 5.10.

$$\mathbf{p} = \mathbf{A_u}\mathbf{v} \tag{5.10}$$

`lexical function` is another example of a parametric function. The noun vectors are considered to be given. The model needs to estimate an $n \times n$ matrix for every adjective. For commonly used values of $n = 300$ dimensions, one matrix will have $300 \times 300 = 90,000$ parameters. Estimating the parameters for a small dictionary of 10,000 adjectives would require the model to estimate 900,000,000 parameters. This increase in the number of parameters makes the model potentially prone to data sparseness issues.

Originally applied by Baroni and Zamparelli (2010) to adjective-noun phrases, the `lexical function` model can be easily applied to other two-component phrases. In the case of compound composition, the vectors of the head constituent is considered to be given and the model has to estimate a matrix for each modifier. The same matrix is reused for different compounds with the same modifier (e.g. a single matrix will be used to model *water*, the common modifier of *water bed*, *water bird* and *water pipe*).

`lexical function` is again a special case of the general linear composition function from Eq. 5.4, where the matrix $\mathbf{A}$ is set to $\mathbf{0}$. The matrix $\mathbf{B}$ is kept and renamed to $\mathbf{A_u}$. A different matrix $\mathbf{A_u} \in \mathbb{R}^{n \times n}$ is used for each of the individual modifiers in the modifier dictionary, $\mathcal{D}_M$. Each $\mathbf{A_u}$ matrix is initialized to $\mathbf{I} + \mathcal{N}(0, 1e-4)$ - that is, the identity

[2]The initial name used by Baroni and Zamparelli (2010) for this composition function was `adjective-specific linear map` (*alm*). The name was changed by the authors to `lexical function` in a later paper, Dinu et al. (2013b).

matrix plus some noise from a normal distribution of mean 0 and standard deviation $1e-4$. This initialization ensures that at the beginning of the training process the result of the composition is approximately equal to the vector of the second component, $\mathbf{v}$. The $\mathbf{A_u}$ matrices are updated during training, such that the composition better matches the vector corresponding to the adjective-noun combination. If the training data does not include any examples of a particular modifier $\mathbf{r}$, its matrix $\mathbf{A_r}$ will not be updated and will remain close to the identity matrix it was initialized with. In such cases the composed representation is the same as the `head` representation.

Of particular importance is the fact that the matrices $\mathbf{A_u}$ are estimated from the training data and have no connection with the modifier vector, $\mathbf{u}$. The modification matrix is learned from scratch, without trying to leverage any of the information already learned by the language model in the form of the vector $\mathbf{u}$.

An extension of the `lexical function` composition model has been proposed by Bride et al. (2015), where the composition uses a third order tensor[3], $\mathcal{A} \in \mathbb{R}^{n \times n \times n}$. In the case of adjective-noun composition, the tensor $\mathcal{A}$ is first (left) multiplied using the adjective vector and then using the noun vector. The resulting n-dimensional vector is the representation of the phrase. The advantage of the generalized lexical function is that it trains a single tensor for all the adjective-noun phrases, not an individual modification matrix for every adjective like in the case of the `lexical function` model. The experiments in Bride et al. (2015) show that the performance of the extended function is on par with the one of `lexical function`. That is why the experiments in this chapter will focus on the evaluation of the originally proposed `lexical function` composition model.

Full Additive Another parametric composition model, `full additive`, was introduced independently by Guevara (2010, 2011) (who calls it a partial least squares regression composition model) and by Zanzotto et al. (2010) (in their work this model is called the estimated additive model). `Full additive` is a direct interpretation of the general linear composition function from Eq. 5.4: the vectors $\mathbf{u}$ and $\mathbf{v} \in \mathbb{R}^n$ corresponding to the two constituents are multiplied via two square matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$. $\mathbf{A}$ and $\mathbf{B}$ are the same for every $\mathbf{u}$ and $\mathbf{v}$. This means that during training the model has to estimate only the parameters in two $n \times n$ matrices. The `full additive` model has a constant number of parameters with respect to the size of the vocabulary.

5.2.3 Non-linear Composition Functions

Linear composition models produce composed representations that are linear combinations of the initial constituent representations. An alternative class of composition functions that were proposed in the literature are the **non-linear composition func-**

[3]The idea of using a third order tensor as a composition function has been also proposed by Mitchell and Lapata (2008, 2010). However, the earlier work does not test such models, since they require large-scale parameter estimation, impossible without a substantial amount of training data.

tions. These functions use non-linear transformations to expand the representation capabilities of composition functions.

Dilation Mitchell and Lapata (2010) introduce `dilation`, a composition function that is quadratic with respect to the modifier vector $\mathbf{u}$. The intuition behind `dilation` is that one can decompose the head vector $\mathbf{v}$ in two vectors, one parallel and one orthogonal to $\mathbf{u}$. The parallel component is then stretched by a factor λ, while the orthogonal component remains unchanged. This operation results in a variant of the head vector $\mathbf{v}$ that has been modified to emphasize the contribution of the modifier vector $\mathbf{u}$. The dilation model targets specifically the idea that the composition operation might 'preselect' some aspects of the meanings of the constituents. An example given by Mitchell and Lapata (2010) are combinations of the adjective *good*, which 'preselects' different aspects of the words *neighbor* and *lawyer* when combined with them: a *good lawyer* and a *good neighbor* differ in the aspects they select in judging the 'goodness' of the modified noun. The mathematical formulation of `dilation` is given by Eq. 5.11.

$$\mathbf{p} = (\mathbf{u} \cdot \mathbf{u})\mathbf{v} + (\lambda - 1)(\mathbf{u} \cdot \mathbf{v})\mathbf{u} \tag{5.11}$$

Matrix Socher et al. (2010) introduce a non-linear composition model where the constituent vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ are first concatenated into a vector $[\mathbf{u}; \mathbf{v}] \in \mathbb{R}^{2n}$ and then multiplied via a matrix $\mathbf{W} \in \mathbb{R}^{n \times 2n}$. The result of the multiplication is an n-dimensional vector, which is passed as a final step through a non-linear function g - the element-wise hyperbolic tangent tanh in the case of Socher et al. (2010). The parameter matrix $\mathbf{W}$ has to be estimated during the training process and serves as a combination matrix for all the possible input vectors $\mathbf{u}$ and $\mathbf{v}$. Since this composition function is implemented via a neural network, a bias term $\mathbf{b} \in \mathbb{R}^n$ is added after the multiplication of the matrix $\mathbf{W}$ with the vector concatenation $[\mathbf{u}; \mathbf{v}]$. The complete form of this composition function, henceforth the `matrix` composition function, is given in Eq. 5.12.

$$p = g(\mathbf{W}[\mathbf{u}; \mathbf{v}] + \mathbf{b}) \tag{5.12}$$

Mathematically, $\mathbf{W}$ performs a **linear transformation** (Strang, 2016:401) of the concatenated inputs vector, $[\mathbf{u}, \mathbf{v}]$. A transformation $\mathbf{T}$ is linear if it preserves addition and scalar multiplication: (i) $\mathbf{T}(\mathbf{u}+\mathbf{v}) = \mathbf{T}(\mathbf{u}) + [\mathbf{T}](\mathbf{v})$ and (ii) $\mathbf{T}(c\mathbf{v}) = c\mathbf{T}(\mathbf{v})$, where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and $c \in \mathbb{R}$. Examples of linear transformations are rotation by an angle around the origin, reflection about the origin, scaling with respect to the origin etc. However, adding the bias vector $\mathbf{b}$ to the result of the linear transformation shifts (or translates) the entire result, making $\mathbf{W}[\mathbf{u}; \mathbf{v}] + \mathbf{b}$ an **affine transformation**(Strang, 2016:402), which not preserve the origin of the coordinate system. Finally, applying a nonlinearity g like tanh makes the whole `matrix` composition a **non-linear transformation** of the input vectors.

It's worth noting that there is a close relation between the `full additive` and the `matrix` composition models. The `full additive` model can be rewritten as in

Eq. 5.13. This equation shows that the `matrix` model is, in effect, the `full additive` model with an added bias term and a nonlinearity.

$$p = \mathbf{Au} + \mathbf{Bv} = [\mathbf{A}; \mathbf{B}][\mathbf{u}; \mathbf{v}] = \mathbf{W}[\mathbf{u}; \mathbf{v}] \tag{5.13}$$

Full Lexical Socher et al. (2012) introduce a more general composition model that subsumes the `matrix` composition function, the `lexical function` as well as the `full additive` model. Each constituent is modeled via two components: an n-dimensional vector which captures the meaning of the constituent and an $n \times n$ modification matrix which captures how one constituent modifies the meaning of the other constituent it combines with.

During the composition process the vectors $\mathbf{u}$ and $\mathbf{v}$ of the two constituents are multiplied by the modifier matrix corresponding to the other constituent, $\mathbf{A_v}$ and $\mathbf{A_u} \in \mathbb{R}^{n \times n}$. The resulting vectors $\mathbf{A_v u}$ and $\mathbf{A_u v}$ are concatenated and multiplied by a matrix $\mathbf{W} \in \mathbb{R}^{n \times 2n}$. The result is passed through a non-linearity g, just like in the case of the `matrix` model.

Socher et al. (2012) call the model MV-RNN, but Dinu et al. (2013b) name it the `full lexical` composition function, a name that will be used for the rest of the discussion in this chapter. The full form of the `full lexical` model, including the bias terms as in the case of the `matrix` model, is given in Eq. 5.14.

$$p = g(\mathbf{W}[\mathbf{Vu}; \mathbf{Uv}] + \mathbf{b}) \tag{5.14}$$

Each of the components of the `fulllex` model aims at capturing different aspects of the composition. The individual word matrices $\mathbf{A_u}, \mathbf{A_v}$ capture the composition effects that are particular for every word. The word matrices do not depend on the position of the word, i.e. if the word is the first or the second one in the phrase. Only one matrix is trained for each word from the dictionary containing all unique modifiers and heads, $\mathcal{D}_{MH}$. The global matrix $\mathbf{W}$ captures more general, word-independent aspects of the composition.

The `matrix` model is a special case of the `full lexical` model where the word-specific matrices $\mathbf{U}, \mathbf{V}$ are all replaced by identity matrices. This is, in fact, also the way the `full lexical` model is initialized: each individual word matrix is set to $\mathbf{I} + \mathcal{N}(0, 1e-4)$. In effect, the `full lexical` model starts off from the same point as the `matrix` model. Its advantage, however, is the possibility to fine-tune the composition process for each word individually, instead of relying only on the general composition performed using the matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$.

`Full lexical` is similar to `lexical function` in that at training time it estimates a modification matrix for every word in the dictionary. The number of parameters is therefore dependent on the size of the dictionary. This again makes data sparseness an issue that needs to be taken into account when training the model.

Table 5.1 gives examples of composing two vectors using the ten composition functions presented in this section. The parametric composition functions have their

$$\begin{matrix}\mathbf{u}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad\longrightarrow\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}$$

(a) `head`: $\mathbf{p} = \mathbf{v}$

$$\begin{matrix}\mathbf{u}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad\longrightarrow\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}$$

(b) `modifier`: $\mathbf{p} = \mathbf{u}$

$$\begin{matrix}\mathbf{u}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}\quad+\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}4\\5\\-1\\7\\1\end{pmatrix}\end{matrix}$$

(c) `addition`: $\mathbf{p} = \mathbf{u} + \mathbf{v}$

$$\begin{matrix}\mathbf{u}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}\quad\odot\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}3\\6\\-6\\6\\0\end{pmatrix}\end{matrix}$$

(d) `multiplication`: $\mathbf{p} = \mathbf{u} \odot \mathbf{v}$

$$\begin{matrix}\lambda\\0.4\end{matrix}\quad\begin{matrix}\mathbf{u}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}\quad+\quad\begin{matrix}\beta\\0.6\end{matrix}\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}2.2\\2.4\\-1.0\\4\\0.6\end{pmatrix}\end{matrix}$$

(e) `weighted addition`: $\mathbf{p} = \lambda\mathbf{u} + \beta\mathbf{v}$

$$\begin{matrix}\mathbf{A_u}\\ \begin{pmatrix}1.0&0.5&0.8&0.4&0.3\\-1.0&0.7&0.3&1.2&0.0\\0.3&0.2&-0.2&0.0&1.1\\0.7&0.0&-1.0&0.2&0.4\\0.2&-0.8&0.3&0.1&1.0\end{pmatrix}\end{matrix}\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}4.3\\4.7\\3.0\\6.7\\-0.3\end{pmatrix}\end{matrix}$$

(f) `lexical function`: $\mathbf{p} = \mathbf{A_u v}$

$$\begin{matrix}\mathbf{A}\\ \begin{pmatrix}0.1&0.1&0.0&0.5&0.0\\-0.2&0.3&-0.4&0.2&0.3\\0.4&0.1&-0.6&0.4&0.7\\0.2&0.5&-1.1&0.1&-0.4\\0.8&0.0&0.1&0.9&-1.2\end{pmatrix}\end{matrix}\quad\begin{matrix}\mathbf{u}\\ \begin{pmatrix}1\\3\\2\\1\\0\end{pmatrix}\end{matrix}\quad+\quad\begin{matrix}\mathbf{B}\\ \begin{pmatrix}0.4&0.4&0.2&0.5&0.1\\-1.3&0.1&0.4&-1.4&0.3\\0.6&0.3&0.0&0.1&0.6\\0.1&0.6&-0.1&-0.2&-0.5\\-0.3&0.0&-0.2&0.8&0.2\end{pmatrix}\end{matrix}\quad\begin{matrix}\mathbf{v}\\ \begin{pmatrix}3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}5.4\\-12.9\\3.5\\-0.3\\6.6\end{pmatrix}\end{matrix}$$

(g) `full additive`: $\mathbf{p} = \mathbf{Au} + \mathbf{Bv}$

$$\begin{matrix}\mathbf{W}\\ \begin{pmatrix}0.2&0.3&-0.1&0.4&0.3&0.7&-0.1&-0.2&0.8&0.0\\1.1&0.3&0.6&-0.2&0.4&0.4&0.0&0.1&-0.5&0.0\\0.5&0.1&-0.4&0.1&0.8&0.2&-0.1&-0.7&-0.2&1.0\\0.3&0.1&-1.1&0.3&0.9&0.7&-0.1&-0.4&-0.2&0.0\\0.0&-1.0&0.5&0.4&1.2&-0.1&-0.3&0.8&0.1&1.0\end{pmatrix}\end{matrix}\quad\begin{matrix}[\mathbf{u};\mathbf{v}]\\ \begin{pmatrix}1\\3\\2\\1\\0\\3\\2\\-3\\6\\1\end{pmatrix}\end{matrix}\quad+\quad\begin{matrix}\mathbf{b}\\ \begin{pmatrix}-1.0\\0.1\\-0.3\\0.2\\0.1\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}7.6\\1.0\\2.1\\0.8\\-3.2\end{pmatrix}\end{matrix}$$

(h) `matrix`: $g(\mathbf{W}[\mathbf{u};\mathbf{v}] + \mathbf{b})$, where g is the identity function, $f(x) = x$

$$\begin{matrix}\mathbf{W}\\ \begin{pmatrix}0.1&0.9&-1.1&0.2&-0.6&0.8&0.0&0.3&-0.4&-0.1\\-0.1&-0.2&0.2&-0.8&0.0&-0.4&0.3&0.7&-0.9&0.4\\0.2&0.0&-0.3&-0.1&1.0&0.3&-0.4&-0.1&0.2&1.0\\0.6&0.4&0.1&0.1&0.8&0.2&0.2&0.2&0.1&-0.3\\0.2&-0.8&0.0&0.5&-0.2&0.5&-0.7&1.0&1.0&1.2\end{pmatrix}\end{matrix}\quad\begin{matrix}[\mathbf{A_v u};\mathbf{A_u v}]\\ \begin{pmatrix}0.1\\-0.3\\1.2\\1.3\\-0.1\\0.5\\0.4\\-0.3\\0.6\\1.1\end{pmatrix}\end{matrix}\quad+\quad\begin{matrix}\mathbf{b}\\ \begin{pmatrix}-0.1\\1.0\\1.3\\-0.2\\0.5\end{pmatrix}\end{matrix}\quad=\quad\begin{matrix}\mathbf{p}\\ \begin{pmatrix}-1.40\\-0.14\\1.97\\-0.24\\3.02\end{pmatrix}\end{matrix}$$

(i) `full lexical`: $g(\mathbf{W}[\mathbf{A_v u};\mathbf{A_u v}] + \mathbf{b})$; g is the identity function; the terms $\mathbf{A_v u}$ and $\mathbf{A_u v}$ are obtained as shown in example 5.1f, `lexical function`

Table 5.1 Illustrative examples for the composition functions presented in Section 5.2. The parameters to be estimated in each case are highlighted in red. The input to all the composition functions are the 5-dimensional vectors $\mathbf{u}$ and $\mathbf{v}$. The result of the composition is another 5-dimensional vector $\mathbf{p}$.

parameters highlighted in red, to emphasize what needs to be estimated during the training process of each individual model.

Other composition models. Although the idea of creating compositional distributional representations is relatively new, several approaches to composition have been proposed in the literature. Comprehensively testing all the existing proposals is outside the scope of this work. The focus here will be on the ten composition models presented above, which are diverse enough to make this analysis a useful starting point. The next paragraphs sketch the main ideas behind other composition models, which will, however, not be implemented or further evaluated in this thesis.

Coecke et al. (2010) introduces a compositional framework where the compositional process is driven by syntax. The model can be used to define composition models for different constructions, i.e. a transitive verb with its subject and object, an adjective that modifies a noun, etc. The main intuition is that each word can participate in the composition process either as an argument or as a function. In the previous examples, the verb is the function that is applied to its arguments, the subject and the object. The verb is therefore represented by a third order tensor ($\in \mathbb{R}^{n \times n \times n}$), and its arguments by vectors $\in \mathbb{R}^n$. Similarly, the adjective is a function applied to the noun. The adjective is represented by a second order tensor (a matrix, $\in \mathbb{R}^{n \times n}$), while the noun is represented by a vector. The framework is evaluated in Grefenstette and Sadrzadeh (2011), and obtains similar results to Mitchell and Lapata (2008) on the dataset introduced in Mitchell and Lapata (2008). Clark (2012) makes the observation that Coecke et al. (2010)'s framework can be seen as an extension of Baroni and Zamparelli (2010) that covers 'all functional types, not just adjectives'. The `lexical function` composition introduced by Baroni and Zamparelli (2010) and presented in Eq. 5.10 is one of the models that will be tested throughout this chapter.

Socher et al. (2013b) also propose a recursive composition model, targeted at learning phrase and sentence representations. Their model, the Recursive Neural Tensor Network (RNTN), is an extension of Socher et al. (2010) `matrix` model, where the representations of the constituents are allowed to interact not only through a matrix, but also through a third order tensor. This allows for a larger set of interactions between the two constituent vectors. The new model obtains state-of-the-art results at predicting sentiment labels on a supervised dataset. It has to be noted, however, that this model works well only with short word representations (25 to 35 dimensions, as reported by Socher et al. (2013b)). Moreover, the individual word representations are learned via the supervised task, which ensures that the word and phrase representations are compatible. It is unclear if the model would work as well when using pre-trained word representations - the authors only report results where the word representations are learned together with the composition.

Learning dedicated word representations for phrase composition is also the main idea introduced in Lebret and Collobert (2015b). Here, the authors train word and phrase representations using a joint architecture. The idea is to take the probability distributions of target words (using 10,000 context words) and construct their low-

dimensional representation using an autoencoder. The phrase representation is a linear combination of the representations of its constituents - the constituent representations are added together to form the phrase representation. The joint training encourages the autoencoder to find an encoding function that also lends itself easily to addition-based composition.

Lebret and Collobert (2015b) show that the phrase representations are competitive when compared to GloVe or word2vec-induced representations. The word representations, however, fall short when compared to their GloVe and word2vec counterparts (trained on the same corpus with similar hyperparameters and vocabulary). The authors hypothesize that the lower performance of the word embeddings might be due to the limit of 10,000 context words, and argue that the word embeddings might be improved by using more context words.

Another frequent combination is that of composition models and parsing. Such models were proposed, among others, in Socher et al. (2010), Socher et al. (2013a), Legrand and Collobert (2015). In such settings the composition models are trained using the supervised signals provided by annotated treebanks. The proposals differ, however, when it comes to how they approach composition.

Socher et al. (2010) applies the composition function to word pairs. The word pair to be collapsed next is chosen either greedily, or using a sentence-optimized strategy. After composition a word pair is collapsed into a single node, represented using the composed representation. The pairwise composition followed by collapsing is applied until a representation for the whole sentence is obtained.

Legrand and Collobert (2015) use a matrix composition function, similar to the Socher et al. (2010)'s `matrix` model. However, since phrases have variable length, they train a separate composition matrix for every phrase length from 1 to 7. This means that a noun-noun combination, or an adjective-noun phrase, or an adverb-adjective combination will be combined using the composition function for phrases of length 2. This type of modeling might suffer from being too general - a single matrix might not be enough to model the modification patterns in these different phrase types.

Socher et al. (2013a)'s proposal lies at the opposite end of the spectrum: several composition functions are defined using discrete syntactic categories extracted from a probabilistic context-free grammar (PCFG). The composition function to be used depends on the syntactic categories of the phrase constituents. In their model Socher et al. (2013a) report training ∼900 composition matrices in a standard parsing setup using the Wall Street Journal corpus. Although theoretically appealing, the sparsity of the training data might make it difficult to train each composition matrix in an effective manner.

5.3 Constructing Datasets for Evaluating Composition Models

Several of the composition models presented in Section 5.2 were used in the literature for a diverse set of compositional phenomena: Baroni and Zamparelli (2010) investigated the composition of adjective-noun phrases, Kisselew et al. (2015) and Lazaridou et al. (2013) focused on compositional representations of morphologically complex words in German and English respectively, Dinu et al. (2013b) evaluated composition models on intransitive sentences, adjective-noun phrases and determiner phrases.

The next sections present an evaluation of the different composition models on the task of building composed representations for German and English compound words. Since most of these models are parametric, training examples are necessary for learning the optimal parameter values.

Previous studies like Guevara (2010, 2011), Baroni and Zamparelli (2010), Dinu et al. (2013b) use a common procedure for creating evaluation datasets for composition functions. First, a set of word pairs to be modeled is gathered - these could be adjective-noun, determiner-noun, noun-noun combinations, etc. A large support corpus is then used to construct distributional representations for the word pairs, as well as for the individual words in each pair. Continuous, distributed representations for individual words can be constructed using any of the approaches presented in Chapter 4.

Deriving representations for English word pairs requires an additional preprocessing step: all the corpus occurrences of the word pairs of interest must be linked with the underscore character '_'. This tricks the tokenizer into treating each pair as a single token, thus making it possible to record the co-occurrence statistics of the word pairs using the same distributional methods one would use for recording individual word statistics. The tokenization trick will be used in constructing representations for English noun compounds in Section 5.5.2, given that English compounds are generally written as separate words.

German compounds have a strategic advantage for this study: they are generally written as a contiguous word, irrespective of how many constituents they have. For example the English compound 'apple tree' translates into the German compound *Apfelbaum* with the head *Baum* 'tree' and the modifier *Apfel* 'apple'. Because the German compounds are written as a single word, the representations for the compound and its constituents can be directly learned, without the need for the tokenization trick described above.

5.3.1 German Compounds Dataset for Compositionality Tests

The German compounds used in the experiments are a subset of the 54759 compounds available in GermaNet 9.0[4]. The compounds in the list were automatically split into the **immediate constituents** (Henrich and Hinrichs, 2011): a two-part compound

[4] http://www.sfs.uni-tuebingen.de/GermaNet/documents/compounds/split_compounds_from_GermaNet9.0.txt

like *Apfelbaum* 'apple tree' would be split into *Apfel* 'apple' and *Baum* 'tree', while a compound made of multiple tokens, like *Basketballschuh* 'basketball shoe' would be split once, into its immediate constituents *Basketball* 'basketball' and *Schuh* 'shoe'. Of course, it is possible that the compound *Basketball* is part of the dataset, in which case another entry in the list would then be *Basketball* together with its constituents *Basket* 'basket' and *Ball* 'ball'. All the splitting decisions were manually post-corrected.

Each entry in the list is a triple of the form (compound, modifier, head). The entries in the list were filtered, keeping only those where all three words have a minimum frequency of 500 in the support corpus used to create the vector space representations (the support corpus for creating vector representations for German words will be introduced in Section 5.3.2). The reason for the filtering step is that a 'well-learned' representation, based on a sufficiently large number of contexts, should allow for a more accurate reconstruction than a representation based only on a few contexts.

The filtered dataset contains 34497 entries. This dataset was randomized and partitioned into `train`, `test` and `dev` splits according to the 70-20-10 rule, containing 24147, 6901 and 3449 entries respectively. The dataset contains 8580 unique modifiers and heads, and a dictionary of 41732 unique words. In some cases compounds are the constituents of larger compounds. 1345 compounds appear as the modifier or head of another compound.

Henrich and Hinrichs (2011) estimate that nominal compounds make up 95% of the GermaNet compounds. However, according to the GermaNet definition, nominal compounds are only required to have a noun as the head, i.e. as the second constituent. The part of speech restriction does not apply for the modifier (the first constituent of a noun), which means that nominal compounds from GermaNet can also have as modifiers adjectives, verb, adverbs, etc. Table 5.2 presents a detailed overview of the word class statistics[5] on the filtered dataset. The table includes the percentages of every modifier category (see last column).

Three subtypes of compounds can be identified in this dataset. The most frequent case, covering 93.59% of the data, is when the compound is identified in GermaNet as having only *one word class* for the modifier, for example having a noun as a modifier like in *Kaffeebohne* 'coffee bean'. The remaining of the data is split in two subcategories. One where the modifier can be either of *two word classes*, which encompasses 3.69% of the data, and one where the modifier has *no word class* - a subtype covering 2.72% of the data.

Modifiers with two word classes can exhibit different types of indeterminacy: noun/verb, adjective/noun and adjective/verb. An example is *Tanzschuh* 'dance shoe', where the modifier can be either the noun *Tanz* 'dance' or the verb *tanzen* 'to dance'. It is interesting to note that for some of these compounds it is impossible to establish what the correct word class of the modifier is - both are equally likely and the choice of word class does not lead to a shift in the meaning of the compound.

[5]The initial word list did not contain any part-of-speech information, so the information about the actual part-of-speech of every compound constituent was obtained from the database files of GermaNet release 9.0.

Modifier	Example	dev	test	train	category %	total %
No word class	*Biogas* (*bio* + *Gas*) 'biogas'	104	187	649	-	2.72%
Noun	*Kaffeebohne* (*Kaffee* + *Bohne*) 'coffee bean'	2,655	5,410	18,796	83.20%	
Adjective	*Gutschein* (*gut* 'good' + *Schein* 'bill, note') 'voucher'	266	500	1,822	8.02%	
Verb	*Brennholz* (*brennen* 'to burn' + *Holz* 'wood') 'burning wood'	213	389	1,403	6.21%	
Preposition	*Untertitel* (*unter* 'under' + *Titel* 'title') 'subtitle'	48	96	345	1.51%	
Adverb	*Nichtexistenz* (*nicht* + *Existenz*) 'non-existence'	23	47	174	0.76%	
Particle	*Selbstentzündung* (*selbst* + *Entzündung*) 'autoignition'	16	14	62	0.28%	
Pronoun	*Allheilmittel* (*alles* + *Heilmittel*) 'universal remedy'	0	2	4	0.02%	
One word class		3,221	6,458	22,606	-	93.59%
Noun/Verb	*Tanzschuh* (*Tanz* 'dance'/*tanzen* 'to dance' + *Schuh*) 'dance shoe'	121	252	870	97.72%	
Adjective/Noun	*Veterinärmedizin* (*veterinär* 'veterinary'/ *Veterinär* 'veterinarian' + *Medizin*) 'veterinary medicine'	3	3	12	1.42%	
Adjective/Verb	*Kühlsystem* (*kühl* 'cold'/*kühlen* 'to cool' + *System*) 'cooling system'	0	1	10	0.86%	
Two word classes		124	256	892	-	3.69%
Total		3449	6901	24147	-	100%

Table 5.2 Statistics for the German dataset used for compounding experiments. The last column lists the percentage of a particular subtype with respect to the whole dataset (34497 compounds). The next to last column lists the percentages within a particular category.

The subset of compounds where the modifier does not have a word class consists of compounds where the modifier is part of a small class of morphemes that cannot appear as individual words - also known as *bound morphemes* (Henrich and Hinrichs, 2011). An example of this class is the compound *Himbeere* 'raspberry', where the head *Beere* 'berry' can appear independently, but the modifier *Him-* does not carry meaning on its own.

The availability of word class information in GermaNet makes it possible to asses composition models using two datasets: a `mixed` dataset, containing all the 34497 nominal compounds obtained after the filtering step and a more specialized `nn-only` dataset, containing only the compounds marked as having a single noun as modifier. Compounds that are clearly marked in GermaNet as having a noun as a modifier represent a total of 77.86% of the `mixed` compound dataset. The data splits created for the `mixed` dataset were reused to extract a noun-noun compounds dataset, by discarding the compounds whose modifier was something else than a single noun.

The `nn-only` compound dataset contains 26861 noun-noun compounds, whose `dev`, `test` and `train` partitions contain 2655, 5410 and 18796 compounds respectively (see Table 5.2, second row). There are 7131 unique modifiers and heads, and the dictionary, i.e. the total number of constituents and compounds, has 33241 unique entries. 751 compounds act as the head or modifier of another compound. The average number of compounds with the same modifier is 5.48, with a minimum of 1, a median of 2, and a maximum of 198 (the word *Land* 'land' is the most frequently used modifier in the `nn-only` dataset). The average number of compounds sharing the same head is 5.88, with a minimum of 1, a median of 2 and a maximum of 146 (the word *Haus* 'house' is the most frequent head word in the dataset).

The experiments in Sections 5.4.3 through 5.4.6 report the results of training and testing composition models both on the `nn-only` and on the `mixed` dataset.

Table 5.3 lists some entries of the `nn-only` dataset. An entry specifies the lemmas of the modifier, of the head and of the compound. Many compounds in German have linking elements (see compounds in Table 5.3 where the last column is not empty)[6]. Given that linking elements do not play a role in the semantics of the compound (Fleischer and Barz, 2012:186), the composition models presented in this thesis will only operate on the *lemmatized* versions of the compound and its constituents.

The dataset contains the lowercased versions of the compound and its constituents. This decision was prompted by practical reasons, as training a language model using the properly-cased version of the constituents would lead to a substantial increase in the size of the vocabulary.

Modifier lemma	Head lemma	Compound lemma		Linking element
Kunst 'art'	*Galerie* 'gallery'	*Kunstgalerie* 'art gallery'		
Ende 'end'	*Gerät* 'device'	*Endgerät* 'terminal device'	-	*e*
Hund 'dog'	*Steuer* 'tax'	*Hundesteuer* 'dog tax'	+	*e*
Ostern 'Easter'	*Tag* 'day'	*Ostertag* 'Easter day'	-	*n*
Experte 'expert'	*Kommission* 'committee'	*Expertenkommission* 'commission of experts'	+	*n*
Verkehr 'traffic'	*Sprache* 'language'	*Verkehrssprache* 'interlingua'	+	*s*
Stern 'star'	*Himmel* 'sky'	*Sternenhimmel* 'spangled sky'	+	*en*
Rind 'cow'	*Zucht* 'rearing'	*Rinderzucht* 'cattle rearing'	+	*er*
Gesetz 'law'	*Text* 'text'	*Gesetzestext* 'legal text'	+	*es*
Glaube 'belief'	*Artikel* 'article'	*Glaubensartikel* 'article of belief'	+	*ns*
Herz 'heart'	*Angelegenheit* 'matter'	*Herzensangelegenheit* 'matter of heart'	+	*ens*

Table 5.3 Examples of compounds from the `nn-only` dataset. The selection of linking elements is based on the selection in Henrich and Hinrichs (2011). The + sign indicates that the linking element is inserted between the modifier and the head. The - sign indicates that the element is removed from the right end of the modifier.

5.3.2 Word Representations for German

Four vector space language models for German were trained, with 50, 100, 200 and 300 dimensions respectively. The training was performed using the GloVe package (Pennington et al., 2014) and a 10 billion token raw-text corpus extracted from the DECOW14AX corpus (Schäfer, 2015). The corpus was pre-processed in a similar way to the one described in Collobert et al. (2011b): all the words were lowercased, punctuation was removed and each number was replaced by the string 'NUMBER' (irrespective of how many digits it had). The training corpus is not lemmatized - constituent/compound representations are based exclusively on the appearances of the respective word forms in the corpus.

Although using the originally-cased corpus or using a fully lemmatized corpus are interesting alternatives for creating word representations, they will not be pursued in this thesis. The focus here is on testing the performance of different composition models using the same word representations.

The used vocabulary had 1,029,270 (1M) words, and contained all the words with a minimum frequency of 100 (the full vocabulary had 50M unique words). Each model was trained for 15 iterations using GloVe's default training parameters, the only

[6]The selection of linking elements is based on the list mentioned in Henrich and Hinrichs (2011).

modification being the use of a symmetric context when constructing the co-occurrence matrix (10 words to the left and to the right of the target word).

5.4 Evaluating Composition Models

The experiments in this section will focus on the ten composition models described in Section 5.2. Table 5.4 gives an overview of the names of these composition function, as they will be used in the remainder of this chapter, and reiterates their mathematical formulation. The ten composition models in Table 5.4 were implemented using the Torch7 library (Collobert et al., 2011a), whose `nngraph` module allows for an easy creation of different architectures. All models are trained using the `CosineEmbeddingCriterion`. The optimization uses mini-batch `Adagrad` (Duchi et al., 2011).

The models are trained using early stopping (Prechelt, 1998) with a patience of 5 epochs. The last column in Table 5.4 lists the initialization method for the parameters of the model (where needed). Initial tests on the `dev` set showed that initializing parameters from a normal distribution of mean 0 and standard deviation $1e-4$, $\mathcal{N}(0, 1e-4)$ leads to good results for all models. The initialization method will remain fixed throughout the experiments described in this chapter.

Composition Model	Short name	Formula	Initialization
`head`	`head`	$\mathbf{p} = \mathbf{v}$	none
`modifier`	`modifier`	$\mathbf{p} = \mathbf{u}$	none
`addition`	`addition`	$\mathbf{p} = \mathbf{u} + \mathbf{v}$	none
`multiplication`	`mul`	$\mathbf{p} = \mathbf{u} \odot \mathbf{v}$	none
`weighted addition`	`w_addition`	$\mathbf{p} = \lambda\mathbf{u} + \beta\mathbf{v}$	$\lambda, \beta \in \mathcal{N}(0, 1e-4)$
`lexical function`	`lexfunc`	$\mathbf{p} = \mathbf{A_u v}$	$\mathbf{A_w}$: $\mathbf{I} + \mathcal{N}(0, 1e-4)$
`full additive`	`fulladd`	$\mathbf{p} = \mathbf{W_1 u} + \mathbf{W_2 v}$	$\mathbf{W_1}, \mathbf{W_2}$: $\mathcal{N}(0, 1e-4)$; no biases
`dilation`	`dilation`	$\mathbf{p} = (\mathbf{u} \cdot \mathbf{u})\mathbf{v} + (\lambda - 1)(\mathbf{u} \cdot \mathbf{v})\mathbf{u}$	$\lambda \in \mathcal{N}(0, 1e-4)$
`matrix`	`matrix`	$\mathbf{p} = g(\mathbf{W}[\mathbf{u}; \mathbf{v}] + \mathbf{b})$	$\mathbf{W}$: $\mathcal{N}(0, 1e-4)$
`full lexical`	`fulllex`	$\mathbf{p} = g(\mathbf{W}[\mathbf{A_v u}; \mathbf{A_u v}] + \mathbf{b})$	$\mathbf{A_w}$: $\mathbf{I} + \mathcal{N}(0, 1e-4)$ $\mathbf{W}$: $\mathcal{N}(0, 1e-4)$

Table 5.4 Existing composition functions to be evaluated in this chapter. $\mathbf{u}$ and $\mathbf{v}$ are the vector representations of the modifier and the head respectively. $\mathbf{p}$ is the composed representation of the compound, the result of applying a composition model to the constituent representations.

`lexfunc` and `fulllex` use lookup tables to train the individual word matrices, $\mathbf{A_w}$. Each individual word matrix was initialized to the identity matrix $\mathbf{I}$ plus a small amount of noise from a normal distribution, $\mathcal{N}(0, 1e-4)$. This type of initialization was proposed by Socher et al. (2012) and allows the model to use the original vectors initially ($\mathbf{Iv} = \mathbf{v}$), and to navigate away from the identity matrix only when the training data provides enough evidence.

`matrix` and `fulllex` are reimplementations of the models used in Socher et al. (2010, 2012). The existing implementations were part of a more complex recursive architecture aimed at constructing representations for full sentences, and thus not directly reusable for the compound analysis task described here.

Also, note that although implemented using neural networks, some of the weight matrices were not defined using a bias term. The implementations in this thesis were kept as close as possible to the originally proposed functions: i.e. `lexfunc` and `fulladd` do not have a bias term, whereas `matrix` and `fulllex` do feature a separate bias vector (**b**). The formulas in Table 5.4 present the exact mathematical formulation that was implemented for each of the models.

Different implementations for composition functions are different with respect to the way the composition process affects the input word vectors. In Socher et al. (2010, 2012) the input vectors are fine-tuned by the composition process and this might indeed result in a better composition. However, it might also make it more difficult to apply the learned composition functions to compounds whose constituents were not part of the training data.

In contrast, Mitchell and Lapata (2010); Baroni and Zamparelli (2010); Zanzotto et al. (2010); Guevara (2010); Dinu et al. (2013b) all opt for studying composition without modifying the input vectors. When creating compound representations, the composition models are expected to work well with constituents they have not been trained on. Fine-tuning the initial representation for the training compounds can make it more difficult for the composition models to perform well on unseen compounds. This is why the implementations in this thesis belong to the second camp: all the composition models keep the input vectors fixed during the composition process.

5.4.1 Evaluation Criteria

Evaluating composition models essentially amounts to comparing the **observed** compound representations, obtained using distributional methods, to the **composed** representations obtained through composition.

Intuitively, a good composed representation should be among the nearest neighbors of the corresponding observed representation. A composition model is judged to perform well if it builds good composed representations for a large number of previously unseen compounds, i.e. the `dev` and `test` set compounds. This intuition is presented visually in Figure 5.2: composition models work well when the composed representations (displayed using orange filled circles) are close to the observed representations (displayed using blue unfilled circles) of the same compounds. Here the composed representation of *Apfelbaum* 'apple tree', marked with an orange filled circle, is close to the observed representation of *Apfelbaum*, which is marked with a blue unfilled circle. The composed representation also 'fits' the neighborhood of the observed representation: its neighbors are the head *Baum* 'tree', as well as other fruit trees like *Kirschbaum* 'cherry tree'. At the same time the representation of the compound *Apfelbaum* is considerably further away from the representation of its modifier *Apfel* 'apple' and other similar concepts like *Birne* 'pear' and *Kirsch* 'cherry'.

The performance of the composition models will be evaluated using two methods: (i) the **cosine distance** between the composed and the corresponding observed vectors

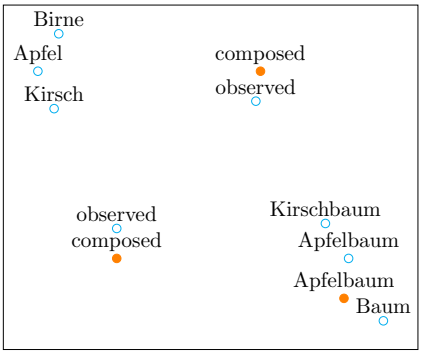


Fig. 5.2 Word representations in an idealized 2-dimensional vector space. Composition models work well when the composed representations (displayed with orange filled circles) are close to the observed representations (displayed with blue unfilled circles) of the same compounds.

and (ii) a modified version of the **rank evaluation** proposed by Baroni and Zamparelli (2010).

Cosine distance The **cosine distance** is based on the cosine similarity, defined in Eq. 4.3 (the formula is repeated, for convenience, in Eq. 5.15). The cosine similarity measures the angle between two vectors: if the vectors are parallel, the angle is 0°, and its cosine is 1. Perpendicular vectors have an angle of 90°, whose cosine is 0. Opposite vectors have an angle of 180° and a cosine of -1. The cosine similarity is defined as the inner product of two vectors, divided by the norm (or length) of the two vectors.

$$cosine\_similarity : \cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = \mathbf{u} \cdot \mathbf{v}, \text{ if } \|\mathbf{u}\|_2 = \|\mathbf{v}\|_2 = 1 \tag{5.15}$$

The range of the cosine similarity is therefore the interval $[-1, 1]$. The cosine similarity is transformed into a distance metric (allowing only positive values) by subtracting it from 1. Although the result is not a proper metric (Chanwimalueang and Mandic, 2017), it has, in practice been extensively used to quantify the similarity of word vectors. The mathematical formulation of the **cosine distance** is given in Eq. 5.16. The cosine distance is 0 when the angle between the vectors is 0°, 1 when the angle is 90°, and 2 when the angle between the vectors is 180°, as illustrated in Fig. 5.3.

$$cosine\_distance : 1 - \cos(\mathbf{u}, \mathbf{v}) = 1 - \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = 1 - \mathbf{u} \cdot \mathbf{v}, \text{ if } \|\mathbf{u}\|_2 = \|\mathbf{v}\|_2 = 1 \tag{5.16}$$

$J_{cosine\_distance}$, the mean cosine distance between the composed ($\mathbf{c}^{composed}$) and observed ($\mathbf{c}^{corpus}$) representations, can be computed for a set of $|C|$ compounds using the formula in Eq. 5.17:

$$J_{cosine\_distance} = \frac{1}{|C|} \sum_{i=1}^{|C|} 1 - \cos(\mathbf{c}^{composed}, \mathbf{c}^{corpus}) \tag{5.17}$$

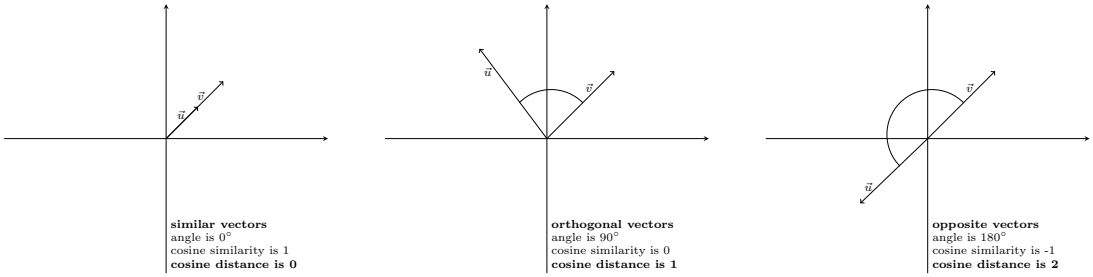
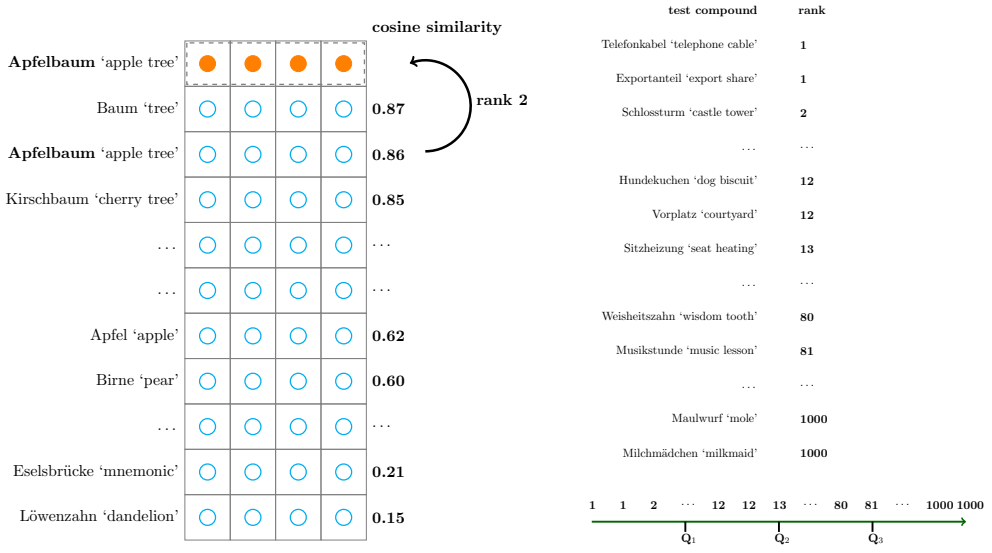


Fig. 5.3 Vector similarity and cosine distance: the smaller the angle between the vectors the smaller the cosine distance and therefore the measured error.

The cosine distance is used as an optimization criterion in training the composition models in this thesis, using the implementation in `CosineEmbeddingCriterion`.[7]

Rank evaluation The rank evaluation idea was introduced by Baroni and Zamparelli (2010). The intuition is that one can estimate the quality of a composition model by computing, for each composed vector in the test set, the cosine similarity to all the observed vectors (both of individual words and of compounds) and recording the position of the corresponding observed representation is the cosine-ordered list.



(a) (b)

Fig. 5.4 Rank evaluation as proposed by Baroni and Zamparelli (2010). (a): Each composed representation is compared to all observed representations in terms of cosine similarity. The rank assigned to the composed representation is given by the position of the corresponding observed representation in the cosine-ordered list. (b): Quartile computation for the ranks of the test set compounds.

[7] `https://github.com/torch/nn/blob/master/doc/criterion.md#nn.CosineEmbeddingCriterion`

Fig. 5.4a illustrates the Baroni and Zamparelli (2010) rank computation process for the compound *Apfelbaum* 'apple tree'. Filled orange dots are used to illustrate composed vectors. Unfilled blue dots are used to represent the original, corpus-based vectors. The cosine similarity is computed between the composed representation of *Apfelbaum* and all the observed representations. In this case, the original representation of the compound *Apfelbaum* is on position 2 in the ordered list, therefore the composed representation is considered to have rank 2.

The best possible result is when the observed representation is the nearest neighbor of the composed representation. This situation corresponds to assigning the rank 1 to the composed vector. The cut-off rank 1000 is assigned to all the representations with ranks above 1000 (1K).

The evaluation on a whole test set involves first computing the rank for each composed representation, as shown above. The list of ranks is then sorted, and the first, second and third quartiles (Q_1, Q_2/median, Q_3) are computed. Fig. 5.4b illustrates the quartile computation process.

First, the median value is used to separate the ordered rank list in two halves. The **median** of a list of numbers is the middle value if there is an odd number of elements in the list, and the mean of the two middle values otherwise. E.g. the median of [1, 2, 3, 4, 5] is 3, and the median of [1, 2, 3, 4] is 2.5. The median is saved as the Q_2 value, and the list is split in two halves. If the list had an odd number of elements, the median value is not included in any of the halves. Q_1 and Q_3 are then computed as the median values of the resulting sublists, each containing half of the initial data.[8]

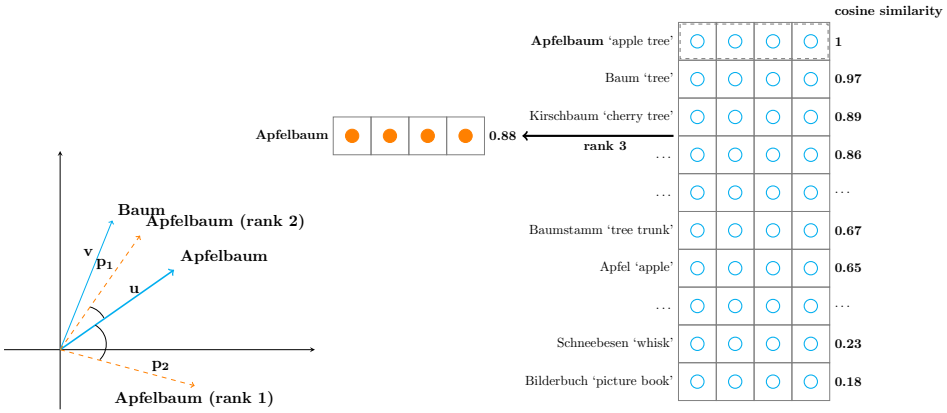
A Q_1 value of 2 means that the first 25% of the data was only assigned ranks 1 and 2. Similarly, Q_2 and Q_3 refer to the ranks assigned to the first 50% and 75% of data, respectively. For the example test dataset sketched in Fig. 5.4b, Q_1=2, Q_2=13 and Q_3=81.

Target-centric rank evaluation The results reported in Dima (2015), Dima (2016) use Baroni and Zamparelli (2010)'s rank evaluation as described above. However, reporting also the average cosine distance between the composed and the observed representations of the test compounds pointed to some shortcomings of the rank computation. Because the rank is computed based on the similarity to the composed representation, the reference representation is going to change from one composition model to the other (or from one variant of a composition model to the next). This makes it possible for the ranks to show improvements even in cases where the cosine similarity is worse.

Fig. 5.5a illustrates a case where such an outcome is possible. The solid blue lines depict the original vectors and the dashed orange lines depict the composed vectors. A first composed vector for *Apfelbaum*, $\mathbf{p_1}$, is closer to the original vector for *Baum*, $\mathbf{v}$, than of the original vector for *Apfelbaum*, $\mathbf{u}$, and is assigned therefore the rank 2. At a later moment, a variant of the composition model creates the composed vector

[8]The quartile computation procedure described in more detail: `https://en.wikipedia.org/wiki/Quartile#Method_1`, last accessed 18 April 2018.

$\mathbf{p_2}$, which is, in terms of cosine similarity, further away from the original vector for *Apfelbaum*, $\mathbf{u}$. However, because the composed vector is taken as reference when computing the cosine similarity, $\mathbf{p_2}$ gets assigned the rank 1, as it is the closes vector to $\mathbf{u}$.



(a) (b)

Fig. 5.5 (a) Problem in the Baroni and Zamparelli (2010) evaluation: $\mathbf{p_1}$, a composed vector that is closer to its original $\mathbf{u}$ is ranked worse that $\mathbf{p_2}$, a composed vector that is further away from the original $\mathbf{u}$ because the vector of reference is the composed vector, which changes from one model to another; (b) The vector of reference is the observed representation of each compound. The composed representations created by each composition model are compared against the other vectors in the vector space, and ranked according to the same frame of reference.

The evaluation can be improved by always taking as a reference the original vector of the word. As the illustration in Fig. 5.5b, the cosine similarity is computed this time between the original vector for *Apfelbaum* and all the other original vectors from the vocabulary. Composed representations are assigned a rank based on their position with respect to this list. Referring back to Fig. 5.5a, the new, target-centric evaluation will correctly assign a better rank to $\mathbf{p_1}$, which is closer to the original representation of *Apfelbaum*, and a worse rank to $\mathbf{p_2}$, which is further away in terms of cosine similarity.

In the next sections the new, target-centric rank evaluation and the cosine distance are going to be used to asses the performance of composition models. The evaluation metrics will serve both for identifying the best hyperparameter settings for the composition models, as well as for comparing the results of different composition models on several benchmark test sets.

5.4.2 The Impact of Embedding Normalization

Several researchers (Ó Séaghdha, 2008; Baroni and Zamparelli, 2010; Levy et al., 2015a) have pointed out that the efficiency of systems that use distributed word representations depends in a great measure of some seemingly trivial preprocessing choices. One such

choice is whether **normalization** should be applied to the initial word space, and if yes, what is the best normalization procedure to use.

Normalizing a vector $\mathbf{u} = (x, y)^\top$ is done by dividing each of its components by the length of the vector. The length of a vector $\mathbf{u}$ is typically denoted as $\|\mathbf{u}\|_2$ and is computed using the formula in Eq. 5.18. The length of the vector is also known as the Euclidean norm or the L_2 norm of a vector.

$$\|\mathbf{u}\|_2 = \sqrt{x^2 + y^2} \tag{5.18}$$

A normalized version of $\mathbf{u}$ can therefore be computed using Eq. 5.19. The normalized vector $\hat{\mathbf{u}}$ is a **unit vector**, that is, its length is equal to 1.

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|_2} \tag{5.19}$$

Normalization has several advantages when building machine learning models. Consider, for example, using a vector whose components are (30, 40), and another vector whose components are (3, 4). The two vectors differ in their magnitude, but have the same orientation. However, because of the difference in magnitude, finding a set of parameters that works well with both vectors can prove difficult. When normalized to unit norm, however, both vectors will be the same: $(\frac{3}{5}, \frac{4}{5})$. This makes finding a set of parameters that will give good results for many of the input vectors a much easier task.

Existing research using as input distributional representations points towards a beneficial effect of normalization: Ó Séaghdha (2008) reports a four point improvement in the results using an L_2 linear kernel when using L_2 normalization of the input vectors. Baroni and Zamparelli (2010) obtain the best results for the `addition` composition model when using a normalized word space, and mention that *"non-normalized addition was also tried, but it did not work nearly as well as the normalized variant"*. However, Baroni and Zamparelli (2010) do not provide any numbers comparing the performance of normalized vs. non-normalized representations. Levy et al. (2015a) take the normalization method to be one of the hyperparameters of a model using word embeddings. They perform preliminary experiments using four L_2-normalization types: none, row, column and both row and column. Again, they do not report any numbers, but specify that the best results were obtained using L_2 row normalization of the word spaces, which they go on to use in their other experiments.

This section explores two ways of normalizing the input word embeddings for the composition task: L_2 row normalization and L_2 column normalization. Furthermore, the results using some form of normalization are contrasted against using the raw, not normalized input vectors. L_2 row normalization means making each individual word vector have a unit norm. L_2 column normalization translates to having each component of the vector space be normalized to unit norm across the vector representations of all the words in the dataset dictionary (33241 entries).

Table 5.5 displays the results of the ten composition models evaluated on the `dev` split of the `nn-only` dataset. The models use the 50-dimensional embeddings. All

parametric models are optimized with AdaGrad, using a batch size of 100 items. The learning rate was chosen separately for each model, by trying out from three different options: $\eta \in 0.001, 0.01, 0.1$. Table B.1 in Appendix B gives a detailed overview of the results of each model using the different learning rates and different normalization types. Most models perform at their best with a learning rate $\eta = 0.01$. The dilation model is the only one that performs decidedly better when using a higher learning rate ($\eta = 0.1$). The three different normalization setups - no normalization, L_2 row normalization (L_2-row) and L_2 column normalization (L_2-col) - are compared using the best performing learning rate for each model.

Comp. Model	η	none				L_2-row				L_2-col			
		Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d
mul	0.01	1K	1K	1K	0.9627	1K	1K	1K	0.9627	1K	1K	1K	**0.9619**
modifier	0.01	356	1K	1K	0.6923	356	1K	1K	0.6923	342	1K	1K	**0.6869**
head	0.01	65	480	1K	0.5912	65	480	1K	0.5912	59	433	1K	**0.5820**
addition	0.01	67	377	1K	0.5731	64	356	1K	0.5712	59	335	1K	**0.5615**
dilation	0.1	50	371	1K	0.5734	50	371	1K	0.5734	44	330	1K	**0.5632**
w_addition	0.01	52	325	1K	0.5627	50	322	1K	0.5615	45	288	1K	**0.5509**
lexfunc	0.01	7	44	377	**0.4524**	7	44	377	**0.4524**	7	45	374	0.4514
fulladd	0.01	4	11	49	0.3677	3	11	49	**0.3678**	4	11	49	0.3696
matrix	0.01	3	10	46	**0.3649**	3	10	46	**0.3649**	3	11	49	0.3674
fulllex	0.01	2	6	30	0.3384	2	6	30	**0.3379**	3	8	36	0.3475

Table 5.5 Results on the dev split of the nn-only dataset, using 50-dimensional embeddings. Row normalization works better with models that have a large number of parameters (underlined). In contrast, models with no or few parameters have better results when using column normalization.

The two normalization types display an interesting dichotomy: models that have a large number of parameters (underlined) work better with row normalized inputs, whereas the ones with few or no parameters work better with column-normalized inputs. A possible explanation for why this might be the case is that the models with no or few parameters benefit from the column normalization step, as this ensures that the individual dimensions of the input representations carry a more consistent meaning across the whole word representation space.

Another observation is that there is, in general, little difference between using the L_2-row normalized input vectors and the raw vectors, for most parametric composition models. A likely reason for this result is the use of the cosine distance as the optimization objective. The cosine distance is based on the cosine similarity, which is defined as the dot product of the normalized vectors. When using raw (un-normalized) vectors the composition models have to perform an implicit normalization in order to minimize the loss.

Although normalization did not bring large improvements in the current experimental setup, machine learning best practices recommend the use of normalized inputs to learning algorithms, for the reasons explained above. Therefore, the experiments that follow all use L_2-row normalized vector spaces for the highly parametric models (underlined), and L_2-col normalization for the other models.

5.4.3 Results on the German nn-only composition dataset

The ten composition models described in Section 5.2 were evaluated using word representations of increasing size - 50, 100, 200 and 300 dimensions (described in Section 5.3.2). All models are trained on the train split. Results are reported both on the dev and on the test split. The models are considered to generalize well if the results on the unseen test dataset are comparable to those obtained on the dev set, which was used to choose the model hyperparameters.

During the training procedure each model is trained on the train set and tested on the dev set. The training procedure stops when the error on the dev set starts increasing, using an early stopping procedure (described in detail in Section 3.3.3). The intuition behind this procedure is that a good model can be obtained by monitoring the error on the dev set and stopping the training as soon as the dev error constantly increases for a number of epochs. In the experiments described here the increase is allowed for 5 consecutive epochs, after which the last model, saved before these 5 epochs, is returned as the best model.

The best model, which has only 'seen' the train and the dev splits during its training phase is then tested on the test split. Because the test split was kept completely separate during the training procedure, the error obtained on the test split - the generalization error - is indicative of the model's generalization capabilities.

	Model	Dro	50d				100d				200d				300d			
			Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d
dev	mul	0	1K	1K	1K	0.9619	1K	1K	1K	0.9769	1K	1K	1K	0.9944	1K	1K	1K	0.9974
	modifier	0	342	1K	1K	0.6869	113	797	1K	0.7060	75	506	1K	0.7457	**62**	**393**	**1K**	0.7630
	head	0	59	433	1K	0.5820	24	157	1K	0.6142	19	112	736	0.6688	**16**	**83**	**561**	0.6911
	addition	0	59	335	1K	0.5615	18	88	447	0.5778	14	57	272	0.6337	**11**	**41**	**181**	0.6558
	dilation	0	44	330	1K	0.5632	16	102	700	0.5899	13	67	427	0.6445	**10**	**50**	**312**	0.6673
	w_addition	0	45	288	1K	0.5509	16	77	410	0.5704	12	52	250	0.6278	**10**	**36**	**181**	0.6505
	lexfunc	0.5	5	32	246	0.4350	4	18	140	0.5019	3	11	69	0.5503	**3**	**9**	**53**	0.5787
	fulladd	0	3	11	49	0.3678	2	6	19	0.4258	2	4	10	0.4783	**2**	**3**	**8**	0.5057
	matrix	0	3	10	46	0.3649	2	6	18	0.4241	2	4	10	0.4767	**2**	**3**	**8**	0.5037
	fulllex	0.25	2	6	27	0.3317	2	4	13	0.3980	2	3	9	0.4642	**2**	**3**	**7**	0.4993
test	mul	0	1K	1K	1K	0.9610	1K	1K	1K	0.9787	1K	1K	1K	0.9969	1K	1K	1K	0.9999
	modifier	0	319	1K	1K	0.6875	120	760.5	1K	0.7070	77	491	1K	0.7468	**60**	**375**	**1K**	0.7645
	head	0	61	451	1K	0.5813	23	162.5	1K	0.6134	18	111	736	0.6688	**16**	**83**	**548**	0.6912
	addition	0	64	325.5	1K	0.5620	18	82	432	0.5776	14	55	276	0.6341	**11**	**40.5**	**191**	0.6565
	dilation	0	49	330.5	1K	0.5625	17	98	660	0.5894	13	67	438	0.6450	**11**	**49**	**319**	0.6679
	w_addition	0	48	279	1K	0.5510	15	73	401	0.5698	12	48.5	256	0.6280	**10**	**36**	**180**	0.6509
	lexfunc	0.5	5	27	253	0.4318	4	16	162	0.4996	3	10	73	0.5482	**3**	**8**	**56**	0.5771
	fulladd	0	3	10	45	0.3663	2	6	19	0.4262	2	4	10	0.4778	**2**	**3**	**8**	0.5055
	matrix	0	3	10	42	0.3635	2	6	19	0.4243	2	4	10	0.4763	**2**	**3**	**7**	0.5034
	fulllex	0.25	2	6	24	0.3311	2	4	12	0.3973	2	3	9	0.4623	**2**	**3**	**7**	0.4982

Table 5.6 Quartiles for the 2655 composed representations of the dev split and the 5410 composed representations of the test split of the nn-only dataset, ranked with respect to the observed representations. Using **cosine distance criterion**.

Table 5.6 presents the evaluation results on the dev and test splits of the nn-only dataset. The best result for each model across the different dimension input vectors

is marked in bold. A first observation that can be made is that the performance of composition models increases with the dimensionality of the input vectors. One model makes an exception: the multiplication model, `mul`, which is consistently the worst performing model on every dimension.

Model	Formula	d=50	d=100	d=200	d=300
`mul`	-	0	0	0	0
`modifier`	-	0	0	0	0
`head`	-	0	0	0	0
`dilation`	λ	1	1	1	1
`addition`	-	0	0	0	0
`w_addition`	λ, β	2	2	2	2
`lexfunc`	$d^2 \times \|V\|$	17,827,500	71,310,000	285,240,000	641,790,000
`fulladd`	$d^2 \times 2$	5,000	20,000	80,000	180,000
`matrix`	$d \times 2d + d$	5,050	20,100	80,200	180,300
`fulllex`	$d^2 \times \|V\| + d^2 \times 2 + d$	17,832,550	71,330,100	285,320,200	641,970,300

Table 5.7 Number of trainable parameters for each model and for each dimensionality of the input word embeddings on the `nn-only` dataset. Vocabulary size $|V| = 7,131$. d is the dimensionality of the input embeddings (one of 50, 100, 200 and 300).

The positive impact of using higher-dimensional input representations can be consistently observed, even for models where the number of parameters is quadratic with respect to the size of the input word embeddings (e.g. `lexfunc`, `fulllex`). Table 5.7 displays, for comparison, the number of parameters for each of the 10 composition models for each input dimensionality. However, the improvements brought about by increasing the dimensionality of the input vectors do eventually flatten out: the improvements of the models using 200 and 300 dimensional inputs are comparatively smaller than the ones between the modes using 50 and 100 dimensional inputs.

Models with a large number of parameters are easily prone to overfitting - i.e. learning unimportant details about the training data at the expense of performing well on the test data. To reduce overfitting a dropout layer was added to the composition models with a high number of parameters. For `fulladd` and `matrix` the dropout layer was added to the $\mathbf{W_1}, \mathbf{W_2}/\mathbf{W}$ matrices. For `lexfunc` and `fulllex` dropout was added to the lookup table. Several dropout rates were tested for each model: 0, 0.25, 0.5 and 0.75. The dropout rate is the number of units that get discarded from the layer - i.e. a dropout rate of 0.25 means 25% of the units are discarded and 75% of the units are kept. The experiments showed that the `fulladd` and `matrix` models do not benefit from the addition of dropout. However, the `lexfunc` and `fulllex` models do show a marked improvement when using dropout, with the best dropout rate being 0.5 for `lexfunc` and 0.25 for `fulllex`. The results in Table 5.6 show the results of the model with the best hyperparameters in each case. The results using all the different hyperparameters are available in Appendix B, Table B.2. As before, models other than `dilation` use a learning rate of 0.01. `dilation` uses a learning rate of 0.1.

The remainder of this section analyses in more detail the results of each of the ten composition models. Appendix C contains tables with example neighbors. For each composition model, a sample of 12 compounds was extracted from the `test` set: four of the highest ranking compounds, four middle-ranking and four low-ranking compounds. The nearest neighbors are computed for each of the sample compounds, using the composed vector as a representation. The representations are obtained using the best performing model, indicated in bold in Table 5.6. The neighbors are searched for in the observed vector space. The cosine similarity is listed next to each neighbor. The examples are in lowercase German, as in the dataset. Due to space considerations, only the examples referred to in this section's text are glossed with their English translation.

`head` For many compounds, the referent of the whole is, in many cases, of the same semantic type as the referent of the head, e.g. an *Apfelbaum* 'apple tree' is a type of *Baum* 'tree'. This is why the `head` model is used as a **strong baseline** for the compound composition task.

Table C.1 contains a set of examples extracted from the results of the `head` model on the `test` dataset. It features three groups of four compounds assigned the best rank (1), the middle rank (83) and the cut-off rank (1000). E.g. *Wasserstrahl* 'jet of water', *Teilprivatisierung* 'partial privatization', *Rechnungsbetrag* 'invoice amount' and *Körpergewicht* 'bodyweight' are all assigned the best rank, 1; *Malteserkreuz* 'Maltese cross' and the other compounds on the second row are assigned the middle rank 83; and *Schlagzahl* lit. 'strike count', 'the number of times the paddle hits the water in an interval of one minute' and the other compounds on the third row are assigned the cut-off rank 1000.

The `head` model does not learn any parameters, it just uses a normalized version of the corpus-based head word representation as the compound representation. The examples show that the compounds for which this model works well tend to be very close in meaning to their head word or even used interchangeably with it. 125 `test` compounds are assigned the rank 1 by the `head` model, with an average cosine similarity of 0.59580. Overall, the results of the `head` composition model are, unsurprisingly, not impressive. They show that it is difficult to model the semantics of a compound without taking into account the contribution of the modifier.

`modifier` A second baseline is the `modifier` model, where the normalized representation of the modifier is used to represent the compound. According to the results in Table 5.6, the `modifier` baseline performs worse than the `head` baseline: the `modifier` model obtains on the `test` compounds an average cosine distance of 0.7645, compared to the 0.6912 average obtained by the `head` model. This suggests that the `head` representation is a better fallback representation to use when a compound representation is unavailable. Table C.2 shows examples of neighbours for compounds represented via their modifier vector.

Only 32 compounds are assigned rank 1 using the `modifier` model - a significant decrease from the 125 in the case of the `head` model. The average similarity of the

rank 1 compounds is 0.58728. The modifier representation is, in general, a poor approximation for the compound representation. E.g. for a compound like *Stadtvilla* 'urban villa', the representation of the modifier *Stadt* 'town' is almost perpendicular to the original representation of the compound 'Stadtvilla', with a cosine similarity of 0.08773. This makes it totally unfit as a representation for the compound - a fact also confirmed by the low rank assigned to it - 1000.

`mul` The `mul` composition function is by far the worst performing model on all input dimensions. The results in Table 5.6 show that the `mul` composition model is unable to produce good compound representations. The average cosine distance on all dimensions is close to 1 for all input dimensionalities, corresponding to an average cosine similarity that is close to 0. This means that in the majority of cases the composed vector is perpendicular to the observed representation of the compound. The multiplicative model performs significantly worse than both the `head` and the `modifier` baselines.

Table C.3 shows, on the first row, the best composed representations obtained using the `mul` model. The ranks assigned to the best compositions produced by the `mul` model - 105, 106, 306, 307 - speak for the poor quality of these composed representations.

However, this result is surprising, considering the good performance of the multiplicative model reported by Mitchell and Lapata (2008), Mitchell and Lapata (2010) and Blacoe and Lapata (2012), where the multiplicative model using a simple semantic space emerged as the best performer for noun-noun combinations.

Is important to note, however, the different evaluation methodology used in those studies. Mitchell and Lapata (2010) work with pairs of phrases ($phrase_1$, $phrase_2$), where $phrase_1$ is '$word_a$ $word_b$' and $phrase_2$ is '$word_c$ $word_d$'. Mitchell and Lapata (2010) build composed representations for each phrase, using multiplication as a composition model. Then they compute the cosine similarity between the two phrase representations. The cosine similarity is then correlated with the average human rating concerning the similarity of the two phrases (with values between 1 - low similarity - and 7 - high similarity), using Spearman's rank correlation coefficient ρ.

Notice that by using this evaluation measure, these studies *never compare a composed representation to an observed representation.* Their goal is not to re-embed the composed phrase into the vector space where the constituent representations reside. As long as the composition function captures enough about the semantics of the two words to make it useful for the rank correlation computation, it is considered to be a satisfactory composition function.

This difference in goals and evaluation measures also explains why the two assessments of element-wise multiplication as a possible composition function are so different. If the goal is just to compare the similarity between two composed representations, the `mul` composition function is a good candidate. If the goal is to create composed representations that fit into the original vector space, `mul` is not a competitive choice.

`addition` The `addition` model outperforms, according to the results in Table 5.6, both the `head` and the `modifier` baselines. Although the additive composition is

symmetric, i.e. will produce the same representation for 'car factory' and 'factory car', this vector combination outperforms the use of either one of the constituent vectors alone.

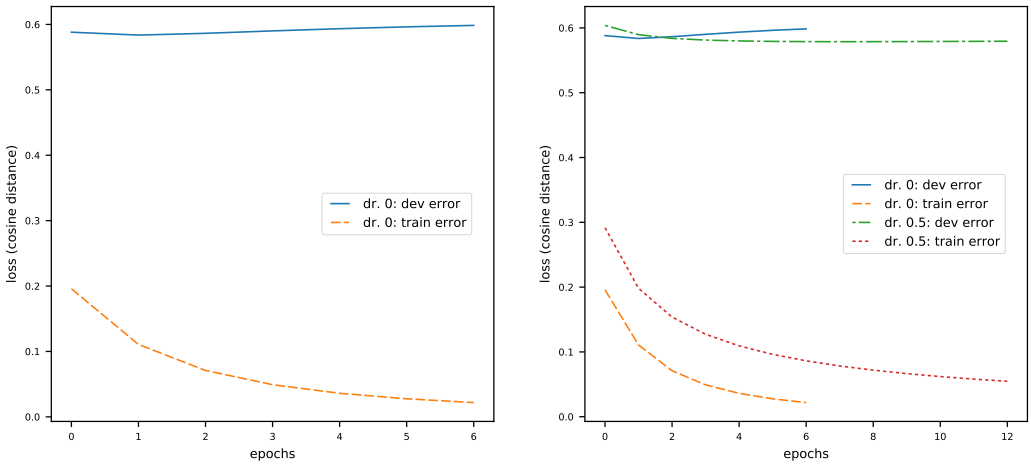
Table C.5 illustrates some nearest neighbors of the composed representations produced by the `addition` model. The average cosine for the 76 compounds in the `test` set that are assigned rank 1 using the `addition` composition model is 0.56764.

`dilation` is the first parametric composition model to be evaluated. It scores, according to the results in Table 5.6, better than the `head` and `modifier` baselines on all input dimensions. Moreover, it achieves better results than the `addition` model, a performance due to the asymmetric treatment of the two input vectors. `dilation` achieves the best rank 1 for 123 compounds from the `test` set, with an average cosine of 0.57343. The learned value for λ as a result of the training phase is 1.9306. Table C.4 illustrates some of the nearest neighbors of the composed representations created using the `dilation` model.

`w_addition` The weighted addition model outperforms the baselines and the `addition` model across all input dimensions, according to the results in Table 5.6. It has two parameters, λ and β, which control the contribution of each of the input representations to the resulting composed representations. As a result of the training process, λ and β are assigned values which correspond to the modifier vector contributing 37.5% and the head vector contributing 62.5% of the information, respectively (for the best model using 300-dimensional input representations). The compound representation is thus a linear combination of the modifier and head representations where the head has a considerably larger contribution than the modifier. This two-parameter model outperforms the `dilation` model, which had a single tunable parameter. The examples in Table C.6 feature neighbors of compound representations obtained using `w_addition`. 91 compounds obtain the rank 1, with an average cosine similarity value of 0.55654.

The results of the `w_addition` model suggest that when a compound representation is not available, using a weighted combination of the head and modifier vectors as a back-off representation is better than using only the `head` representation, or than using an unweighted additive combination of the head and modifier representations.

`lexfunc` The lexical function model is the first model in Table 5.6 to have a sizable number of parameters. The number of parameters is given by the formula $(d \times d) \times |V|$, where d is the size of the input embeddings and $|V|$ is the size of the vocabulary. The model trains a separate modifier matrix for each word in the vocabulary, while at the same time discarding the corpus-learned modifier vector. This approach turns out to work better than any of the previous models, and as a result the `lexfunc` model is the first model to consistently produce composed representations ranked with the best possible rank, 1. Despite the large number of parameters, the performance of the `lexfunc` model shows improvements even when using 300-dimensional input vectors.



(a) (b)

Fig. 5.6 Learning curves for the `lexfunc` model. X axis: the number of training epochs; Y axis: the loss of the model. (a) no dropout; (b) no dropout vs. 0.5 dropout on the weights in the lookup table. Dropout makes the model generalize better, leading to better results on the `test` set.

The increased number of parameters, however, makes `lexfunc` quickly overfit the training data, as shown by the learning curves of the model in Fig. 5.6a. The dashed orange line represents the loss of the model on the `train` set, while the solid blue line at the top of the figure represents the loss on the `dev` set. A model is said to overfit when there is a large difference between the `train` and the `dev` loss, as is the case here.

To compensate for overfitting, the `dev` subset was used to experiment with different dropout rates between 0 and 0.75, in 0.25 increments. The dropout technique, introduced in Section 3.3.3, involves randomly zeroing out or 'dropping' some of the network weights, with probability p, each time a new training example is presented for training. This has the effect of preventing the co-adaptation of units - i.e. units should not rely on all their inputs when computing the activation, but should instead be robust to missing inputs. As the results in Appendix B, Table B.2 show, `lexfunc` performs better when using dropout. Its performance peaks when using a dropout rate of 0.5 on the lookup table.

As the learning curves in Fig. 5.6b show, using dropout makes the model generalize better, leading to better results on the `test` set. The `dev` error with a 0.5 dropout - the dot-dash green line - goes lower than the solid blue line, which was the `dev` error without any dropout. Adding dropout also decreases the gap between the `dev` and the `train` error (the dot-dash green line vs. the dotted red line). The benefits of dropout come at the expense of the larger number of epochs needed for the model to converge, as can be seen in Fig. 5.6b.

Table C.7 contains nearest neighbors for composed representations obtained with `lexfunc`. On the `test` set the `lexfunc` model achieves the maximum rank 1 for 651

words. These representations have an average cosine similarity of 0.59182 to their respective observed representations.

fulladd The full additive model has far fewer parameters than the lexfunc model. It trains only two matrices of size $d \times d$, which are used to transform the head and the modifier input representations. The fulladd composition model using 300-dimensional input representations obtains very good results in terms of quartiles: 2 - 3 - 8, surpassing the result of the lexfunc model. The median rank 3 means that fulladd creates meaningful composed representations for more than half of the test set compounds. This is because in many cases synonyms, spelling variants or other forms of the word (e.g. plurals) can turn out to be closer to the original representation. For most compounds, obtaining a rank $\leqslant 5$ means that the composed vector is actually a meaningful, useful representation of that concept. 1193 of the test compounds representations composed using fulladd are assigned rank 1, with an average cosine similarity of 0.55958 to their corresponding observed representations.

As the examples in Table C.8 show, the composed representations obtain good ranks when the neighborhood of the original compound representation is homogeneous, as is the case for the compounds on the first two rows. Compounds with high ranks from the last row have, in contrast, a much 'looser' neighborhood. The composed representation of the compound *Pechstein* 'pitchstone' signals a problem related to the process of creating distributed representations for common nouns. The word *Pechstein* turns out to be very often in the support corpus the last name of a public person. The proper name sense overtakes the common noun sense in the observed representation: the neighbors of the observed representation of *Pechstein* contain the words *Dopingfall* 'doping case' and *Weltcup* 'world cup'. Pechstein turns out to be the family name of Claudia Pechstein, a renown German speed skater. This means that the representation created by the distributional model is not the representation of the compound *Pechstein* meaning 'pitchstone', but the representation of the proper name *Pechstein*. The gold representation provided to the composition model is therefore incorrect, and the composition model cannot, starting from the representations of the words *Pech* 'pitch' and *Stein* 'stone' arrive at the this named entity representation.

Such problems could be avoided by using a named-entity annotated support corpus for creating representations: words that are part of named entities should be represented separately from words with identical spelling that are not part of a named entity. This idea, however, will not be further explored in this thesis.

As for the lexfunc model, different dropout rates were tested. The results in Appendix B, Table B.2 show that fulladd does not benefit from dropout: the model performs optimally with a 0 dropout rate. This is to be expected, given that fulladd has only 0.00028% of the number of parameters in lexfunc (cf. Table 5.7).

matrix The matrix model obtains its best ranks, 2-3-7, using 300-dimensional inputs. Table C.9 illustrates the neighborhoods of the composed representations created using the matrix model. 1217 of the test compounds are assigned the highest rank 1, and

are at an average cosine distance of 0.56108 from their observed representations. As in the case of the `fulladd` model, using dropout did not improve the performance of the model (see Appendix B, Table B.2).

The matrix model is the first to use a nonlinearity in its formulation. Experiments on the `dev` set showed similar performance when not using the nonlinearity: for 50 dimensional input vectors, the cosine distance loss was 0.3649, exactly the same as when using a nonlinearity, while the quartiles showed minors fluctuations: 3 - 10 - 47 (without nonlinearity) vs. 3 - 10 - 46 (with nonlinearity).

`fulllex` The full lexical composition function obtains the best results on the `nn-only test` compounds using the 300-dimensional input vectors: 2 - 3 - 7, and a cosine distance of 0.5016, as reported in Table 5.6. 1099 compounds are assigned the best rank, 1, with an average cosine similarity of 0.56269. `fulllex` outperforms `lexfunc` model, although the two functions have a similar number of parameters.[9] However, while `lexfunc` discards the initial modifier vector, `fulllex` uses both the modifier and the head vectors. This suggests that composition models should make use of all the information made available by the language model, and not discard any vectors.

Similar to `lexfunc`, `fulllex` also benefits from the use of dropout. The results in Appendix B, Table B.2 show that `fulllex` performs at its best with a 0.25 dropout rate applied to the lookup table. As in the case of the `matrix`, removing the outer nonlinearity did not affect the performance of the model. Table C.10 shows the neighbors of a selection of compound representations obtained using the `fulllex` composition model.

Overview Figure 5.7 shows the ranks from Table 5.6 as a box-and-whiskers plot. For each model, the orange line marks the median rank, the lower edge of the box marks the first quartile (Q_1), and the top edge marks the third quartile (Q_3). The length of the box is the interquartile range, IQR = Q_1 - Q_3. The lines connected to the boxes are the lower and upper whiskers, computed as Q_1 - 1.5 * IQR, and Q_3 + 1.5 * IQR. The green circles show the rank outliers, which fall outside the interval covered by the lower and upper whiskers.

Figure 5.8 is another visualization of the results from Table 5.6. It plots the ordered `test` set ranks for each composition model. The figure uses the results of the 300-dimensional models all composition functions. The dotted green vertical lines are visual markers for 25%, 50% and 75% of the test data. As mentioned before, a good composed representation generally has the rank $\leqslant$5 (see the horizontal red line labeled 'rank 5'). The visualization of a perfect composition model would be a flat line close to rank 5 for all the examples in the `test` set. This, however, is not the case with any of the tested models. The best performing models, `fulladd`, `matrix` and `fulllex` assign ranks $\leqslant$ 5 to 66.9%, 67.4% and 68.8% of the `test` data, respectively. All the models

[9]Modulo the matrix transformation of `fulllex`, with $d^2 \times 2 + d$ parameters which stands only for a small fraction of the total number of parameters, $d^2 \times |V| + d^2 \times 2 + d$.

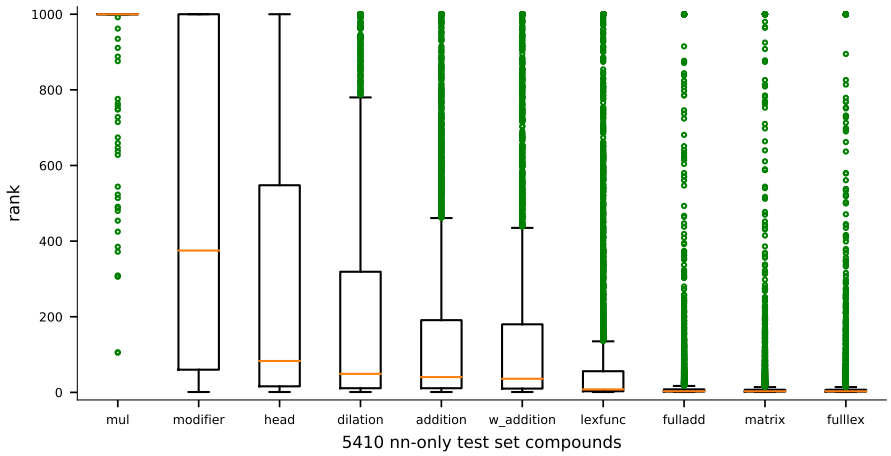


Fig. 5.7 Box plots of the `test` set ranks for the ten composition models (results from Table 5.6). The orange line is the median. The green circles depict the outliers.

produce representations assigned the cut-off rank 1000 - see the flat line at the top of the figure.

Figure 5.8 allows for an easy, visual estimation of the performance of each model: the multiplication (`mul`) is by far the worst performer, with most of the representations assigned rank 1000. The next best model is `modifier`, which shows a quite sizable improvement over `mul`, and is one of the baselines. The stronger `head` baseline is again visibly better than the `modifier` baseline, and the `dilation` model performs a bit better than it. Another improvement is brought about by the `addition` and `w_addition` models, which perform very similar with the weighted addition model obtaining slightly better results. The lexical function model (`lexfunc`) is only able to outperform the baseline and the additive models, despite its large number of parameters. The best results are obtained by the `fulladd`, `matrix` and `fulllex` models, whose performance is very close.

The close performance of these three models, despite the large difference in number of parameters between `fulllex` and the `fulladd`/`matrix` models, invites the following questions: is a single linear/affine transformation, like the one in `fulladd`/`matrix`, all that is needed for composing representations? Why does the additional matrix for every word in the vocabulary bring so little in `fulllex`'s case? Is it the lack of data to train a full matrix for many of the words? Or the increase in the number of parameters?

The next sections introduce two new composition approaches that address the questions posed above. The `mask` models, presented in Section 5.4.4, tailor the composition individually for each word, like `fulllex`, but use less parameters. Section 5.4.5 introduces `multimatrix`, a composition model that tackles both the lack of sufficient data for many examples, as well as the parameter problem of `fulllex`.

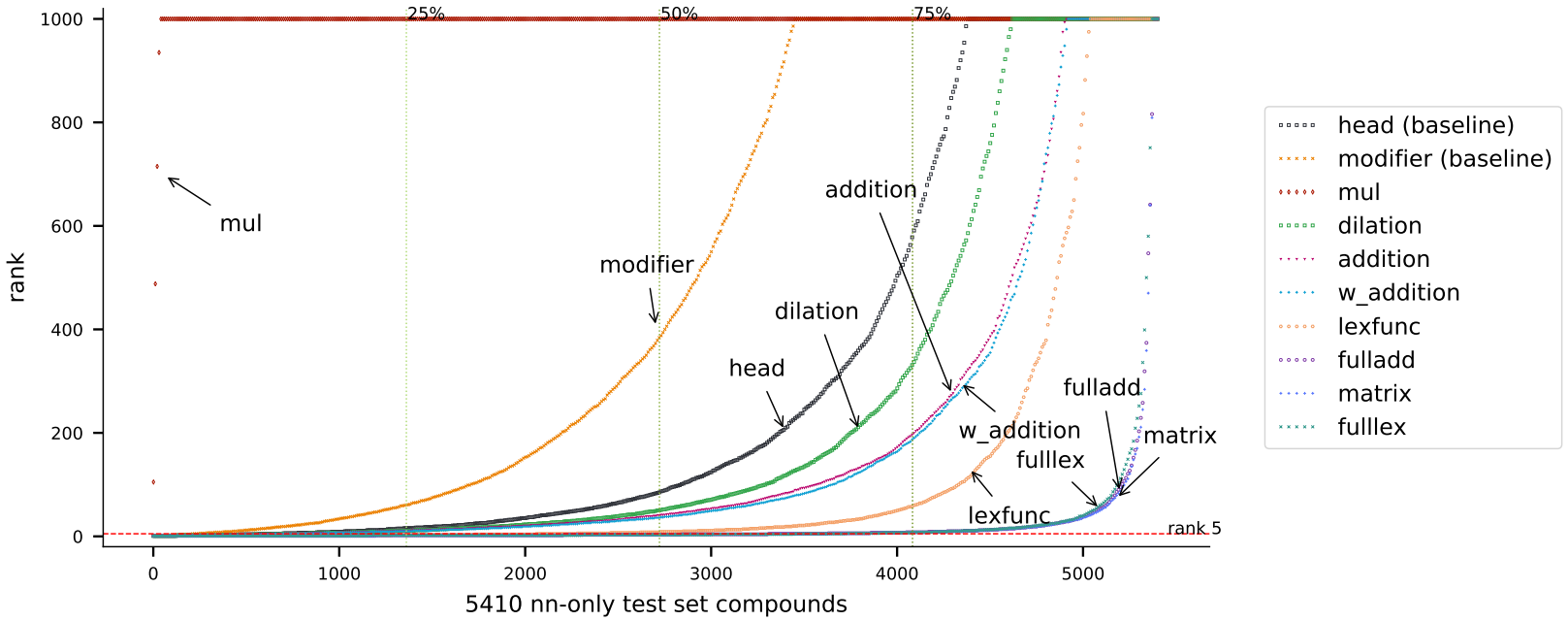


Fig. 5.8 Ordered `test` set ranks for the ten composition models (results from Table 5.6). The dotted green vertical lines are visual markers for 25%, 50% and 75% of the test data. A good composed representation generally has the rank $\leqslant 5$ (see the horizontal red line labeled 'rank 5').

5.4.4 The `mask` models

The mask models are based on the idea that when a word w enters a composition process, there is some variation in its meaning depending on whether it is the first or the second element of the composition. Take, for instance, the compounds *company* <u>*car*</u> and <u>*car*</u> *factory*. In *company* <u>*car*</u>, *car* has its primary denotation, that of a 'vehicle'. In contrast, in the compound <u>*car*</u> *factory*, what matters more is the *car*'s 'product' aspect, the fact that it is an 'artifact produced in a factory'. A good representation of the word *car* should encode both aspects: a car is a 'vehicle' and it is also an 'artifact'. Likewise, a good composition model should be able to select from the constituent representations those aspects that are relevant for a particular composition.

The composition model is given the possibility to accommodate these meaning variations by training, for each word in the dictionary, two **masks**: one for the case when it is the first word in the composition process, and one for when it is the second word.

The **masks** of the word w represented by $\mathbf{u} \in \mathbb{R}^n$, are two vectors $\mathbf{u}', \mathbf{u}'' \in \mathbb{R}^n$. Each time w is the first word in the composition process, it is represented as the element-wise multiplication of the vector $\mathbf{u}$ and its modifier mask $\mathbf{u}'$, $\mathbf{u} \odot \mathbf{u}'$. When w is the second word in the composition, it is represented by the element-wise multiplication of $\mathbf{u}$ and its head mask $\mathbf{u}''$, $\mathbf{u} \odot \mathbf{u}''$, as illustrated in Figure 5.9.

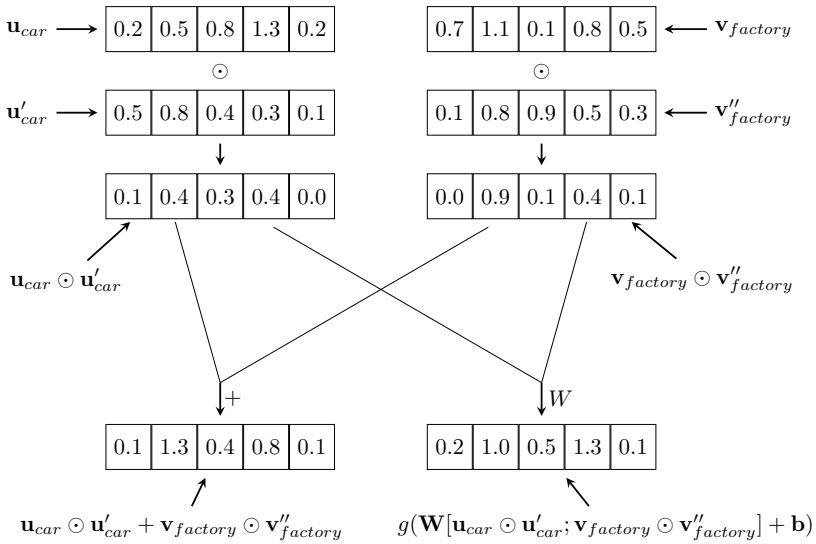


Fig. 5.9 The mask model representation for the compound *car factory*. $\mathbf{u}'$ is the mask used when the word is on the first position (<u>*car*</u> *factory*). $\mathbf{v}''$ is the mask used to modify the representation $\mathbf{v}$ of the word in the second position (*car* <u>*factory*</u>). If the word *car* would instead be in the second position (as in *company* <u>*car*</u>), it would be represented by $\mathbf{u} \odot \mathbf{u}''$. Similarly, if the word *factory* would be in the first position (as in <u>*factory*</u> *car*), it would be represented by $\mathbf{v} \odot \mathbf{v}'$.

The mask vectors are initialized with a vector of all ones, $\mathbf{1} \in \mathbb{R}^n$, and estimated with the help of the training data.[10] It is important to note that the initial vector representations, $\mathbf{u}$ and $\mathbf{v}$, remain fixed during the learning process. The learning process only affects the mask vectors. The composed representation of a compound like *car factory* is obtained by combining the masked representations of the two constituents, $\mathbf{u}_{car} \odot \mathbf{u}'_{car}$ and $\mathbf{v}_{factory} \odot \mathbf{v}''_{factory}$. The way in which the masked representations are combined leads to two variants of the model:

(i) $\mathbf{p} = \mathbf{u} \odot \mathbf{u}' + \mathbf{v} \odot \mathbf{v}''$, called `addmask`, where the masked representations are combined via component-wise addition, and

(ii) $\mathbf{p} = g(\mathbf{W}[\mathbf{u} \odot \mathbf{u}'; \mathbf{v} \odot \mathbf{v}''] + \mathbf{b})$, called `wmask`, where the masked representations are combined via a global matrix $\mathbf{W} \in \mathbb{R}^{n \times 2n}$ and a nonlinearity $g = \tanh$.

Training the mask models entails estimating modifier and head masks for every word in the vocabulary V. The two masks to be learned for every word can be written as two matrices $\mathbf{W}_M, \mathbf{W}_H \in \mathbb{R}^{d \times |\mathcal{V}|}$, where d is the size of the initial word representations. The masks of the word $w_i \in V$ are the i^{th} rows in $\mathbf{W}_M$ and $\mathbf{W}_H$. The masks for all vocabulary words are estimated using lookup tables[11] (Collobert et al., 2011b), which map matrix indices to the corresponding row vectors.

The masked representation of the modifier is obtained by first feeding the index of the word to $LT_{\mathbf{W}_M}$, the modifier lookup table, to obtain the modifier mask, and then multiplying the modifier mask with the initial representation of the modifier. The masked representation of the head is obtained in a similar manner via a lookup operation in $LT_{\mathbf{W}_H}$, the head lookup table. The `addmask` and `wmask` models differ only in the composition method used after the masking process: the masked representations are directly added together in the case of `addmask` and are passed through a composition matrix $\mathbf{W} \in \mathbb{R}^{n \times 2n}$ and a nonlinearity g in the case of `wmask`.

$\mathbf{W}_M$ and $\mathbf{W}_H$ are initialized with all ones and are modified via backpropagation during the training process. The 'all ones' initialization ensures that at the beginning of the training phase the composition function is being fed the initial modifier and head representations. As a consequence, at the beginning of the training process the `addmask` model behaves just like the `addition` model, whereas the `wmask` model behaves initially like the `matrix` model.

In terms of number of parameters, the `mask` models compare favorably to the `fulllex` model, as illustrated by the model sizes listed in Table 5.8. Even though the `mask` models estimate more parameters than the simple `matrix` model, the number of parameters is still linear in the size of the vocabulary. Because the masks are only d-dimensional instead of being $d \times d$ dimensional, there are two orders of magnitude fewer parameters to train than in the case of the `fulllex` model. Also, note that the `fulllex` model trains a single modification matrix for each word, irrespective of its position in the composition, whereas the `mask` models train two separate vector masks, one for each position.

[10] Experiments in initializing the vectors with $\mathbf{1} \in \mathbb{R}^n$ plus Gaussian noise did not show improvements over the simple initialization with ones.

[11] introduced in Chapter 4, see esp. Eq. 4.15.

Composition	Formula	d=50	d=100	d=200	d=300
addmask	$d \times \|V\| \times 2$	713,100	1,426,200	2,852,400	4,278,600
wmask	$d \times \|V\| \times 2 + d \times 2d + d$	718,150	1,446,300	2,932,600	4,458,900
matrix	$d \times 2d + d$	5,050	20,100	80,200	180,300
fulllex	$d^2 \times \|V\| + d \times 2d + d$	17,832,550	71,330,100	285,320,200	641,970,300

Table 5.8 Number of trainable parameters for each model and for each dimensionality of the input word embeddings on the nn-only dataset. Vocabulary size $|V| = 7,131$.

Table 5.9 displays the results of the mask models on the dev and test splits of the nn-only dataset. The wmask model systematically outperforms the addmask model on every input dimension, showing that performing an affine transformation on the masked representations an then applying a non-linearity is preferable to just adding the masked representations together.

	Model	Dro	50d				100d				200d				300d			
			Q_1	Q_2	Q_3	cos-d	Q_1	Q_2	Q_3	cos-d	Q_1	Q_2	Q_3	cos-d	Q_1	Q_2	Q_3	cos-d
dev	addmask	0.5	3	8	43	0.3508	2	5	19	0.4138	2	4	12	0.4724	**2**	**3**	**8**	0.5025
	wmask	0.25	2	5	20	0.3224	2	3	10	0.3867	1	3	6	0.4454	**1**	**2**	**5**	0.4776
test	addmask	0.5	3	7	37	0.3479	2	5	17	0.4107	2	4	10	0.4697	**2**	**3**	**8**	0.5005
	wmask	0.25	2	5	19	0.3213	2	3	10	0.3851	1	3	6	0.4437	**1**	**2**	**5**	0.4763

Table 5.9 Quartiles for the 2655 composed representations of the dev split and the 5410 composed representations of the test split of the nn-only dataset, ranked with respect to the observed representations. Using cosine distance criterion. Lower ranks are better. Best rank is 1. Both models use a learning rate $\eta = 0.1$, dropout 0.5 for addmask and 0.25 for wmask. A tanh nonlinearity is used for wmask.

addmask Table C.11 shows sample neighborhoods of the representations created by the addmask composition model. Rank 1 is achieved for 1243 compounds in the test set, with an average cosine similarity of 0.58511. A rank $\leqslant$5 is obtained for 67.5% of the test data. Comparing to the results in Table 5.6, the addmask model performs on par with the fulladd and the matrix models, but is outperformed by the fulllex model. The addmask model benefits from the addition of a 0.5 dropout layer on the masked representations, as can be seen from the results in Table B.2 in Appendix B.

wmask obtains the best results so far in terms of ranks assigned to the 25%, 50% and 75% percent of the test data: 1 - 2 - 5. It achieves the best rank 1 for 1657 of the test dataset compounds, with an average cosine similarity of 0.58349. 75.7% of the test compounds receive a meaningful representation with a rank $\leqslant$ 5 when using the wmask composition model. Table C.12 lists neighbors of the wmask-composed representations. The use of a 0.25 dropout rate improves the performance of the model. Comparative results using different dropout rates can be seen in Table B.2 in Appendix B.

The `mask` models have the advantage of being computationally cheaper than models like `lexfunc` and `fulllex`. They train only $2n$ parameters for each word in the vocabulary and manage to strike a balance and learn a dedicated representation for each constituent with less parameters. Moreover, the `mask` models are able to use the information extracted from the corpus (as vector representations) for both the head and the modifier. This is in contrast to `lexfunc`, where the modifier representation (a matrix) is estimated solely on the basis of the training data.

The `wmask` model is able to outperform `fulllex` while using only a fraction of the latter's number of parameters. However, although smaller, the number of parameters is still dependent on the number of words in the vocabulary, and can become unmanageable for large vocabularies. The `mask` models do not address `fulllex`'s **curse of dimensionality** problem (Bengio et al., 2003): the masks are still trained for each word individually, and a customized composition is only available for those constituents that occur in the `train` set on their respective positions. For all other words, the composition defaults to the `matrix` model, and does not benefit from the masking process.

5.4.5 The `multimatrix` model

What if instead of training a matrix or mask vector to modify each individual constituent one could leverage the fact that similar words have similar vectors and learn to compose related words the same way? Take for example the compound *Wacholderbeerensaft* 'juniper berries juice'. It is an infrequent compound - it only occurs once in the 10 billion tokens support corpus described in Section 5.3.2. Despite its low frequency, a human has no problem understanding that this compound refers to the juice pressed out of the juniper berries, even when juniper berries are not a typical berry. This insight comes from the numerous other berries that can be pressed into juice that were previously encountered, a pattern that seems to make sense in this situation as well. The capacity to generalize from previously encountered examples to new but similar examples is key to creating meaningful composed representations for the majority of the existing and newly coined compounds.

The `multimatrix` model offers a scalable solution to the problem of providing input-specific compositions for the majority of the compounds in the vocabulary, even if they do not occur in the training data. The idea is to perform multiple affine transformations of the input vectors $\mathbf{u}$ and $\mathbf{v} \in \mathbb{R}^d$, each resulting in an d-dimensional composed vector $\mathbf{p}_k$. k, the number of transformations to be performed, is a hyperparameter of the model. These k affine transformations are parametrized by $\mathbf{W}_i \in \mathbb{R}^{d \times 2d}$ and $\mathbf{b}_i \in \mathbb{R}^d$, $i \in [1, k]$. The resulting composed representations $\mathbf{p}_i$, $i \in [1, k]$ are concatenated and passed through a nonlinearity g. After that another affine transformation, parametrized by $\mathbf{W} \in \mathbb{R}^{d \times kd}$ and $\mathbf{b} \in \mathbb{R}^d$, is applied. The end result is a composed representation of size d. The mathematical formulation of the `multimatrix` composition model is given by Eq. 5.20:

$$\mathbf{p} = \mathbf{W}g([\mathbf{W}_1[\mathbf{u};\mathbf{v}] + \mathbf{b}_1; \ldots; \mathbf{W}_k[\mathbf{u};\mathbf{v}] + \mathbf{b}_k]) + \mathbf{b} \tag{5.20}$$

The first layer of transformations provides the premises for the input representations to be combined in multiple ways. This broadens the spectrum of combinations that the two constituents can be part of. The transformations in the first layer are also responsible for the name of the new model, `multimatrix`. Each of these transformations can be thought of as single a `matrix` model, and the full layer as performing, in parallel, multiple `matrix`-like compositions.

The second layer combines the k transformations obtained in the first layer into a single composed representation. The objective of being near to the corpus-observed representation of the compound can only be fulfilled by picking the most relevant parts from all the available transformations of the inputs.

The remainder of the section provides details about the hyperparameter values and how they were chosen. The results of the different hyperparameter tests for the `multimatrix` model are available in Appendix B, Table B.3.

The nonlinearity between the two layers of transformations plays an important role. Without it, the model is reduced to performing only linear combinations of the input representations. If the nonlinearity is removed from Eq. 5.20 and the equation is rewritten as in Eq. 5.21, one can observe that the `multimatrix` model would become just another `matrix` model, where W' is the matrix obtained after multiplying $\mathbf{W}[\mathbf{W}_1; \ldots; \mathbf{W}_k]$. This is because matrix multiplication is an associative operation, so $\mathbf{W}([\mathbf{W}_1; \ldots; \mathbf{W}_k][u; v])$ is the same as $(\mathbf{W}[\mathbf{W}_1; \ldots; \mathbf{W}_k])[u; v]$.

$$\begin{aligned}\mathbf{p} &= \mathbf{W}([\mathbf{W}_1; \ldots; \mathbf{W}_k][\mathbf{u};\mathbf{v}] + [\mathbf{b}_1; \ldots; \mathbf{b}_k]) + \mathbf{b} \\ &= \mathbf{W}[\mathbf{W}_1; \ldots; \mathbf{W}_k][\mathbf{u};\mathbf{v}] + \mathbf{W}[\mathbf{b}_1; \ldots; \mathbf{b}_k] + \mathbf{b} \\ &= \mathbf{W}'[\mathbf{u};\mathbf{v}] + \mathbf{W}[\mathbf{b}_1; \ldots; \mathbf{b}_k] + \mathbf{b}\end{aligned} \tag{5.21}$$

The best performing nonlinearity on the `nn-only` dataset is the rectified linear unit - `ReLU` (Glorot et al., 2011), introduced in Section 3.1. The form of the function is reiterated in Eq. 5.22.

$$g(x) = \max(0, x) \tag{5.22}$$

Experiments using $g(x) = x$, the identity function, and $g(x) = \tanh(x)$ gave markedly worse results. The model performed relatively well using the parametric PReLU nonlinearity, $g(x) = \max(0, x) + a\min(0, x)$, both when using the same a for all inputs dimension or one a per input dimension.

The model has a fairly large number of parameters: $k(d \times 2d + d)$ for the first layer of transformations and $d \times kd + d$ for the second layer. k values between 40 and 140, in increments of 20, were tested, and the best performance on the `nn-only dev` set was obtained using 120 transformation matrices. This is a drastic reduction in parameter size when compared to `fulllex`, which required a separate matrix for each

word in the vocabulary. Table 5.10 shows the comparative number of parameters for the `multimatrix` model, the two `mask` models, the `matrix` model and the `fulllex` model. As the table shows, for $|V|$ =7131 and $k = 120$ the number of parameters in the `multimatrix` model is just 0.05% of the number of parameters in the `fulllex` model, independent of the input vector size d. This means that even if working with larger vocabularies - that are likely to need more transformation matrices - the `multimatrix` model will remain competitive in terms of parameter requirements. Compared to the `mask` models, `multimatrix` does require a larger parameter set. The increase in parameter size, however, is a reasonable trade-off when taking into account the improvement in generalization capabilities of the `multimatrix` model.

Composition	Formula	d=50	d=100	d=200	d=300
multimatrix	$k(d \times 2d + d) + d \times kd + d$	906,050	3,612,100	14,424,200	32,436,300
addmask	$d \times \|V\| \times 2$	713,100	1,426,200	2,852,400	4,278,600
wmask	$d \times \|V\| \times 2 + d \times 2d + d$	718,150	1,446,300	2,932,600	4,458,900
matrix	$d \times 2d + d$	5,050	20,100	80,200	180,300
fulllex	$d^2 \times \|V\| + d \times 2d + d$	17,832,550	71,330,100	285,320,200	641,970,300

Table 5.10 Number of trainable parameters for each model and for each dimensionality of the input word embeddings on the `nn-only` dataset. Figures for vocabulary size $|V| = 7,131$ and $k = 120$.

The number of parameters was also the motivation for running a number of experiments focused on the effect of dropout. Dropout is applied to the concatenated outputs resulting from the first layer of transformations. With no dropout, the loss of the model on the `dev` set with 50 dimensional inputs is 0.3282. Dropout values from 0 to 0.75 in 0.25 increments were applied, and the best results on the same `dev` set, 0.3102, was obtained using a dropout rate of 0.75. This means that 75% of the weights of the concatenated vector were randomly set to 0 for each training example.

The optimization was performed using AdaGrad, just like in the case of the other models. The learning rates 0.001, 0.01 and 0.1 were tested, with 0.1 providing the best results, which are presented in Table 5.11.

	Model	50d				100d				200d				300d			
		Q_1	Q_2	Q_3	cos-d	Q_1	Q_2	Q_3	cos-d	Q_1	Q_2	Q_3	cos-d	Q_1	Q_2	Q_3	cos-d
dev	multimatrix	2	4	15	0.3102	1	3	7	0.3659	1	2	5	0.4190	1	2	4	0.4488
test	multimatrix	2	4	14	0.3098	1	3	7	0.3654	1	2	4	0.4185	1	2	4	0.4483

Table 5.11 Quartiles for the 2655 composed representations of the `dev` split and the 5410 composed representations of the `test` split of the `nn-only` dataset, ranked with respect to the observed representations. Using cosine distance criterion. Lower ranks are better. Best rank is 1. Using a learning rate $\eta = 0.1$, dropout 0.75, 120 transformation matrices, `ReLU` nonlinearity.

2165 of the 5410 representations in the `test` set are assigned rank 1 when using `multimatrix` composed representations. 81.8% of all the `test` instances receive meaningful representations, assigned a rank ⩽5. Table C.13 shows neighbors of the representations created by the `multimatrix` model for a sample of the `test` compounds.

inst		fulladd	matrix	lexfunc	fulllex	addmask	wmask	multimatrix
145	`test` modifier ∈ `train` as head	24	21	368	34	79	**19**	21
410	`test` modifier ∉ `train`	21	19	390	27	46	14	**11**
146	`test` head ∈ `train` as modifier	24	**22**	98	46	59	27	23
358	`test` head ∉ `train`	44	42	122	44	77	37	**30**

Table 5.12 Analysing the composition performance in difficult cases from the `nn-only` dataset. Showing the mean rank assigned by different composition models to four subsets of the `test` data: test compounds where the modifier occurs in the `train` data only on the head position (first row); test compounds where the modifier does not occur in the `train` data (second row); test compounds where the head occurs in `train` only as modifier (third row); test compounds where the head does not occur in `train` (last row). Models without the global matrix composition (`lexfunc`, `addmask`) perform the worst. Models using only global composition matrices (`fulladd`, `matrix`) have a surprisingly good and stable performance. Models that use global matrices and word-specific transformations (`fulllex` and `wmask`) perform in some scenarios worse than the ones using only global matrices. However, using per-word masks works better for unknown words than when using per word matrices (`wmask` outperforms `fulllex` in all cases). Using shared transformations (`multimatrix`) produces the best average ranks in the majority of cases.

Ideally composition models should produce good representations also in cases where the head or the modifier is not part of the compounds in the training data. This is a difficult setup for models that rely on transformations tailored for each word in the vocabulary, because they will regress to defaults for part of the input.

The performance of the composition models under such circumstances was assessed on four subsets of the `test` data: (i) 145 compounds where the modifier appears in `train` only as a head. Such an example is *Tortenboden* 'cake base', where the modifier *Torte* 'cake' appears in `train` only as the head of the compounds *Erdbeertorte* 'strawberry cake' and *Sachertorte* 'Sacher cake'; (ii) 410 `test` compounds where the modifier is never part of the `train` compounds; (iii, iv) similarly, 146 and 358 `test` compounds where the head appears only in a modifier position and where the head is not a constituent of the `train` compounds, respectively.

Table 5.12 displays the mean rank on these subsets for each of the seven best performing models. A surprising result is the relatively good performance of the `fulladd` and `matrix` models, which use only global composition matrices. These models perform better in these particular evaluation scenarios than the `fulllex` and `addmask` models. The `lexfunc` model is clearly disadvantaged in such low data regimes, particularly when the modifier information is missing. The performance of the `wmask`

compared to the addmask model speaks again for the importance of using a global combination matrix over a simple additive combination of the input representations. wmask outperforming fulllex speaks for a better decoupling between the vector masks and the global composition matrix, leading to a better performance in such cases.

The multimatrix model obtains the best mean ranks when the modifier or head is completely unavailable in the training data. It behaves slightly worse than wmask if the modifier is available as a head (21 vs. 19), showing that positional information captured by wmask can make a difference in such cases. multimatrix ranks a bit under the matrix model (23 vs. 22) when the head is only seen as a modifier during training. The model copes better with unknown modifiers than with unknown heads.

The evaluation up to this point focused exclusively on the performance of the composition models on the very homogeneous nn-only dataset. The next section presents the results of the composition models on the mixed German dataset, where the composition models will have to deal with a more diverse set of possible compound modifiers.

5.4.6 Results on the German mixed composition dataset

As mentioned in Section 5.3.1, the nn-only dataset used in the previous sections is a filtered version of the larger mixed dataset, containing all the nominal compounds in GermaNet release 9.0. In the mixed dataset the part-of-speech restrictions are somewhat relaxed: the head of the compound must be a noun as before, but the modifier can be something other than a noun (e.g. an adjective, a verb, etc. - see full list with examples in Table 5.2). Dima (2015) reports results for the twelve composition models on the mixed dataset, and the interested reader is referred to the paper for more details. The goal of this section is to establish if the more diverse training material has any impact on learning composed representations for noun-noun compounds.

The experiments in this section will use the train and dev splits of the mixed dataset to build composition models and will be tested on the test portion of the nn-only dataset. This is possible because the train, test and dev portions of the mixed dataset are supersets of the train, test and dev portions of the nn-only dataset. This setup makes it possible to directly compare the results in the previous section to the results to be reported here.

The question this section seeks to answer is: which training material leads to an overall better composition model - the clean, noun-noun only data in the nn-only dataset, or the mixed one? Since the nn-only represents 77.86% of the mixed dataset, its results are useful for understanding the impact of training with noisy data. The nine parametric composition models presented in the previous sections were tested in this new setup, using the best performing 300-dimensional input vectors for each model. The non-parametric models are not influenced by the quality or amount of training data, therefore their performance will not change if the training set is changed.

Table 5.13 reports the results obtained by the parametric composition models in this new setup on the right, and the results previously obtained by the models when training

Composition	nn_only 300d				mixed 300d			
	Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d
dilation	11	49	319	0.6679	12	57	375	0.6662
w_addition	10	36	180	0.6509	12	44	229	0.6516
lexfunc	3	8	56	0.5771	3	9	66	0.5767
fulladd	2	3	8	0.5055	2	3	9	0.5044
matrix	2	3	7	0.5034	2	3	9	0.5030
fulllex	2	3	7	0.4982	2	3	8	0.4927
addmask	2	3	8	0.5005	2	3	9	0.4965
wmask	1	2	5	0.4763	1	2	6	0.4730
multimatrix	1	2	4	0.4483	1	2	4	0.4442

Table 5.13 Quartiles for the 5410 composed representations of the `test` split of the `nn-only` dataset, using models trained on the `nn-only` training data (left) and `mixed` dataset training data (right). Using 300-dimensional input representations in both cases. Most models work best when trained on the smaller but more homogeneous training data in the `nn-only` dataset. `multimatrix` is the only model that seems to benefit from the additional data, despite its noisiness.

on the `nn-only` data. The comparison reveals an interesting pattern: all composition models other than `multimatrix` perform better when trained on the smaller but more homogeneous `nn-only` data. Additional data acts as noise when it is not of the same type. The fact that the `mixed` training data contains additional compounds where the modifiers are something else than a noun turns out to have a detrimental effect on the ability of the composition models to produce good composed representations. The effect is larger for models with less parameters (`dilation`, `w_addition`) and smaller for the models with more parameters. The `multimatrix` model is the only one that seems to benefit from the additional training data, obtaining a small improvement in cosine distance when trained on the `mixed` dataset training data.

This investigation pointed out a shortcoming of the current analysis: if a compound is marked by GermaNet as having two modifiers (e.g. *Tanzschuh* 'dance shoe' has both the noun *Tanz* 'dance' and the verb *tanzen* 'to dance'), it is likely that the composition process would benefit from using the information captured by the distributional model in both the nominal and the verbal form. The meaning of the compound is likely to contain aspects from both word forms, and by excluding any of them the composition has less information to work with.

A similar intuition applies for morphologically-derived forms. The semantics of a compound like *Fabrikarbeiter*, 'factory worker', where the head *Arbeiter* 'worker' is derived from the verb *arbeiten* 'to work', depends directly on the semantics of the verb. The modifier fills in an argument of the verb - in this case the location where the activity specified by the verb takes place.

Obtaining better composed representations might hinge on finding ways to provide the automatic composition process with the same information that a person processing a compound has access to. The input of the composition function should not be limited to the modifier and head, but should include, where available, additional information - like verbs or adjectives with the same lexical root as the modifier and the head.

5.5 Composition Models evaluated on English Compounds

The results in this section expand on the results in Dima (2016), which focused on the `fulladd` and `matrix` models, by testing all the thirteen composition models presented in the previous section. The English compounds dataset used here is the same as in Dima (2016). The evaluation methodology, however, is no longer based on the composition-centric approach from Baroni and Zamparelli (2010), but uses the target-centric approach described in Section 5.4.1.

5.5.1 English Compositionality Dataset (`en-comcom`)

The `en`glish `com`positionality dataset containing `com`pounds, code name `en-comcom`, was constructed from two existing compound datasets and a selection of the nominal compounds in the WordNet database. The first existing compound dataset was described in Tratz (2011) and contains 19158 compounds[12]. The second existing compound dataset was proposed in Ó Séaghdha (2008) and contains 1443 compounds[13].

Additional compounds were collected from the WordNet 3.1 database files (Fellbaum, 1998)[14], more specifically from the noun database file `data.noun`. The WordNet compound collection process involved 3 steps: (i) collecting all candidate compounds, i.e. words that contained an underscore or a dash (e.g. *abstract_entity*, *self-service*); (ii) filtering out candidates that included numbers or dots, or had more than 2 constituents; (iii) filtering out candidates where either one of the constituents had a part-of-speech tag that was different from `noun` or `verb`.

The part-of-speech tagging of the candidate compounds was performed using the *spaCy* Python library for natural language processing[15]. The reason for allowing both `noun` and `verb` as accepted part-of-speech tags was that given the extremely limited context available when POS-tagging a compound, the tagger would frequently label as `verb` multi-sense words that were actually nouns in the given context (e.g. *eye drop*, where *drop* was tagged as a verb).

The downside of this automatic POS-tagging step is that some non-compounds also slipped in, e.g. 'do_nothing' or 'co_op'. A visual inspection showed that there were not many cases of this type, but as a result the WordNet data is noisier than the German `nn-only` dataset. As the difference in the results on the German `nn-only` and `mixed` datasets showed, noisy data can have a negative impact on the performance of composition models. However, the unavailability of a large scale hand tagged compounds corpus makes such automatic approaches the only viable solution to gathering enough

[12]The dataset is part of the semantically-enriched parser described in Tratz (2011) which can be obtained from `http://www.isi.edu/publications/licensed-sw/fanseparser/`

[13]Available at `http://www.cl.cam.ac.uk/~do242/Resources/1443_Compounds.tar.gz`

[14]Available at `http://wordnetcode.princeton.edu/wn3.1.dict.tar.gz`

[15]`https://spacy.io/`

data for training composition models. The final compound list extracted from WordNet 3.1 contained 18775 compounds.

The compounds collected from all three resources were combined into one list. The list was de-duplicated and filtered for capitalized compounds (the Tratz (2011) dataset contained a small amount of person names). A final filtering step removed all the compounds where either of the two constituents of the compound or the compound itself did not have a minimum frequency of 100 in the support corpus. As in the case of the German dataset, the frequency filtering step was motivated by the assumption that the compositional process can be better modeled using 'well-learned' word vectors, derived from a reasonable number of contexts.

The final `en-comcom` dataset contains 27220 compounds, formed through the combination of 5335 modifiers and 4761 heads. The set of unique modifiers and heads contains 7646 words, with 2450 being used both as modifiers and as heads. The dictionary for the final dataset contains therefore 34866 unique words. The dataset was partitioned again into `train`, `test` and `dev` splits containing 19054, 5444 and 2722 compounds respectively.

5.5.2 Word Representations for English

The process of creating English word representations for compositionality experiments is similar to the one used for creating German word representations. The support corpus was obtained by concatenating the raw text from the ENCOW14AX corpus (Schäfer, 2015) and the pre-processed 2014 English Wikipedia dump described and made available in Müller and Schütze (2015). A preprocessing step similar to the one described in Müller and Schütze (2015) was applied to the concatenated corpus: the text was lower-cased and the digits were replaces with 0s.

An additional preprocessing step was necessary for creating compound representations. The initial corpus was recoded such that the two-part compounds in the `en-comcom` dataset would be considered a single token. The recoding process involved replacing all the different spelling variants of a compound - written as two separate words, contiguously or with a dash (as in *dress code*, *dresscode* or *dress-code*), as well as their respective plural forms (*dress codes*, *dresscodes*, *dress-codes*) with an artificial underscore-based form (e.g. *dress_code*). However, the plural first constituents were not modified (i.e. *savings account*). Also, the spelling variation which is the result of different spelling standards as in *color scheme* (American English) and *colour scheme* (British English) was left in place. The result was a 9 billion words raw-text corpus with a corresponding vocabulary containing 424,014 words (both simplex words and compounds) with minimum frequency 100 (the full vocabulary had 16M words).

The raw-text corpus was the basis for training word representations in 4 different sizes (50, 100, 200 and 300 dimensions) with the GloVe package (Pennington et al., 2014). Each model was trained for 15 iterations using a 10-word symmetric context for constructing the co-occurrence matrix.

5.5.3 Embedding Normalization

This section reports on the results obtained on the `en-comcom dev` subset using the three normalization setups previously explored for German. The three setups are: `none` - use the vectors space as produced by GloVe, without any normalization; L_2`-row` - normalize each word vector to unit norm; L_2`-col` - normalize each column of the whole vector space to unit norm.

Table 5.14 displays results using input representations of size 50 and the cosine distance as a training criterion. The learning rate used for each model is listed in the column titled η. The best learning rate was chosen for each model individually (either 0.01 or 0.1). The results with different learning rates for each model are available in Appendix B, Table B.4.

Comp. Model	η	none				L_2-row				L_2-col			
		Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d	Q1	Q2	Q3	cos-d
mul	-	1K	1K	1K	0.9343	1K	1K	1K	0.9343	1K	1K	1K	0.9629
modifier	-	237	1K	1K	0.7262	237	1K	1K	0.7262	**159**	**1K**	**1K**	**0.6836**
head	-	276	1K	1K	0.7302	276	1K	1K	0.7302	**168**	**1K**	**1K**	**0.6805**
addition	-	137	1K	1K	0.6888	131	1K	1K	0.6870	**68**	**886**	**1K**	**0.6252**
dilation	0.1	174	1K	1K	0.7023	174	1K	1K	0.7023	**88**	**1K**	**1K**	**0.6462**
w_addition	0.1	136	1K	1K	0.6888	132	1K	1K	0.6870	**67**	**864**	**1K**	**0.6251**
lexfunc	0.1	**5**	**39**	**913**	**0.4634**	**5**	**39**	**913**	**0.4634**	5	40	908	0.4574
fulladd	0.01	**3**	**13**	**93**	**0.3727**	3	13	95	0.3733	3	14	105	0.3787
matrix	0.01	**3**	**11**	**74**	**0.3617**	3	11	75	0.3636	3	12	77	0.3694
fulllex	0.01	2	6	55	0.3330	**2**	**5**	**53**	**0.3337**	2	7	78	0.3488
addmask	0.1	3	12	203	0.3849	3	12	208	0.3859	**3**	**13**	**208**	**0.3834**
wmask	0.1	2	8	64	0.3487	**2**	**6**	**47**	**0.3360**	2	7	51	0.3394
multimatrix	0.1	2	6	46	0.3364	**2**	**6**	**44**	0.3357	2	7	56	0.3459

Table 5.14 Results on the `dev` split of the `en-comcom` dataset, using 50 dimensional embeddings and different normalization variants. Column normalization works best for models with no or few parameters. Models with more parameters (underlined) behave similarly under the three normalization setups, with slightly worse results in the column normalization case.

As in the case of the German experiments, the non-parametric composition functions (`head`, `modifier`, `addition` and `mul`) and the models with few parameters (`w_addition`, `dilation`) work best with the column-normalized embedding space, L_2`-col`, and perform visibly worse with the other normalization variants. The explanation for these results is that these models have no or little ability to influence how the input vectors are transformed into the composed representation. However, they can gain information from the column-wise normalization, which makes it easier to compare the information coming from different vectors on the same input dimension .

For models with a larger number of parameters (underlined in Table 5.14) there is little difference between using the raw vectors space (the `none` setup) and the one using unit vectors (the L_2`-row` setup). Using the column-normalized space L_2-col makes these models have, in general, weaker results.

The experiments reported in the rest of this section will use the L_2-col normalization for the no/low parametric models and L_2-row for the models with more parameters.

5.5.4 Results on the English Compositionality Dataset

Table 5.15 displays the results of the thirteen composition models that were previously evaluated on German. As an overall observation, comparing to the results on the German nn-only dataset in Table 5.6, the majority of models obtain markedly worse ranks on the English dataset. The performance of the model might be affected by several factors: (i) the en-comcom dataset is, as previously mentioned, more noisy because of the automatic filtering step; (ii) en-comcom's size may be an issue: because it is smaller, there is less data available for training the composition models. However, the fact that non-parametric models also have worse results, is an argument in favor of the noisy data explanation.

	Composition	50d				100d				200d				300d			
		Q1	**Q2**	**Q3**	**cos-d**	**Q1**	**Q2**	**Q3**	**cos-d**	**Q1**	**Q2**	**Q3**	**cos-d**	**Q1**	**Q2**	**Q3**	**cos-d**
dev	mul	1K	1K	1K	0.9629	1K	1K	1K	0.9999	1K	1K	1K	1.0024	1K	1K	1K	1.0045
	head	168	1K	1K	0.6805	85	1K	1K	0.7095	62	825	1K	0.7497	**47**	**611**	**1K**	**0.7554**
	modifier	159	1K	1K	0.6836	78	1K	1K	0.7109	52	748.5	1K	0.7446	**37**	**495**	**1K**	**0.7505**
	dilation	88	1K	1K	0.6462	45	604.5	1K	0.6734	29	369.5	1K	0.7144	**22**	**235**	**1K**	**0.7197**
	addition	68	886	1K	0.6252	32	332.5	1K	0.6502	22	177	1K	0.6900	**15**	**102.5**	**1K**	**0.6921**
	w_addition	67	864	1K	0.6251	31	328	1K	0.6501	21	177	1K	0.6900	**15**	**103**	**1K**	**0.6921**
	lexfunc	5	39	915	0.4635	4	24.5	623	0.5162	3	18	474	0.5669	**3**	**15**	**453**	**0.5935**
	addmask	2	10	164	0.3770	2	5	57	0.4217	1	4	27	0.4653	**1**	**3**	**18**	**0.4877**
	fulladd	3	13	95	0.3733	2	6	34	0.4173	2	4	14	0.4581	**2**	**3**	**10**	**0.4786**
	matrix	3	11	75	0.3636	2	6	28	0.4077	2	4	12	0.4488	**1**	**3**	**9**	**0.4699**
	wmask	2	6	47	0.3360	2	4	19	0.3870	1	3	10	0.4347	**1**	**3**	**8**	**0.4617**
	fulllex	2	5	38	0.3228	1	3	16	0.3707	1	2	8	0.4213	**1**	**2**	**7**	**0.4501**
	multimatrix	2	5	31	0.3215	1	3	10	0.3609	**1**	**2**	**5**	**0.4009**	**1**	**2**	**5**	**0.4241**
test	mul	1K	1K	1K	0.9633	1K	1K	1K	0.9972	1K	1K	1K	0.9989	1K	1K	1K	1.0040
	head	167	1K	1K	0.6727	86.5	1K	1K	0.7034	56	674	1K	0.7420	**41**	**450**	**1K**	**0.7479**
	modifier	175	1K	1K	0.6851	88	1K	1K	0.7129	54	738.5	1K	0.7460	**40**	**471**	**1K**	**0.7504**
	dilation	91	1K	1K	0.6400	44	540.5	1K	0.6689	28	294	1K	0.7082	**20**	**187**	**1K**	**0.7133**
	addition	67	789	1K	0.6211	32	322	1K	0.6473	20	169	1K	0.6860	**14**	**100**	**1K**	**0.6873**
	w_addition	68	782	1K	0.6209	31.5	320.5	1K	0.6472	20	169	1K	0.6860	**14**	**100**	**1K**	**0.6872**
	lexfunc	4	35	701	0.4551	3	20	472	0.5098	3	15	322.5	0.5600	**3**	**14**	**301**	**0.5882**
	addmask	2	9	118	0.3730	2	5	46	0.4196	1	4	21	0.4627	**1**	**3**	**15**	**0.4849**
	fullladd	3	13	86.5	0.3714	2	6	30	0.4175	2	4	13	0.4579	**1**	**3**	**9**	**0.4785**
	matrix	3	11	66	0.3632	2	6	24	0.4094	2	3	11	0.4500	**1**	**3**	**8**	**0.4710**
	wmask	2	6	40	0.3342	1	4	17	0.3866	1	3	8	0.4342	**1**	**3**	**7**	**0.4615**
	fulllex	2	5	33	0.3214	1	3	13	0.3692	1	2	8	0.4214	**1**	**2**	**7**	**0.4508**
	multimatrix	2	5	29	0.3220	1	3	11	0.3637	**1**	**2**	**5**	**0.4042**	**1**	**2**	**5**	**0.4269**

Table 5.15 Quartiles for the 2722 composed representations of the dev split and the 5444 composed representations of the test split of the en-comcom dataset, ranked with respect to the observed representations. Best result for each model (across the different input dimensionalities) is marked in bold.

The two baseline models, head and the modifier, obtain modest results. What is striking, however, is that the modifier baseline comfortably outperforms the head

baseline. This suggests that in the case of English compounds there is a majority of cases in which the modifier representation carries more information that the head representation. Tables C.14 and C.15 in Appendix C present nearest neighbors obtained using the representations created by the `head` and the `modifier` models. 60 words obtain rank 1 using the `head` representation, with an average cosine similarity of 0.57150. Although its overall results are better, in the `modifier` representation case only 55 words obtain rank 1, with an average cosine similarity of 0.59314.

The multiplicative model, `mul` is, as in the German case, the worst performing model. Table C.16 presents the nearest neighbors obtained for compounds using the `mul` composed representation. Again, the cosine distance between these representations and the original representations is very low, meaning that the multiplicative representations integrate poorly into the original vector space.

The `addition` and `w_addition` models obtain almost the same results on the `en-comcom` dataset. It follows that a simple `addition` model is, for English compounds, the best alternative for a composition model when training data is not available. Perhaps unsurprisingly, the `addition` model is also the default choice when it comes to representing a larger phrase, or even a sentence or paragraph (Roth and Woodsend, 2014; Hermann and Blunsom, 2014). Tables C.18 and C.19 present the nearest neighbors when using `addition` and `w_addition`. Both models obtain the best rank for 74 `test` compounds, with an average cosine similarity between the observed and the composed representation of 0.5893.

`dilation` obtains, as in the German compounds case, modest results, and only manages to outperform the baseline models. Example neighbors are presented in Table C.17. 81 words are assigned rank 1, with an average cosine similarity between the observed and the composed representation of 0.59594.

Training a matrix for each word leads to a jump in the performance: 612 compounds are assigned rank 1 when using `lexfunc` composed representations, with an average cosine similarity of 0.61611. `fulllex` assigns the best rank to 1881 composed representations, with an average cosine similarity of 0.63913. The benefits of using both the head and the modifier representations and a matrix to combine them can be clearly seen in `fulllex`'s case: `lexfunc test` quartiles for the best model are 3-14-301, while the corresponding quartiles for `fulllex` are 1-2-7. Tables C.20 and C.21 illustrate the nearest neighbors of the representations created by these composition models for English compounds.

`fulladd` and `matrix` obtain, as expected, results similar to each other. They manage to outperform the `lexfunc` model, despite the per-word matrix trained by `lexfunc`. These models show that reusing the same composition matrix for different words can lead to a better generalization performance than when the model learns to compose each modifier using an independent, per-word matrix. 1455 words are assigned the rank 1 using the `fulladd` representations, with an average cosine similarity of 0.60495. With the `matrix` model 1523 words achieve the best rank, with an average cosine of 0.60690. Example neighborhoods for the `fulladd` and `matrix` representations can be seen in Tables C.22 and C.23, respectively.

The mask models are overtaken by several models on the en-comcom dataset. addmask's results are weaker than the ones of the simple fulladd and matrix models. wmask performs better than fulladd/matrix, but is outperformed by fulllex and multimatrix. The average cosine between the 1529 addmask representations assigned the rank 1 and the corresponding original representations is 0.63038. In the case of wmask, the 1724 word vectors assigned the rank 1 have an average cosine similarity of 0.62752 to the observed vectors. Neighbors of the addmask/wmask representations are presented in Tables C.24/C.25.

The best result on the en-comcom dataset is obtained by the multimatrix model: its quartiles on the test subset are 1-2-5. As the quartiles show, over 75% (in fact 78.03%) of the test data receives a good representation with rank $\leqslant 5$ using multimatrix. 2389 out of the 5444 test compounds receive the best rank 1, with an average cosine similarity of 0.6546. Notice that the average cosine similarity value obtained by multimatrix is the largest obtained by any of the models.

inst		fulladd	matrix	lexfunc	fulllex	addmask	wmask	multimatrix
144	test modifier ∈ train as head	59	**44**	622	52	291	62	65
436	test modifier ∉ train	101	**58**	626	86	251	86	67
194	test head ∈ train as modifier	85	**67**	287	83	271	74	83
370	test head ∉ train	143	**76**	335	89	273	119	94

Table 5.16 Analysing the composition performance in difficult cases from the en-comcom dataset. Showing the mean rank assigned by different composition models to four subsets of the test data: test compounds where the modifier occurs in the train data only on the head position (first row); test compounds where the modifier does not occur in the train data (second row); test compounds where the head occurs in train only as modifier (third row); test compounds where the head does not occur in train (last row).

As in the German dataset case, four subsets of challenging data were extracted from the en-comcom test compounds: (i) compounds where the modifier only occurs as a head in train; (ii) compounds whose modifier does not appear in train; (iii) compounds where the head can only be found in the modifier position in the train data and (iv) compounds where the head never appears in the train data (neither as head, nor as modifier).

Table 5.16 displays the average rank of the seven best performing composition models. The ranks are much higher than the average ranks for German on the same type of data. The reason for this increase in average rank may be the noisier dataset, which seems to make it more difficult for all models to operate with less data. The most consistent results are given by the matrix model. lexfunc and addmask, models without a matrix-like component, obtain extremely bad ranks.

There is a subset of compounds for which composition models fail to produce a representation that is close to the original representation. A look at the compounds assigned high ranks by the best performing model, 300-dimensional multimatrix,

reveals several patters: they can be highly technical terms (e.g. plant names like *spring_beauty*, *rose_hip*, medical terms *thyroid_cartilage*, *cross_section*, names of instruments *walkie_talkie*, terms like *head_start*, *test_bed*, *safety_net*), named entities (e.g. *borscht_belt* - the nickname for past summer resorts in the Catskill Mountains, NY; *wonder_woman* and *iron_man*, names of comics superheroes; *half_nelson*, the name of a movie), loan words (e.g. *modus_operandi*, 'mode of operating', from Latin; *foie_gras* 'fat liver', from French) or lexicalized forms such as *whistle_blower* - informer of illegal activities; *mother_tongue* - native language, *silver_lining* - a hopeful stance on a bad situation.

For all of these compounds, the compositional approach fails, and the reasons why vary from one example to the other. For some - like the medical terms - the ability of the composition models might be improved by training on additional compounds from the medical domain. The named entities should ideally be isolated before the composition step. For most of the remaining compounds, however, the best approach might be just to know that they have a specialized meaning, and to store the learned representation directly and do not attempt to obtain it compositionally from the meanings of the constituents. In some cases the surrounding context might be needed to decide if one should use the compositional or the learned representation, e.g. *spring_beauty* might, in particular contexts, refer to something else than the plant.

Finding ways to best represent both compositional and non-compositional compounds with the help of integrative composition models is the objective of the experiments described in the next chapter.

5.6 Comparing to Other Compositionality Studies

Other studies have investigated the possibility of building composed representations, both above and below the word level. They cover, inter alia, the composition of adjective-noun phrases like *vast amount* (Mitchell and Lapata, 2010; Guevara, 2010; Dinu et al., 2013b), of intransitive verbs and their subjects - *skin glows* (Mitchell and Lapata, 2008; Dinu et al., 2013b), of transitive verbs and their direct objects - *buy land* (Mitchell and Lapata, 2010), of determiner phrases like *no home* or *two opponents* (Bernardi et al., 2013; Dinu et al., 2013b) and of noun-noun combinations like *telephone number* (Mitchell and Lapata, 2010). Moreover, a number of studies investigated the use of composition models for modeling several morphological derivation patterns in English, e.g. *re-* + *build* → *rebuild*, from Lazaridou et al. (2013)) and German (e.g. *taub* 'deaf' + *-heit* → *Taubheit* 'deafness', from Padó et al. (2016).

An important factor to consider in all these studies is how the composition models are evaluated. As mentioned before in the discussion of the results of the multiplicative model in Section 5.4.3, studies like Mitchell and Lapata (2008, 2010); Blacoe and Lapata (2012) compare two composed representations using cosine similarity and then correlate the cosine similarity with human similarity judgments. Although this type of evaluation can be revealing from a cognitive perspective, the use of human judgments

limits its potential because of the high cost of obtaining the manual annotations. This evaluation will judge a composition model to work well even if the resulting composed representations are incomparable with the original representations of the constituents.

Another popular evaluation metric is comparing the composed representation of a word combination with the representation of a single word with a similar sense. Evaluations of this type compare for example *close interaction* to *contact* (Zanzotto et al., 2010) or *heavy particle* to *baryon* (Dinu et al., 2013b). However, this setup makes it difficult to figure out when the composition model fails because of its own inability to model composition, and when does it fail because of the approximate semantic match between the word combination and the single-word target.

The evaluation methodology used in this thesis, a modified version of the evaluation originally proposed by Guevara (2010) and Baroni and Zamparelli (2010), builds the same type of distributional representations for the constituents of a phrase and for the phrase itself. The quality of the composition is then judged by comparing the original representation of the phrase to the composed representation. This type of setup has the advantage that new training material can easily be created, without the need for expensive human annotations. Moreover, this type of evaluation inherently leads to composed representations that are compatible with the original representations of the constituent words, while avoiding the possible ambiguities related to the approximate matching between a word combination and its single word correspondent.

In the experiments on adjective-noun (AN) composition in Guevara (2010), the `fulladd` model is found to be superior to component-wise addition and multiplication. Guevara (2010) reports that the composed representations build by the `fulladd` model have the observed representation in its top-10 list of neighbors in 57.6% of the test cases. Noteworthy in Guevara (2010)'s evaluation is an additional setup where the composition is evaluated not only with respect to the target representation itself, but also with respect to its 10 nearest neighbors Such a training/evaluation setup would promote even further the idea that the compositionally build representation should 'fit' the neighborhood of the observed representation. Investigating the benefits of this alternative training/evaluation setup is, however, left for future work.

Baroni and Zamparelli (2010) use a rank-based evaluation to judge the performance of the `lexfunc` model, while also comparing to the `additive` and `mul` models. The study uses a large adjective-noun English dataset. However, the dataset contains only 36 target adjectives to be modeled, albeit with a large number of adjective-noun combinations for each individual adjective (734 combinations on average). The composition model is trained using a leave-one-out policy, where the composed vector of a particular combination is estimated using as training all the vectors of the other combinations with the same adjective. In their study `lexfunc` obtains better results than the `additive` and `mul` models, obtaining the quartiles 17-170-$\geqslant$1K.

5.7 Compositional Models - Main Results

This chapter presented an empirical evaluation of several composition models for the task of creating word representations for noun-noun compounds, starting from the representations of their constituents. Another dimension of the study concerned the cross-lingual investigation of how composition models work for the same type of construction in two different languages, German and English. This section summarizes the main results of the chapter.

A main take-away point is that parametric composition models consistently outperform non-parametric ones. This result is a strong indicator that whenever training data is available it is better to train a dedicated, parametric composition model than to use, for example, a simple `addition` model.

Models with no or few parameters (`head`, `modifier`, `addition`, `mul`, `w_addition`, `dilation`) perform better when using L_2 column normalization. Using no normalization or L_2 row normalization works better with models that have a large number of parameters (`lexfunc`, `fulladd`, `matrix`, `fulllex`,`addmask`, `wmask`, `multimatrix`) - see Sections 5.4.2 and 5.5.3 for details.

The two baseline models, `head` and `modifier`, have a mediocre performance. The `head` model clearly outperforms the `modifier` model on the German datasets, but is outperformed by the `modifier` model on the English dataset (details in Section 5.5.4).

`addition` outperforms the `head` and `modifier` baselines; however, its main disadvantage is that it does not take into account the position of the constituents - compounds like *car factory* and *factory car* will receive the same representation using the `addition` model. `mul` is the worse composition model, its composed representations are incompatible with the original vector space both for the German and for the English compounds (Sections 5.4.3 and 5.5.4).

The experiments in Section 5.4.6 showed that the cleaner `nn-only` training set, containing only noun-noun compounds, leads to better results than using a larger but more diverse `mixed` dataset, where the modifiers could also be verbs, adjectives, etc. It follows that when building a composition dataset it is advisable to isolate a particular construction and build composition models that target that construction in particular.

The newly introduced mask models, `addmask` and `wmask`, obtain competitive results both the German and the English datasets. This confirms the intuition - already presented for the `lexfunc` and the `fulllex` models - that training dedicated components for each word can lead to better composed representations. Training two vectors for each modifier and head instead of two matrices makes the `mask` models use only a fraction of the corresponding `lex` models (`lexfunc`/`fulllex`) - see Section 5.4.4.

Both the `mask` and the `lex` models use a modeling idea that is potentially detrimental for composition: the individual treatment of constituents (building separate matrices or masks for each) might lead to a loss in generalization behavior. In contrast, the `fulladd`/`matrix` models are at the opposite end of the spectrum, with a one-size-fits-all approach where a single matrix is expected to account for all the possible composition typologies. Ideally, composition models should aim for a middle point, where the

composition would abstract away from the individual word level but specialize for frequently encountered semantic composition patterns.

The proposed `multimatrix` model satisfies exactly these requirements, and leads to a significant improvement over the other models for both the German and the English dataset. It uses a set of affine transformations to compose the two inputs in different ways, which are then integrated into a single composed vector via a nonlinearity and another affine transformation.

Chapter 6

The Challenge of Non-Compositional Compounds

Dass Zitronenfalter keine Zitronen falten, Kammerjäger keine Kammern jagen, Landstreicher keine Land streichen, Briefschlitze keine Briefe schlitzen und Heckenschützen keine Hecke schützen wissen wir natürlich - aber woher eigentlich?

- Verena Klos, *Komposition und Kompositionalität*

6.1 Identifying Non-Compositional Compounds

When the process of creating the meaning of the compound involves combining (some parts of) the meanings of its modifier and head, the newly coined complex word is considered to be compositional. In such cases the meaning of the constituents contributes directly to the meaning of the compound. Lexicalization, on the other hand, describes *non-compositional* cases where the meaning of one of the constituents *deviates* from the meaning they are usually associated with. In fully lexicalized cases the meaning of the compound simply does not match the compositional meaning that could be derived from the meaning of the constituent words.

Take for example the compound *Löwenzahn* 'dandelion', lit. 'lion's tooth': it is hopeless to start from the words *Löwe* 'lion' and *Zahn* 'tooth' and seek to derive the meaning of 'dandelion'. Only after one learns the correct referent, i.e. sees the flower, it is possible to grasp the metaphor behind the creative process which resulted in this particular name. In any case, the correct referent of *Löwenzahn* has to be stored in an individual's mental lexicon in order to correctly identify its meaning next time it is encountered, since the compositional process will not be of any help.

Similarly, meaning deviations like the one in the example above make it impossible for composition models to produce the correct vector representation of *Löwenzahn* starting from the representations of *Löwe* and *Zahn*. In such cases the correct repre-

sentation can only be learned from the contexts in which the compound *Löwenzahn* appears in the support corpus.

Bauer (1983:45) describes a continuum of types of complex words, arranged with respect to their formation status and to how dependent their interpretation is on the context: (i) **nonce formations**, *"coined by a speaker/writer on the spur of the moment to cover some immediate need"*, where there is a large ambiguity with respect to the meaning of the compound which has to be resolved using the immediate context (e.g. *Algenwein* 'algae wine', a new word described by Lemnitzer (2007) to mean *alcoholic beverage made of fermented algae*); (ii) **institutionalized lexemes**, whose potential ambiguity is canceled by the frequency of use and familiarity with the term, and whose more established meaning could be inferred based on the meanings of the constituents and prior world experience, without the need for an immediate context (e.g. *orange juice*); (iii) **lexicalized lexemes**, where the meaning has become a convention which cannot be inferred from the constituents alone and can only be successfully interpreted if the term is familiar or if the context provides enough clues (e.g. *couch potato*[1]).

A person learning the German language can relate to these three stages. Upon the first encounter with the word *Löwenzahn*, if one knows that the constituents *Löwe* and *Zahn* mean 'lion' and 'tooth' one can think of multiple interpretations as in stage 1: 'tooth from a lion', 'tooth as sharp/strong/large as a lion's', 'person with particularly sharp/strong teeth' (similar to 'hawk eye'). However, as soon as the learner becomes aware of the real referent, she speeds though the institutionalized to the lexicalized stage, and the idiosyncratic meaning of the word has to be stored for future use. Failure to record compounds that significantly deviate from their 'expected' interpretations will hamper further progress in understanding the German language.

Non-compositional compounds can have a derailing effect for the modeling of compositional compounds. For composition models like the ones presented in the previous chapter, it might be desirable to filter out compounds that exhibit large meaning drifts. This is because training on lexicalized compounds will result in patterns that do not generalize to other compounds. Moreover, composition models cannot be expected to learn how to construct a meaning representation that significantly deviates from the expected compositional representation.

This chapter takes an in-depth look at the performance of composition models on lexicalized compounds. Section 6.1.1 introduces a small, manually annotated lexicalization dataset which is then used in Section 6.1.2 to analyze the results of the composition models. Section 6.2 connects the investigations in this chapter to similar investigations in the literature.

6.1.1 `de-ulex` Dataset of German Compounds

A subset of the annotation scheme introduced in Section 2.2, pp. 42 focuses on lexicalization. 648 (approx. 8%) of the `de-nncom-sem` dataset to be introduced in

[1] a *couch potato* is not a potato, but a person who exercises little and spends most of the time in front of a TV.

Section 7.1.1 are annotated as being partially or completely lexicalized. The annotation guidelines regarding lexicalization specify that the compounds can be annotated with one or more of the labels `lex_M`, `lex_H`, `lex_MS`, `lex_HS` and `lex_R`.

The label `lex_M` is used when the modifier deviates from its regular meaning (e.g. *Pfefferkuchen* 'ginger bread', lit. 'black pepper cake'); similarly `lex_H` is used when the head deviates from its regular meaning (e.g. *Notnagel* 'person who can fill in for others in emergency case', lit. 'emergency nail'). In these examples one constituent has an opaque meaning, while the other maintains its conventional meaning. However, in cases like *Eselsbrücke* 'mnemonic', lit. 'donkey bridge' neither constituent contributes to the meaning of the new compound. Moreover, the head and the modifier are in an atypical relation. Such examples are marked with three lexicalization labels: `lex_M`, `lex_H` and `lex_R`, to mark the fact that the modifier, the head and the relation they are in have deviated from typical use.

`lex_MS` and `lex_HS` are used to label compounds where the referent of the modifier or the head physically resembles the typical referent: e.g. *Ohrensessel* 'wing chair', lit. 'ears chair', where the top part of the chair loosely resembles a pair of ears or *Schokoladenhase* 'chocolate bunny' where the chocolate is shaped to look like a bunny.

As mentioned before, 648 compounds from the annotated dataset to be introduced in Section 7.1.1 are annotated with at least one of the five lexicalization labels. These compounds were used to create a dataset for bootstrapping the automatic identification of lexicalized German compounds.

A balanced dataset was constructed from the 648 lexicalized compounds plus another 648 compounds that were not marked in the dataset as being lexicalized. The goal was to have a sample of data containing both compositional and non-compositional compounds, and learn how to tell them apart. From the 1296 compounds only 1050 had the head, modifier and compound itself occur with a frequency above 100 in the support corpus used to create the word representations described in Chapter 5.3.2. These 1050 compounds form the `de-ulex` dataset, which comprises of 525 compositional and 525 non-compositional compounds. The `de-ulex` dataset will be used in the next section to discover criteria for distinguishing when a compound is compositional and when it is lexicalized.

6.1.2 More, or Less Compositional?

Even for the best of the composition models presented in the previous chapter there were `test` compounds for which the composed representation was far away from when the corpus-learned representation. A look at the four example compounds from Table C.13 in Appendix C, whose `multimatrix` representations are assigned the worst rank possible, 1000, shows that these examples have something in common. *Nachtschatten* 'solanum', lit. 'night shadow' is a botanical genus, whose best known members are the potato, the tomato and the eggplant. *Besenreiser* 'spider veins', lit. 'broom twigs' is a medical condition affecting the veins. *Wertschätzung*, meaning

'esteem' or 'appreciation' is an abstract concept with the literal meaning of 'value estimate'. And *Tierkreis* 'zodiac' is literally translated as 'animal circle'.

What these compounds have in common is that their actual meaning cannot be derived from the meaning of their constituents. The compositional representation produced by the composition models does not correspond to the actual, lexicalized meaning. Table 6.1 illustrates this point. It shows the neighbors of the original and the `multimatrix`-composed representations of the just mentioned compounds. The neighbors are taken from the full vector space of word representations, containing ∼1 million words (described in Section 5.3.2). In all cases the neighborhood of the original compound representation captures the actual meaning of the word, whereas the composed representation is in the 'garbage bin' of cosine similarity, with most composed representations being nearly orthogonal to their observed counterparts (orthogonal vectors have a cosine similarity of 0).

nachtschatten:1000	**besenreiser:1000**	**wertschätzung:1000**	**tierkreis:1000**
nachtschatten 1.00000	besenreiser 1.00000	wertschätzung 1.00000	tierkreis 1.00000
lotos 0.36870	varizen 0.66002	akzeptanz 0.67903	tierkreiszeichen 0.62106
schwarzer 0.36613	besenreisern 0.61733	anerkennung 0.64735	siderischen 0.49127
walfisch 0.35867	krampfadern 0.60190	zuneigung 0.64412	zodiak 0.48716
tollkirsche 0.34875	pigmentstörungen 0.57887	empathie 0.59704	tierkreises 0.46585
herbstzeitlose 0.34442	couperose 0.56271	entgegengebracht 0.58749	ekliptik 0.46328
klabauter 0.33420	aknenarben 0.54388	entgegenbringen 0.58140	aszendent 0.45524
weisser 0.33350	dehnungsstreifen 0.52476	respekt 0.58109	sternbilder 0.44575
täuber 0.32934	schwangerschaftsstreifen 0.52181	sympathie 0.58090	frühlingspunkt 0.43581
schluckspecht 0.32924	besenreißer 0.51405	dankbarkeit 0.58040	astrologischen 0.42823
stechapfel 0.32594	äderchen 0.50821	zuwendung 0.57655	aszendenten 0.42551
...	...	...	...
lohndumping 0.12018	höflichere 0.08151	rwert 0.09490	karpaltunnelsyndroms 0.11724
strandpiraten 0.12018	gke 0.08151	freistellung 0.09489	geretteten 0.11724
prüß 0.12018	muskelmänner 0.08151	waschergebnis 0.09489	ferias 0.11724
weinthal 0.12018	geldkoffer 0.08151	geschichtsbewusste 0.09489	bagatelldelikte 0.11723
weimeraner 0.12018	regenbogenpresse 0.08151	mittelbarkeit 0.09489	wirtschaftskrimineller 0.11723
nachtschatten_c 0.12018	besenreiser_c 0.08151	wertschätzung_c 0.09489	tierkreis_c 0.11723
offenau 0.12018	eröffnungsposting 0.08151	pflegealltag 0.09489	peloponnes 0.11723
zivilrechtlich 0.12018	ethereal 0.08151	frauenbewegungen 0.09489	panzerkolonnen 0.11723
dragees 0.12017	aufgeschwollen 0.08151	schweiz 0.09489	drangehalten 0.11723
ungef 0.12017	dirndln 0.08151	musikszene 0.09489	tcg 0.11723
fiorella 0.12017	silur 0.08151	abraumhalde 0.09489	vorausbuchungen 0.11723

Table 6.1 Neighbours of a sample of lexicalized compounds from the `nn-only test` set where the composed representation was assigned a cut-off rank. Neighbours are arranged in decreasing cosine similarity order to the original representation. The composed representation is marked in each case with an _*c*. Both the original and the composed representations are highlighted in red. The composed representation of such lexicalized compounds is commonly very far away from the original representation, as the cosine similarities show. The composed representations were obtained using the best performing composition model, the 300-dimensional `multimatrix` model.

As these examples show, a large dissimilarity between the original and the composed representation can mean that the compound in question is partially or completely lexicalized. The hypothesis that a compound's degree of lexicalization can be judged

using a good quality composition model was put to the test using the compounds from the `de-ulex` set.

As mentioned above, the `de-ulex` set contains 525 compounds that were marked with different types of lexicalization, and an equal amount of compounds that were not marked as being lexicalized. A first step in testing the hypothesis formulated above was to compute the cosine distance between the original, corpus-observed representation of each compound and its corresponding composed representation, using the best performing composition model from the previous chapter - the 300-dimensional `multimatrix` model.

The average of the cosine similarity between the original and the composed representation was computed separately for the group of compounds annotated as lexicalized and for the ones that were compositional according to the annotation. The average of the compositional compounds was 0.6338. Surprisingly, the average of the lexicalized ones was also relatively high - 0.5566. Even for highly lexicalized compounds like *Löwenzahn* 'dandelion', lit. 'lion tooth' there was little difference between the composed and the original representation - 0.7456.

Why? Because *Löwenzahn* was part of the data the composition model was trained on, and the model did its best and learned the idiosyncratic meaning of the lexicalized compounds. 495 of the 1050 compounds in the `de-ulex` dataset, both lexicalized and compositional, were part of the `nn-only` `train` portion. To avoid the situation in which the model learns any of the vectors, the `de-ulex` compounds were filtered out of the `nn-only` `train` set. Given that the original `train` set had 18796 compounds, removing the 495 overlapping compounds did not affect the performance of the composition model on the `test` set (the quartiles remained 1-2-4).

What did happen, however, is that when using the retrained composition model there was a visible decrease in the average cosine similarity of the `de-ulex` compounds which were annotated as lexicalized: from 0.5566 down to 0.3615. The average of the compositional vectors also decreased to 0.5191. The cosine similarity between the corpus-based and the composed representation of *Löwenzahn* was also much lower: 0.1579, because the compound was not seen during training.

Figure 6.1 compares the lexicalization labels assigned via the manual annotation to the empirical score assigned by the automatic analysis. The score, shown on the X-axis of Figure 6.1, is the cosine similarity between the original and the composed representation for all the 1050 compounds in the `de-ulex` dataset. The large blue circles mark the compounds that were annotated as lexicalized (i.e. *Milchstraße*, 'Milky Way', lit. 'milk street'), while the small orange circles mark the compositional ones (*Sporthalle* 'sports hall', 'gym').

The green color marks some of the examples where the annotation and the cosine similarity agree, while red signals examples where the annotation and the similarity disagree. The agreement can arise in two situations: either when compounds annotated as lexicalized obtain a low cosine similarity value (the green examples on the left of Figure 6.1, e.g. *Milchstraße*), or when compounds annotated as compositional obtain a high cosine similarity (the green examples on the right of the figure, e.g. *Sporthalle*).

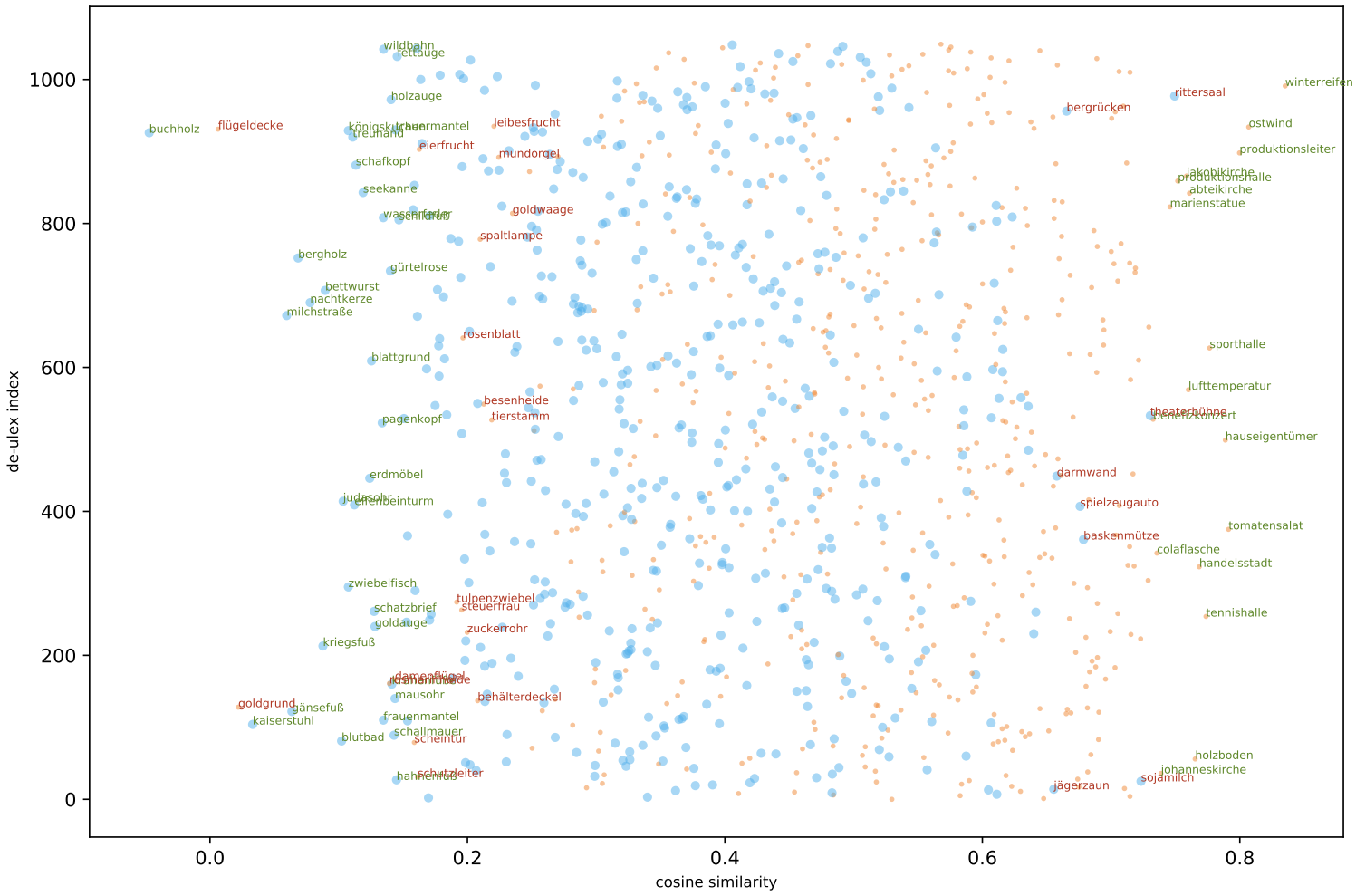


Fig. 6.1 Continuum between lexicalization and compositionality. On the X-axis, the cosine similarity between the original and the `multimatrix`-composed representations of the compounds in the `de-ulex` dataset. The large blue dots mark compounds annotated as lexicalized. Small orange dots mark compounds considered to be compositional. Green labels mark some example compounds where the cosine similarity and the annotation seem to agree, while red labels mark examples where the cosine similarity and the annotation seem to disagree.

Similarly, disagreements can also occur in two circumstances: (i) for compounds with a low cosine similarity that are annotated as compositional (red examples on the left side, e.g. *Flügeldecke* 'elytron', lit. 'wing blanket'); (ii) for compounds with a high cosine similarity that are annotated as lexicalized (red examples on the right side, e.g. *Sojamilch* 'soy milk').

The figure is a visual representation of the fact that lexicalization is a continuum and that there are no sharp boundaries that clearly separate the lexicalized compounds from the compositional ones. There are, however, trends - i.e. many of the compounds annotated as lexicalized are on the left side of the figure, with a cosine similarity between 0.1 and 0.4 (, *Seekanne* 'fringed water lily', lit. 'lake pot'). This region contains, however, also compounds that had *not* been marked as lexicalized, i.e. *Eierfrucht* 'aubergine', lit. 'eggs fruit', or *Zuckerrohr* 'sugarcane', lit. 'sugar tube'.

A similar situation occurs on the right end of Figure 6.1: compounds like *Tomatosalat* 'tomato salad' and *Colaflasche* 'cola bottle' have a high cosine similarity, which corresponds to their annotation as non-lexicalized compounds. From a distributional perspective, however, compounds like *Sojamilch* 'soy milk' and *Rittersaal* 'knight's hall' seem to be closer to being compositional, even if the 'soy milk' is not 'milk' in the sense of 'cow milk', and the 'knight's hall' of many castles has in the meantime a much wider user pool (not necessarily knights).

Other interesting examples are compounds which could be interpreted either compositionally or non-compositionally, depending on the context they are used in. For example *Buchholz* can be interpreted as 'beech wood' in the compositional case, but is also a frequent family or town name. Similarly *Rosenblatt* can either mean the 'leaf or petal of a rose' in the compositional case, but is again a frequent family name. As Fig. 6.1 shows, in both cases the distributional representation captures the family name meaning, leading to a large discrepancy between the corpus-learned and the composed representation. Also, the annotation did indicate *Buchholz* as a possible name, but *Rosenblatt* was annotated as being compositional. Given the use of large, unannotated corpora for the creation of word representations, it is difficult to predict when such a collision might happen, and when will the vector no longer correspond to the compound but to the proper name.

Such compounds - whose meaning strongly deviates from the compositional meaning - can have an undesired effect on the overall efficiency of the composition model. At the beginning of this section the composition model showed that it had the ability to learn quite well the idiosyncratic representation of *Löwenzahn*. If many such idiosyncratic compounds - either lexicalized like *Löwenzahn* or doubling as a name like *Buchholz* - are used as training material, will the composition models have a harder time modeling the compositional examples? The following experiment aims to provide an answer to the question: are composition models affected by the presence of lexicalized compounds in the training data or not?

In order to find an answer, the best composition model was trained using portions comprising 20%, 40%, 60%, 80% and 100% of the `train` dataset. Three setups are then tested. In `setup 1` the `train` data is randomized and then split; in `setup 2` the

`train` data is sorted from most compositional to least compositional and then split; in `setup 3` the `train` data is sorted from least compositional to most compositional and then split. Composition models are expected to generalize faster when trained first on compositional compounds (`setup 2`) than when trained first on non-compositional examples (`setup 3`) or on randomly chosen examples (`setup 1`). This is because training on compositional compounds leads to learning patterns that can be reused to compose similar examples. In contrast, training on non-compositional examples - like *Löwenzahn* - results in patterns that can only be used for composing that particular word.

The manually annotated compounds are only a small fraction of the GermaNet compounds in the `nn-only` dataset used for the composition experiments, and manually annotating each compound with a 'compositionality score' would be a laborious, time intensive undertaking.

Therefore, an automatic method was devised to arrange the `train` compounds based on their (automatically-perceived) compositionality. The full `nn-only` dataset, comprising 26861 compounds, was split into 10 equal folds of data. Ten separate `multimatrix` composition models were trained, each using nine of the folds as training material and the tenth fold as the dev/test data. All models were trained using the hyperparameters that worked best for `multimatrix` in the previous chapter: $\eta = 0.1$, 0.75 dropout, 120 transformation matrices and a `ReLU` nonlinearity.

The cosine similarity between the composed and the original representation was computed for each compound *when it was part of the* `test` *fold*. In this way each compound gets to be tested without it being part of the training data, making it possible to compare the original and the composed representations without the interference of overfitting. The concatenated list of all the cosine similarities computed for all the 10 test folds serves as an approximation of how compositional each compound is: the higher the cosine similarity - the more compositional the compound and vice-versa.

The original `train` subset of the `nn-only` dataset was then re-sorted according to the cosine similarities, and 5 `train` splits were created, each containing an increasing percentage of the data: 20%, 40%, 60%, 80% and 100%. In `setup 1` each training example is randomly assigned to a data split. In `setup 2` the first subset contains the `train` compounds identified as most compositional, whereas in `setup 3` the first subset contains the least compositional `train` compounds.

Table 6.2 shows the quartiles, the average cosine distance on the `nn-only test` set and the number of compounds in the `test` set that received a rank $\leqslant$5. The models in `setup 2`, using the `train` data sorted from most compositional to least compositional, obtain a smaller (i.e. better) average cosine distance and rank consistently more than 100 compounds with ranks $\leqslant$5 than the ones in `setup 1` (which uses randomized training data). The difference between `setup 1` and `setup 2` evens out when the last 20% of data is added. However, in `setup 2` the last 20% of data are the compounds where the cosine similarity between the original and the composed representations is less than 0.4560, so on the non-compositional side of the spectrum. These last 20% of training data bring no improvement in terms of cosine distance (compare 0.4479

#	%train	setup1: random					setup 2: decr. cosine					setup 3: incr. cosine				
		Q_1	Q_2	Q_3	cos-d	$\leqslant 5$	Q_1	Q_2	Q_3	cos-d	$\leqslant 5$	Q_1	Q_2	Q_3	cos-d	$\leqslant 5$
3760	20%	2	4	12	0.5305	3098	2	4	14	0.5149	3261	6	15	38	0.6080	1348
7520	40%	2	3	7	0.4939	3804	1	2	6	0.4753	3959	2	5	13	0.5386	2851
11280	60%	1	2	5	0.4736	4120	1	2	4	0.4564	4231	2	3	7	0.5012	3666
15040	80%	1	2	4	0.4589	4281	1	2	4	0.4479	4389	1	2	5	0.4733	4154
18796	100%	1	2	4	0.4483	4428	1	2	4	0.4482	4430	1	2	4	0.4480	4427

Table 6.2 Three different training setups: `setup 1` - `train` data chosen randomly; `setup 2` - `train` data chosen from most to least compositional; `setup 3` - `train` data chosen from least to most compositional. Non-compositional compounds supply less information than the compositional ones. However, as long as the majority of the data is compositional, there is no downside to also having less compositional instances in the training data.

to 0.4482), and only 41 additional compounds are assigned a rank $\leqslant$5 (from 4389 to 4430). This is not the case for the models in `setup 1`, using randomly sampled data - there the improvements are more evenly spread out.

`setup 3` shows dramatic differences to the previous two setups in terms of the quality of the models trained on 20%, 40% and 60% of the data. In this case, the first 20% of training data contains compounds with a cosine similarity between -0.0366 and 0.4561, while the next two 20% parts bring the maximum cosine similarity to 0.5451 and 0.6118, respectively.

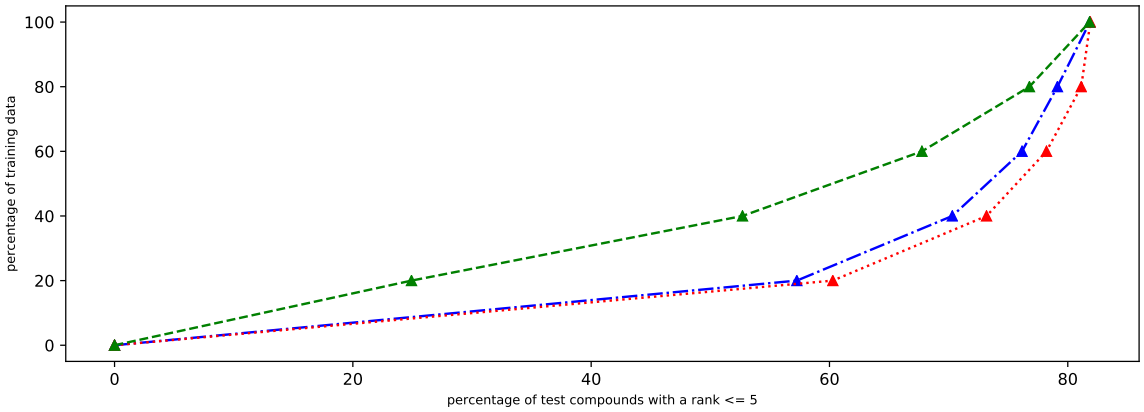


Fig. 6.2 Percentage of `test` compounds with rank $\leqslant$5 (X-axis) against the percentage of `train` data (Y-axis). The dashed green line plots the performance of the model trained on cosine-sorted data (from least compositional to most compositional). The dash-dotted blue line plots the performance of the models trained on randomly sampled data. The dotted red line plots the performance of the models using the cosine-sorted training data (from most to least compositional).

Figure 6.2 shows the percentage of `test` compounds that received a rank $\leqslant$5 (X-axis) against the percentage of `train` data that was used (Y-axis). The dashed green

line shows the performance of the models in `setup 3`, using the lexicalized training data first. The dash-dotted blue line plots the performance of the models trained on randomly sampled `train` data (`setup 1`). The dotted red line plots the performance of the models in `setup 2`, using the training data sorted from most to least compositional. Although the models in `setup 3` do learn from the first 20% of data, their performance is nowhere near the models in the other two training setups. This shows that while the composition models manage to learn even from the less compositional examples, they do not obtain as much information as from more compositional instances. As can be seen in Figure 6.2, models trained on the 40% least compositional training examples perform worse than the ones trained on 20% of the most compositional instances.

This analysis shows, in `setup 2`, that while training on the last 20% of data - containing the lexicalized compounds - does not have a negative impact, composition models do not learn much from them. `setup 3` shows that using a high percentage of lexicalized examples as training data has a negative impact on the performance of the composition models.

It should be noted, however, that it would nevertheless be advisable to filter out the lexicalized compounds from the training data, as shown above. Consider for example compounds like the compound *Frauenschuh*, a type of orchid, lit. 'women's shoe' being used and interpreted in its compositional sense in the context of buying shoes, but being used in its non-compositional sense at a purchase in a florist shop. Such ambiguities cannot be solved without access to context information. In contexts that warrant the use of the compositional interpretation, the representation created by a composition model should be used, whereas in the other contexts the learned, non-compositional representation should be chosen. However, if the composition model is trained on the non-compositional representation, it will always produce the lexicalized representation of the compound, even in cases where the compositional interpretation would be needed.

6.2 Other Compositionality Explorations

Previous research targeting the compositionality of German and English noun-noun compounds (Reddy et al., 2011; Im Walde et al., 2013; Roller et al., 2013) investigated the link between human ratings of compositionality and different representations of the modifier, the head, and of the compound as a whole. While the compositionality experiments in Chapter 5 and the automatic interpretation experiments in Chapter 7 analyze German and English compounds in parallel, the remainder of this chapter focuses on replicating experiments concerning the compositionality of German noun-noun compounds. This is due to the shortage of annotated resources for English: the dataset of English noun-noun compounds annotated for compositionality of Reddy et al. (2011) contains only 90 compounds, making it impractical for automatic approaches.

Roller et al. (2013) describe a dataset of 244 German noun-noun compounds, where each compound was rated by multiple human annotators. The ratings are meant to capture three separate aspects: (i) how compositional the meaning of the **modifier** is

with respect to that of the compound; (ii) how compositional the meaning of the **head** is with respect to the compound and (iii) how compositional the **whole compound** is perceived to be. All three aspects are rated with a number from 1 to 7, where 1 rates a completely **opaque** meaning (non-compositional), while 7 rates a completely **transparent** meaning (compositional). Multiple independent annotators provide ratings for each of the compounds. The end result are three average scores associated to each compound, quantifying the compositionality of the modifier with respect to the compound, the head's compositionality and how compositional the whole compound is.

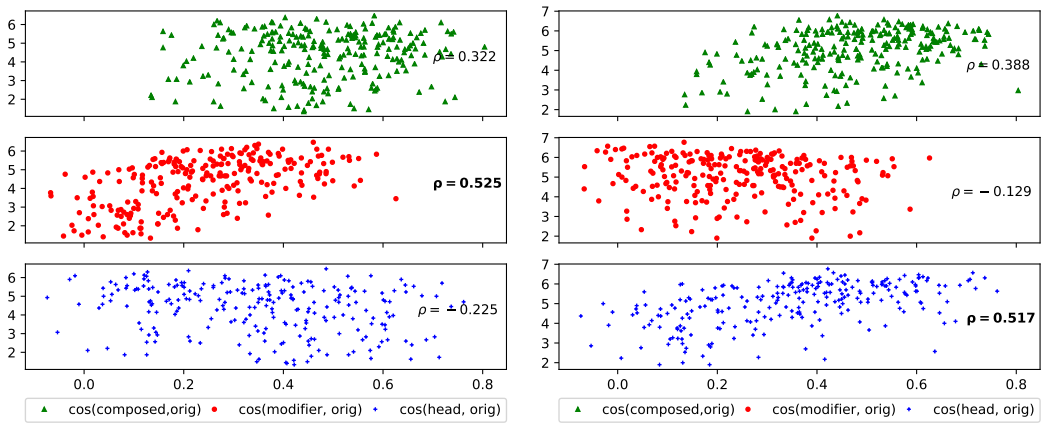
Im Walde et al. (2013) use Spearman's rank-order correlation coefficient ρ (Siegel and Castellan, 1988) to quantify the correlation between the average human ratings and the representations created by vector space models for the modifiers, heads and compounds in the Roller et al. (2013) dataset.

Spearman's ρ is a measure of how monotonic the relationship between two variables x and y is, i.e. how often do x and y 'change' together. Spearman's ρ takes values between -1 and 1. -1 is a perfect correlation meaning that when one of the variables increases, the other decreases, or that x and y decrease together. 1 is a perfect correlation meaning that x and y 'grow' at the same time (but not necessarily at the same rate). A value of 0 means that there is no correlation between the two variables.

Im Walde et al. (2013) try to predict the average human rating and use as predictors (i) the cosine similarity between the modifier and the compound ($\cos(m, c)$), (ii) the similarity between the head and the compound ($\cos(h, c)$) and (iii) the similarity between the addition or multiplication of head and modifier vectors and the compound ($\cos(m + h, c)$ and $\cos(m \cdot h, c)$). Their best predictions are made using vector representations based on co-occurring nouns (in a window of 20 words to the left and to the right of the target word).

The experiments of Im Walde et al. (2013) were replicated using their dataset of 244 noun-noun compounds. The 300-dimensional word representations for German, trained using GloVe (as described in Section 5.3.2) were used to represent the modifiers, heads and compounds in the dataset. An additional representation that was used was the 300-dimensional composed representation created using `multimatrix`, the best performing model from Chapter 5. To avoid that the composed representations are in any way biased by training, a new instance of the model was retrained where the 244 noun-noun compounds in the dataset used by Roller et al. (2013) and Im Walde et al. (2013) were filtered out of the original `nn-only train` set.

Figures 6.3a and 6.3b plot the average modifier and head ratings (Y-axis, ranging from 1 to 7) against three cosine similarity measures. The green triangles have on the X-axis the cosine similarity between the `multimatrix` composed representation and the original representation of the compound (cos(composed, orig)). The X-axis corresponding to the red circles is the cosine similarity between the modifier and the original compound representation (cos(modifier, orig)), while in the case of the blue plus signs on the X-axis there is the cosine similarity between the head and the original compound representation (cos(head, orig)).



(a) modifier ratings (b) head ratings

Fig. 6.3 (a) On the Y-axis: the average human rating regarding how compositional the **modifier** is wrt. the compound; On the X-axis: green triangles - cosine similarity between the `multimatrix`-composed and the original compound representation; red dots - cosine similarity between the modifier and the original compound representations; blue plus signs - cosine similarity between the head and the original compound representations. The best Spearman correlation coefficient $\rho = 0.525$ is achieved, as expected, when using cos(modifier, orig). (b) On the Y-axis: the average human rating regarding how compositional the **head** is wrt. to the compound; the X-axis as in subfigure (a); the best $\rho = 0.517$ is this time achieved using cos(head, orig).

The best predictors for the average human ratings of the modifier and the head are the cosine similarities between the modifier and the original compound representation, and between the head and the original representation, respectively. This result brings no surprises: Im Walde et al. (2013) also report a $\rho = 0.5698$ and $\rho = 0.5745$ when predicting the modifier-compound and the head-compound compositionality using the cosine between the representations of the modifier/head and that of the compound. The lower ρ values obtained here are most likely the result of the word embeddings using all the words in the context, not only the nouns as in Im Walde et al. (2013), and a smaller context window of only 10 words (instead of 20). The results are comparable to those of Im Walde et al. (2013) when using a 10-word window for creating the representations.

In the case of the compound whole average human ratings, the starting hypothesis was that the cosine similarity between the composed representation and the original compound representation (which was learned from the corpus) should provide the best predictions. However, the plots in Fig. 6.4 contradict this hypothesis: the best Spearman rank correlation coefficient is obtained by cos(modifier, orig). The similarity between the modifier and the original compound representations seems to be a better predictor of the compositionality or non-compositionality of the compound as a whole. Im Walde et al. (2013) make a similar point: in their experiments the best results are obtained using cos(modifier, compound) and cos(modifier·head, compound). In the

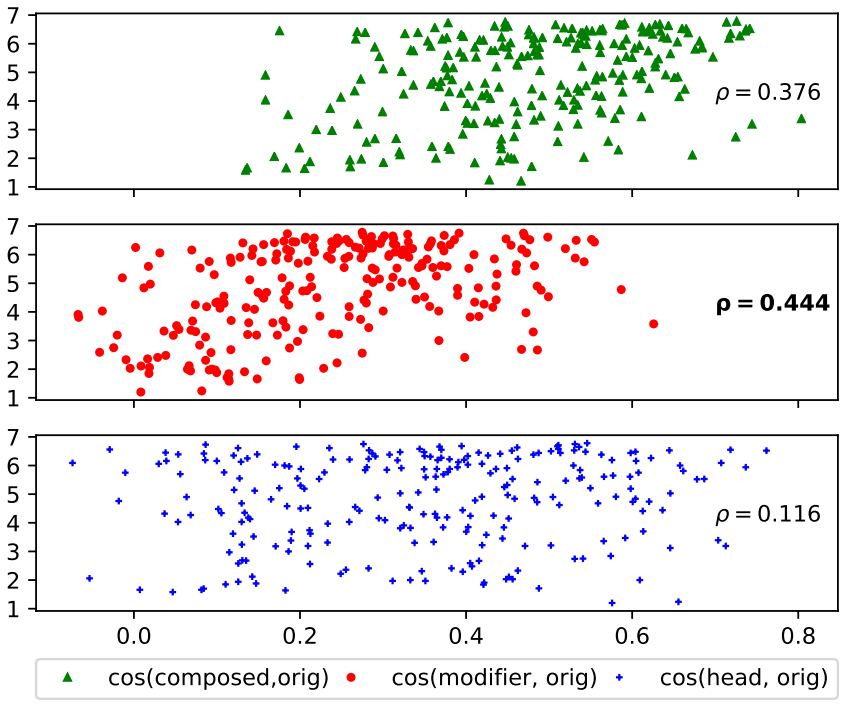


Fig. 6.4 Correlation for compound-whole ratings.

experiments here, however, the similarity using the compositional representation is clearly not as good a predictor as the one using the modifier.

The idea of using a composition function to distinguish between compositional and non-compositional phrases has been put to the test in various settings: using high-dimensional co-occurrence based representations and simple additive/multiplicative compositions (Reddy et al., 2011), using low-dimensional word representations (Salehi et al., 2015) or using more complex composition functions (Yazdani et al., 2015). However, for the German noun-noun dataset of Roller et al. (2013), there is no better Spearman correlation coefficient than the ρ =0.45 obtained by Im Walde et al. (2013), although on an English dataset containing 90 compounds a far better ρ =0.714 (Reddy et al., 2011) is obtained.

Bell and Schäfer (2013, 2016) describe a model for analyzing a complex nominal AB with the constituents A and B. Two aspects of the semantics of compounds are underlined: first, that the two constituents A, B are connected via an underspecified semantic relation R. Second, that both the constituents and the compound as a whole are likely to undergo metaphoric or metonymic meaning shifts. Bell and Schäfer (2016) gives as an example the English compound *buttercup*, where both constituents have a metaphorical meaning shift: *butter* is shifted to *smth. which has the color of butter*, whereas *cup* is shifted to *smth. that has the shape of a cup*. Additionally, the entire construction is then metonymically shifted such that the flower's color/shape are used to denote the the plant as a whole.

A closer look at compound-whole human ratings of the examples in the Roller et al. (2013) dataset showed interesting types of discrepancies between how the humans perceived the compound and how well the composition model is able to model it. Many of the compounds perceived as compositional by humans but which are difficult for the composition model are compounds with a metonymic shift of the head. Examples of this type are *Eisberg* 'iceberg', lit. 'ice mountain', which humans rated with an average score of 6.39 while the cosine similarity between the composed and the original representation was only 0.27; *Sandburg* 'sand castle', human rating 6.42, avg. cosine 0.26; *Schneeball* 'show ball', avg. human rating 6.35, avg. cosine 0.32 (a rating of 7 means that the compound is transparent/compositional; for the composition model a cosine similarity of 1 means that the vectors have the same direction, while orthogonal vectors have similarity 0). This suggests that such metaphorical meaning shifts affecting the head make it far more difficult to model the compound compositionally, but seem to have a lesser effect on the human perception of compositionality.

Similarly, a subset of compounds that are assigned a very low rating wrt. the transparency of the modifier but a high rating wrt. to the transparency of the head are far less of a challenge for the composition model. For example the compound *Jägerzaun* 'rustic fence', lit. 'hunter's fence' has an average modifier rating of 2.38, an average head rating of 5.00 and a whole rating of 2.11, while the cosine between the composed and the original compound representation is 0.67. Similarly for the compound *Blumenkohl* 'cauliflower', lit. 'flower cabbage' humans rate the relationship between *Blumen* and *Blumenkohl* with 2.1, between *Kohl* and *Blumenkohl* with 5.83 and the whole compound with 3.19. The cosine between the composed and original representation of the compound is 0.74. In these cases, the opacity of the modifier seems to play a bigger role in the human perception of compositionality, but is less of a challenge for the compositional models.

Further work is needed to apply the insights gained from this study on a larger scale to composition models. Maybe composition models must first learn how to pick up the meaning shift that humans have no trouble with - e.g. from an object to the shape of an object (like in *Schneeball*) or from an object to a small-scale, idealized version of it (e.g. *Sandburg*). The next chapter will focus on identifying the semantic relation between the constituents of a compound, using again compositional representations as a proxy for the meaning of compounds.[2]

[2]Both *Schneeball* and *Sandburg* are annotated in the dataset used in Chapter 7 as APPEARANCE*: the first constituent names a material that is formed to resemble the entity denoted by the second constituent.

Chapter 7

Automatic Interpretation of Noun-Noun Compounds

Writing code is an activity whose main result are good, useful programs, each tackling one of many different tasks that can nowdays be solved with the help of a computer. However, a by-product of programming are also the *bugs*, the parts of the program that do not perform the tasks that they should be performing, to the distress of the programmer. The art and science of discovering and removing the *bugs* - i.e. the programming faults - is called *debugging*.

A popular method for debugging is called *rubber duck debugging*. The reader encountering this three part compound for the first time will (if not aware of this technique) face some difficulty in interpreting it. As a first step she might try to figure out which makes more sense *((rubber) duck debugging)* or *((rubber duck) debugging)*? Given that *rubber* is a material, it is more likely for a *duck* to be made of *rubber* than for the *(duck) debugging* to be made of *rubber*. Then, if *rubber duck* is taken to be a concept, to solve the riddle one then has to figure out what does a *rubber duck* have to do with *debugging*? As it turns out, the *rubber duck* plays the role of a partner to which the programmer explains the program. By explaining the program line by line to the *rubber duck*, in as simple terms as she can, the programmer is usually able to figure out where the bug is (Hunt and Thomas, 1999:95).

As this example showed, understanding the meaning of a compound can be a non-trivial exercise even for humans. As mentioned in Chapter 2, researchers studying the way humans process novel compounds saw patterns of interpretation being used. Ryder (1994:87) describes the interpretation process of a listener as follows:

> When presented with a novel combination of established forms, such as a new compound, a listener will automatically generalize across knowledge from all the previous contexts in which he has experienced the two words and any other linguistic experiences he perceives as similar, in order to create an interpretation compatible with whatever contextual information he has.

In the example above, the reader might have known the compound *rubber duck*, or if not the relatively frequent pattern '*rubber* ANIMAL' might provide enough grounds

for interpreting the first part of the compound. However, discovering the link between *rubber duck* and *debugging* is far more difficult, given the lack of similar compounds.

Klos (2011:155) names as the first result of her analysis the fact that humans are aware that compounds have, when presented in isolation, multiple possible interpretations. If for *rubber duck* the interpretation seems relatively straightforward, the *rubber duck debugging* is an example where a person might come up with multiple plausible interpretations, which might be closer or further away from the institutionalized use of the compound.

As discussed in Chapter 2, computational approaches have framed the semantic interpretation of compounds as a **classification task**. Given a compound like *rubber duck* and an inventory of semantic relations like PART, MATERIAL, TIME, LOCATION, the automatic classifier has to choose only one of the semantic relations, deemed the most plausible by the classifier. For some compounds, choosing just one semantic relation might work - e.g. the MATERIAL relation should be the preferred choice of a well-trained classifier for the compound *rubber duck*. However, it is very likely that the inventory of semantic relations does not contain a suitable label for the link between *rubber duck* and *debugging*. Such infrequent patterns are likely to get bundled up into an 'other relation' category.

For some compounds the internal semantic relation is easy to recognize, while others can be ambiguous in their interpretation. If there is ambiguity in the interpretation of the constituents, or the semantic relations have some overlap, or the compound genuinely has multiple interpretations, choosing the correct semantic relation might require the use of additional information from the sentential context.

Most of the existing compound datasets annotated with semantic relations provide, however, the compounds and the semantic relation annotations **out-of-context**, i.e. without illustrative sentences containing the compound together with the annotated semantic relation. This is true also for the German and English datasets used in this chapter (described in Sections 7.1.1 and 7.2.1). The datasets contain low frequency compounds, whose interpretation might require taking the immediate context into account, as well as lexicalized compounds whose identification requires the use of context. The out-of-context setup can make the interpretation task more difficult: a human that thinks of multiple possible interpretations for a compound has an easier time choosing (the correct) interpretation when context is available, sometimes even completely re-analyzing and overriding the initial analysis. Meyer (1993:26) describes the process as follows:

> Compounds are ambiguous. So if a certain interpretation of a compound is chosen in a discourse, other possibilities are still open such that it is uncertain whether the interpretation used is the right one. So if it turned out that the compound is used with a wrong meaning, a kind of semantic backtracking process could lead to a new interpretation.

Given the available data, the automatic interpretation experiments in this chapter will be carried out in the artificial and more difficult setup where the compounds are interpreted **without context**, and where only **one** semantic relation is considered

correct in each case. While the automatic interpretation of compounds might prove easier in a given sentential context, such experiments are left for future work since they require dedicated datasets, where the compound-internal semantic relation is annotated in its context of use.

Evaluating the Quality of a Classification When automatically interpreting compounds, the task of the classifier is to decide which of the r possible labels in a semantic relation inventory is the correct one. Such classifiers are called **multi-label classifiers** because they need to choose between r labels, with r>2.

Several measures can be used for evaluating the performance of a multi-label classification algorithm. The first is to build a **contingency table** (Manning and Schütze, 1999:577) for each label. A contingency table provides information about each of four possible scenarios: (i) how many examples labeled with a label l were actually assigned label l (**true positives**, `tp`); (ii) how many examples of different labels were assigned label l (**false positives**, `fp`); (iii) how many examples with label l received another label (**false negatives**, `fn`) and (iv) how many examples that were not labeled with l were also predicted to have a label other than l (**true negatives**, `tn`). Table 7.1 shows an example of a contingency table for a given label l.

	items labeled with l	items not labeled with l
prediction was l	`tp` (true positives)	`fp` (false positives)
prediction was not l	`fn` (false negatives)	`tn` (true negatives)

Table 7.1 Contingency table for the label l.

Several scores can be computed based on the information in a contingency table (Manning and Schütze, 1999:269):

- **accuracy** = $\frac{\texttt{tp+tn}}{\texttt{tp+tn+fp+fn}}$, the proportion of correct predictions from the total number of predictions
- **precision** = $\frac{\texttt{tp}}{\texttt{tp+fp}}$, the proportion of correct predictions of the label l out of all the predictions of the label l
- **recall** = $\frac{\texttt{tp}}{\texttt{tp+fn}}$, the proportion of correct predictions of the label l out of the total number of instances labeled l in the test set

Depending on the application, one might compromise precision for recall or the other way around. When precision and recall are equally important a combined score - the F_1 score - is usually reported. The $\mathbf{F_1}$ **score** is defined as the harmonic mean between precision and recall (Eq. 7.1).

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{7.1}$$

Accuracy and F_1 score offer complementary insights into the results of a classifier. The accuracy indicates how many times the classifier was right in choosing a particular label. The F_1 score, on the other side, favors classifiers that obtain more true positives.

In the case of multi-label classification, accuracy and F_1 score are computed separately for each label. The instances labeled with a particular label l are the positive instances. All the instances labeled with labels other than l are considered to be negative examples for the label l. The per-label scores are then averaged to obtain an overall, **macro-averaged** classification score. Macro-averaging treats the score of each label equally - each is given an equal weight in the overall score, even as the number of instances is different from label to label.

A variant of macro-averaging - the **weighted** F_1 score - weighs the score of each label by the number of support instances for that label. In the weighted F1 score case, labels that have few examples will have less impact on the overall score, whereas labels with many examples will make the most significant contribution to the overall score. In other words, the penalty for making errors is higher when there are many examples with a particular label l, and smaller when there are only a handful of examples of l.

The datasets used in the experiments in this chapter all have different number of instances per label, making the **weighted F_1 score** the most appropriate score to report. The evaluation is carried out using a public implementation available in `scikit-learn` (Pedregosa et al., 2011) - the `metrics.precision_recall_fscore_support` method.

7.1 Automatic Interpretation of German Compounds

7.1.1 `de-nncom-sem`: The Dataset of German Noun-Noun Compounds Annotated with Semantic Relations

The dataset of annotated German noun-noun compounds contains a total of 8005 compounds, formed through the combination of 497 distinct heads and 3002 distinct modifiers. The most frequent head is the noun *Haus* 'house', which combines with 161 distinct modifiers. The most frequent modifier is the noun *Wasser* 'water', which forms compounds with 49 different heads. There are on average 16.11 compounds with the same head and 2.67 compounds with the same modifier.

Each compound in the dataset is labeled with a hybrid label. The hybrid label consists of a property and a preposition, all from the predefined inventory of 57 properties and 19 prepositions described in Section 2.2. Table 2.5 in Section 2.2 presents the statistics of the dataset in terms of percentage of dataset compounds for each property, along with the most frequently co-occurring prepositions and representative examples for each property.

As mentioned in Section 2.2, the inventory is a mixture of three property types: there are 26 direct-only properties, 14 bi-directional properties and 3 indirect-only properties. MEMBER, for example, is a bi-directional property: the compound *Kinderchor* 'children chorus' is labeled with its direct sense, MEMBER, because the denotatum of the modifier

Kinder 'children' are members of the group defined by the head *Chor* 'chorus'. The inverse situation happens in the case of the compound *Marinesoldat* 'marine soldier', where the modifier names the group *Marine* 'marine' and the head *Soldat* 'soldier' names the member of the group. *Marinensoldat* is therefore annotated with the inverse sense of the property, MEMBER*. Direct-only and inverse-only properties are those properties that occur in the annotated dataset only in one of the two possible directions.

If the 57 properties are taken individually, the most frequent property is PART*, which labels 535 compounds - 6.68% of the total dataset. The least frequent property is COMPARISON*, which labels only 5 compounds - 0.06% of the dataset.

If the bi-directional properties are counted together, the inventory contains 43 properties. The most frequent is the bi-directional PART/* property, which labels 984 compounds - 12.29% of the total dataset. The least frequent property in this case is ACCESS*, which labels 18 compounds - 0.22% of the dataset. Compacting the bi-directional properties seems to be a good way of gaining more training examples for each class. The experiments in the next sections will quantify the impact of the property collapse with respect to the performance of the automatic classification.

The annotated dataset was split into four parts. First, the 481 IAA compounds were removed from the dataset to form a separate `test-iaa` portion. The remaining 7524 were then split in standard `train`, `test` and `dev` splits, containing 5265, 1505 and 754 compounds respectively (70-20-10%).

Table 7.2 lists the overlap in terms of modifiers and heads between the `train` split of the dataset and the other splits. A high overlap in terms of modifiers and heads between the `train` subset and the `test` and `dev` subsets might make the model learn to associate particular heads and modifiers to particular properties or prepositions. A good model, however, should be able to generalize above the lexical level, and make useful predictions even when it encounters compounds with completely new heads/modifiers. The `test-iaa` split is expected to be the most challenging but also the most informative portion of the test data, given the zero overlap in terms of heads between it and the `train` split.

Data split	Size	Modifier overlap	Head overlap
`test`	1505	63.09%	98.36%
`dev`	754	65.66%	98.62%
`test-iaa`	481	73.64%	0.00%

Table 7.2 Amount of modifier/head overlap between the `train` split of the dataset (5266 compounds) and the `dev`, `test` and `test-iaa` splits. The `test-iaa` split should provide the most realistic estimate of the generalization power of the model, since there is no overlap in terms of the head constituent between it and the `train` split.

Because some of the property labels in the dataset are not frequent enough, the different splits might not contain instances labeled with all the properties. Figure 7.1 shows the number of compounds labeled with each of the 57 individual properties, first for the whole dataset and then for the different data splits. Both the `test` and the `dev` subsets have only one compound labeled with some of the least frequent properties -

like INGREDIENT* and COMPARISON*. The distribution of properties in the `test-iaa` subset is different when compared to the distribution of the other subsets. The fact that `test-iaa` contains compounds with a particular subset of heads has a bearing on the distribution of properties used to annotate them. There are, for example, far more compounds annotated with the properties ATTRIBUTE, PURPOSE__OF__USE and RAW__PRODUCT in the `test-iaa` subset than in the whole dataset. Conversely, the `test-iaa` subset was annotated using only 43 of the 57 properties available in the annotation inventory, with 14 individual properties missing completely in the `test-iaa` annotation. This suggests that there is a strong correlation between the identity and semantics of the head constituent and the range of possible properties.

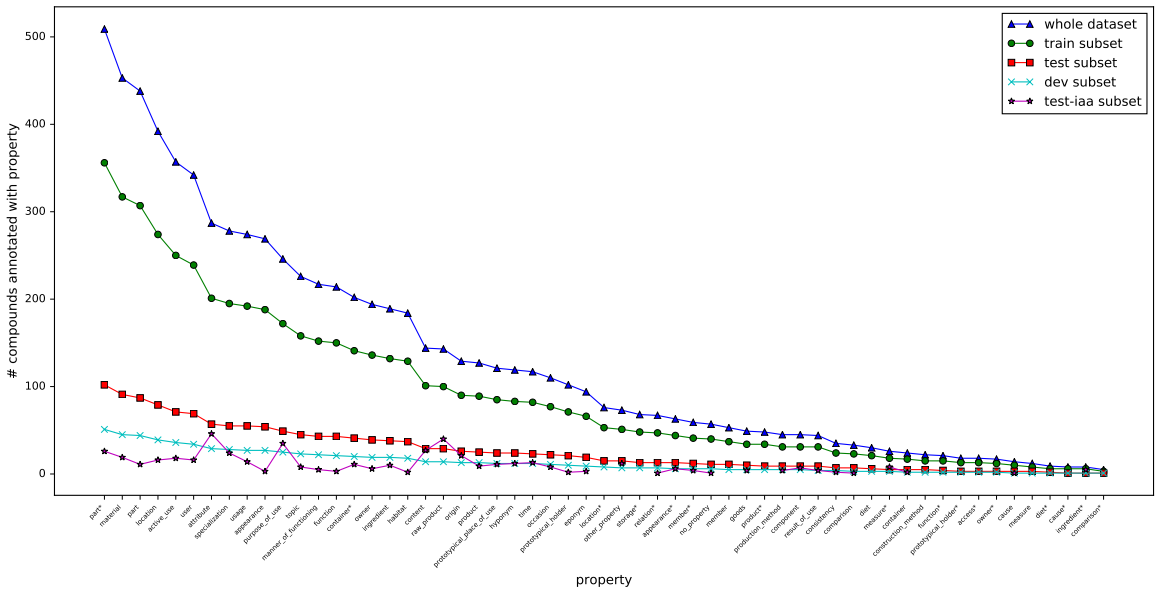


Fig. 7.1 Number of compounds annotated with each of the **57 individual properties** in the whole dataset and the `train`, `test`, `dev` and `test-iaa` subsets respectively.

Figure 7.2 presents the distribution of properties in the different data splits, this time using the 43 collapsed properties. This time there are only four properties that are not represented in the `test-iaa` subset: STORAGE, DIET/*, CONSTRUCTION__METHOD and ACCESS*. Collapsing categories seems particularly useful in this case, when comparing the 4 missing categories with the 14 that were missing in the individual case.

The experiments in the next sections will report on the performance of different classifiers both using the 57 **individual properties** and using the 43 **collapsed properties**. In both cases, the number of examples shows marked variations from label to label, making the weighted F_1 score the most appropriate scoring variant for this classification setup.

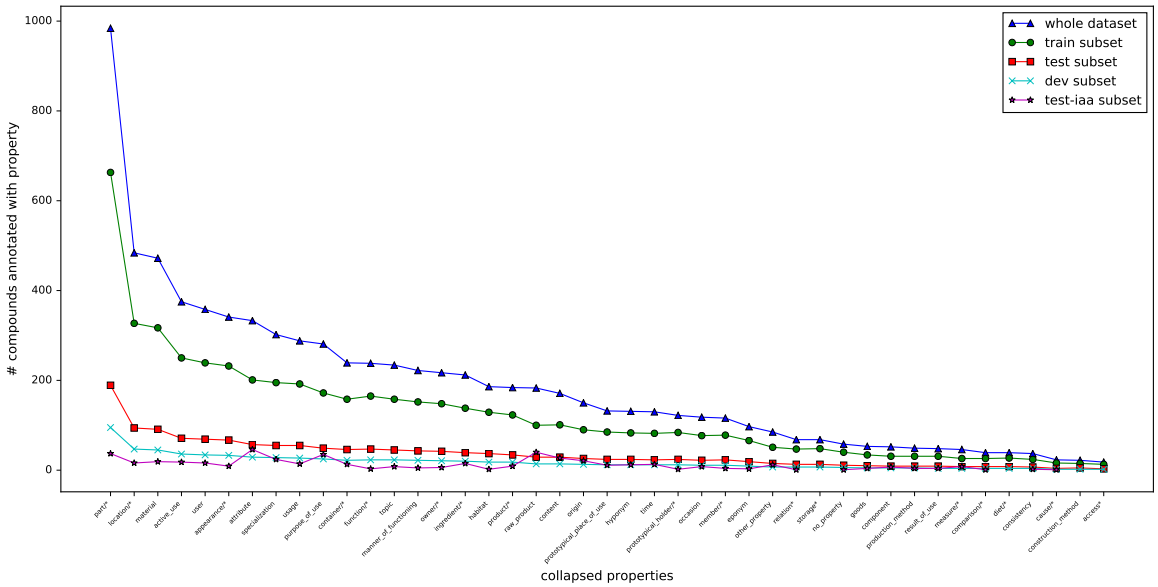


Fig. 7.2 Number of compounds annotated with the **43 collapsed properties** in the whole dataset and the `train`, `test`, `dev` and `test-iaa` subsets respectively.

7.1.2 Experiments on `de-nncom-sem`

Word Representations The first step in experimenting with automatic classifiers is connecting the constituents and the compounds in the dataset to word representations. The experiments in this chapter will make use of the 300-dimensional set of distributional word representations created in the previous chapter (for details see Section 5.3.2), normalized to unit norm. The vocabulary used to train these representations contained 1,029,270 individual words of minimum frequency 100. The 8005 compounds in `de-nncom-sem` use 11028 individual words. There are 497 heads, 3001 modifiers and 8005 compounds, with 385 words that appear both as heads and modifiers, and 90 words that appear both as modifiers and as compounds.

Each of the words - the compound as well as its constituents - was assigned a word representation by searching in the embedding dictionary. As expected, the words in the dataset cover a wide spectrum of frequencies: from very frequent - e.g. *Zeit* 'time', which appears over 94 million times in the support corpus, to moderately frequent, e.g. *Gesichtscreme* 'face cream' appears 3294 times, or low frequent - *Klavierdeckel* 'piano lid' occurs only 102 times. Moreover, a subset of the individual words do not appear in the embedding dictionary - meaning that if they appear in the support corpus their frequency must be below 100 occurrences. 9700 of the 11028 individual words were found in the embedding vocabulary and received a custom, trained representation. All 497 head nouns, 2969 of the 3001 modifiers and 6708 of the 8005 compounds were part of the word embeddings vocabulary and received a custom vector representation. The fact that only 1% of the modifiers were below the frequency threshold - compared to 16% of the compounds - shows again the importance of an alternative, compositional

modeling of compounds. Almost one in five compounds in the `de-nncom-sem` dataset is infrequent enough to make its direct modeling using word representations impossible.

In the context of distributional word representations, unknown words are usually modeled by an **unknown word vector**, which is either trained together with the other word representations[1] or obtained later by averaging over all the representations in the vocabulary. Following the suggestion of the GloVe developers, unknown words will be represented in the latter way, as the averaged (and L_2-normalized) vector over all the 1M word vectors in the vocabulary. The experiments using the original compound representations as input will make use of this averaged unknown word vector both for modeling the 16% infrequent compounds and the 1% infrequent modifiers. The experiments using composed representations will use composed vectors to represent the infrequent compounds and will only use the unknown vector to model the 1% infrequent modifiers.

7.1.3 Classification using individual properties

Baselines Typical baselines for a classification task are the **random baseline** and **most frequent class baseline**. For the property classification case, this amounts to assigning all compounds in the `test` set either a random property label or the most frequent property label seen in the `train` data.

Table 7.3 shows the performance of these baselines applied to the compounds in `de-nncom-sem`. The models in this section will be evaluated in terms of their accuracy and of their weighted F_1 score. Each baseline is computed 10 times, and the average accuracy and weighted F_1 score is reported.

The baseline results leave much room for improvement. The **random property baseline** (#1) has the worse performance on each data split, leading to a correct assignment of the individual property for ∼2% of the data. The accuracy of the overall **most frequent property baseline** (#2) ranges from 5.41% to 6.78% on the different data splits. As mentioned above, the most frequent property is, in this case, the property PART*, annotating compounds like *Autodach* 'car roof', where the modifier *Auto* names the whole, and the head *Dach* names the part.

In the particular case of noun-noun compounds, there are other baselines that might provide useful insights into the underlying data patterns. A variation of the most frequent baseline is to assign, to each compound, the **most frequent property** encountered with the **modifier** (#3) or the **head** (#4). In cases where there is no data about a particular head or modifier, these baselines would back off to the overall most frequent property - PART* (baseline #2). These baselines are estimated with the help of the data in the `train` split. In cases where two or more properties occur with

[1]The unknown word representation is used as a representation for every token that occurs fewer times than the minimal frequency threshold. Its representation is thus learned just like any other word representation, based on all the contexts containing infrequent tokens.

#	Baseline	test		dev		test-iaa	
		Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
1	`random property`	1.79%	0.021	1.82%	0.022	1.73%	0.022
2	`mf prop overall (mf_o)`	6.78%	0.009	6.76%	0.009	5.41%	0.006
3	`mf prop modifier (mf_m)`	29.77%	0.303	28.45%	0.287	25.90%	0.255
4	`mf prop head (mf_h)`	35.18%	0.327	35.07%	0.314	5.41%	0.006
5	`prop_m`$\cap$`prop_h` if$\neq \phi$ else `mf_o`	30.47%	0.341	33.22%	0.364	5.41%	0.006
6	`prop_m`$\cap$`prop_h` if$\neq \phi$ else `mf_m` else `mf_o`	35.28%	0.365	36.96%	0.379	**26.15%**	**0.259**
7	`prop_m`$\cap$`prop_h` if$\neq \phi$ else `mf_h` else `mf_o`	46.57%	0.450	46.33%	0.439	5.41%	0.006
8	`prop_m`$\cap$`prop_h` if$\neq \phi$ else `mf_m` else `mf_h` else `mf_o`	**46.63%**	**0.451**	**46.66%**	**0.442**	26.03%	0.257

Table 7.3 Baselines for **individual property classification**. Results reported in terms of accuracy (Acc.) and weighted F_1 score (wF_1). The best `test` set performance is obtained by taking the intersection of the modifier and head properties and choosing the property with the highest probability. If the intersection is empty, the most frequent property of the modifier, head or overall most frequent property are considered (in this order, the first to be defined is chosen).

maximum frequency for a given head, one of them is randomly chosen.[2] The quality of the predictions is then judged on the `test`, `dev` and `test-iaa` data.

Knowing what property is most frequently associated with a particular modifier substantially improves the results, leading to a correct prediction in 25.90% - 29.77% on the different data splits. An intriguing result can be observed in the case of the `test-iaa` split. Although the modifiers in the `test-iaa` split have a larger overlap with the modifiers in the `train` split when compared to the `test` and `dev` splits (73.64% vs. 63.09%/65.66%, cf. Table 7.2), the performance of baseline #3 decreases on the `test-iaa` split when compared to the `test` and `dev` splits. A likely explanation is that the new heads contained in the `test-iaa` subset enter different relations with the existing modifiers, thus requiring other properties than those already associated with them in the training data. An example of this type is the modifier *Linde* 'lime tree', which appears in the `train` set in the compounds *Lindenblatt* 'lime tree leaf' and *Lindenblüte* 'lime tree flower', both annotated with the property PART*. In the `test-iaa` subset the modifier *Linde* appears in the context of the compound *Lindenholz* 'lime tree wood', annotated with the property RAW PRODUCT.

Using the property that is most frequently associated with the head (baseline #4) leads to even better results: ∼35% of the compounds in the `test` and `dev` splits are, in this case, correctly classified. On the `test-iaa` subset, however, because there is no overlap with the heads in the training set, this baseline is reduced to guessing the overall most frequent property (baseline #2).

Another possible baseline is to consider the best property from the intersection between the set of properties associated to the head and the modifier (baseline #5). It can be computed by multiplying the individual probability distributions over properties for the modifier and the head, and renormalizing the resulting product distribution

[2]The random choice is made for each data instance individually: e.g. if the head occurs 25% of times with $prop_1$, 25% with $prop_2$, and a smaller number of times with other properties, there's a uniform probability for each compound using this head to be assigned either $prop_1$ or $prop_2$.

to sum to 1. Having a non-empty intersection occurs, however, only in ∼50% of the compounds in the `test` and `dev` data splits. In the rest of the cases baseline #5 backs off to predict the overall most frequent property.

Since the intersection of properties is non-empty only in half of the test cases, baseline #5 could therefore be augmented by adding the most frequent property of the head and modifier where available. Adding the most frequent property assigned to the modifier (baseline #6) lead to an accuracy of over 35% on the `test` and `dev` data, a result comparable to the one obtained with baseline #4.

Adding the most frequent property assigned to the head (baseline #7) worked even better, leading to accuracies above 46% on the `test` and `dev` data. The head information, however, did not bring improvements on the `test-iaa` dataset, given its zero overlap in terms of heads with the `train` subset. A combined baseline (baseline #8), augmented with both the information about the most frequent property of the head and of the modifier obtained the overall best results from all the baselines that were tested: ∼46% accuracy on the `test` and `dev` subsets, and ∼26% accuracy on the `test-iaa` subset.

Experimenting with different types of baselines lead to substantial improvements over the classical most frequent property baseline. Knowing the distribution of properties that a particular modifier or head typically occurs with can greatly improve classification results. However, even the more sophisticated baselines lack a central property: they cannot generalize to new constituents. When dealing with compounds with new constituents, they are reduced to guessing the most frequent property.

The next section introduces automatic interpretation models that aim to overcome these generalization issues through the use of distributional word representations. For cases where the distributional representations of the compounds are not available because of their low frequency, compound representations are created using the best performing composition models described in Chapter 5.

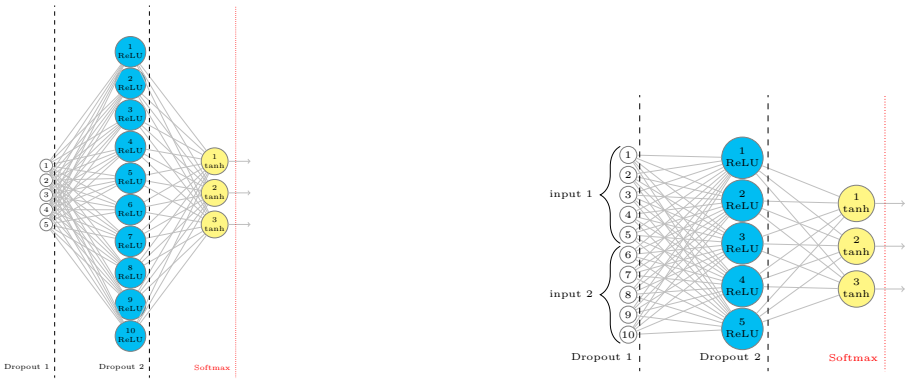
Experiments using as input the learned compound representation. Consider a compound like *Apfelbaum* 'apple tree'. To predict the semantic relation[3] connecting its constituents, *Apfel* 'apple' and *Baum* 'tree', a good starting point should be the distributional representation of the compound itself. This is the representation learned **directly** from co-occurrence data for the word *Apfelbaum*, as previously described in Section 7.1.2.

Several classifiers were tried out for this classification task. The **logistic regression** (`LR`) and **k nearest neighbors** (`KNN`) classifiers as implemented in `scikit-learn` v.0.19.1 (Pedregosa et al., 2011) are used as strong baselines. The following hyperparameters differ from the default settings (they were optimized according to their performance on the `dev` subset): logistic regression uses the `multinomial` loss and the `sag` optimizer, while the nearest neighbors classifier uses $k = 12$ neighbors and the

[3]Out of the 57 semantic relations available in the individual property inventory.

`distance` weighting function, where the closer neighbors have a higher impact than the ones that are further away.

The `LR` and `KNN` baselines are compared to a multilayer perceptron implemented in Torch (Collobert et al., 2011a). Figure 7.3a shows the neural network architecture that was used for this task: an input layer of size n, a hidden layer of size $2n$ and an output layer of size r. n is the size of the word representations which equals 300 for the experiments presented here.



(a) Single input classification network (`SMLP`) (b) Two input classification network (`DMLP`)

Fig. 7.3 Different classification architectures. Leftmost: the input layer; inputs are displayed as small while circles. Middle: hidden layer; hidden units are displayed as large blue circles with `ReLU` activation. Right: output layer; output notes displayed as yellow circles with `tanh` activation.

The sizes of the input and output layers are fixed by virtue of the input features (the word representations), and by the number of classes in the classification task. The size of the hidden layer is a hyperparameter of the model. Three hidden layer sizes were tested out: $n/2$, n and $2n$, where n is the size of the input representation. Table B.7 in Appendix B presents details on the impact of different hyperparameter settings. The classifier has an initial learning rate of 0.14 and a learning rate decay of $1e-5$. It is trained using a negative log-likelihood criterion, optimized with mini-batch AdaGrad (Duchi et al., 2011), with a mini-batch size of 100 examples. Models are trained using early stopping (Prechelt, 1998) with a patience of 100 epochs (see Section 3.3.3 for details about the use of early stopping as a regularization technique). The values for the learning rate, batch size and patience were chosen after an initial set of experiments over the `dev` set and were kept unchanged for the rest of the experiments.

The best results on the `dev` subset were obtained using a hidden layer twice the size of the input layer. Other hyperparameters were a 0.5 dropout immediately after the input layer (Dropout 1), the use of a `ReLU` activation function for the hidden layer and of a `tanh` activation function for the output layer (see Sections 3.3.3 for details about the use of dropout, and Section 3.1 for details about the different activation functions). A `softmax` layer was added on top of the network to transform the activations of the

output layer into a probability distribution over all the classification labels and thus be able to predict the "winning" label out of the r possible ones. The acronym SMLP is used for the single input classification network in the discussion that follows.

Classifier	Input	test		dev		test-iaa	
		Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
KNN	compound_original	33.89%	0.333	35.01%	0.340	26.20%	0.236
LR	compound_original	32.89%	0.313	34.48%	0.317	28.07%	0.239
SMLP	compound_original	**36.94%**	**0.367**	**42.18%**	**0.410**	**30.77%**	**0.274**
KNN	head	32.62%	0.306	34.62%	0.311	18.50%	0.164
LR	head	**36.08%**	**0.321**	**37.14%**	**0.320**	**27.44%**	**0.199**
SMLP	head	34.29%	0.303	36.74%	0.314	25.99%	0.211
KNN	modifier	34.22%	0.308	32.89%	0.308	27.23%	0.252
LR	modifier	35.35%	0.318	35.81%	0.326	29.11%	0.246
SMLP	modifier	**36.74%**	**0.342**	**38.20%**	**0.359**	**28.48%**	**0.260**

Table 7.4 Results for **individual property classification**, using as input the original representation of the compound, its head or its modifier. LR (logistic regression) and KNN (k nearest neighbours) are strong baselines. SMLP is the single input multilayer perceptron from Figure 7.3a.

Table 7.4 contains the classification results. When using the original, corpus-learned representation of the compound as an input feature (compound_original), the SMLP classifier was able to predict the semantic relation correctly for 36.94% of the test subset, and 30.77% of the test-iaa subset, surpassing the logistic regression (LR) and nearest neighbors (KNN) classifiers. However, these results are disquieting in the light of the best baseline (#8) results, where the accuracy of the predictions measured 46.63% on the test subset and 26.03% on the test-iaa subset (from Table 7.3). The results show that the compound representation contains, in this case, not enough information to make an informed prediction about the internal semantic relation.

Other inputs that could be used in a similar way are the corpus-learned representations of the two constituents: the head and the modifier. The next two sections in Table 7.4 present the results of using the head (head) and the modifier (modifier) representations as inputs for predicting the semantic relation. However, neither of these representations lead to a better result, irrespective of the classifier that is used.

Overall, none of the original representations on its own leads to a result that is above the best baseline (#8) on the test subset, while the results on the test-iaa subset improve but still remain unsatisfactory. Can a composed representation, making use of both modifier and head representations, do better?

The results in Table 7.5 show this to be the case. The classification input is in this case a compound representation created using a composition operation. The composition operation starts from the representations of the modifier and the head and produces a compound representation, which is then normalized to unit norm and used as input for the classifier. Using a composed representation brings improvements across the board, for all data subsets. It improves over using any of the original vectors as input, as well as over the best baseline results (baseline #8).

Inputs	Dropout 1	test		dev		test-iaa	
		Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
composed_addition	0.3	54.22%	0.527	58.49%	0.560	32.85%	0.306
composed_matrix	0	54.55%	0.530	57.16%	0.553	38.88%	0.354
composed_multimatrix	0.1	**56.81%**	**0.550**	**58.89%**	**0.565**	**44.70%**	**0.411**

Table 7.5 Results for **individual property classification**, using as input composed representations of the compound.

The results in Table 7.5 were obtained using the same SMLP architecture depicted in Figure 7.3a. The only hyperparameter that was tuned was the amount of dropout to be applied to the input vectors. Since the input is now a composed representation - so a combination of two original vectors - one might expect that the amount of information that could be safely "dropped-out" without hurting the performance of the classifier to be different depending on the composition function that was used. The results in Table B.8 in Appendix B show the results obtained on the dev subset using the different hyperparameter settings. The best dropout setting when using composed vectors as input turned out to be different for each composition function: as high as 0.3 when making use of the addition-based representation, 0 when using the matrix-composed representation and only 0.1 when having the multimatrix representation as input.

Combining both constituent representations significantly boosts the performance of the SMLP classifier: the test accuracy reaches 56.81%, while the test-iaa accuracy gets to 44.70%. When compared to the results in Table 7.4 (test: 36.94% acc., test-iaa: 30.77% acc.), it is indisputable that a compound representation based on both constituents contains far more information than what the compound_original representation could offer. The poor classification results using only the original representation of the compound might be due, on one side, to the fact that the unknown word representation is used to model infrequent compounds, which amount to 16% of the dataset. Given the productivity of the compounding process, it is very likely that the classifier will face the task of interpreting compounds whose corpus-based representation is unavailable. Using a combination of the constituent representations will make this task easier. On the other side, the semantic property annotation is aimed at capturing the interaction between the two constituents of a compound. The classification might be easier to accomplish starting from the constituent representations than from the representation of the compound, which might not preserve enough information about its constituents to make the semantic relation identification possible.

The results on the test-iaa subset from Table 7.5 also reveal a qualitative difference between the different composed vectors. Using the multimatrix-composed representation leads to a 11.85% increase in accuracy when compared to using a composed vector obtained via addition. This difference can solely be attributed to the fact that the multimatrix model was trained to learn a good composition using a large amount of example compounds, whereas the addition model performs the combination directly, unaware of any other training examples.

The experiments in this section have shown that a classifier based on a composed representation outperforms classifiers based on the original compound representation. The improvements are already visible when using a simple `addition`-based representation to combine the representations of the modifier and the head. Even better results can be obtained using a composed representation produced by a pre-trained composition model. However, when the constituent representations are combined into a new, composed representation, the composition process will perform a compression operation. The two n-dimensional constituent representations are combined into an n-dimensional representation for the compound. How well does composition preserve the semantic information? Is composition lossy or not? The next section will try to answer these questions.

Experiments using two representations as input. If composed representations capture all the necessary details from the modifier and head representations, there ought to be no difference between a classifier that starts off with the two constituent representations and one that uses a composed representation. This hypothesis was tested using a new architecture, depicted in Fig. 7.3b, which takes as input two vectors ($2n$) and uses an n-dimensional hidden layer with a `ReLU` activation. Just like in the case of the previous classifier, the classification is performed using a $n \times r$ linear layer with a `tanh` activation function, followed by a `softmax`. Two possibilities for placing a dropout layer are considered: one immediately after the inputs, and the other after the hidden layer. Different sizes of the hidden layer, as well as different dropout rates were explored and are presented in more detail in Table B.9, Appendix B.

Inputs	Dr1	Dr2	`test`		`dev`		`test-iaa`	
			Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
`modifier`; `head`	0	0.2	61.06%	0.597	64.46%	**0.633**	44.28%	0.433
`composed_multimatrix`; `head`	0.1	0.1	53.95%	0.526	58.62%	0.562	40.54%	0.381
`modifier`; `composed_multimatrix`	0.3	0.3	**61.93%**	**0.608**	**64.59%**	0.631	**47.82%**	0.457

Table 7.6 Results for **individual property classification**, using as input two representations. Hidden layer size: $2n \times n$.

Table 7.6 shows the results of classifiers using two representations as input. In the first experiment, the inputs are the two representations of the constituents, `modifier` and `head`. There is a notable improvement over the results of the `SMLP` classifier using a composed representation as input: 61.06% accuracy on the `test` subset versus the previous best result, 56.81% accuracy (cf. Table 7.5). On the `test-iaa` subset there is only a minimal difference between the two results: 44.28% vs. 44.70%, with the composed vector obtaining a slightly improved performance. This result suggests that the composition process discards some semantic information needed for distinguishing between the different semantic relations.

To find out which information gets discarded, two variations were tried out: one where the `head` representation is kept but the `modifier` representation is replaced by the best composed representation - `composed_multimatrix`. And a second one where

the `head` representation is discarded and the `composed_multimatrix` representation is used together with the `modifier` representation.

Surprisingly, pairing the `head` and the composed representation and discarding the `modifier` representation lead to worse results than when using the composed representation alone: 53.95% vs. 56.81% on the `test` subset and 40.54% vs. 44.70% on the `test-iaa` subset.

Conversely, discarding the `head` representation and pairing the `multimatrix` and the `modifier` representations lead to better results - both when compared to the classifiers using only composed vectors as well as the classifier using the original representation of the compound. The results improved both on the `test` subset - 61.93% vs. 61.06% and on the `test-iaa` subset - 47.82% vs. 44.70%.

The result suggests that the `multimatrix`-composed representation is biased towards capturing more information about the head constituent and less about the modifier. This behavior is understandable, given that the majority of compounds are endocentric, i.e. the compound is a subtype of the head. Therefore, the focus on capturing information about the head seems like a reasonable strategy towards obtaining a good representation. However, the modifier usually names the noteworthy aspect that makes it worthwhile to construct a compound rather just using the head word - i.e. a *dog house* would not quite qualify as a *house*. Composition models could be improved by making sure that they capture the essential information about both constituents, and not focus excessively on the head constituent.

Analyzing the performance of the best model on individual property classification The best classifier for individual property classification uses as input the modifier representation and the composed `multimatrix` representation of the compound (`modifier`; `composed_multimatrix`, Table 7.6). This model obtained, on the `test` subset, an accuracy of 61.93% and a weighted F_1 score of 0.608. Figure 7.4 shows the performance of this classifier on the `test` set, broken down per individual property. The F_1 score, with a minimum value of 0 and a maximum value of 1, is marked with blue circles. The absolute frequency of each label in the `train` set is marked using green triangles. Going from left to right, the number of `train` examples per label constantly increases (from 3 - COMPARISON* to 356 - PART*).

One might expect that the F_1 score increases with the number of examples - after all, having more training data for a class should increase the classifier's ability to correctly identify it. This is, however, not the case. The labels on which the classifier performs best come from all bands of frequency - some with as little as 6 training examples - DIET*, F_1 score: 1, others with as many as 317 training examples - MATERIAL, F_1 score: 0.85.

In contrast, some properties seem inherently difficult to classify, even when enough training data is available. This is the case of the two properties that label compounds where the internal semantic relation is either unknown (NO PROPERTY - 0.14, 40 `train` examples) or is too specific to warrant a separate category (OTHER PROPERTY - 0.07, 51 `train` examples). Given the heterogeneous nature of the compounds labeled with

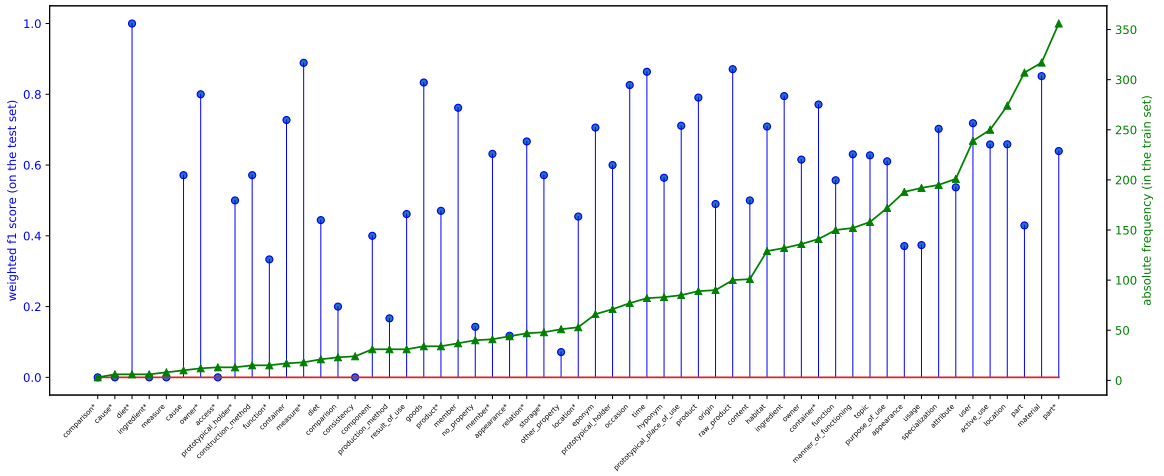


Fig. 7.4 Individual property classification: `test` F_1 score vs. `train` frequency.

these two properties, it is unlikely that a classifier can ever learn to distinguish them very well. While they make sense when designing an annotation inventory, one can hardly expect a classifier to correctly identify the characteristic features of such groups.

Other properties that have a low F_1 score despite a large number of training examples are APPEARANCE (F_1 score 0.37, 188 `train` examples), USAGE (F_1 score 0.37, 192 `train` examples) and PART (F_1 score 0.43, 307 `train` examples). As seen in Figure 7.6, and discussed in more detail below, these labels have a higher probability to be confused with other labels in the inventory, leading to worse classification results.

On the `test-iaa` subset the best classifier obtained an accuracy of 47.82% and an weighted F_1 score of 0.457. Figure 7.5 compares the weighted F_1 score on the `test-iaa` set with the amount of instances labeled with a particular property in the `train` set. The figure shows only the 43 individual properties that were used to annotate compounds in the `test-iaa` subset, and not the full set of 57 properties that are shown in the previous figure (7.4). Remember that the heads of the compounds in this dataset are not known to the classifier - the overlap with the compounds in the `train` data occurs only at the modifier level.

Figure 7.5 shows that the lack of information about the head has a greater impact on properties like HYPONYM (whose F_1 score drops from 0.56 to 0), FUNCTION (0.56 → 0.1), TOPIC (0.63 → 0.15), OWNER (0.62 → 0.17), PURPOSE OF USE (0.61 → 0.19) or PRODUCT (0.79 → 0.32). For example the lack of training data with the head *Klinik*, and the existence of the compound *Blutbank* 'blood bank' annotated as STORAGE* in the `train` set made the classifier infer that *Augenklinik* should also be labeled as STORAGE* (meaning *a place for storing eyes*) rather than as TOPIC (*clinic for eye treatment*), the correct label.

However, the lack of head information seems not to affect labels like COMPARISON (e.g. *Rabensohn* 'bad, uncaring son', lit. 'raven son'), RELATION* (e.g. *Mannschaftskapitän* 'team captain') or TIME (e.g. *Winterreifen* 'winter tire'). The reason for this discrepancy is the strong association between these labels and particular modifiers,

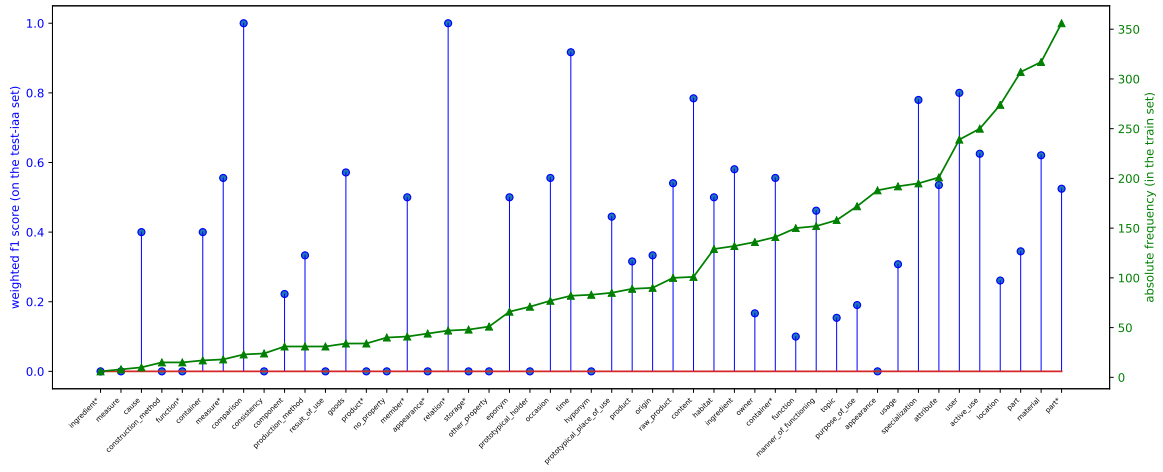


Fig. 7.5 Individual property classification: `test-iaa` F_1 score vs. `train` frequency.

which make it possible to identify the correct label while relying mostly on the information provided by the modifier. E.g. the `train` subset contains 5 other compounds with the modifier *Winter* 'winter', all labeled with the property TIME.

Figure 7.6 shows the confusion matrix for the `test` set. Because of the large difference in the amount of `test` instances annotated with each property, it was more illustrative to normalize the counts per property. This makes it easy to figure out what proportion of the `test` compounds labeled with a particular semantic property were also predicted to have that semantic label. E.g. the classifier predicted the label APPEARANCE for 0.43 of the compounds annotated as APPEARANCE, while the PART label was incorrectly predicted for 0.17 of the APPEARANCE compounds. The ideal case are properties like DIET*, who has a 1.00 on the main diagonal. This means that all the compounds that had been annotated as DIET* were also correctly predicted as DIET* by the classifier.

A number close to 1 on the main diagonal suggests therefore that the classifier had no trouble identifying that property, whereas a small proportion or nothing at all on the main diagonal means that the classifier had trouble recognizing the traits of that property and cannot easily distinguish them from the characteristics of other properties.

Several points can be made based on the information in Figure 7.6. First, 11 of the 57 properties were badly misclassified. The properties ACCESS* (13), CAUSE* (6), COMPARISON* (3), CONSISTENCY (24), INGREDIENT* (6), MEASURE (8) were fully misclassified, meaning that none of the `test` compounds annotated with this semantic property were correctly identified by the classifier. The properties APPEARANCE* (44), COMPARISON (23), NO PROPERTY (40), OTHER PROPERTY (51) and PRODUCTION METHOD (31) were misclassified in a very large proportion - over 85% of the data was assigned an incorrect label by the classifier.

Many of these properties are cases where having more training data would most likely improve the classification results. E.g. MEASURE, a property labeling compounds

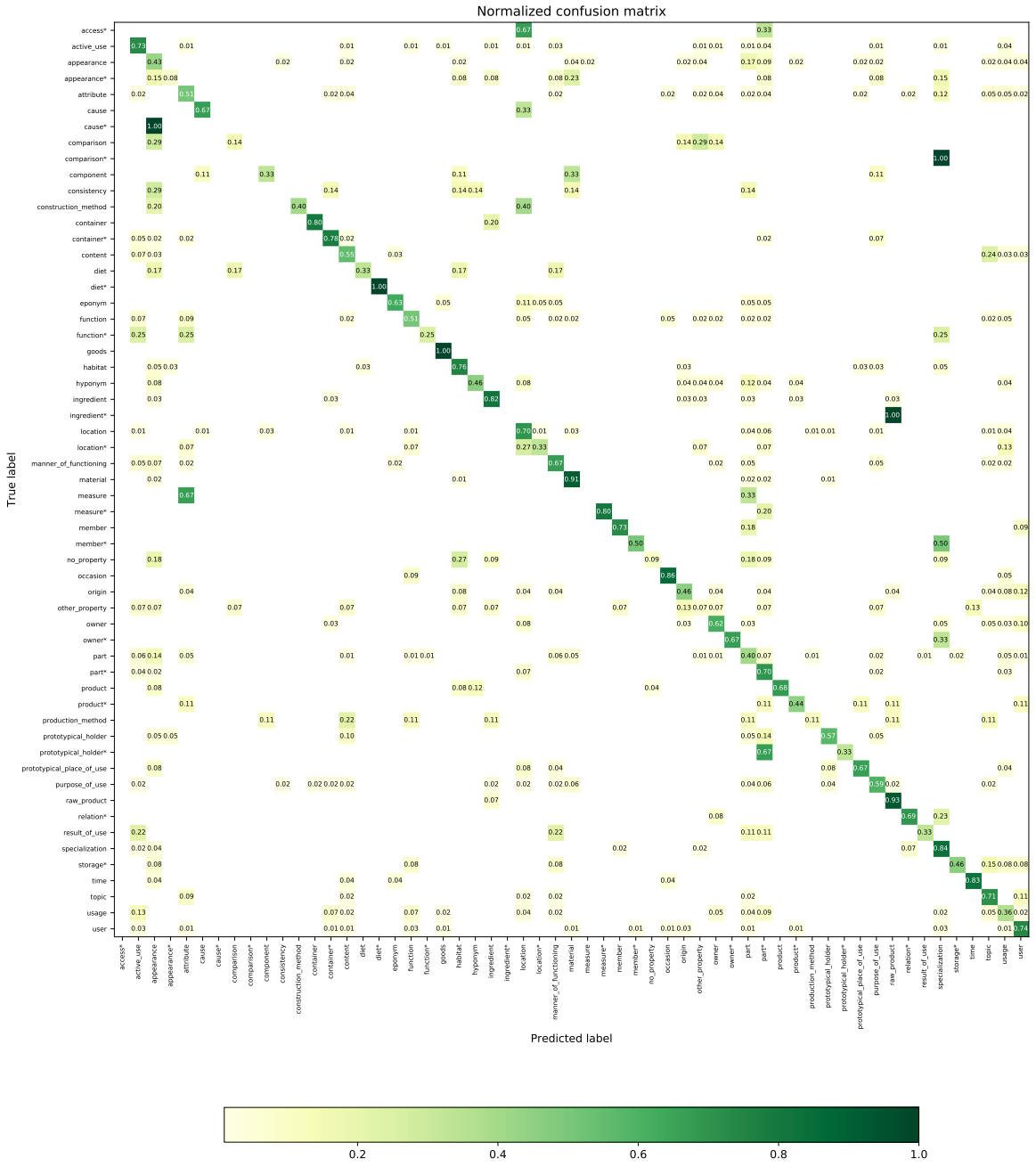


Fig. 7.6 Confusion matrix on the `test` subset for the best performing model on individual property classification. Using as input the composed `multimatrix` vector and the modifier vector (`modifier; composed_multimatrix`) in Table 7.6.

like *Literflasche* 'liter bottle', is a clear compounding pattern with a smaller number of instances, given its specificity (the modifier has to be a measure for the head). However, it will always have far fewer instances than a more productive pattern like PART.

As discussed above, the NO PROPERTY and OTHER PROPERTY labels will probably remain difficult to classify given the heterogeneity of compounds that are assigned these labels.

A second subset contains 14 properties which were correctly classified only in a relatively small proportion - up to 50%: APPEARANCE (188), COMPONENT (31), CONSTRUCTION METHOD (15), DIET (21), FUNCTION* (15), HYPONYM (83), LOCATION* (53), ORIGIN (90), PART (307), PRODUCT* (34), PROTOTYPICAL HOLDER (71), RESULT OF USE (31), STORAGE* (48), USAGE (192). While in some cases the limited amount of data is the cause of the poor performance of the classifier, this subset also contains a selection of properties with a relatively high number of training examples - like APPEARANCE - 188 examples, USAGE - 192 examples or PART - 307 examples. Why would the classifier have trouble with them? The answer: because of their generality, these properties are easier to confuse with other properties. E.g. the compound *Ringbuch* 'ring binder' is labeled with the semantic property PART, since the *Ring* 'ring' is part of the *Buch* 'book'. The classifier, however, chooses the property APPEARANCE as a more likely candidate, given the `train` examples *Ringmauer* 'ring wall' and *Ringbahn* 'circular railway'. The interpretation that would be assigned in this case would be that of a book that is shaped as a ring - a rather unlikely artifact. However, is it important to note that the pattern 'ring + artifact' meaning 'artifact with a circular/ring shape' is not unusual, as illustrated by the existence of other examples like *Ringbürste* 'ring brush' or *Ringdichtung* 'plug washer'.

The third subset are the 32 properties for which over 50% of the instances received the correct label, where the classifier was able to identify enough of the main traits of the properties to make the identification possible in the majority of cases.

The confusion matrix also revealed that sometimes the classifier has trouble distinguishing between the direct and the inverse relation: e.g. the classifier predicts for the compound *Straßentunnel* 'street tunnel' the semantic relation LOCATION, which would correspond to the interpretation 'street located in a tunnel', but the correct semantic relation is LOCATION*, with the interpretation 'tunnel that is located on a street'. Another frequent confusion is between PART and PART*: e.g. *Fensterfront* 'window facade' is automatically labeled as PART*, an interpretation where the head *Front* should be part of the modifier *Fenster*, while in reality the relation is reversed - the *Fenster* is part of the *Front*. Correctly establishing the directionality of the relation in these cases would require an amount of world knowledge that the classifier is currently unable to derive directly from the word vectors.

The impact of the amount of training data To asses the impact of the amount of training data on classification performance, the best performing classifier was retrained using 25%, 50% and 75% of the original `train` subset. These percentages correspond to

1316, 2632 and 3948 compounds, respectively, given that the full `train` subset contains 5265 compounds.

As Figure 7.7 shows, there is still a relatively large gap between the `train` and the `dev`/`test`/`test-iaa` accuracies. This indicates that the classifier suffers from a high variance problem, and that the addition of new training examples will improve the quality of its predictions.

While increasing the size of the training set is left for future work, the amount of manual work can be drastically reduced by using the best classifier to predict the semantic relation in new compounds and then manually correcting the classification output.

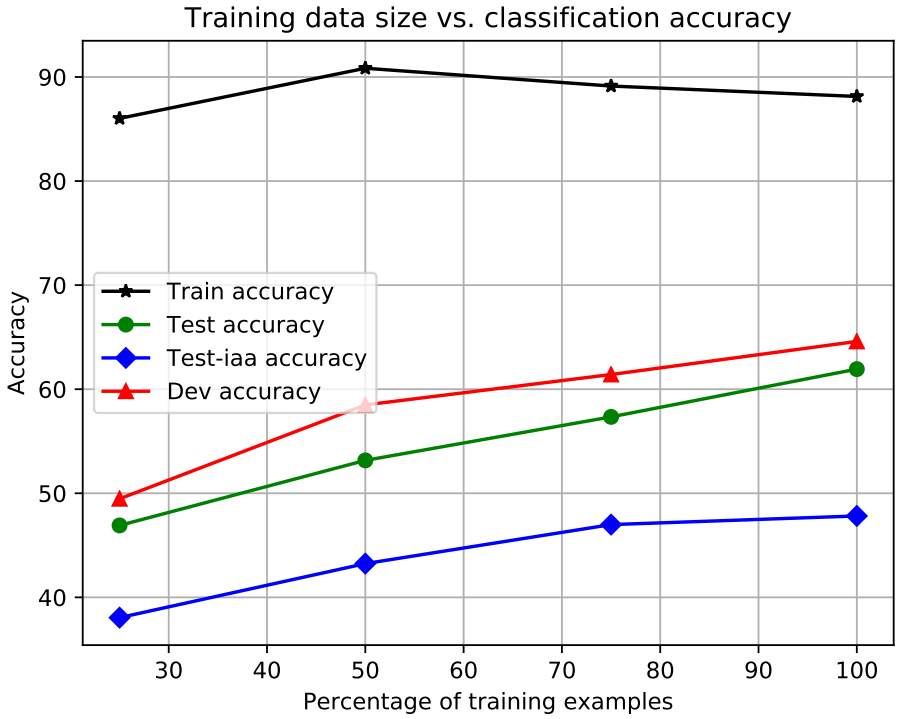


Fig. 7.7 More training data increases the `test` accuracy of the classifier. The classifier has a high variance problem, in which case adding more training data is likely to improve the classification results.

7.1.4 Classification using collapsed properties

The results presented thus far focused on classifying compounds according to each of the individual semantic properties in the annotation inventory. However, as discussed above, and also in more detail in Section 2.2, 14 of the 57 properties have an inverse property in the inventory.

This section presents experiments using 43 collapsed categories, where direct property and their inverses (like MEMBER and MEMBER*) are considered to be one property. The goal of these experiments is to find out if collapsing the categories offers any advantages from a semantic perspective - i.e. if it is easier to recognize that two entities enter a particular semantic relation without having to decide on the directionality of the relation. Another practical reason for collapsing the categories is that, as pointed out in Section 7.1.1, Fig. 7.2, each collapsed category will have more training examples than the individual categories. As shown in the previous section, increasing the number of training examples per property should improve the classifier's performance.

Baselines The eight baselines computed in the case of individual property classification were also used for evaluating the difficulty of the collapsed property classification task. A marked increase, compared to the previous baseline, is observed, as expected, in the case of the overall most frequent property baseline (#2). The collapsed PART/* property labels 12.29% of the total dataset, and this high percentage is apparent in the baseline results presented in Table 7.7.

#	Baseline	test		dev		test-iaa	
		Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
1	`random property`	2.11%	0.024	2.48%	0.029	2.45%	0.030
2	`mf prop overall` (`mf_o`)	12.56%	0.028	12.60%	0.028	7.69%	0.011
3	`mf prop modifier` (`mf_m`)	33.49%	0.322	31.13%	0.303	27.46%	0.263
4	`mf prop head` (`mf_h`)	36.37%	0.337	35.49%	0.318	7.69%	0.011
5	`prop_m`∩`prop_h` if $\neq \phi$ else `mf_o`	34.88%	0.355	36.67%	0.370	7.69%	0.011
6	`prop_m`∩`prop_h` else `mf_m` else `mf_o`	39.08%	0.385	39.46%	0.391	27.23%	0.260
7	`prop_m`∩`prop_h` else `mf_h` else `mf_o`	47.71%	0.462	46.95%	0.447	7.69%	0.011
8	`prop_m`∩`prop_h` else `mf_m` else `mf_h` else `mf_o`	**47.74%**	0.463	**47.14%**	0.449	**27.32%**	0.261

Table 7.7 Baselines for **collapsed property classification**.

For the strongest baseline (#8), however, the ∼1% increase in `test` accuracy (from 46.63%, in Table 7.3, to 47.74%) is small when considering that by using this set of labels the classifier does not have to guess an important piece of information - namely the directionality of the semantic relation. Overall, the decrease in the number of properties does not lead to a more simple classification problem - the complexity of the classification decisions remain largely on the same level as in the individual property classification case. The experiments presented next follow the same classification setup used before in the individual property case.

Table 7.8 displays the results of classifying compounds according to the 43 collapsed semantic properties. The input is, for the first section of Table 7.8, the representation of the compound - either taken directly from the learned word representations (for `compound_original`, `head` and `modifier`) or composed using either `addition` or `multimatrix`. The classifier used in this case is the `SMLP` classifier from Figure 7.3a.

The results in the second section of Table 7.8 always use two vectors as input: either the representations of the two constituents, `modifier` and `head`, or a `multimatrix`-

Classif.	Input	Dr 1	Dr2	test		dev		test-iaa	
				Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
SMLP	compound_original	0	0.4	37.74%	0.361	42.97%	0.404	32.02%	0.291
	head	0	0	34.49%	0.303	36.87%	0.319	25.16%	0.194
	modifier	0	0.4	38.14%	0.355	38.99%	0.367	30.98%	0.280
	compound_addition	0	0.3	56.81%	0.557	61.41%	0.596	34.51%	0.320
	compound_multimatrix	0	0.3	57.34%	0.559	60.61%	0.589	40.33%	0.387
DMLP	modifier; head	0.2	0.2	**62.66%**	**0.615**	**67.37%**	**0.659**	45.95%	0.434
	compound_multimatrix; head	0	0	55.61%	0.543	60.08%	0.580	42.41%	0.395
	modifier; compound_multimatrix	0.2	0.2	62.33%	0.613	66.31%	0.649	**48.44%**	**0.468**

Table 7.8 Results for **classification on 43 collapsed semantic properties**, using as input either one representation (original, or obtained using a composition model) or two concatenated representations.

composed representation of the compound and one of the constituent representations. The classifier used in this case is the DMLP classifier from Figure 7.3b.

The overall trend when comparing the results in Table 7.8 to those obtained for the classification of individual property is that collapsing the categories only results in minor improvements. The best result, obtained using the modifier and head representation as input to the DMLP classifier is 62.66% accuracy and 0.615 F_1 score on the test subset, is slightly higher than the best result on the individual property classification task (61.93% accuracy, 0.608 weighted F_1 score).

The difficulty of the classification task does not show a significant decrease when collapsing the 14 bi-directional properties. This results suggests that the difficulty of a classification task relates also to the homogeneity of the examples annotated with each label. In the individual property classification case the classifier had to decide among more semantic relations, but the class membership was more clear. In the collapsed property classification case, there are fewer semantic relations to choose in between, but the class itself is more heterogeneous, making the class boundaries less crisp.

In light of these results, an annotation inventory with multiple, well-defined semantic relations is preferable to an inventory containing fewer semantic relations that label a more heterogeneous set of examples.

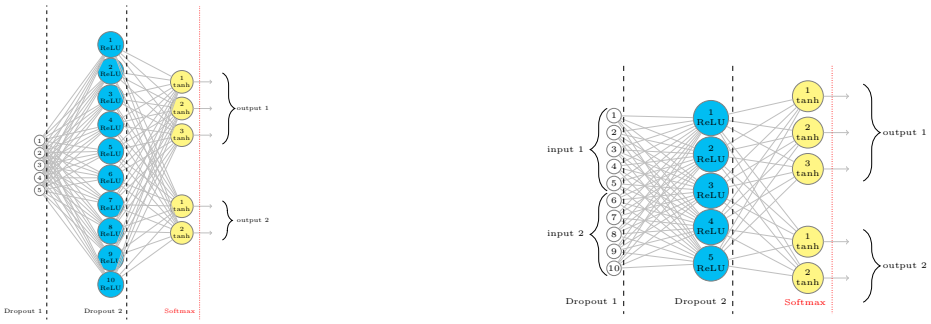
Integrating information from prepositions The German dataset was annotated with hybrid labels - a property and preposition combination. As detailed in Chapter 2, Section 2.2, in its instantiation for German the annotation inventory contains 19 unique prepositions. The annotators were instructed to select the preposition that felt the most natural for paraphrasing the compound.

For example, *Armbanduhr* 'wrist watch', lit. 'bracelet watch' is annotated with the hybrid label (PART, *mit* 'with'), because the preposition *mit* is used to the paraphrase the compound as *Uhr mit Armband* 'watch with bracelet'. In a small number of cases (206 out of the 8005 compounds in the annotated dataset), multiple prepositions were considered to be equally good for paraphrasing particular compounds. *Kartoffelsalat* 'potato salad', whose hybrid label is (INGREDIENT, (*aus* 'of', *mit* 'with')) is a point in case: it can be paraphrased either as 'salad (made) of potatoes' or as 'salad with

potatoes'. Given the small percentage of multi-preposition annotations a single, random preposition[4] was considered for each of the multi-preposition compounds.

Sorokin et al. (2015) showed that the results of semantic property classification can be improved by using a single classifier to predict both the compound-internal semantic property and the preposition that is most frequently used to paraphrase the compound. The assumption of such **multitask learning** (Caruana, 1997) setups is that one can improve the classification performance of a task of interest by making use of **related tasks** as auxiliary classification targets.

In the `de-nncom-sem` case, each compound is annotated both with a semantic relation and a preposition. Given that the main focus of this chapter is predicting the semantic relation between the constituents of a compound, the preposition prediction task was used as an auxiliary task. The classification architectures used in the previous sections, `SMLP` and `DMLP` (from Figures 7.3a and 7.3b), were updated to include two separate output layers: one predicting the semantic relation (from the 57 individual properties in the inventory) and one predicting the preposition (from the 19 available prepositions). The errors from the two classification systems are joined together in a common error signal. The errors were weighted such that the property classification contributed 0.9 and the preposition classification 0.1 to the total error. At the same time, the errors from both classifiers are backpropagated through the shared hidden layer, making adjustments for each of the classification tasks. Details about the results with different dropout rates are available in Appendix B, Table B.11. Figures 7.8a and 7.8b illustrate the updated architectures.



(a) Single input classification network with two output layers(`SMLP20`)

(b) Two input classification network with two output layers(`DMLP20`)

Fig. 7.8 Different classification architectures. Leftmost: the input layer; inputs are displayed as small while circles. Middle: hidden layer; hidden units are displayed as large blue circles with `ReLU` activation. Right: output layer; output notes displayed as yellow circles with `tanh` activation.

The results in Table 7.9 show that the use of preposition prediction as an auxiliary task for individual property classification did not result in a significant improvement. The classifier using the concatenated `modifier`; `head` representation obtained, in the

[4] A random preposition from the ones manually assigned to that compound by the human annotators.

Input	Dr 1	Dr2	test		dev		test-iaa	
			Acc.	wF_1	Acc.	wF_1	Acc.	wF_1
`compound_multimatrix`	0.2	0	56.15%	0.545	59.42%	0.570	40.96%	0.380
`modifier`; `head`	0.2	0.2	**62.13%**	**0.609**	**65.38%**	**0.637**	44.70%	0.427
`compound_multimatrix`; `head`	0.1	0.1	54.42%	0.528	57.96%	0.553	39.71%	0.377
`modifier`; `compound_multimatrix`	0.1	0.1	60.47%	0.592	64.59%	0.630	**48.86%**	**0.467**

Table 7.9 Results for **individual property classification**, using the prediction of the **preposition** associated with the compound as an auxiliary task.

multitask setup, an F_1 score of 0.609, compared to the 0.597 F_1 score obtained by the single-task individual property classifier. For the `compound_multimatrix` and `modifier`; `compound_multimatrix` case, the performance of the classifier is rather degraded through the addition of the extra output - from 0.550 to 0.545 F_1 score for `compound_multimatrix` and from 0.608 to 0.592 for `modifier`; `compound_multimatrix`.

A likely reason for this result is that both the single and the multi-task classifiers rely on the same input representations which, in contrast to Sorokin et al. (2015)'s setup, does not include co-occurrence information of nouns with the prepositions in the dataset. The word representations capture global co-occurrence patterns, and are unlikely to focus on prepositions as disambiguating contexts. Prepositions co-occur frequently with many other words, making them of peripheral importance in the representation of most nouns.

Another way to use the preposition annotations would be to include the gold preposition as an additional input when classifying the semantic relations. While adding information about the gold preposition might bring improvements - cf. Girju et al. (2005), it detracts from the generality of the approach, as the classification relies on the existence of this gold preposition annotation. The preposition classification resulted in an accuracy of ∼60%, making the use of automatically generated preposition labels not a viable option. Further research is needed to improve the classification in this case. Chapter 8 contains suggestions for alternative types of context that could supplement the input information and lead to improvements both in preposition classification as a separate task and the multi-task setup.

7.2 Automatic Interpretation of English Compounds

The automatic labeling of the semantic relations in English noun-noun compounds has been an active field of investigation, and several datasets that identify compound-internal relations have been made available. In their overview Tratz and Hovy (2010) cite 8 datasets, containing a range of annotations: from 385 to 2169 annotated compounds. Tratz and Hovy (2010) go on to propose their own annotated dataset, containing 17509 compounds annotated with 43 semantic relations. The dataset was updated in Tratz (2011) to contain 19158 compounds annotated with 37 semantic relations. Given that machine learning models require many training examples to make good predictions, the experiments in this section will focus exclusively on the Tratz (2011) dataset, the largest dataset of English compounds annotated with semantic relations.

7.2.1 The Tratz (2011) Dataset

The Tratz (2011) dataset contains 19158 compounds, 3446 unique modifiers and 3375 unique heads. There are, on average, 5.68 compounds with the same head and 5.56 compounds sharing the same modifier. *System* is the most frequent head, forming compounds with 228 modifiers (e.g. *flight system*). The most frequent modifier is *government*, which forms compounds with 235 heads (e.g. *government activity*).

Table 2.3 in Chapter 2 displays the 37 semantic relations together with example annotated compounds. The Tratz (2011) relation inventory makes similar distinctions to the ones made by the inventory used for the German `de-nncom-sem` dataset. The category names chosen by Tratz (2011) overlap in a large proportion to the ones proposed in the inventory in Section 2.2. However, the type of semantic distinctions that dictate assigning a particular compound to a category or another are very similar for some relations, but diverge for others.

For example, the relation OWNER-USER from Tratz (2011)'s inventory, which labels compounds like *family estate* and *taxpayer money*, corresponds to two relations in the inventory introduced in Section 2.2 - namely the direct relation OWNER and the relation USER. Tratz (2011)'s LOCATION relation, however, labels compounds with more diverse relations, which in the inventory of semantic relations proposed in this thesis are more finely differentiated. E.g. in the English dataset *shipyard worker* is annotated as LOCATION whereas in the German dataset *Werftarbeiter* 'shipyard worker' is annotated as SPECIALIZATION. Because the annotation proposed here is head-driven, compounds like *Textilarbeiter* 'textile worker' and *Wissensarbeiter* 'knowledge worker' have the same label, SPECIALIZATION, while in the Tratz (2011) annotation *rail worker* is annotated as LOCATION, *food worker* as OTHER, *construction worker* as PERFORM&ENGAGE_IN and *city worker* as EMPLOYER. This approach makes it sometimes difficult to choose a single correct annotation: *hospital worker*, annotated as LOCATION, might as well be annotated as EMPLOYER. This type of ambiguity in the definition of relations can make it difficult for a classifier to make correct predictions.

Other compounds annotated as LOCATION in the Tratz (2011) dataset are *toilet paper* - annotated as PROTOTYPICAL PLACE OF USE in the German dataset, *garage door* - which would be annotated as ACCESS* in the German dataset. Compounds like *Bergdorf* 'mountain village' and *Körperhaar* 'body hair' would receive the annotation LOCATION in both datasets.

Dima (2016) used distributed word representations in combination with composition models to predict the 37 relations for the compounds in the Tratz (2011) dataset. However, the results reported by Dima (2016) point to a problematic aspect of the Tratz (2011) data: the relations for which the classifier performs the best are also the ones with a small number of unique heads or unique modifiers in the dataset. For example the property ADJ-LIKE NOUN only has 7 distinct modifiers for 254 compounds, while AMOUNT_OF has 15 unique heads for 168 compounds. Both properties have an error rate of under 10%. The use of pre-trained composition models does not bring any improvements over the use of a simple classifier with only the concatenated constituent representations as input. Dima (2016) attributes this behavior to a process of *lexical memorization* (Levy et al., 2015b), the situation in which a classifier assigns a label based on the identity of a particular input - e.g. assigning the property ADJ-LIKE NOUN whenever one of the seven modifiers associated with ADJ-LIKE NOUN class occurs.[5] In effect, (Tratz, 2011:202) defines the ADJ-LIKE NOUN relation as *"in n_1n_2, n_1 is one of a handful of adjective-like nouns such as 'key', 'chief', and 'core"'*. This, however, makes the identification of the relation equal to memorizing which modifiers signal the semantic relation. Generalizing to unseen examples is impossible on datasets where the set of modifiers is disjoint between the training and the test examples. The particularities of such relations make them too 'easy' for the classifier to learn, and can lead to unrealistic estimations about the capacity of the classifier to capture more complex semantic relations and to generalize to unseen examples.

Dataset	`train`	`dev`			`test`		
	inst.	inst.	modifier overlap	head overlap	inst.	modifier overlap	head overlap
`tratz-fine, random`	14,369	958	84.10%	84.54%	3,831	77.52%	78.63%
`tratz-fine, lexical-mod`	9,783	5,400	0.0%	70.07%	3,975	0.0%	73.68%
`tratz-fine, lexical-head`	9,185	5,819	65.74%	0.0%	4,154	70.39%	0.0%
`tratz-fine, lexical-full`	4,730	1,614	0.0%	0.0%	869	0.0%	0.0%

Table 7.10 Dataset statistics for the four variations of the Tratz (2011) dataset proposed by Shwartz and Waterson (2018). The columns 'modifier/head overlap' show what percentage of the constituents in the `dev`, respectively `test` split occurs also in the `train` split. The `random` dataset has a very high constituent overlap, `lexical-mod` has no modifier overlap, `lexical-head` has no head overlap. The compounds in the `dev` and `test` splits of the `lexical-full` setup share neither modifier nor head with the `train` compounds.

Shwartz and Waterson (2018) follow up on the issues raised by Dima (2016) and, in testing their own methods for semantic relation identification, propose four dataset

[5]These are *chief*, *core*, *head*, *key*, *mainstream*, *mass*, *minimum*.

variations based of the original Tratz (2011) data: (1) `random`, where the entire dataset is split into `train`, `test` and `dev` portions (75:20:5); (2) `lexical-full`, where the `train`, `test` and `dev` splits have disjoint vocabularies; (3) `lexical-mod`, where the vocabularies of the `train`, `test` and `dev` splits are disjoint with respect to the modifier (i.e. a modifier that appears in `train` will not be part of compounds the `test` or `dev` splits); (4) `lexical-head`, where the vocabularies of the three splits are disjoint with respect to the head of the compound.

The experiments presented in this section report on these four dataset variations proposed by Shwartz and Waterson (2018). This will allow both for a better understanding of the real generalization capabilities of the different classifiers, and for a direct comparison of the results. As a note, Shwartz and Waterson (2018) also report results for classifying the compounds according to the 12 coarse relation groups defined by Tratz (2011) (these are the headings shown in bold in Table 2.3, Chapter 2, e.g. *Topic Group*). However, Shwartz and Waterson (2018)'s results when using the coarse relations, just as the experiments on collapsed property classification for German in Section 7.1.2, show only small improvements when compared to the fine-grained setup. Collapsing the categories is likely to have a blurring effect - i.e. more diverse compounds are assigned the same category, making it more difficult for the classifier to capture the defining traits of each label. In what follows, the experiments will concentrate on identifying the set of 37 fine-grained semantic relation proposed by Tratz (2011).

The statistics of the four dataset variations proposed by Shwartz and Waterson (2018) are shown in Table 7.10. The high constituent overlap of the `random` dataset is confirmed: 84.10% of the compounds in the `random` dataset's `dev` split have modifiers that occur in the `train` data, and 84.54% of the heads occur in `train`. The `lexical-mod` and `lexical-head` variations have zero overlap with the `train` data in terms of modifiers and heads, respectively. The `lexical-full` dataset has zero overlap with the `train` data both in terms of modifiers and of heads, but is at the same time the smallest of the four dataset variants, with only 7213 of the initial 19158 compounds (37.65%) from the Tratz (2011) dataset.

Baselines Table 7.11 presents the baselines for the four dataset variations. 8 different baselines are considered, as in the German dataset case. The average of 10 separate runs is reported for each baseline. The first two baselines are the classical random baseline (`random relation`) and most frequent baseline (`mf_o`). The next two baselines predict, for the compounds in the `test`/`dev` splits, the most frequent relation associated with that particular modifier (`mf_m`) or head (`mf_h`) in the `train` split. On the `random` dataset variation, the `mf_m` and `mf_h` baselines obtain already very good results: 0.345 and 0.532 in weighted F_1 score. However, on dataset variations where there is no overlap between the constituents in the `train` split and the ones in the `test`/`dev` splits, i.e. where there is no information about the tested constituents in the `train` set, these baselines revert to the most frequent relation baseline.

The next four baselines combine the separate information about the typical modifier relations and the typical head relations and take the most frequent relation that occurs

#	Baseline	test		dev	
		Acc.	wF_1	Acc.	wF_1
		tratz-fine, random			
1	random relation	2.73%	0.034	2.79%	0.034
2	mf relation overall (mf_o)	17.55%	0.052	17.14%	0.050
3	mf relation modifier (mf_m)	37.21%	0.345	37.49%	0.343
4	mf relation head (mf_h)	55.73%	0.532	53.93%	0.510
5	mf_m ∩ mf_h if $\neq \phi$ else mf_o	52.50%	0.519	48.64%	0.469
6	mf_m ∩ mf_h if $\neq \phi$ else mf_m else mf_o	57.67%	0.563	57.02%	0.547
7	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_o	64.10%	0.625	60.75%	0.585
8	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_m else mf_o	**66.17%**	**0.652**	**63.74%**	**0.624**
		tratz-fine, lex-mod			
1	random relation	2.74%	0.035	2.65%	0.033
2	mf relation overall (mf_o)	19.05%	0.061	17.73%	0.053
3	mf relation modifier (mf_m)	19.05%	0.061	17.73%	0.053
4	mf relation head (mf_h)	53.67%	0.517	**55.28%**	**0.533**
5	mf_m ∩ mf_h if $\neq \phi$ else mf_o	19.05%	0.061	17.73%	0.053
6	mf_m ∩ mf_h if $\neq \phi$ else mf_m else mf_o	19.05%	0.061	17.73%	0.053
7	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_o	53.65%	0.517	**55.28%**	**0.533**
8	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_m else mf_o	**53.79%**	**0.519**	55.26%	**0.533**
		tratz-fine, lex-head			
1	random relation	2.88%	0.036	2.68%	0.034
2	mf relation overall (mf_o)	17.12%	0.050	19.46%	0.063
3	mf relation modifier (mf_m)	36.28%	0.344	**36.39%**	**0.345**
4	mf relation head (mf_h)	17.12%	0.050	19.46%	0.063
5	mf_m ∩ mf_h if $\neq \phi$ else mf_o	19.05%	0.061	17.73%	0.053
6	mf_m ∩ mf_h if $\neq \phi$ else mf_m else mf_o	36.49%	**0.346**	36.34%	**0.345**
7	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_o	17.12%	0.050	19.46%	0.063
8	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_m else mf_o	**36.51%**	**0.346**	36.31%	0.344
		tratz-fine, lex-full			
1	random relation	2.35%	0.029	2.78%	0.037
2	mf relation overall (mf_o)	**17.97%**	**0.055**	**20.89%**	**0.072**
3	mf relation modifier (mf_m)	17.97%	0.055	20.89%	0.072
4	mf relation head (mf_h)	17.97%	0.055	20.89%	0.072
5	mf_m ∩ mf_h if $\neq \phi$ else mf_o	17.97%	0.055	20.89%	0.072
6	mf_m ∩ mf_h if $\neq \phi$ else mf_m else mf_o	17.97%	0.055	20.89%	0.072
7	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_o	17.97%	0.055	20.89%	0.072
8	mf_m ∩ mf_h if $\neq \phi$ else mf_h else mf_m else mf_o	17.97%	0.055	20.89%	0.072

Table 7.11 Baselines for the four Tratz (2011) dataset variations proposed in Shwartz and Waterson (2018).

with both modifier and head (baseline #5). Since the intersection of typical properties can be empty, the backup strategy is, again, to predict the most frequent relation overall. Baselines #6, #7 and #8 refine the backup strategy. For baseline #6 the first backup considered is the relation most frequently associated with the modifier, for #7, the one most frequently associated with the head while for #8 the backup strategy is first the most frequent relation of the head, than of the modifier.

The baselines show a marked decrease in performance as the amount of overlap between the constituents in the `train` split and those in the `test` and `dev` splits decreases. Moreover, the very high weighted F_1 scores of baseline #8 on the `random` dataset variation show already how much can be learned just from knowing the identity of the constituents and their most frequently associated relations in the `train` data.

In what follows the combinations of inputs and classifiers that were tested on the German dataset will be tried on the four variations of the Tratz (2011) data.

Word Representations The 300-dimensional, unit normalized (`L2-row`) word representations described in Section 5.5.2 were used as inputs for classifying the semantic relations in English compounds. The goal of the experiments is to identify the semantic relation between a compound's constituents while relying only on constituent information - i.e. without using the compound representation that was learned using the underscore trick (e.g. learn the representation of *apple tree* by recoding each appearance of the compound as *apple_tree*). The reason for this setup is that the majority of the compounds for which identifying the relation will be needed are the novel compounds - compounds where there is no large body of previous occurrences. Relying on information about the compound as a whole makes the approach not applicable in cases where it is most needed.

Classifiers and Inputs The `SMLP` and `DMLP` classifier from Figures 7.3a and 7.3b are retrained for predicting the semantic relations from the Tratz (2011) data.

Four input types are tested for the `SMLP` classifier: using only the `modifier` or the `head` representation, using the unit-normalized `addition` of the two constituent representations and using the composed representation obtained by applying the pre-trained `multimatrix` composition model to the modifier and head representations. The `SMLP` classifier has a $n \times 2n$ hidden layer size, and a $2n \times r$ output layer, where $n = 300$ is the dimensionality of the input representation and $r = 37$ is the number of semantic relations to be classified.

Three input variations are tested for the `DMLP` classifier: using a concatenation of the constituent representations (`modifier`; `head`); using the `multimatrix`-composed representation concatenated with the `head` representation, and using the `modifier` together with the `multimatrix`-composed compound representation. The `DMLP` classifier uses a $2n \times n$ hidden layer and a $n \times r$ output layer.

All the classifiers are trained using a negative log-likelihood criterion. The optimization is done using mini-batch Adagrad (Duchi et al., 2011), with a mini-batch size of 100 examples. The learning rate is initially set to 0.14, with a learning rate decay of $1e-5$. The models are trained using early stopping (Prechelt, 1998) on the `dev` set, with a patience of 100 epochs.

The seven classifiers were tried out on each of the four dataset variations. The only hyperparameters that were tuned for each classifier/dataset pair was the amount of dropout to be applied. Section B.4 of Appendix B details the performance of the classifiers with different dropout rates. As a rule, dropout rates from 0 to 0.4 were tried, in 0.1 increments. In cases where the best result was obtained using 0.4 dropout, further values were tested until a decrease in the `dev` set performance was observed. The best performing classifier - with the highest weighted F_1 score, was then tested on the `test` split of the data. Table 7.12 reports the accuracy and weighted F_1 score on both the `test` and the `dev` splits.

Dataset	Classifier	Input	Dr 1	Dr2	test		dev	
					Acc.	wF_1	Acc.	wF_1
tratz-fine, random		baseline #8	-		66.17%	0.652	63.74%	0.624
		Shwartz and Waterson (2018), Int	0.1		-	0.714	-	-
	SMLP	modifier	0.6	0.0	38.42%	0.343	40.50%	0.359
	SMLP	head	0.4	0.0	56.96%	0.547	56.26%	0.537
	SMLP	addition	0.2	0.0	69.38%	0.685	68.37%	0.676
	SMLP	compound_multimatrix	0.2	0.0	71.08%	0.702	70.67%	0.700
	DMLP	**modifier**; **head**	0.1	0.1	**77.92%**	**0.772**	**78.71%**	**0.783**
	DMLP	compound_multimatrix; head	0.1	0.1	72.93%	0.721	72.76%	0.715
	DMLP	modifier; compound_multimatrix	0.1	0.1	72.83%	0.721	73.07%	0.723
tratz-fine, lex-mod		baseline #8	-		53.79%	0.519	55.26%	0.533
		Shwartz and Waterson (2018), Int	0.1		-	0.613	-	-
	SMLP	modifier	0.4	0.0	29.76%	0.250	30.22%	0.256
	SMLP	head	0.4	0.0	54.79%	0.534	56.76%	0.553
	SMLP	addition	0.3	0.0	54.36%	0.533	55.31%	0.539
	SMLP	compound_multimatrix	0.2	0.0	58.99%	0.579	59.20%	0.579
	DMLP	**modifier**; **head**	0.1	0.1	**66.62%**	**0.656**	**68.37%**	**0.670**
	DMLP	compound_multimatrix; head	0.1	0.1	63.25%	0.620	64.67%	0.633
	DMLP	modifier; compound_multimatrix	0.2	0.2	62.14%	0.606	63.33%	0.616
tratz-fine, lex-head		baseline #8	-		36.51%	0.346	36.31%	0.344
		Shwartz and Waterson (2018), Int	0.1		-	0.510	-	-
	SMLP	modifier	0.3	0.0	37.17%	0.338	38.79%	0.352
	SMLP	head	0.4	0.0	25.25%	0.230	38.06%	0.360
	SMLP	addition	0.3	0.0	41.48%	0.392	43.48%	0.415
	SMLP	compound_multimatrix	0.0	0.0	48.36%	0.469	51.80%	0.508
	DMLP	modifier; head	0.3	0.3	52.50%	0.503	55.30%	0.530
	DMLP	compound_multimatrix; head	0.3	0.3	46.34%	0.449	49.91%	0.481
	DMLP	**modifier**; **compound_multimatrix**	0.0	0.0	**54.33%**	**0.531**	**55.97%**	**0.553**
tratz-fine, lex-full		baseline #2	-		17.97%	0.055	20.89%	0.072
		Shwartz and Waterson (2018), Int	0.1		-	**0.421**	-	-
	SMLP	modifier	0.2	0.0	27.73%	0.237	27.76%	0.251
	SMLP	head	0.4	0.0	23.13%	0.231	36.62%	0.355
	SMLP	addition	0.1	0.0	34.75%	0.321	37.42%	0.353
	SMLP	compound_multimatrix	0.2	0.0	39.82%	0.380	40.89%	0.395
	DMLP	modifier; head	0.3	0.3	43.61%	0.410	**47.58%**	0.446
	DMLP	compound_multimatrix; head	0.3	0.3	39.13%	0.380	43.25%	0.411
	DMLP	**modifier**; **compound_multimatrix**	0.3	0.3	**44.65%**	0.419	46.65%	**0.448**

Table 7.12 Semantic property classification on the English Tratz (2011) dataset, using the four variations proposed by Shwartz and Waterson (2018). The variations range from a datasplit with substantial constituent overlap (`random`) to no modifier overlap (`lex-mod`), no head overlap (`lex-head`) and no constituent overlap (`lex-full`). The performance of the classifiers decreases as the lexical overlap between the `train` data and the `test`/`dev` splits decreases. The results on the `lex-full` dataset are realistic estimates about the generalization capabilities of the classifier in the absence of overlap between the constituents in the `train` and `test` data.

Each of the four subsections of Table 7.12 focuses on one of the four dataset variations proposed in Shwartz and Waterson (2018). The first two lines in each of the section report the result of the best performing baseline, as well as the result obtained by Shwartz and Waterson (2018) for that dataset variation using their integrated classifier (`Int`).

Shwartz and Waterson (2018)'s integrated classifier uses as input a combination of the 300-dimensional distributional representations of the constituents and a path

representation. The path representation between the two constituents w_1 and w_2 is defined in terms of all the dependency paths that connect the two words in the corpus. A single path can contain multiple edges - i.e. a path for the compound *coffee cup* can be formed from the phrase *cup of coffee*. This path has two edges: one connecting *cup* to *of* and another connecting *of* to *coffee*. Each edge is represented by a 4-tuple containing its lemma, its part of speech, its dependency label and its direction vectors. Each path embedding is obtained by running an LSTM over the edge representations and taking the last output. The path representation of a compound is a 50-dimensional vector obtained by averaging over individual path embeddings, where each path is weighted by its frequency.

Shwartz and Waterson (2018) also propose a variant of the integrated classifier which uses the original compound representation in addition to the constituent and the path representations (`Int-NC`). However, the only setup where the original compound information helps is on the `tratz-fine, lex-full` dataset, where `Int-NC` outperforms the `Int` with a 0.008 increase in F_1 score (from 0.421 to 0.429). On the other dataset variants adding information about the original compound leads to a lower performance.

Returning to the classifiers tested in this thesis, the results of the `SMLP` classifier using as input the `modifier` and `head` representations reveal a complex picture about guessing the semantic relation using only information about one of the constituents.

On the `random` dataset, using the `head` representation leads to a substantial, 0.20 increase in F_1 score on the `test` split when compared to using the `modifier` representation - from 0.343 to 0.547. This seems to corroborate (Ó Séaghdha, 2008:97)'s observation that *"knowledge about compound heads is more informative for compound interpretation, at least when classifying with distributional information"*. The same trend is visible from the results on the other dataset variations: `modifier` drops 0.09 in F_1 score on the `lex-mod` dataset, where there is 0 modifier overlap. However, `head` drops an impressive 0.30 in F_1 score on the datasets for which there is 0 head overlap (`lex-head` and `lex-full`). Moreover, `head` also shows a large discrepancy between the results on the `dev` and `test` splits on the datasets with no head overlap (e.g. from 0.355 on the `dev` to 0.231 on the `test` of the `lex-full` dataset).

It must be noted, however, that both the results of the `modifier` and the `head` on the no-overlap dataset `lex-full` are above the best baseline (compare baseline 0.055 F_1 score to 0.237 of `modifier` and 0.231 of the `head`). In this setup - with no constituent overlap - both modifier and head information are equally informative, with the `modifier` having a slightly better performance.

The `addition` setup uses constituent representations that were first pairwise added and then unit-normalized. The `addition` representation preforms on all dataset variations better than or at least on par with the best performing constituent representation, while at the same time outperforming the baseline results on every dataset.

The `SMLP` classifier performs at its best using as input the `multimatrix`-composed representation. Although this representation starts off with the same provisions as `addition`, i.e. the representations of the two constituents, the value of pre-training the composition model becomes apparent, particularly in the setups where there is no

overlap in terms of the head constituent between the compounds in the `train` and `test` data splits (e.g. 0.392 for `addition`, but 0.469 for `compound_multimatrix` on the `lex-head` dataset).

Three inputs were tested with the `DMLP` classifier: using the concatenated `modifier` and `head` representations, or replacing one of the constituent representations with the `multimatrix`-composed representation. On the `random` and `lex-mod` datasets the best results (marked in bold in Table 7.12) were obtained using only the constituent representations. Although they show substantial improvements both from the baselines and from the previously reported results (0.772 weighted F_1 score on `random`, 0.656 on the `lex-mod` setup), they are probably not a realistic estimate of the classifier's real capacity to grasp the patterns underlying the semantic relations. To illustrate this point, it's worth comparing the results of the best performing classifier on the `random` dataset with that of the best performing classifier on the `lex-mod` dataset. What are the semantic relations that the classifier struggles to identify when switching from the `random` to the `lex-mod` dataset?

An example of such a relation is the EMPLOYER relation, which labels compounds like *hospital personnel*. Even as the head *personnel* occurs in the `train` set three out of eight times in a compound labeled as EMPLOYER, the best performing classifier assigns it the label LOCATION rather than EMPLOYER. The training data contains only one instance with the head *personnel* tagged as LOCATION - namely *ground personnel*.

A look at the `train` data can shed some light on this behavior. There are 50 unique modifiers in the 264 compounds labeled with the relation EMPLOYER in the `train` set, but none of them are similar to *hospital*. By contrast, there are 156 unique modifiers for the 527 compounds labeled as LOCATION in the `train` data, and several have similar meanings to *hospital*: *nursing home*, *pharmacy*, *hospice*. The classifier seems to be predicting the relation label that is closer to the modifier *hospital* it has encountered for the first time - and this turns out to be LOCATION.

For the dataset variations where there is no overlap between the head constituents in the `train` and `test`/`dev` data the `DMLP` classifier performs best when using the `modifier`; `compound_multimatrix` combination as input. The results, however, are far less impressive, and amount only to a 0.531 F_1 score on the `lex-head` dataset, and a 0.419 F_1 score on the `lex-full` dataset.

The semantic relations on which the classifier does well on the `lex-full` dataset are, in effect, the only ones where the classifier can be said to have generalized above the word level. These are MEASURE (F_1 0.765), SUBSTANCE-MATERIAL-INGREDIENT (F_1 0.723) and TIME-OF1 (F_1 0.652). By contrast, the relations on which the classifier struggles are the ones with a low F_1 measure, like MEANS (F_1 0), WHOLE+PART_OR_MEMBER_OF (F_1 0.1), PART&MEMBER_OF_COLLECTION&CONFIG&SERIES (F_1 0.19).

What makes these groups of relations different in the former group the majority of the modifiers belong to a well established class of things: i.e. they are measures (e.g. *inch water*, *percent change*, *mile road*), time expressions (e.g. *afternoon newspaper*, *fall semester*, *morning workout*) or substances/materials (e.g. *sand beach*, *aluminum can*, *leather jacket*). It is then of no surprise that such a tightly knit cluster of modifiers

makes the classifier's job easier, resulting in good predictions for similar pairs like *kilometer walk*, *tablespoon oil* for MEASURE, *summer session*, *summer tour* for TIME-OF1 and *glass bowl*, *steel wall* or *brick cabin* - for SUBSTANCE-MATERIAL-INGREDIENT.

These examples seem to illustrate the textbook version of **lexical similarity** (Turney, 2006; Ó Séaghdha, 2008), where compounds like *plastic knife* and *metal spoon* are similar because of the pairwise similarity of their constituents (*plastic*, *metal*) and (*knife*, *spoon*).

In contrast, the relations for which the classifier has a hard time making good predictions show a different pattern. The constituents come from more diverse word classes: e.g. the compounds *steam boat*, *bus trip*, *video conference*, all labeled as MEANS, have constituents from more diverse classes. While all express a means relation, the means has at the same time a very different interpretation, one that is particular for each head: the *boat* is powered by *steam*, the *bus* is the vehicle for the *trip* and the *video* names the medium of the *conference*. However, to analyze *plane ride* correctly one has to consider really only those compounds annotated as MEANS that have a head similar to *trip*. The compounds with semantically different heads but with the same relation are more of a distraction than of help in correctly analyzing *plane ride*. In effect, learning that a compound like *gas cooker* should also be annotated as MEANS seems impossible.

A similar situation occurs in WHOLE+PART_OR_MEMBER_OF's case. Many things can be described as wholes with parts: *whale tongue*, *album cover*, *reactor core*. Generalizing from such a diverse set of `train` examples to compounds like *train car*, *apartment window* and *audience member* is a task that the classifier using only distributional information as input seems ill-suited to solve.

From a theoretical perspective, what is still needed is to ensure that the classifier is aware of and can use the **relational similarity** (Turney, 2006; Ó Séaghdha, 2008) between the two pairs of constituents. Relational similarity entails that two pairs of words are similar if they occur in similar contexts. I.e. "My disposable *knife* is made of *plastic*" and '*Spoons* are made using high-quality *metal*.' However, this type of relational knowledge is, in many cases, encyclopaedic and thus seldom available in a large number of contexts. For example the statement *The whale has a tongue* is not frequently encountered, even as the size of the corpus grows. It's common-sense knowledge that mammals have tongues. The only highlight-worthy aspect is maybe the size of the whale's tongue, which can weigh as much as an elephant. Predicting the relation between *whale* and *tongue* might put the distributional classifier in a difficult situation - it needs to somehow learn, from the little training data that it has, that *whales* are mammals, that *tongues* are body parts and that the relation between a mammal and one of its body parts is WHOLE+PART_OR_MEMBER_OF, this, of course, while also figuring out the correct word classes and prototypical combinations that are as well tagged with WHOLE+PART_OR_MEMBER_OF (the *album cover*, the *reactor core*, etc.).

The English WordNet (Fellbaum, 1998) might be seen as an obvious source for this type of encyclopaedic information. Tratz (2011), in fact, used WordNet information -

such as the synonyms and hypernyms of a word, the terms in the words gloss, etc.[6] Shwartz and Dagan (2018) replicated the setup described in Tratz (2011) and report, however, only a 0.340 F_1 score for the `lexical-full` dataset, lower than the best result reported here, 0.419 and than Shwartz and Waterson (2018) best results, 0.429. A reason for this subpar performance is likely to be the limited coverage of such lexical resources, but also the fact that in the end the classifier still has to use a set of features to learn that a particular word belongs to the mammal class **during** the semantic relation classification task. Chapter 8 describes possible directions of research that can incorporate the information from such lexical resources while sidestepping the limited coverage issue.

Comparison to other Studies One of the earliest experiments in the automatic classification of English noun compounds is due to Lauer (1995), who reports an accuracy of 47% at predicting one of 8 possible prepositions in a set of 385 compounds. Rosario and Hearst (2001) obtain 60% accuracy at the task of predicting one of 18 relations using neural networks and a dataset of 1660 compounds from the medical domain. The domain-specific inventory Rosario and Hearst (2001) use was obtained through iterative refinement by considering a set of 2245 extracted compounds and looking for commonalities among them. Girju et al. (2005) use WordNet-based models and SVMs to classify nouns according to an inventory containing 35 semantic relations, and obtain accuracies ranging from 37% to 64%. Kim and Baldwin (2005) report 53% accuracy on the task of identifying one of 20 semantic relations using a WordNet-based similarity approach, given a dataset containing 2169 noun compounds.

Ó Séaghdha and Copestake (2013) experiment with the dataset of 1443 compounds introduced in Ó Séaghdha (2008) and obtain 65.4% accuracy when predicting one of 6 possible classes using SVMs and a combination of various types of kernels. Tratz and Hovy (2010) classify English compounds using a new taxonomy with 43 semantic relations, and obtain 79.3% accuracy using a Maximum Entropy classifier and 79.4% accuracy using $\text{SVM}_{multiclass}$ on their dataset comprising 17509 compounds and 63.6%(MaxEnt)/63.1%($\text{SVM}_{multiclass}$) accuracy on the Ó Séaghdha (2008) data. Dima and Hinrichs (2015) report 77.7% accuracy on identifying the 37 relations in the Tratz (2011) dataset using a combination of distributional representations as input to a neural network classifier. Dima (2016) reports an accuracy of 79.36% on the Tratz (2011) dataset, and 61.12% accuracy on the Ó Séaghdha (2008) data.

All these efforts have concentrated on English compounds, despite the fact that compounding is a pervasive linguistic phenomenon in many other languages. Recent work by Verhoeven et al. (2012) applied the guidelines proposed by Ó Séaghdha (2008) to annotate compounds in Dutch and Afrikaans with 6 category tags: BE, HAVE, IN, INST, ACTOR and ABOUT. The reported F_1-scores are 47.8% on the 1447 compounds Dutch dataset and 51.1% on the 1439 compounds Afrikaans dataset.

For German, Sorokin et al. (2015) report an F_1 score of 0.601 at the task of identifying one of the 38 semantic relations and 0.639 at the task of the 17 prepositions

[6] A complete overview is available in Chapter 2, p. 27.

defined in their hybrid inventory on a dataset of 4607 noun-noun compounds. Sorokin et al. (2015)'s system uses SVMs and a combination of distributional and knowledge-based features.

Semantic Relation Classification - Main Results The semantic relation classification experiments brought insights both into the complexities of identifying the semantic relation for a given compound and into the advantages but also the limitations of using composed representations as classification features.

The experiments have shown that using the composed representation as input when doing semantic relation classification is better than using any of the original, corpus-learned representations - either of the constituents or of the compound as a whole. Moreover, composed representations obtained via a trained composition model like `multimatrix` are superior to additive representations - in particular for compounds whose head constituent is not part of the training data.

The experiments also showed that the composed representations capture more information about the head than about the modifier, leading to a loss of expressiveness when used as features for a semantic task; a classifier combining the modifier and the composed representation is better suited to identify the semantic relation in compounds than classifiers that rely only on the constituent representations or that combine the head and the composed representation.

In the German case, more data is needed to improve the classification capabilities of classifiers. However, the experiments on the larger English dataset pointed to a possible pitfall that should be avoided when enlarging the dataset: the chosen data must not permit the classifier to perform lexical memorization, i.e. to learn a mapping from specific constituents to relation labels, by including too many examples with the same constituents and the same relation type. The most realistic classification setup is one where there is no overlap between the constituents of the compounds used for training and those used for the development and testing of machine learning classifiers.

The results of the classifiers on the datasets with little lexical overlap show that the task of semantic relation identification for noun-noun compounds is far from solved, obtaining F_1 scores around 0.5. The best classifiers are able to identify relations where the constituents come from closely knit classes with high attributional similarity like MATERIAL or TIME, but break down for more open-ended relations like PART OF. In most cases the classifiers have a hard time grasping the defining features of each semantic relation directly from the distributional representations.

Using fewer categories that conflate several patterns does not lead to a sizable improvement in classification performance, even when the difference is only the directionality of the semantic relation. Having many well defined categories is therefore preferable both from a semantic and from a computational perspective.

Chapter 8

Conclusions

8.1 Contributions

This thesis showed that neural network-based composition models can be successfully used to represent and interpret the semantics of German and English nominal compounds. The experiments showed that composition models produce versatile compound representations, useful for distinguishing compound-internal semantic relations and for recognizing lexicalized compounds. The main contributions of this thesis are summarized next.

A hybrid annotation scheme and a dataset of semantically annotated German compounds. The hybrid annotation scheme described in Section 2.2 classifies compounds in terms of a combined label consisting of a property and preposition. The work on the hybrid annotation scheme was part of the SFB 833 A3's project work, and several colleagues contributed. An initial version of the annotation scheme was developed with input from Prof. Dr. Erhard Hinrichs, Dr. Yannick Versley, Dr. Verena Henrich and Christina Hoppermann. The second iteration greatly benefited from the valuable insights of late Dr. Heike Telljohann. The annotation of the compounds in the dataset was performed in parallel by one of the students helping with the annotation, Nadine Balbach, Tabea Sanwald and Kathrin Adlung, and one of the experienced annotators, Christina Hoppermann and Dr. Telljohann.

Since the author is not a native speaker of German, her contribution focused mainly on the theoretical aspects of developing the annotation scheme, i.e. deciding about which categories should be included, and how should these categories be defined and differentiated from other categories, and computing the inter-annotator agreement. Another contribution was developing a web-based annotation tool that allowed versioning the annotation associated with each compound as the annotation scheme evolved.

New composition models: `addmask`, `wmask`, `multimatrix`. The experiments in Chapter 5 showed that composed distributional representations are a viable way of representing compounds. Surveying the performance of existing composition models

indicated several shortcomings of existing models: they were either discarding useful information (`lexfunc`), had an extremely large number of parameters and worked only for the constituents in the training data (`lexfunc`, `fulllex`) or used a one-fits-all composition (`matrix`, `fulladd`).

The new models addressed the limitations of existing models both by training dedicated vector masks for each vocabulary word, like the `addmask` and `wmask` models, and by allowing the input representations to be transformed in multiple ways and then integrating the different transformations, like the `multimatrix` model.

`multimatrix`, the best performing composition model, was able to build meaningful composed representations for 81.8% of the German test compounds and 78.03% for the English test compounds. The proposed composition models are, however, not specific to compounds - they can in theory be used to create representations for other binary constructions - e.g. for adjective-noun phrases or prepositional phrases - provided that the necessary training data is available. Their usefulness for each new construction must, however, be separately investigated.

Composed representations to identify lexicalized compounds The experiments in Chapter 6 showed that lexicalized compounds can be identified by comparing the composed and the observed representation of a compound. While definitive guidelines for annotating lexicalized compounds are still a desideratum, using the representations created by a composition model can streamline the process of automatically identifying larger sets of potentially lexicalized compounds.

Composed representations as features for semantic relation classification. Chapter 7 has shown that composed representations are the best representation choice for compounds in semantic tasks where simplex and complex words should have representations of the same length. Using a `multimatrix`-composed representation gave substantially better results than using the original representation when classifying the internal relations in German compounds (compare 0.550 to 0.367 `test` F_1 score). The results are particularly compelling on the subset of compounds used for the inter-annotator agreement, which shared none of the head constituents with the test set - compare 0.411 F_1 to 0.274 `test-iaa` F_1 score.

Extrinsic evaluation of composition models The analysis of the semantic relation classification task in Chapter 7 showed that some of the modifier information which is needed for identifying semantic relations is discarded during composition. The experiments showed that evaluating composed representations in an extrinsic setting brings useful insights and should be an integral part of evaluating the performance of composition models.

This thesis has lead to the development of a number of resources made available to the scientific community.

- **Torch7 implementations of the composition models**, available on the author's GitHub page[1]
- **Word representations for German and English** - containing representations of words and compounds in different dimensions[2].
- **Composition datasets** for benchmarking composition models for English and German compounds[2].
- **Semantically annotated dataset of German compounds**, containing 8005 compounds annotated with semantic relations and prepositions[2].

8.2 Looking Back, Looking Forward

Looking back, this thesis did not yet provide an answer on the matter of how many semantic relations can be found between the constituents of a compound. The proposed hybrid annotation scheme contained 57, but, as mentioned, the inventory of semantic relations was constructed bottom-up using the compounds in the German dataset.

The parallel investigation of semantic relations and composition models meant, however, that this question was at the periphery of every analysis, especially those involving the ∼30,000 compounds used in the composition experiments, and the multitude of additional compounds in the ∼1 million words distributional vocabulary.

There are, undoubtedly, more semantic relations to be recognized if a larger scale analysis is undertaken. They form, most likely, a *"continuum of patterns"*, as described by Ryder (1994): some with many instances and possibly even sub-patterns, like PART, and some characteristic only for a handful of compounds, like MEASURE. While their manual annotation is a prohibitively expensive undertaking, the use of distributional word representations in combination with machine learning algorithms might allow for the problem to be turned on its head and framed as **discovery of relations** instead of **classification of relations**.

If one looks at deictic compounds, the number of possible semantic relations is probably infinite, as postulated before by Downing (1977) and Finin (1980). However, discovering most of the typical, habitual relations that form the basis of creating new compounds that have a naming function should be an attainable goal.

Looking forward, the distributional representations used for representing words need to become better. As the results in Chapter 7 showed, the classifier using distributional representations as input (both original and composed) had difficulties in recognizing

[1] `https://github.com/corinadima`

[2] Word representations and datasets available through TALAR, the Tübingen Archive of Language Resources, at `http://hdl.handle.net/11022/0000-0007-CFE2-1`.

the defining traits of the different semantic relations when working with disjoint sets of training and test constituents. The classifiers were expected to know when a particular representation belonged to an artifact, a shape, a group, a body part, an animal, a plant, a human, an organization, a place, an edible thing etc. These are, as Warren (1978) showed, the type of features that people instinctively use when interpreting compounds. While distributional representations of words might inherently capture this information, semantic tasks require a more direct access to categorizations of words in terms of such microfeatures (Rumelhart et al., 1986b).

Recent research (Faruqui et al., 2015; Speer and Lowry-Duda, 2017) has shown the benefits of incorporating information from external knowledge source like WordNet (Fellbaum, 1998) and ConceptNet (Speer et al., 2017) into word representations. A possible way to improve the performance of classifiers that use distributional word representations as inputs is to first pretrain them to learn how to distinguish microfeatures like the ones described above. In a subsequent step the classifier is aimed at the end-task, possibly using a progressive network architecture like the one proposed by Rusu et al. (2016).

Another aspect that noun representations need to become more sensitive to are their lexical preferences. The hybrid label annotation showed that semantic relations and prepositions have a mutual disambiguation effect when interpreting noun-noun compounds. However, to be able to bank on this mutual disambiguation, word representations of nouns need to be sensitive to the preposition co-occurrences. Dependency-based embeddings like the ones proposed by Levy and Goldberg (2014a) allow for a more deliberate choice of the context that a word representation is based on. For example the context can be restricted only to words with particular parts of speech and/or words that are connected via specific dependency relations. Such representations are better suited to model the preferences of nouns with regard to prepositions, and should provide additional information both for composing and interpreting compounds.

The discovery of semantic relations does not concern only the internal semantics of compounds. The CONTAINER semantic relation in the compound *tea cup* can also be expressed via the prepositional paraphrase *cup for tea*. Because "*compounding straddles the boundary between morphology and syntax, behaving in some ways like word-formation and in others like phrase-formation*" (Bender, 2013:15), the lessons learned for compounds might be of help in investigating particular types of phrases. From a computational perspective, one could argue that just like simplex words and compounds should have comparable representations (remember the example from Chapter 5 - the representation *programmer* should be comparable to that of *software developer*), compounds and their prepositional paraphrases should also have comparable representations - i.e. the representations of *tea cup* and *cup for tea* should be comparable and possibly even interchangable.

Composing representations for such phrases is not a trivial task, for a careful analysis of the semantics of the different prepositions is needed. A statement like **I invited my friend over for a cup for tea* is wrong because *cup for tea* expresses a CONTAINER relation (in the example one could substitute *cup* for the phrase *cup for*

tea). A friend is usually invited over for 'consumables' like beverages, drinks, talks, etc, not for 'things' like *cups*. The correct version is *I invited my friend over for a cup of tea*, where *cup of tea* expresses a MEASURE relation, and where the phrase *cup of tea* could be substituted for *tea*.

This parallelism between the compound and the semantics of its *for* and *of* paraphrases can be observed for other examples as well: *wine bottle* - *bottle for wine* (CONTAINER); *bottle of wine* (MEASURE), *beer glass* - *glass for beer* (CONTAINER), *glass of beer* (MEASURE). It suggests that, like the semantic relations in compounding, phrases of this type are based on patterns and could thus be automatically identified and suitably represented using machine learning techniques.

The construction of vector representations for complex words starting from the representations of the parts that constitute them was the central topic of this thesis. Composition models are, however, not specific to compounds. The models presented in Chapter 5 can be readily applied to phrases made of two components, e.g. adjective-noun phrases like *white car*. However, allowing only for fixed size inputs - i.e. of length 2, 3, or more, can lead to arbitrary distinctions in some cases - e.g. the representations of *tea cup* and *cup for tea* would have to be constructed by different composition models because of their mismatch in length - despite their similar semantics.

Recurrent neural networks like LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014) are capable of creating representations for inputs of variable length. Such networks build the representation of a longer phrase incrementally, processing a word from the phrase at each step. The representation they build at a given step depends both on the previous state (i.e. the words from the input that the network has already seen) and on the current input (the current word). Using such models for composition would remove the artificial constraints imposed by architectures which only support a fixed number of inputs.

The composition models in Chapter 5 were applied for German compounds like *Apfelbaum* 'apple tree' by using the splitting information available in GermaNet. However, the splitting information might not be available for newly coined compounds, rendering composition models that rely on it unusable in such contexts. Schütze (1992) represents words via their fourgrams, e.g. the representation of the word *rule* is the average of the representations of the fourgrams _RUL, RULE and ULE_. The representation of each fourgram is based on its co-occurrences with a selection of 5000 fourgrams, to which singular value decomposition is applied - leading to a 97-dimensional representation.

Recently Bojanowski et al. (2017) have shown that adding subword information to word representations improves the quality of the word representations and their usefulness for language modeling. Bojanowski et al. (2017) represent a word as a bag of character n-grams, an approach in a similar vein to that of Schütze (1992).

Combining the idea of representing words via their subword units and that of recurrent neural networks for composition should result in more robust composition models. Such composition models could handle both a variable number of words in a phrase (*tea cup*, *cup for tea*), as well as words or phrases that are written as a

contiguous string of characters (*Apfelbaum*). A welcome side-effect of such an approach would be that spelling variations of the same word - like *database*, *data-base*, *data base* could be represented using the same composition model, and would ideally receive similar vector representations.

Related Publications The following peer-reviewed publications were published on topics addressed in this thesis.

- Dima, C. (2016). On the Compositionality and Semantic Interpretation of English Noun Compounds. In *Proceedings of the 1st Workshop on Representation Learning for NLP @ ACL 2016*, pages 27–39, Berlin, Germany
- Dima, C. (2015). Reverse-engineering Language: A Study on the Semantic Compositionality of German Compounds. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, pages 1637–1642, Lisbon, Portugal
- Dima, C. and Hinrichs, E. W. (2015). Automatic Noun Compound Interpretation using Deep Neural Networks and Word Embeddings. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS 2015)*, pages 173–183, London, UK
- Sorokin, D., Dima, C., and Hinrichs, E. W. (2015). Classifying Semantic Relations in German Nominal Compounds using a Hybrid Annotation Scheme. *Journal of Cognitive Science*, 16(3):260–285
- Dima, C., Henrich, V., Hinrichs, E. W., and Hoppermann, C. (2014). How to Tell a Schneemann from a Milchmann: An Annotation Scheme for Compound-Internal Relations. In *Proceedings of 9th International Conference on Language Resources and Evaluation (LREC 2014)*, pages 1194–1201, Reykjavik, Iceland

References

Aichner, T. and Coletti, P. (2013). Customers' online shopping preferences in mass customization. *Journal of Direct, Data and Digital Marketing Practice*, 15(1):20–35.

Asimov, I. (1988). *Prelude to Foundation*. The Foundation Novels. Doubleday.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Ballesteros, M., Dyer, C., and Smith, N. A. (2015). Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, pages 349–359.

Barker, K. and Szpakowicz, S. (1998). Semi-automatic recognition of noun modifier relationships. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 96–102. Association for Computational Linguistics.

Baroni, M., Dinu, G., and Kruszewski, G. (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 (ACL 2014), Volume 1: Long Papers*, pages 238–247.

Baroni, M., Matiasek, J., and Trost, H. (2002). Predicting the components of German nominal compounds. In Harmelen, F. v., editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, pages 470–474.

Baroni, M. and Zamparelli, R. (2010). Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, pages 1183–1193.

Bauer, L. (1983). *English word-formation*. Cambridge University Press.

Bauer, L. (2001). Compounding. In Haspelmath, M., editor, *Language Typology and Language Universals*. Mouton de Gruyter, The Hague.

Bauer, L. (2009). Typology of compounds. In Lieber, R. and Štekauer, P., editors, *The Oxford Handbook of Compounding*. Oxford University Press, Oxford, UK.

References

Bell, M. J. and Schäfer, M. (2013). Semantic transparency: challenges for distributional semantics. In *Proceedings of the IWCS 2013 Workshop Towards a Formal Distributional Semantics*, pages 1–10.

Bell, M. J. and Schäfer, M. (2016). Modelling semantic transparency. *Morphology*, 26(2):157–199.

Bender, E. M. (2013). *Linguistic fundamentals for natural language processing: 100 essentials from morphology and syntax*. Morgan & Claypool Publishers.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Bernardi, R., Dinu, G., Marelli, M., and Baroni, M. (2013). A relatedness benchmark to test the role of determiners in compositional distributional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), Volume 2: Short Papers*, pages 53–57.

Blacoe, W. and Lapata, M. (2012). A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP 2012)*, pages 546–556.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association of Computational Linguistics*, 5(1):135–146.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM.

Bottou, L. (2012). Stochastic gradient descent tricks. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, pages 421–436. Springer.

Brants, T. and Franz, A. (2006). Web 1T 5-gram Corpus Version 1.1 (LDC2006T13).

Bride, A., Van de Cruys, T., and Asher, N. (2015). A Generalisation of Lexical Functions for Composition in 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 (ACL 2015), Volume 1: Long Papers*, pages 281–291.

Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Burnard, L. (1995). Users Guide for the British National Corpus Version 1.0.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Chanwimalueang, T. and Mandic, D. P. (2017). Cosine similarity entropy: Self-correlation-based complexity analysis of dynamical systems. *Entropy*, 19(12):652.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 740–750.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29.

Clark, S. (2012). Vector space models of lexical meaning. *Handbook of Contemporary Semantics, 2nd ed.*

Coecke, B., Sadrzadeh, M., and Clark, S. (2010). Mathematical Foundations for Distributed Compositional Model of Meaning. Lambek Festschrift. *Linguistic Analysis*, 36:345–384.

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011a). Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, Advances in Neural Information Processing Systems (NIPS) Workshop*.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine learning*, pages 160–167. ACM.

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

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Dahl, G., Mohamed, A.-r., Hinton, G. E., et al. (2010). Phone recognition with the mean-covariance restricted boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*, pages 469–477.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Dima, C. (2015). Reverse-engineering Language: A Study on the Semantic Compositionality of German Compounds. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, pages 1637–1642, Lisbon, Portugal.

References

Dima, C. (2016). On the Compositionality and Semantic Interpretation of English Noun Compounds. In *Proceedings of the 1st Workshop on Representation Learning for NLP @ ACL 2016*, pages 27–39, Berlin, Germany.

Dima, C., Henrich, V., Hinrichs, E. W., and Hoppermann, C. (2014). How to Tell a Schneemann from a Milchmann: An Annotation Scheme for Compound-Internal Relations. In *Proceedings of 9th International Conference on Language Resources and Evaluation (LREC 2014)*, pages 1194–1201, Reykjavik, Iceland.

Dima, C. and Hinrichs, E. W. (2015). Automatic Noun Compound Interpretation using Deep Neural Networks and Word Embeddings. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS 2015)*, pages 173–183, London, UK.

Dinu, G., Nghia, T. P., and Baroni, M. (2013a). DISSECT - DIStributional SEmantics Composition Toolkit. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, pages 31–36, Sofia, Bulgaria.

Dinu, G., Nghia, T. P., and Baroni, M. (2013b). General estimation and evaluation of compositional distributional semantic models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality @ ACL 2013*, Sofia, Bulgaria.

Downing, P. (1977). On the creation and use of english compound nouns. *Language*, pages 810–842.

Dowty, D. R., Wall, R., and Peters, S. (1981). *Introduction to Montague semantics*, volume 11 of *Synthese Language Library*. Springer Science & Business Media.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Eisenberg, P. (2013). *Grundriß der deutschen Grammatik: Band 1: Das Wort*. J. B. Metzler Verlag, Weimar.

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

Fano, R. M. (1961). *Transmission of information: A statistical theory of communications*. The MIT Press, Cambridge, Mass.

Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (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*, pages 1606–1615.

Fellbaum, C. (1998). *WordNet*. Wiley Online Library.

Finin, T. W. (1980). *The semantic interpretation of compound nominals*. PhD thesis, Ph. D. thesis, University of Illinois at Urbana-Champaign.

Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. In *Studies in Linguistic Analysis (special volume of the Philological Society)*, volume 1952-59, pages 1–32. The Philological Society, Oxford.

Fleischer, W. and Barz, I. (2012). *Wortbildung der deutschen Gegenwartssprache.* Walter de Gruyter, Berlin.

Girju, R. (2006). Out-of-context noun phrase semantic interpretation with cross-linguistic evidence. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 268–276. ACM.

Girju, R., Moldovan, D., Tatu, M., and Antohe, D. (2005). On the semantics of noun compounds. *Computer speech & language*, 19(4):479–496.

Gleitman, L. R. and Gleitman, H. (1970). *Phrase and Paraphrase: Some Innovative Uses of Language.* New York: W. W. Norton.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing.* Morgan & Claypool Publishers.

Goldberg, Y. and Levy, O. (2014). word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722.*

Grefenstette, E. and Sadrzadeh, M. (2011). Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1394–1404. Association for Computational Linguistics.

Grolier Inc. (1992). The New Grolier Multimedia Encyclopedia.

Guevara, E. (2010). A regression model of adjective-noun compositionality in distributional semantics. In *Proceedings of the Workshop on GEometrical Models of Natural Language Semantics @ ACL 2010*, pages 33–37. Association for Computational Linguistics.

Guevara, E. (2011). Computing semantic compositionality in distributional semantics. In *Proceedings of the Ninth International Conference on Computational Semantics*, pages 135–144. Association for Computational Linguistics.

Hamp, B. and Feldweg, H. (1997). Germanet - a lexical-semantic net for German. In *Proceedings of ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, pages 9–15. Citeseer.

Harris, Z. S. (1954). Distributional Structure. *Word*, 10(2-3):146–162. Reprinted in Harris (1970), pp. 775 – 795.

Harris, Z. S. (1970). *Papers in Structural and Transformational Linguistics*, volume 1 of *Formal Linguistics Series*. D. Reidel Publishing Company, Dordrecht-Holland.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

Henrich, V. and Hinrichs, E. (2010). GernEdiT - The GermaNet Editing Tool. In *Proceedings of the ACL 2010 System Demonstrations*, pages 19–24, Uppsala, Sweden. Association for Computational Linguistics.

Henrich, V. and Hinrichs, E. W. (2011). Determining Immediate Constituents of Compounds in GermaNet. In *Proceedings of Recent Advances in Natural Language Processing (RANLP 2011)*, pages 420–426, Hissar, Bulgaria.

Hermann, K. M. and Blunsom, P. (2014). Multilingual models for compositional distributed semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Volume 1: Long Papers*, pages 58–68. Association for Computational Linguistics.

Hindle, D. (1990). Noun classification from predicate-argument structures. In *Proceedings of the 28th annual meeting on Association for Computational Linguistics (ACL 1990)*, pages 268–275. Association for Computational Linguistics.

Hinton, G. E., McClelland, J., and Rumelhart, D. E. (1986). Distributed representations. *Parallel Distributed Processing: Explorations in the Microstructure, vol. 1: Foundations*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hunt, A. and Thomas, D. (1999). *The pragmatic programmer*. Addison Wesley.

Im Walde, S. S., Müller, S., and Roller, S. (2013). Exploring vector space models to predict the compositionality of german noun-noun compounds. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, volume 1, pages 255–265.

Jackendoff, R. (2016). English noun-noun compounds in Conceptual Semantics. In ten Hacken, P., editor, *The Semantics of Compounding*, pages 15–37. Cambridge University Press.

Jurafsky, D. and Martin, J. H. (2017). *Speech and Language Processing, 3rd edition draft*. https://web.stanford.edu/ jurafsky/slp3/.

Kastovsky, D. (2009). Diachronic perspectives. In Lieber, R. and Štekauer, P., editors, *The Oxford handbook of compounding*. Oxford University Press, Oxford, UK.

Kim, S. N. and Baldwin, T. (2005). Automatic interpretation of noun compounds using WordNet similarity. In *Proceedings of the Second International Joint Conference on Natural Language Processing (IJCNLP-05)*, pages 945–956. Springer.

Kisselew, M., Padó, S., Palmer, A., and Šnajder, J. (2015). Obtaining a Better Understanding of Distributional Models of German Derivational Morphology. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS 2015)*, pages 58–63, London, UK.

Klos, V. (2011). *Komposition und Kompositionalität: Möglichkeiten und Grenzen der semantischen Dekodierung von Substantivkomposita*. Walter de Gruyter.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105.

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, pages 159–174.

Langacker, R. W. (1987). *Foundations of cognitive grammar: Theoretical prerequisites*, volume 1. Stanford university press.

Lapata, M. and Keller, F. (2004). The web as a baseline: Evaluating the performance of unsupervised web-based models for a range of NLP tasks. In *the Proceedings of the Human Language Technology Conference/North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*.

Lauer, M. (1995). *Designing statistical language learners: Experiments on compound nouns*. PhD thesis, Ph. D. thesis, Macquarie University.

Lazaridou, A., Marelli, M., Zamparelli, R., and Baroni, M. (2013). Compositional-ly Derived Representations of Morphologically Complex Words in Distributional Semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, pages 1517–1526, Sofia, Bulgaria.

Lebret, R. and Collobert, R. (2014). Word Embeddings through Hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2014)*, pages 482–490, Gothenburg, Sweden. Association for Computational Linguistics.

Lebret, R. and Collobert, R. (2015a). Rehabilitation of Count-Based Models for Word Vector Representations. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, volume 9041 of *Lecture Notes in Computer Science*, pages 417–429. Springer International Publishing.

Lebret, R. and Collobert, R. (2015b). "The Sum of Its Parts": Joint Learning of Word and Phrase Representations with Autoencoders. *In ICML Deep Learning Workshop*.

Lebret, R. P. (2016). *Word Embeddings for Natural Language Processing*. PhD thesis, Ph. D. thesis, EPFL.

References

LeCun, Y., Bottou, L., O., G. B., and Müller, K.-R. (2012). Efficient BackProp. *Neural Networks: Tricks of the Trade, 2nd edition*.

Legrand, J. and Collobert, R. (2015). Joint RNN-based greedy parsing and word composition. *Proceedings of the International Conference on Learning Representations (ICLR 2015)*.

Lemnitzer, L. (2007). *Von Aldianer bis Zauselquote: neue deutsche Wörter, wo sie herkommen und wofür wir sie brauchen*. Gunter Narr Verlag.

Levi, J. N. (1978). *The syntax and semantics of complex nominals*. Academic Press.

Levy, O. and Goldberg, Y. (2014a). Dependency-Based Word Embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Volume 2: Short Papers*, pages 302–308.

Levy, O. and Goldberg, Y. (2014b). Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2177–2185.

Levy, O., Goldberg, Y., and Dagan, I. (2015a). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Levy, O., Goldberg, Y., and Ramat-Gan, I. (2014). Linguistic Regularities in Sparse and Explicit Word Representations. In *Proceedings of the Eighteenth Conference on Computational Language Learning (CoNLL 2014)*, pages 171–180.

Levy, O., Remus, S., Biemann, C., and Dagan, I. (2015b). Do supervised distributional methods really learn lexical inference relations? In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 970–976.

Libben, G. (2006). Why Study Compound Processing? An overview of the issues. In Libben, Gary and Jarema, Gonia, editor, *The Representation and Processing of Compound Words*, chapter 1, pages 1–22. Oxford University Press, Oxford, UK.

Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the 17th International Conference on Computational Linguistics, Volume 2*, pages 768–774. Association for Computational Linguistics.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luis, T. (2015). Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, pages 1520–1530.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge University Press.

Manning, D. C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press.

Melinger, A., Im Walde, S. S., and Weber, A. (2006). Characterizing response types and revealing noun ambiguity in German association norms. In *Proceedings of the EACL Workshop Making Sense of Sense: Bringing Computational Linguistics and Psycholinguistics together*, pages 41–48. Citeseer.

Meyer, C. M., Mieskes, M., Stab, C., and Gurevych, I. (2014). DKPro Agreement: An Open-Source Java Library for Measuring Inter-Rater Agreement. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 105–109.

Meyer, R. (1993). *Compound Comprehension in Isolation and in Context*. Max Niemeyer Verlag, Tuebingen, Germany.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3111–3119.

Mikolov, T., Yih, W.-T., and Zweig, G. (2013c). Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 13, pages 746–751.

Mitchell, J. and Lapata, M. (2008). Vector-based Models of Semantic Composition. In *Proceedings of ACL-08: HLT*, pages 236–244.

Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Müller, T. and Schütze, H. (2015). Robust morphological tagging with word representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Nastase, V. and Szpakowicz, S. (2003). Exploring noun-modifier semantic relations. In *Fifth International Workshop on Computational Semantics (IWCS-5)*, pages 285–301.

Nicoladis, E. (2006). Preschool children's acquisition of compounds. In Libben, Gary and Jarema, Gonia, editor, *The Representation and Processing of Compound Words*, pages 96–124. Oxford University Press, Oxford, UK.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Niwa, Y. and Nitta, Y. (1994). Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 304–309. Association for Computational Linguistics.

Ó Séaghdha, D. (2008). *Learning compound noun semantics*. PhD thesis, Computer Laboratory, University of Cambridge. Published as University of Cambridge Computer Laboratory Technical Report 735.

References

Ó Séaghdha, D. and Copestake, A. (2013). Interpreting compound nouns with kernel methods. *Natural Language Engineering*, 19(03):331–356.

Padó, S., Herbelot, A., Kisselew, M., and Šnajder, J. (2016). Predictability of distributional semantics in derivational word formation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*, pages 1285–1296.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, volume 12, Doha, Qatar.

Plag, I. (2003). *Word-Formation in English.* Cambridge University Press.

Prechelt, L. (1998). Early stopping - but when? In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the trade*, pages 55–69. Springer.

Reddy, S., McCarthy, D., and Manandhar, S. (2011). An empirical study on compositionality in compound nouns. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 210–218.

Roller, S., im Walde, S. S., and Scheible, S. (2013). The (un) expected effects of applying standard cleansing models to human ratings on compositionality. In *Proceedings of the 9th Workshop on Multiword Expressions*, pages 32–41.

Rosario, B. and Hearst, M. (2001). Classifying the semantic relations in noun compounds via a domain-specific lexical hierarchy. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP-01)*, pages 82–90.

Rosario, B., Hearst, M. A., and Fillmore, C. (2002). The descent of hierarchy, and selection in relational semantics. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL 2002)*, pages 247–254. Association for Computational Linguistics.

Roth, M. and Woodsend, K. (2014). Composition of Word Representations Improves Semantic Role Labelling. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 407–413.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure, vol. 1: Foundations.*

Rumelhart, D. E., Smolensky, P., McClelland, J., and Hinton, G. E. (1986b). Schemata and sequential thought processes in pdp models. *Parallel Distributed Processing: Explorations in the Microstructure, vol. 2: Psychological and Biological models.*

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Ryder, M. E. (1994). *Ordered chaos: The interpretation of English noun-noun compounds*. University of California Press.

Salehi, B., Cook, P., and Baldwin, T. (2015). A word embedding approach to predicting the compositionality of multiword expressions. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 977–983.

Santos, C. D. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

Schäfer, R. (2015). Processing and querying large web corpora with the COW14 architecture. In *Challenges in the Management of Large Corpora (CMLC-3)*.

Schütze, H. (1992). Dimensions of Meaning. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, Supercomputing '92, pages 787–796, Los Alamitos, CA, USA. IEEE Computer Society Press.

Schütze, H. (1992). Dimensions of meaning. In *Proceedings of Supercomputing'92*, pages 787–796. IEEE.

Shwartz, V. and Dagan, I. (2018). Paraphrase to explicate: Revealing implicit noun-compound relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Volume 1 (Long Papers)*, pages 1200–1211, Melbourne, Australia.

Shwartz, V. and Waterson, C. (2018). Olive oil is made of olives, baby oil is made for babies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, volume 2, pages 218–224, New Orleans, Louisiana.

Siegel, S. and Castellan, N. (1988). Nonparametric systems for the behavioural sciences. *McGraw Hill International Editions*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Socher, R., Bauer, J., Manning, C. D., and Ng, A. Y. (2013a). Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers*, pages 455–465.

Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (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*, pages 1201–1211.

Socher, R., Manning, C. D., and Ng, A. Y. (2010). Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., Potts, C., et al. (2013b). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1631, page 1642.

Sorokin, D., Dima, C., and Hinrichs, E. W. (2015). Classifying Semantic Relations in German Nominal Compounds using a Hybrid Annotation Scheme. *Journal of Cognitive Science*, 16(3):260–285.

Spärck Jones, K. (1983). Compound noun interpretation problems. Technical report, University of Cambridge, Computer Laboratory.

Speer, R., Chin, J., and Havasi, C. (2017). Conceptnet 5.5: An open multilingual graph of general knowledge. In *AAAI*, pages 4444–4451.

Speer, R. and Lowry-Duda, J. (2017). Conceptnet at semeval-2017 task 2: Extending word embeddings with multilingual relational knowledge. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 85–89.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Strang, G. (2016). *Introduction to Linear Algebra*. Wesley - Cambridge Press.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112.

Telljohann, H., Hoppermann, C., Dima, C., Hinrichs, E., Henrich, V., and Versley, Y. (2017). Stylebook für die Annotation deutscher Nominalkomposita. In *Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany*.

Tratz, S. (2011). *Semantically-enriched parsing for natural language understanding*. PhD thesis, PhD Thesis, University of Southern California.

Tratz, S. and Hovy, E. (2010). A taxonomy, dataset, and classifier for automatic noun compound interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*, Uppsala, Sweden.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics.

Turney, P. D. (2006). Similarity of semantic relations. *Computational Linguistics*, 32(3):379–416.

Turney, P. D. and Littman, M. L. (2005). Corpus-based learning of analogies and semantic relations. *Machine Learning*, 60(1-3):251–278.

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

Twain, M. (1880). *The awful German language*. BVK.

Verhoeven, B., Daelemans, W., and van Huyssteen, G. (2012). Classification of noun-noun compound semantics in dutch and afrikaans. In *Proceedings of the Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa*, pages 121–125.

Warren, B. (1978). Semantic patterns of noun-noun compounds. *Acta Universitatis Gothoburgensis. Gothenburg Studies in English Goteborg*, 41:1–266.

Yazdani, M., Farahmand, M., and Henderson, J. (2015). Learning semantic composition to detect non-compositionality of multiword expressions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1733–1742.

Zanzotto, F. M., Korkontzelos, I., Fallucchi, F., and Manandhar, S. (2010). Estimating Linear Models for Compositional Distributional Semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1263–1271.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer.

Appendix A

Mathematical Notation

symbol	meaning
$\alpha, \beta \in \mathbb{R}$	**scalars** (integer or real values)
$\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{b} \in \mathbb{R}^n$	real-valued **vectors** with n components
$\mathbf{x} = (x_1, x_2, \ldots, x_k)^\top$, $x_i \in \mathbb{R}$	n-dimensional **vector** and its components; each x_i is a scalar; the $^\top$ notation indicates transposition, i.e. transforms row vectors to column vectors
$\|\mathbf{u}\|_1 = \|x\| + \|y\|$	the $\mathbf{L}_1$ **norm** of a vector $\mathbf{u} = (x, y)$
$\|\mathbf{u}\|_2 = \sqrt{x^2 + y^2}$	the $\mathbf{L}_2$ **norm** of a vector $\mathbf{u} = (x, y)$; also known as the **Euclidean norm** of a vector
$\hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|_2}$	a **unit vector**; its length is equal to 1;
$[\mathbf{u}; \mathbf{v}] \in \mathbb{R}^{2n}$	**concatenation** of two n-dimensional vectors, $\mathbf{u}$ and $\mathbf{v}$; the result is a $2n$-dimensional vector
$\mathbf{u} \odot \mathbf{v}$	component-wise **multiplication** of two vectors
$\mathbf{u} + \mathbf{v}$	component-wise **addition** of two vectors
$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i$	the **dot product** of two vectors
$\mathbf{A}, \mathbf{B}, \ldots, \mathbf{M_1}, \ldots, \mathbf{M_i} \in \mathbb{R}^{n \times n}$	real-valued **matrices** with n rows and n columns
$\mathbf{A}^\top$	the **transpose** of matrix A
$f, g, h, J : \mathbb{R} \rightarrow [0, 1]$	mathematical **functions**; the first interval after the semicolon marks the *domain* of the function, and the second interval marks its *codomain*; examples are the hyperbolic tangent (tanh), the logistic function etc.
$\ln : \mathbb{R}^+ \rightarrow \mathbb{R}$	the **natural logarithm** (logarithm in base e)
$\exp(x) = e^x : \mathbb{R} \rightarrow \mathbb{R}$	the **exponential function**; e is the base of the natural logarithm and is approx. equal to 2.71828
∇J	**gradient** of a function J
$\frac{\partial J}{\partial w_1}$	**partial derivative** of the function J with respect to the parameter w_1

Appendix B

Hyperparameter tuning

B.1 Hyperparameter tuning for composition models on the German nn-only dataset

norm	model	η=0.001				η=0.01				η=0.1			
		Q1	**Q2**	**Q3**	**cos-d**	**Q1**	**Q2**	**Q3**	**cos-d**	**Q1**	**Q2**	**Q3**	**cos-d**
none	dilation	275	1K	1K	0.6867	182	1K	1K	0.6553	50	371	1K	0.5734
	w_addition	52	322	1K	0.5628	52	325	1K	0.5627	52	323	1K	0.5627
	fulladd	**3**	**11**	**48**	**0.3667**	4	11	49	0.3677	3	11	50	0.3679
	matrix	3	10	47	0.3658	**3**	**10**	**46**	**0.3649**	4	11	49	0.3696
	lexfunc	7	49	448	0.4586	7	44	377	0.4524	8	52	365	0.4599
	fulllex	3	9	42	0.3573	2	6	30	0.3384	6	22	122	0.4079
L_2-row	dilation	275	1K	1K	0.6867	182	1K	1K	0.6553	50	371	1K	0.5734
	w_addition	51	319	1K	0.5616	50	322	1K	0.5615	51	325	1K	0.5615
	fulladd	3	11	48	0.3669	3	11	49	0.3678	3	11	49	0.3682
	matrix	3	11	47	0.3663	**3**	**11**	**46**	**0.3649**	3	10	47	0.3657
	lexfunc	7	49	448	0.4586	7	44	377	0.4524	8	52	365	0.4599
	fulllex	3	10	51	0.3676	**2**	**6**	**30**	**0.3379**	4	12	65	0.3764
L_2-col	dilation	253	1K	1K	0.6769	166	1K	1K	0.6459	**44**	**330**	**1K**	**0.5632**
	w_addition	47	284	1K	0.5511	**45**	**288**	**1K**	**0.5509**	**45**	**290**	**1K**	**0.5509**
	fulladd	3	11	49	0.3686	4	11	49	0.3696	3	11	51	0.3701
	matrix	4	11	49	0.3701	3	11	49	0.3674	3	10	49	0.3674
	lexfunc	7	50	440	0.4574	**7**	**45**	**374**	**0.4514**	8	54	364	0.4604
	fulllex	3	11	60	0.3737	3	8	36	0.3475	3	10	60	0.3696

Table B.1 Hyperparameter optimization for producing the results in Table 5.5. Trying out different learning rates (η) and different types of normalization: none - no normalization; L_2-row - L_2 normalization per word vector; L_2-col - L_2 normalization per column.

model	dropout position	learning rate	dropout rate	epochs	Q1	Q2	Q3	cos-d
fulladd	after W_1, W_2	0.01	**0**	29	3	11	49	**0.3678**
fulladd	after W_1, W_2	0.01	0.25	36	3	11	48	0.3686
fulladd	after W_1, W_2	0.01	0.5	36	4	12	51	0.3716
fulladd	after W_1, W_2	0.01	0.75	25	4	14	57	0.3785
matrix	after W	0.01	**0**	28	3	10	46	**0.3649**
matrix	after W	0.01	0.25	23	3	10	47	0.3652
matrix	after W	0.01	0.5	23	3	10	47	0.3656
matrix	after W	0.01	0.75	25	3	11	47	0.3668
lexfunc	after lookup	0.01	0	14	7	44	377	0.4524
lexfunc	after lookup	0.01	0.25	32	5	32	292	0.4383
lexfunc	after lookup	0.01	**0.5**	56	5	32	246	**0.4350**
lexfunc	after lookup	0.01	0.75	93	6	34	243	0.4390
fulllex	after lookup	0.01	0	8	2	6	30	0.3379
fulllex	after lookup	0.01	**0.25**	18	2	6	27	**0.3317**
fulllex	after lookup	0.01	0.5	39	2	6	27	0.3324
fulllex	after lookup	0.01	0.75	107	3	7	31	0.3419
addmask	after lookup	0.01	0	926	3	11	57	0.3675
addmask	after lookup	0.01	0.25	1129	3	11	58	0.3688
addmask	after lookup	0.01	0.5	1372	4	12	67	0.3755
addmask	after lookup	0.01	0.75	1716	5	17	95	0.3936
addmask	after lookup	0.1	0	39	3	8	49	0.3563
addmask	after lookup	0.1	0.25	69	3	8	44	0.3516
addmask	after lookup	**0.1**	**0.5**	100	3	8	43	**0.3508**
addmask	after lookup	0.1	0.75	180	3	8	44	0.3532
wmask	after lookup	0.01	0	78	2	6	26	0.3342
wmask	after lookup	0.01	0.25	227	2	6	23	0.3314
wmask	after lookup	0.01	0.5	165	3	9	36	0.3575
wmask	after lookup	0.01	0.75	55	11	37	127	0.4313
wmask	after lookup	0.1	0	12	2	5	22	0.3265
wmask	after lookup	**0.1**	**0.25**	29	2	5	20	**0.3224**
wmask	after lookup	0.1	0.5	56	2	6	25	0.3331
wmask	after lookup	0.1	0.75	72	5	13	49	0.3773

Table B.2 Quantifying the effect of different dropout rates on the performance of the composition models on the `nn-only dev` set, using 50-dimensional input representations. The mask models converge slower and have weaker results using a learning rate of 0.01, but perform better with a higher learning rate of 0.1.

model	nonlinearity	cos-d	Q_1	Q_2	Q_3
`multimatrix`	identity	0.3649	3	10	46
`multimatrix`	tanh	0.3450	2	7	30
`multimatrix`	**ReLU**	**0.3102**	**2**	**4**	**15**
`multimatrix`	PReLU	0.3117	2	4	16
`multimatrix`	PReLU, per channel a	0.3122	2	4	16
model	**dropout**	**cos-d**	Q_1	Q_2	Q_3
`multimatrix`	0	0.3282	2	5	22
`multimatrix`	0.25	0.3230	2	5	20
`multimatrix`	0.5	0.3157	2	4	18
`multimatrix`	**0.75**	**0.3102**	**2**	**4**	**15**
model	**learning rate**	**cos-d**	Q_1	Q_2	Q_3
`multimatrix`	0.001	0.3389	2	7	26
`multimatrix`	0.01	0.3144	2	4	17
`multimatrix`	**0.1**	**0.3102**	**2**	**4**	**15**
model	**# matrices**	**cos-d**	Q_1	Q_2	Q_3
`multimatrix`	40	0.3202	2	5	18
`multimatrix`	60	0.3157	2	4	17
`multimatrix`	80	0.3129	2	4	16
`multimatrix`	100	0.3110	2	4	16
`multimatrix`	120	**0.3102**	**2**	**4**	**15**
`multimatrix`	140	0.3091	2	4	16

Table B.3 Hyperparameter optimization for the `multimatrix` model, described in Section 5.4.5. The nonlinearity tryouts use dropout=0.75, learning rate=0.1, k=120; the dropout tryouts use nonlinearity=ReLU, learning rate=0.1, k=120; the matrices tryouts use dropout=0.75, learning rate=0.1, nonlinearity=ReLU; the learning rate tryouts use dropout=0.75, k=120, nonlinearity=ReLU.

B.2 Hyperparameter tuning for composition models on the English en-comcom dataset

norm	model	lr=0.01					lr=0.1				
		Q1	Q2	Q3	cos-d	epochs	Q1	Q2	Q3	cos-d	epochs
none	dilation	1K	1K	1K	0.8111	6	174	1K	1K	0.7023	6
	w_addition	139	1K	1K	0.6888	6	136	1K	1K	0.6888	6
	fulladd	3	13	93	**0.3727**	34	3	13	99	0.3736	97
	matrix	3	11	74	**0.3617**	36	3	12	76	0.3660	43
	lexfunc	5	59.5	1K	0.4823	22	5	39	913	0.4634	7
	fulllex	2	6	55	**0.3330**	8	4	21	215	0.3964	12
	addmask	3	15	238	0.3967	1112	3	12	203	0.3849	39
	wmask	2	8	49	0.3426	40	2	8	64	0.3487	11
	multimatrix	2	7	41	**0.3352**	11	2	6	46	0.3364	9
L_2-row	dilation	1K	1K	1K	0.8111	6	174	1K	1K	0.7023	6
	w_addition	131	1K	1K	0.6870	6	132	1K	1K	0.6870	6
	fulladd	3	13	95	0.3733	34	3	13	100	0.3743	93
	matrix	3	11	75	0.3636	21	3	11	75	0.3646	21
	lexfunc	5	59.5	1K	0.4823	22	5	39	913	0.4634	7
	fulllex	2	5	53	0.3337	8	3	11	111	0.3641	8
	addmask	3	15	242	0.3978	1113	3	12	208	0.3859	39
	wmask	2	7	50	0.3409	58	2	6	47	**0.3360**	11
	multimatrix	2	7	41	0.3363	11	2	6	44	0.3357	11
L_2-col	dilation	699	1K	1K	0.7642	6	88	1K	1K	**0.6462**	6
	w_addition	67	864	1K	**0.6251**	6	67	864	1K	**0.6251**	6
	fulladd	3	14	105	0.3787	34	4	14	106	0.3794	97
	matrix	3	12	77	0.3694	21	3	12	78	0.3691	32
	lexfunc	5	60.5	1K	0.4712	16	5	40	908	**0.4574**	7
	fulllex	2	7	78	0.3488	9	3	10	98	0.3650	7
	addmask	3	16	268	0.3942	994	3	13	208	**0.3834**	36
	wmask	2	7	51	0.3430	91	2	7	51	0.3394	13
	multimatrix	2	7	52	0.3416	29	2	7	56	0.3459	23

Table B.4 Trying out different learning rates (η) and different types of normalization on the dev subset of en-comcom: none - no normalization; L_2-row - L_2 normalization per word vector; L_2-col - L_2 normalization per column. The best cosine distance for each model is shown in bold. The column epochs displays the number of epochs needed for each model to converge when using early stopping with patience 5 epochs.

model	dropout position	learning rate	dropout rate	epochs	Q1	Q2	Q3	cos-d
fulladd	after W_1, W_2	0.01	**0**	34	3	13	95	**0.3733**
fulladd	after W_1, W_2	0.01	0.25	34	4	14	92	0.3738
fulladd	after W_1, W_2	0.01	0.5	23	4	15	95	0.3768
fulladd	after W_1, W_2	0.01	0.75	34	4	17	95	0.3820
matrix	after W_1, W_2	0.01	**0**	21	3	11	75	**0.3636**
matrix	after W_1, W_2	0.01	0.25	18	3	11	77	0.3641
matrix	after W_1, W_2	0.01	0.5	26	3	11	76	0.3642
matrix	after W_1, W_2	0.01	0.75	26	3	12	79	0.3653
lexfunc	after lookup	0.1	**0**	7	5	39	913	**0.4634**
lexfunc	after lookup	0.1	0.25	8	5	43	978	0.4652
lexfunc	after lookup	0.1	0.5	11	5	49	1K	0.4682
lexfunc	after lookup	0.1	0.75	18	6	47	985	0.4711
fulllex	after lookup	0.01	0	8	2	5	53	0.3337
fulllex	after lookup	0.01	0.25	17	2	5	42	0.3251
fulllex	after lookup	0.01	**0.5**	36	2	5	38	**0.3228**
fulllex	after lookup	0.01	0.75	121	2	5	41	0.3282
addmask	after lookup	0.1	0	39	3	12	208	0.3859
addmask	after lookup	0.1	0.25	68	2	11	175	0.3793
addmask	after lookup	0.1	**0.5**	105	2	10	164	**0.3770**
addmask	after lookup	0.1	0.75	190	2	10	157	0.3781
wmask	after lookup	0.1	**0**	11	2	6	47	**0.3360**
wmask	after lookup	0.1	0.25	20	2	7	42	0.3356
wmask	after lookup	0.1	0.5	40	2	9	47	0.3482
wmask	after lookup	0.1	0.75	50	6	23	102	0.3969
multimatrix	between layers	0.1	0	10	2	7	45	0.3398
multimatrix	between layers	0.1	0.25	12	2	6	41	0.3342
multimatrix	between layers	0.1	0.5	24	2	6	38	0.3300
multimatrix	between layers	0.1	**0.75**	79	2	6	34	**0.3292**

Table B.5 Quantifying the effect of different dropout rates on the performance of the composition models on the en-comcom dataset, dev, using 50-dimensional input representations.

model	nonlinearity	cos-d	Q_1	Q_2	Q_3
multimatrix	**ReLU**	**0.3215**	2	5	31
multimatrix	PReLU	0.3221	2	5	31
model	**# matrices**	**cos-d**	Q_1	Q_2	Q_3
multimatrix	40	0.3292	2	6	34
multimatrix	60	0.3263	2	6	35
multimatrix	80	0.3232	2	5	31
multimatrix	100	0.3230	2	5	33
multimatrix	**120**	**0.3215**	2	5	31
multimatrix	140	0.3216	2	5	31

Table B.6 Hyperparameter optimization for the multimatrix model on the en-comcom dataset (50-dimensional dev).

B.3 Hyperparameter tuning for classification on the German de-nncom-sem dataset

input	hidden layer	dropout	nonlin_h	nonlin_o	Acc.	wF1	Epochs
compound_original	$n/2$	0	ReLU	tanh	37.93%	0.3729	140
compound_original	$n/2$	0.1	ReLU	tanh	37.00%	0.3623	235
compound_original	$n/2$	0.2	ReLU	tanh	39.79%	0.3874	216
compound_original	$n/2$	0.3	ReLU	tanh	39.26%	0.3844	334
compound_original	$n/2$	0.4	ReLU	tanh	37.80%	0.3630	261
compound_original	n	0	ReLU	tanh	37.53%	0.3676	154
compound_original	n	0.1	ReLU	tanh	38.99%	0.3830	210
compound_original	n	0.2	ReLU	tanh	39.79%	0.3924	164
compound_original	n	0.3	ReLU	tanh	41.51%	0.4023	305
compound_original	n	0.4	ReLU	tanh	39.52%	0.3808	217
compound_original	$2n$	0	ReLU	tanh	35.94%	0.3534	129
compound_original	$2n$	0.1	ReLU	tanh	40.85%	0.4009	366
compound_original	$2n$	0.2	ReLU	tanh	40.58%	0.3940	210
compound_original	$2n$	0.3	ReLU	tanh	41.25%	0.4027	168
compound_original	$2n$	0.4	ReLU	tanh	41.64%	0.4084	323
compound_original	**2n**	**0.5**	ReLU	tanh	**42.18%**	**0.4096**	265
compound_original	$2n$	0.6	ReLU	tanh	41.25%	0.3952	243
modifier	n	0	ReLU	tanh	36.74%	0.3477	171
modifier	n	0.1	ReLU	tanh	36.87%	0.3451	136
modifier	n	0.2	ReLU	tanh	36.07%	0.3396	140
modifier	n	0.3	ReLU	tanh	37.14%	0.3474	183
modifier	n	0.4	ReLU	tanh	37.27%	0.3448	129
modifier	$2n$	0	ReLU	tanh	36.21%	0.3351	178
modifier	$2n$	0.1	ReLU	tanh	37.53%	0.3479	156
modifier	**2n**	**0.2**	ReLU	tanh	**38.20%**	**0.3586**	163
modifier	$2n$	0.3	ReLU	tanh	36.21%	0.3381	163
modifier	$2n$	0.4	ReLU	tanh	37.67%	0.3521	174
head	n	0	ReLU	tanh	32.10%	0.2661	151
head	n	0.1	ReLU	tanh	35.28%	0.3002	241
head	n	0.2	ReLU	tanh	34.75%	0.2996	226
head	n	0.3	ReLU	tanh	35.94%	0.3082	180
head	n	0.4	ReLU	tanh	35.28%	0.3055	232
head	$2n$	0	ReLU	tanh	34.48%	0.2909	442
head	$2n$	0.1	ReLU	tanh	35.41%	0.2995	449
head	**2n**	**0.2**	ReLU	tanh	**36.74%**	**0.3143**	628
head	$2n$	0.3	ReLU	tanh	35.28%	0.2933	204
head	$2n$	0.4	ReLU	tanh	34.88%	0.2995	240

Table B.7 Results for **individual property classification** using the single input classification network from Fig. 7.3a, different dropout rates and different sizes of the hidden layer. Input: the original representations of the compound, modifier and head (300-dimensional). The best hyperparameters (in bold) were used to produce the results in Table 7.4.

B.3 Hyperparameter tuning for classification on the German `de-nncom-sem` dataset

model	hidden layer	dropout	nl_h	nl_o	Acc.	wF1	Epochs
composed_addition	n	0	ReLU	tanh	56.23%	0.5410	237
composed_addition	n	0.1	ReLU	tanh	56.10%	0.5366	305
composed_addition	n	0.2	ReLU	tanh	57.29%	0.5516	498
composed_addition	n	0.3	ReLU	tanh	56.37%	0.5358	209
composed_addition	n	0.4	ReLU	tanh	54.77%	0.5200	379
composed_addition	$2n$	0	ReLU	tanh	55.70%	0.5409	388
composed_addition	$2n$	0.1	ReLU	tanh	58.22%	0.5631	384
composed_addition	$2n$	0.2	ReLU	tanh	57.82%	0.5581	408
composed_addition	**2n**	**0.3**	ReLU	tanh	**58.49%**	**0.5602**	344
composed_addition	$2n$	0.4	ReLU	tanh	57.56%	0.5528	345
composed_matrix	n	0	ReLU	tanh	55.17%	0.5333	550
composed_matrix	n	0.1	ReLU	tanh	56.76%	0.5483	513
composed_matrix	n	0.2	ReLU	tanh	56.76%	0.5409	512
composed_matrix	n	0.3	ReLU	tanh	54.51%	0.5182	165
composed_matrix	n	0.4	ReLU	tanh	52.92%	0.4938	156
composed_matrix	**2n**	**0**	ReLU	tanh	**57.16%**	**0.5529**	240
composed_matrix	$2n$	0.1	ReLU	tanh	55.70%	0.5324	314
composed_matrix	$2n$	0.2	ReLU	tanh	56.90%	0.5473	418
composed_matrix	$2n$	0.3	ReLU	tanh	56.50%	0.5408	490
composed_matrix	$2n$	0.4	ReLU	tanh	56.23%	0.5419	408
composed_multimatrix	n	0	ReLU	tanh	56.10%	0.5338	201
composed_multimatrix	n	0.1	ReLU	tanh	57.43%	0.5548	393
composed_multimatrix	n	0.2	ReLU	tanh	57.56%	0.5494	266
composed_multimatrix	n	0.3	ReLU	tanh	56.90%	0.5412	455
composed_multimatrix	n	0.4	ReLU	tanh	55.44%	0.5240	470
composed_multimatrix	$2n$	0	ReLU	tanh	57.56%	0.5513	274
composed_multimatrix	**2n**	**0.1**	ReLU	tanh	**58.89%**	**0.5654**	344
composed_multimatrix	$2n$	0.2	ReLU	tanh	58.36%	0.5605	298
composed_multimatrix	$2n$	0.3	ReLU	tanh	58.75%	0.5630	516
composed_multimatrix	$2n$	0.4	ReLU	tanh	57.03%	0.5389	534

Table B.8 Results for **individual property classification** using the single input classification network from Fig. 7.3a, different dropout rates and different sizes of the hidden layer. Input: a composed representation - obtained either via `addition`, `matrix` or `multimatrix` composition (300-dimensional). The best hyperparameters (in bold) were used to produce the results in Table 7.5.

input	h_sz	dr1	dr2	nl_h	nl_o	Acc.	wF1	Epochs
c_1; c_2	n	0	0	ReLU	tanh	61.67%	0.6017	292
c_1; c_2	n	0	0.1	ReLU	tanh	63.53%	0.6228	736
c_1; c_2	**n**	0	**0.2**	ReLU	tanh	64.46%	**0.6327**	360
c_1; c_2	n	0	0.3	ReLU	tanh	63.53%	0.6194	353
c_1; c_2	n	0	0.4	ReLU	tanh	64.06%	0.6269	323
c_1; c_2	n	0	0	ReLU	tanh	61.67%	0.6017	292
c_1; c_2	n	0.1	0.1	ReLU	tanh	63.66%	0.6198	325
c_1; c_2	n	0.2	0.2	ReLU	tanh	63.26%	0.6130	304
c_1; c_2	n	0.3	0.3	ReLU	tanh	**64.59%**	0.6273	608
c_1; c_2	n	0.4	0.4	ReLU	tanh	61.94%	0.5911	230
c_1; c_2	$2n$	0	0	ReLU	tanh	62.07%	0.6030	369
c_1; c_2	$2n$	0.1	0.1	ReLU	tanh	64.19%	0.6268	255
c_1; c_2	$2n$	0.2	0.2	ReLU	tanh	**64.59%**	0.6245	298
c_1; c_2	$2n$	0.3	0.3	ReLU	tanh	64.46%	0.6293	423
c_1; c_2	$2n$	0.4	0.4	ReLU	tanh	63.53%	0.6158	492
composed_multimatrix; head	n	0	0	ReLU	tanh	58.36%	0.5608	408
composed_multimatrix; head	n	**0.1**	**0.1**	ReLU	tanh	**58.62%**	**0.5624**	564
composed_multimatrix; head	n	0.2	0.2	ReLU	tanh	57.29%	0.5452	516
composed_multimatrix; head	n	0.3	0.3	ReLU	tanh	55.57%	0.5259	444
composed_multimatrix; head	n	0.4	0.4	ReLU	tanh	52.39%	0.4928	469
composed_multimatrix; modifier	n	0	0	ReLU	tanh	63.13%	0.6128	180
composed_multimatrix; modifier	n	0.1	0.1	ReLU	tanh	63.53%	0.6209	224
composed_multimatrix; modifier	n	0.2	0.2	ReLU	tanh	63.53%	0.6207	450
composed_multimatrix; modifier	n	**0.3**	**0.3**	ReLU	tanh	**64.59%**	**0.6307**	375
composed_multimatrix; modifier	n	0.4	0.4	ReLU	tanh	62.47%	0.6031	390

Table B.9 Results for **individual property classification** using the two inputs classification network from Fig. 7.3b, different dropout rates and different sizes of the hidden layer. Input: the constituent representations, `modifier` and `head` and/or the `multimatrix`-composed representations. The best hyperparameters (in bold), are used to produce the results in Table 7.6.

B.3 Hyperparameter tuning for classification on the German de-nncom-sem dataset

input	hl	dr1	dr2	nl_h	nl_o	Acc.	wF1	Epochs
compound_original	$2n$	0	0	ReLU	tanh	38.99%	0.3683	248
compound_original	$2n$	0	0.1	ReLU	tanh	42.18%	0.4014	427
compound_original	$2n$	0	0.2	ReLU	tanh	42.31%	0.4024	205
compound_original	$2n$		00.3	ReLU	tanh	41.78%	0.3944	211
compound_original	$2n$	0	**0.4**	ReLU	tanh	**42.97%**	**0.4037**	163
modifier	$2n$	0	0	ReLU	tanh	38.59%	0.3558	161
modifier	$2n$	0	0.1	ReLU	tanh	38.06%	0.3460	127
modifier	$2n$	0	0.2	ReLU	tanh	38.33%	0.3624	258
modifier	$2n$	0	0.3	ReLU	tanh	38.20%	0.3633	236
modifier	$2n$	0	**0.4**	ReLU	tanh	**38.99%**	**0.3667**	184
head	$2n$	0	**0**	ReLU	tanh	**36.87%**	**0.3194**	260
head	$2n$	0	0.1	ReLU	tanh	36.60%	0.3137	324
head	$2n$	0	0.2	ReLU	tanh	36.07%	0.3089	215
head	$2n$	0	0.3	ReLU	tanh	35.94%	0.3079	216
head	$2n$	0	0.4	ReLU	tanh	35.68%	0.2993	218
compound_addition	$2n$	0	0	ReLU	tanh	59.55%	0.5752	501
compound_addition	$2n$	0	0.1	ReLU	tanh	61.01%	0.5954	428
compound_addition	$2n$	0	0.2	ReLU	tanh	59.42%	0.5757	266
compound_addition	$2n$	0	**0.3**	ReLU	tanh	**61.41%**	**0.5957**	285
compound_addition	$2n$	0	0.4	ReLU	tanh	61.14%	0.5891	427
compound_multimatrix	$2n$	0	0	ReLU	tanh	60.61%	0.5866	310
compound_multimatrix	$2n$	0	0.1	ReLU	tanh	**60.74%**	0.5877	316
compound_multimatrix	$2n$	0	0.2	ReLU	tanh	59.68%	0.5722	251
compound_multimatrix	$2n$	0	**0.3**	ReLU	tanh	60.61%	**0.5894**	593
compound_multimatrix	$2n$	0	0.4	ReLU	tanh	58.75%	0.5676	600
modifier; head	n	0	0	ReLU	tanh	63.53%	0.6215	294
modifier; head	n	0.1	0.1	ReLU	tanh	64.59%	0.6287	233
modifier; head	n	**0.2**	**0.2**	ReLU	tanh	**67.37%**	**0.6594**	550
modifier; head	n	0.3	0.3	ReLU	tanh	63.66%	0.6192	426
modifier; head	n	0.4	0.4	ReLU	tanh	63.79%	0.6197	563
compound_MM; head	n	**0**	**0**	ReLU	tanh	**60.08%**	**0.5796**	468
compound_MM; head	n	0.1	0.1	ReLU	tanh	58.36%	0.5689	300
compound_MM; head	n	0.2	0.2	ReLU	tanh	57.96%	0.5581	409
compound_MM; head	n	0.3	0.3	ReLU	tanh	56.50%	0.5406	448
compound_MM; head	n	0.4	0.4	ReLU	tanh	54.77%	0.5181	487
modifier; compound_MM	n	0	0	ReLU	tanh	62.33%	0.6090	274
modifier; compound_MM	n	0.1	0.1	ReLU	tanh	64.85%	0.6359	288
modifier; compound_MM	n	**0.2**	**0.2**	ReLU	tanh	**66.31%**	**0.6494**	330
modifier; compound_MM	n	0.3	0.3	ReLU	tanh	64.72%	0.6299	257
modifier; compound_MM	n	0.4	0.4	ReLU	tanh	62.33%	0.6032	195

Table B.10 Results for different dropout rates, results for **collapsed property classification**, using 300-dimensional input representations: either the original representations of the compound, modifier and head or the composed representation obtained via `addition` or `multimatrix` composition, or two representations - one of a constituent and a composed representation. The hyperparameters marked in bold were used to produce the results in Table 7.8.

input	hl	dr1	dr2	nl_h	nl_o	IProp Acc.	IProp wF1	Prep Acc.	Epochs
compound_multimatrix	$2n$	0	0	ReLU	tanh	58.75%	0.5686	**60.08%**	346
compound_multimatrix	$2n$	0.1	0	ReLU	tanh	57.43%	0.5520	59.02%	281
compound_multimatrix	$2n$	**0.2**	**0**	ReLU	tanh	**59.42%**	0.5698	57.69%	329
compound_multimatrix	$2n$	0.3	0	ReLU	tanh	58.09%	0.5531	57.69%	295
compound_multimatrix	$2n$	0.4	0	ReLU	tanh	55.84%	0.5351	55.97%	377
modifier; head	n	0	0	ReLU	tanh	63.00%	0.6167	61.61%	422
modifier; head	n	0.1	0.1	ReLU	tanh	65.12%	0.6344	**63.53%**	495
modifier; **head**	n	**0.2**	**0.2**	ReLU	tanh	**65.38%**	**0.6368**	63.00%	407
modifier; head	n	0.3	0.3	ReLU	tanh	63.13%	0.6094	60.61%	314
modifier; head	n	0.4	0.4	ReLU	tanh	63.00%	0.6071	60.48%	580
modifier; compound_multimatrix	n	0	0	ReLU	tanh	61.67%	0.6020	62.86%	254
modifier; **compound_multimatrix**	n	**0.1**	**0.1**	ReLU	tanh	**64.59%**	**0.6295**	63.93%	249
modifier; compound_multimatrix	n	0.2	0.2	ReLU	tanh	64.19%	0.6256	**64.72%**	449
modifier; compound_multimatrix	n	0.3	0.3	ReLU	tanh	62.60%	0.6060	63.26%	328
modifier; compound_multimatrix	n	0.4	0.4	ReLU	tanh	63.66%	0.6148	62.73%	488
compound_multimatrix; **head**	n	0	0	ReLU	tanh	57.82%	0.5557	**60.08%**	370
compound_multimatrix; head	n	**0.1**	**0.1**	ReLU	tanh	**57.96%**	0.5526	58.49%	331
compound_multimatrix; head	n	0.2	0.2	ReLU	tanh	57.03%	0.5414	57.03%	379
compound_multimatrix; head	n	0.3	0.3	ReLU	tanh	54.91%	0.5190	54.77%	365
compound_multimatrix; head	n	0.4	0.4	ReLU	tanh	53.05%	0.4991	52.92%	395

Table B.11 Hyperparameter optimization for classification networks using individual property classification as the main task and preposition classification as an auxiliary task. The best settings for input, shown in bold, are used to produce the results in Table 7.9.

B.4 Hyperparameter tuning for classification on the English Tratz (2011) dataset

dataset, split	input	dr1	dr2	Acc.	wF_1
tratz-fine, random	modifier	0	0	39.14%	0.3439
tratz-fine, random	modifier	0.1	0	39.56%	0.3452
tratz-fine, random	modifier	0.2	0	39.87%	0.3508
tratz-fine, random	modifier	0.3	0	39.67%	0.3476
tratz-fine, random	modifier	0.4	0	40.40%	0.3558
tratz-fine, random	modifier	0.5	0	40.29%	0.3579
tratz-fine, random	**modifier**	**0.6**	0	**40.50%**	**0.3591**
tratz-fine, random	modifier	0.7	0	38.83%	0.3315
tratz-fine, lex-mod	modifier	0	0	28.33%	0.2376
tratz-fine, lex-mod	modifier	0.1	0	28.46%	0.2357
tratz-fine, lex-mod	modifier	0.2	0	29.46%	0.2522
tratz-fine, lex-mod	modifier	0.3	0	29.11%	0.2407
tratz-fine, lex-mod	**modifier**	**0.4**	0	30.22%	**0.2558**
tratz-fine, lex-mod	modifier	0.5	0	**30.74%**	0.2551
tratz-fine, lex-mod	modifier	0.6	0	30.72%	0.2505
tratz-fine, lex-head	modifier	0	0	37.27%	0.3336
tratz-fine, lex-head	modifier	0.1	0	38.48%	0.3445
tratz-fine, lex-head	modifier	0.2	0	38.31%	0.3414
tratz-fine, lex-head	**modifier**	**0.3**	0	**38.79%**	**0.3516**
tratz-fine, lex-head	modifier	0.4	0	38.44%	0.3426
tratz-fine, lex-full	modifier	0	0	27.88%	0.2470
tratz-fine, lex-full	modifier	0.1	0	25.84%	0.2114
tratz-fine, lex-full	**modifier**	**0.2**	0	**27.76%**	**0.2505**
tratz-fine, lex-full	modifier	0.3	0	26.33%	0.2294
tratz-fine, lex-full	modifier	0.4	0	27.70%	0.2505

Table B.12 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using 300-dimensional representations of the `modifier` and the SMLP classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

dataset, split	input	dr1	dr2	Acc.	wF$_1$
tratz-fine, random	head	0	0	55.11%	0.5270
tratz-fine, random	head	0.1	0	55.22%	0.5258
tratz-fine, random	head	0.2	0	55.32%	0.5272
tratz-fine, random	head	0.3	0	55.11%	0.5197
tratz-fine, random	**head**	**0.4**	0	**56.26%**	**0.5373**
tratz-fine, random	head	0.5	0	55.95%	0.5340
tratz-fine, lex-mod	head	0	0	54.26%	0.5256
tratz-fine, lex-mod	head	0.1	0	56.54%	0.5526
tratz-fine, lex-mod	head	0.2	0	56.22%	0.5496
tratz-fine, lex-mod	head	0.3	0	56.19%	0.5478
tratz-fine, lex-mod	**head**	**0.4**	0	56.76%	**0.5528**
tratz-fine, lex-mod	head	0.5	0	**56.83%**	0.5526
tratz-fine, lex-head	head	0	0	33.80%	0.3252
tratz-fine, lex-head	head	0.1	0	34.18%	0.3271
tratz-fine, lex-head	head	0.2	0	35.11%	0.3306
tratz-fine, lex-head	head	0.3	0	36.67%	0.3479
tratz-fine, lex-head	**head**	**0.4**	0	**38.06%**	**0.3599**
tratz-fine, lex-head	head	0.5	0	36.29%	0.3348
tratz-fine, lex-full	head	0	0	32.90%	0.3100
tratz-fine, lex-full	head	0.1	0	32.71%	0.3072
tratz-fine, lex-full	head	0.2	0	32.96%	0.3120
tratz-fine, lex-full	head	0.3	0	33.89%	0.3373
tratz-fine, lex-full	**head**	**0.4**	0	**36.62%**	**0.3549**
tratz-fine, lex-full	head	0.5	0	35.25%	0.3243

Table B.13 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using 300-dimensional representations of the `head` and the SMLP classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

B.4 Hyperparameter tuning for classification on the English Tratz (2011) dataset

dataset, split	input	dr1	dr2	Acc.	wF_1
tratz-fine, random	addition	0	0	68.06%	0.6722
tratz-fine, random	addition	0.1	0	68.16%	0.6738
tratz-fine, random	**addition**	**0.2**	0	**68.37%**	**0.6755**
tratz-fine, random	addition	0.3	0	67.01%	0.6599
tratz-fine, random	addition	0.4	0	66.91%	0.6582
tratz-fine, lex-mod	addition	0	0	53.24%	0.5166
tratz-fine, lex-mod	addition	0.1	0	54.69%	0.5342
tratz-fine, lex-mod	addition	0.2	0	55.22%	0.5380
tratz-fine, lex-mod	**addition**	**0.3**	0	55.31%	**0.5388**
tratz-fine, lex-mod	addition	0.4	0	**55.57%**	0.5374
tratz-fine, lex-head	addition	0	0	42.14%	0.4034
tratz-fine, lex-head	addition	0.1	0	42.88%	0.4143
tratz-fine, lex-head	addition	0.2	0	42.72%	0.4103
tratz-fine, lex-head	**addition**	**0.3**	0	**43.48%**	**0.4149**
tratz-fine, lex-head	addition	0.4	0	42.95%	0.4086
tratz-fine, lex-full	addition	0	0	36.86%	0.3445
tratz-fine, lex-full	**addition**	**0.1**	0	37.42%	**0.3531**
tratz-fine, lex-full	addition	0.2	0	**37.48%**	0.3379
tratz-fine, lex-full	addition	0.3	0	37.24%	0.3365
tratz-fine, lex-full	**addition**	0.4	0	36.62%	0.3417

Table B.14 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using 300-dimensional representations obtained via component-wise `addition` of the constituent representations and the `SMLP` classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

dataset, split	input	dr1	dr2	Acc.	wF_1
tratz-fine, random	compound_multimatrix	0	0	67.64%	0.6743
tratz-fine, random	compound_multimatrix	0.1	0	70.35%	0.6960
tratz-fine, random	**compound_multimatrix**	**0.2**	0	**70.67%**	**0.7009**
tratz-fine, random	compound_multimatrix	0.3	0	68.16%	0.6727
tratz-fine, random	compound_multimatrix	0.4	0	66.81%	0.6579
tratz-fine, lex-mod	compound_multimatrix	0	0	58.50%	0.5711
tratz-fine, lex-mod	compound_multimatrix	0.1	0	59.11%	0.5771
tratz-fine, lex-mod	**compound_multimatrix**	**0.2**	0	59.20%	**0.5785**
tratz-fine, lex-mod	compound_multimatrix	0.3	0	**59.26%**	0.5782
tratz-fine, lex-mod	compound_multimatrix	0.4	0	58.74%	0.5732
tratz-fine, lex-head	**compound_multimatrix**	0	0	**51.80%**	**0.5076**
tratz-fine, lex-head	compound_multimatrix	0.1	0	50.56%	0.4976
tratz-fine, lex-head	compound_multimatrix	0.2	0	50.13%	0.4944
tratz-fine, lex-head	compound_multimatrix	0.3	0	49.11%	0.4815
tratz-fine, lex-head	compound_multimatrix	0.4	0	49.60%	0.4886
tratz-fine, lex-full	compound_multimatrix	0	0	40.21%	0.3865
tratz-fine, lex-full	compound_multimatrix	0.1	0	39.53%	0.3776
tratz-fine, lex-full	**compound_multimatrix**	**0.2**	0	**40.89%**	**0.3953**
tratz-fine, lex-full	compound_multimatrix	0.3	0	40.52%	0.3942
tratz-fine, lex-full	compound_multimatrix	0.4	0	40.33%	0.3901

Table B.15 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using a 300-dimensional, `multimatrix`-composed representation and the SMLP classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

dataset, split	input	dr1	dr2	nl_h	nl_o	Acc.	wF$_1$
tratz-fine, random	modifier; head	0	0	ReLU	tanh	76.07%	0.7541
tratz-fine, random	**modifier; head**	**0.1**	**0.1**	ReLU	tanh	**78.48%**	**0.7793**
tratz-fine, random	modifier; head	0.2	0.2	ReLU	tanh	78.16%	0.7738
tratz-fine, random	modifier; head	0.3	0.3	ReLU	tanh	76.70%	0.7561
tratz-fine, random	modifier; head	0.4	0.4	ReLU	tanh	76.49%	0.7534
tratz-fine, lex-mod	modifier; head	0	0	ReLU	tanh	66.37%	0.6491
tratz-fine, lex-mod	**modifier; head**	**0.1**	**0.1**	ReLU	tanh	68.37%	**0.6697**
tratz-fine, lex-mod	modifier; head	0.2	0.2	ReLU	tanh	67.96%	0.6631
tratz-fine, lex-mod	modifier; head	0.3	0.3	ReLU	tanh	**68.44%**	0.6677
tratz-fine, lex-mod	modifier; head	0.4	0.4	ReLU	tanh	66.94%	0.6498
tratz-fine, lex-head	modifier; head	0	0	ReLU	tanh	52.68%	0.5104
tratz-fine, lex-head	modifier; head	0.1	0.1	ReLU	tanh	53.80%	0.5187
tratz-fine, lex-head	modifier; head	0.2	0.2	ReLU	tanh	54.62%	0.5225
tratz-fine, lex-head	**modifier; head**	**0.3**	**0.3**	ReLU	tanh	**55.30%**	**0.5296**
tratz-fine, lex-head	modifier; head	0.4	0.4	ReLU	tanh	54.97%	0.5223
tratz-fine, lex-full	modifier; head	0	0	ReLU	tanh	44.76%	0.4288
tratz-fine, lex-full	modifier; head	0.1	0.1	ReLU	tanh	45.51%	0.4289
tratz-fine, lex-full	modifier; head	0.2	0.2	ReLU	tanh	46.75%	0.4390
tratz-fine, lex-full	**modifier; head**	**0.3**	**0.3**	ReLU	tanh	**47.58%**	**0.4456**
tratz-fine, lex-full	modifier; head	0.4	0.4	ReLU	tanh	47.30%	0.4398

Table B.16 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using 300-dimensional representations of the `modifier` and `head` as input and the DMLP classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

dataset, split	input	dr1	dr2	Acc.	wF$_1$
tratz-fine, random	compound_multimatrix; head	0	0	72.65%	0.7218
tratz-fine, random	**compound_multimatrix; head**	**0.1**	**0.1**	**72.76%**	**0.7154**
tratz-fine, random	compound_multimatrix; head	0.2	0.2	72.34%	0.7145
tratz-fine, random	compound_multimatrix; head	0.3	0.3	70.25%	0.6924
tratz-fine, random	compound_multimatrix; head	0.4	0.4	69.21%	0.6762
tratz-fine, lex-mod	compound_multimatrix; head	0	0	63.91%	0.6246
tratz-fine, lex-mod	**compound_multimatrix; head**	**0.1**	**0.1**	**64.67%**	**0.6331**
tratz-fine, lex-mod	compound_multimatrix; head	0.2	0.2	64.46%	0.6284
tratz-fine, lex-mod	compound_multimatrix; head	0.3	0.3	64.74%	0.6316
tratz-fine, lex-mod	compound_multimatrix; head	0.4	0.4	63.31%	0.6127
tratz-fine, lex-head	compound_multimatrix; head	0	0	49.24%	0.4785
tratz-fine, lex-head	compound_multimatrix; head	0.1	0.1	49.46%	0.4758
tratz-fine, lex-head	compound_multimatrix; head	0.2	0.2	49.77%	0.4795
tratz-fine, lex-head	**compound_multimatrix; head**	**0.3**	**0.3**	**49.91%**	**0.4808**
tratz-fine, lex-head	compound_multimatrix; head	0.4	0.4	49.91%	0.4778
tratz-fine, lex-full	compound_multimatrix; head	0	0	41.51%	0.3959
tratz-fine, lex-full	compound_multimatrix; head	0.1	0.1	42.13%	0.4006
tratz-fine, lex-full	compound_multimatrix; head	0.2	0.2	41.64%	0.3960
tratz-fine, lex-full	**compound_multimatrix; head**	**0.3**	**0.3**	**43.25%**	**0.4107**
tratz-fine, lex-full	compound_multimatrix; head	0.4	0.4	41.31%	0.4101

Table B.17 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using two 300-dimensional representations: a composed representation, using the `multimatrix` model and the `head` representation. Using the DMLP classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

B.4 Hyperparameter tuning for classification on the English Tratz (2011) dataset

dataset, split	input	dr1	dr2	Acc.	wF_1
tratz-fine, random	modifier; compound_multimatrix	0	0	69.62%	0.6914
tratz-fine, random	**modifier; compound_multimatrix**	**0.1**	**0.1**	**73.07%**	**0.7229**
tratz-fine, random	modifier; compound_multimatrix	0.2	0.2	71.71%	0.7080
tratz-fine, random	modifier; compound_multimatrix	0.3	0.3	69.21%	0.6789
tratz-fine, random	modifier; compound_multimatrix	0.4	0.4	67.85%	0.6606
tratz-fine, lex-mod	modifier; compound_multimatrix	0	0	60.35%	0.5898
tratz-fine, lex-mod	modifier; compound_multimatrix	0.1	0.1	62.56%	0.6116
tratz-fine, lex-mod	**modifier; compound_multimatrix**	**0.2**	**0.2**	**63.33%**	**0.6158**
tratz-fine, lex-mod	modifier; compound_multimatrix	0.3	0.3	62.30%	0.6046
tratz-fine, lex-mod	modifier; compound_multimatrix	0.4	0.4	58.81%	0.5631
tratz-fine, lex-head	**modifier; compound_multimatrix**	**0**	**0**	55.97%	**0.5525**
tratz-fine, lex-head	modifier; compound_multimatrix	0.1	0.1	56.02%	0.5499
tratz-fine, lex-head	modifier; compound_multimatrix	0.2	0.2	56.04%	0.5470
tratz-fine, lex-head	modifier; compound_multimatrix	0.3	0.3	56.20%	0.5434
tratz-fine, lex-head	modifier; compound_multimatrix	0.4	0.4	**56.32%**	0.5466
tratz-fine, lex-full	modifier; compound_multimatrix	0	0	45.66%	0.4453
tratz-fine, lex-full	modifier; compound_multimatrix	0.1	0.1	46.28%	0.4440
tratz-fine, lex-full	modifier; compound_multimatrix	0.2	0.2	45.79%	0.4407
tratz-fine, lex-full	**modifier; compound_multimatrix**	**0.3**	**0.3**	**46.65%**	**0.4478**
tratz-fine, lex-full	modifier; compound_multimatrix	0.4	0.4	46.59%	0.4444

Table B.18 Semantic property classification on the English Tratz (2011) dataset. Results for different dropout rates, using two 300-dimensional representations: the `modifier` representation and a `multimatrix`-composed representation. Using the DMLP classifier. The hyperparameters marked in bold were used to produce results for Table 7.12.

Appendix C

Examples of Neighbours of the Composed Representations

This appendix contains tables with example neighbours for the composed representations created by various composition models. For each composition model, a random sample of 12 compounds was extracted from the `test` set: four of the best ranking compounds, four middle-ranking and four of the worst ranked compounds.

The nearest neighbours are computed for each of the sample compounds, using the composed vector as a representation. The representations are obtained in each case using the best performing model, indicated in bold in Table 5.6. The neighbours are searched for in the observed vector space. The cosine similarity is listed next to each neighbour.

The examples are written as in the datasets: lowercase for the German compounds, with an underscore for the English ones. The observed representation always has a cosine similarity of 1, because the observed vector is compared to itself. The composed representation has an '_c' appended at the end and is marked in red in each case. E.g. when the rank is 1 the composed representation is closest to the observed representation. When the rank is higher than 5 the composed representation is on the last row. The rank and the difference in cosine similarity give an impression of how far away the composition is from the correct representation.

C.1 Composed Representations for German Compounds, `nn-only` dataset

wasserstrahl:1	**teilprivatisierung:1**	**rechnungsbetrag:1**	**körpergewicht:1**
wasserstrahl 1.00000	teilprivatisierung 1.00000	rechnungsbetrag 1.00000	körpergewicht 1.00000
wasserstrahl_c 0.55930	teilprivatisierung_c 0.61081	rechnungsbetrag_c 0.57306	körpergewicht_c 0.63996
strahl 0.55477	privatisierung 0.60631	betrag 0.56232	gewicht 0.63287
gartenschlauch 0.53208	börsengang 0.46165	kursgebühr 0.53677	körpergröße 0.62053
wasserschlauch 0.53077	bahnreform 0.45870	zahlungseingang 0.53130	körperfett 0.53772
laserstrahl 0.50183	rückkauf 0.44314	überweisung 0.53004	fettanteil 0.49844
luftstrom 0.49770	staatsunternehmen 0.42808	warenwert 0.52819	dosis 0.49696
malteserkreuz:83	**zugzwang:83**	**befreiungsfront:83**	**stadtstaat:83**
malteserkreuz 1.00000	zugzwang 1.00000	befreiungsfront 1.00000	stadtstaat 1.00000
andreaskreuz 0.41591	erfolgsdruck 0.50046	befreiungsarmee 0.57572	weltstadt 0.46776
kuppeldach 0.40613	erfolgszwang 0.45422	befreiungsbewegung 0.54077	inselstaat 0.45954
giebeldach 0.40562	zeitdruck 0.41054	befreiungsorganisation 0.44165	millionenstadt 0.41129
ahornblatt 0.40257	zeitnot 0.41017	einheitsliste 0.43790	handelsstadt 0.40916
ziegeldach 0.39877	schusslinie 0.38739	fortschrittspartei 0.43377	inselreich 0.40415
...	...	...	...
malteserkreuz_c 0.29086	zugzwang_c 0.24360	befreiungsfront_c 0.27199	stadtstaat_c 0.28780
schlagzahl:1000	**schildbürger:1000**	**hilfswerk:1000**	**leistungsstand:1000**
schlagzahl 1.00000	schildbürger 1.00000	hilfswerk 1.00000	leistungsstand 1.00000
fehlerquote 0.45229	täuschungsmanöver 0.36015	hilfsdienst 0.54199	entwicklungsstand 0.70655
pulsfrequenz 0.44008	verkehrsplaner 0.35997	caritas 0.49962	ausbildungsstand 0.66531
atemfrequenz 0.43836	streich 0.31941	hilfsorganisation 0.48616	leistungsvermögen 0.60301
seitenwechsel 0.42078	geschichtensammlung 0.31907	katastrophenhilfe 0.48607	leistungsniveau 0.59627
kampfstärke 0.41400	goldgräber 0.31138	ortsverband 0.47512	wissensstand 0.57662
...	...	...	...
schlagzahl_c 0.15467	schildbürger_c 0.09728	hilfswerk_c 0.11973	leistungsstand_c 0.13264

Table C.1 Neighbours of example compounds from the `nn-only test` set, using the `head` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (83) and the cut-off rank (1000). Rank specified next to the compound on the header row, e.g. *Wasserstrahl* 'jet of water', has rank 1.

tariftreuegesetz:1	**schubkraft:1**	**mittagszeit:1**	**höllenfeuer:1**
tariftreuegesetz 1.00000	schubkraft 1.00000	mittagszeit 1.00000	höllenfeuer 1.00000
tariftreuegesetz_c 0.58098	schubkraft_c 0.61961	mittagszeit_c 0.65324	höllenfeuer_c 0.43100
tariftreue 0.58085	schub 0.60430	mittag 0.64868	totenreich 0.42253
einwanderungsgesetz 0.51666	zugkraft 0.60215	mittagspause 0.64601	hölle 0.41414
ladenschlussgesetz 0.45362	auftrieb 0.55064	mitternacht 0.60622	scheiterhaufen 0.37570
finanzausgleichsgesetz 0.45336	abtrieb 0.47455	nachmittag 0.58042	himmelreich 0.35617
haushaltsrecht 0.43740	antriebskraft 0.47065	mittagshitze 0.56101	paradies 0.34748
kostendeckung:375	**heilsbringer:375**	**kirchspiel:375**	**rebellengruppe:375**
kostendeckung 1.00000	heilsbringer 1.00000	kirchspiel 1.00000	rebellengruppe 1.00000
bedarfsdeckung 0.55000	hoffnungsträger 0.62704	rittergut 0.53496	splittergruppe 0.51143
kostenreduzierung 0.42775	retter 0.54264	pfarrsprengel 0.51299	untergrundbewegung 0.50939
kosteneinsparung 0.41999	lichtgestalt 0.53975	pfarrbezirk 0.50748	oppositionsgruppe 0.50215
kostenreduktion 0.41896	wunderheiler 0.46035	pfarre 0.48444	untergrundorganisation 0.49018
schuldentilgung 0.41752	sündenbock 0.45390	kirchengemeinde 0.46484	frauenorganisation 0.46723
...	...	...	...
kostendeckung_c 0.23425	heilsbringer_c 0.21548	kirchspiel_c 0.23243	rebellengruppe_c 0.21373
stadtvilla:1000	**losverfahren:1000**	**gastvortrag:1000**	**rangabzeichen:1000**
stadtvilla 1.00000	losverfahren 1.00000	gastvortrag 1.00000	rangabzeichen 1.00000
penthousewohnung 0.50038	zufallsprinzip 0.56979	fachvortrag 0.74564	dienstgradabzeichen 0.65698
industriehalle 0.49141	zufallsgenerator 0.52098	festvortrag 0.71923	kragenspiegel 0.53794
neubauwohnung 0.47510	losentscheid 0.51416	antrittsvorlesung 0.65226	uniform 0.48495
neubausiedlung 0.47306	auswahlverfahren 0.48468	vortragsabend 0.62737	parteiabzeichen 0.48035
maisonettewohnung 0.47268	studienplatz 0.44089	diavortrag 0.59991	abzeichen 0.45276
...	...	...	...
stadtvilla_c 0.08773	losverfahren_c 0.05999	gastvortrag_c 0.08392	rangabzeichen_c 0.13207

Table C.2 Neighbours of example compounds from the `nn-only test` set, using the `modifier` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (375) and the cut-off rank (1000).

temperaturmessung:105	**eiswüste:106**	**hefeteig:306**	**geschäftsgebaren:307**
temperaturmessung 1.00000	eiswüste 1.00000	hefeteig 1.00000	geschäftsgebaren 1.00000
temperatursensor 0.55149	sumpflandschaft 0.51753	teig 0.69971	gebaren 0.63749
messung 0.51300	sandwüste 0.50431	mehl 0.56247	firmenpolitik 0.52558
druckmessung 0.48740	steppenlandschaft 0.48304	blätterteig 0.55626	geschäftspolitik 0.50110
positionsbestimmung 0.47085	wüste 0.43948	brotteig 0.55242	preispolitik 0.45619
entfernungsmessung 0.46391	eisscholle 0.43368	pizzateig 0.51847	anspruchsdenken 0.40727
...	...	...	...
temperaturmessung_c 0.30230	eiswüste_c 0.29213	hefeteig_c 0.23067	geschäftsgebaren_c 0.20327
reformkurs:1000	**schulzeit:1000**	**verbindungslinie:1000**	**erzählform:1000**
reformkurs 1.00000	schulzeit 1.00000	verbindungslinie 1.00000	erzählform 1.00000
demokratisierungsprozess 0.50042	studienzeit 0.70350	grenzlinie 0.50943	romanform 0.54450
reformpolitik 0.45847	jugendzeit 0.68140	schnittpunkt 0.50175	dialogform 0.50634
reformprogramm 0.45296	kindheit 0.61092	symmetrieachse 0.45167	darstellungsweise 0.47987
reformwillen 0.41501	schule 0.58646	nahtstelle 0.41824	literaturgattung 0.47877
reformprozess 0.40998	abitur 0.57461	schnittlinie 0.41689	versform 0.47829
...	...	...	...
reformkurs_c -0.11848	schulzeit_c -0.23277	verbindungslinie_c 0.07165	erzählform_c 0.07803
kriegsgefangener:1000	**familienausflug:1000**	**heimatkalender:1000**	**einsatzfeld:1000**
kriegsgefangener 1.00000	familienausflug 1.00000	heimatkalender 1.00000	einsatzfeld 1.00000
zwangsarbeiter 0.56735	tagesausflug 0.67161	heimatbuch 0.46712	anwendungsgebiet 0.69981
kriegsgefangenschaft 0.56660	betriebsausflug 0.61388	musenalmanach 0.43831	einsatzgebiet 0.63412
gefangener 0.56521	ausflug 0.60310	kunstkalender 0.36653	betätigungsfeld 0.61496
kriegsgefangene 0.53552	fahrradtour 0.60296	nachrichtenblatt 0.36466	tätigkeitsfeld 0.58381
internierung 0.50939	einkaufsbummel 0.55363	jahrbuch 0.35119	anwendungsfeld 0.57015
...	...	...	...
kriegsgefangener_c -0.03176	familienausflug_c 0.01378	heimatkalender_c -0.02351	einsatzfeld_c 0.09593

Table C.3 Neighbours of example compounds from the `nn-only test` set, using the `mul` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (105, 106, 306, 307), the middle and the cut-off rank (1000).

flugticket:1	**suppenteller:1**	**finanzkapital:1**	**kantonsspital:1**
flugticket 1.00000	suppenteller 1.00000	finanzkapital 1.00000	kantonsspital 1.00000
flugticket_c 0.65155	suppenteller_c 0.52512	finanzkapital_c 0.52548	kantonsspital_c 0.53133
ticket 0.60378	teller 0.51351	kapital 0.49287	marienhospital 0.52146
zugticket 0.58610	plastikbecher 0.43856	finanzsystem 0.44540	spital 0.50101
bahnticket 0.56020	zuckerdose 0.43355	kapitalismus 0.43818	kreiskrankenhaus 0.50023
visum 0.53509	löffel 0.43283	bankensystem 0.41986	frauenklinik 0.49694
fahrkarte 0.52867	kaffeetasse 0.42468	geldkapital 0.40945	kantonsschule 0.44357
stellengesuch:49	**endpunkt:49**	**routinearbeit:49**	**seestraße:49**
stellengesuch 1.00000	endpunkt 1.00000	routinearbeit 1.00000	seestraße 1.00000
inserat 0.54958	anfangspunkt 0.74470	schreibtischarbeit 0.55576	schlossstraße 0.61548
stellenangebot 0.54293	zielpunkt 0.64873	verwaltungsarbeit 0.54241	poststraße 0.60313
stellenanzeige 0.49637	ausgangspunkt 0.50388	planungsarbeit 0.51491	klosterstraße 0.60291
kontaktanzeige 0.47328	zielort 0.48114	büroarbeit 0.49090	kirchstraße 0.59774
jobportal 0.46667	endziel 0.45363	redaktionsarbeit 0.44574	waldstraße 0.57648
...	...	...	...
stellengesuch_c 0.28169	endpunkt_c 0.31385	routinearbeit_c 0.30051	seestraße_c 0.39328
kultursommer:1000	**transferliste:1000**	**kontrollzentrum:1000**	**wertmarke:1000**
kultursommer 1.00000	transferliste 1.00000	kontrollzentrum 1.00000	wertmarke 1.00000
konzertreihe 0.48630	rednerliste 0.40224	menüpunkt 0.49868	schwerbehindertenausweis 0.48191
musikfestival 0.46145	beobachtungsliste 0.39132	benutzerprofil 0.49165	zeitkarte 0.46883
musikfest 0.41195	gewinnerseite 0.34903	systemsteuerung 0.46697	ausweis 0.41929
kindertag 0.40739	dopingliste 0.32852	menüleiste 0.45865	bibliotheksausweis 0.41067
eröffnungskonzert 0.40729	fahndungsliste 0.32293	menüeintrag 0.41040	rabattkarte 0.40621
...	...	...	...
kultursommer_c 0.10624	transferliste_c 0.12538	kontrollzentrum_c 0.04169	wertmarke_c 0.05797

Table C.4 Neighbours of example compounds from the `nn-only test` set, using the `dilation` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (49) and the cut-off rank (1000).

betonstahl:1	**risikofaktor:1**	**oberstleutnant:1**	**spammail:1**
betonstahl 1.00000	risikofaktor 1.00000	oberstleutnant 1.00000	spammail 1.00000
betonstahl_c 0.40406	risikofaktor_c 0.56219	oberstleutnant_c 0.77960	spammail_c 0.50243
beton 0.39909	stressfaktor 0.55618	oberst 0.76499	postsendung 0.46579
stahlbeton 0.37356	bluthochdruck 0.53696	generalmajor 0.73439	briefsendung 0.46022
portlandzement 0.36614	arteriosklerose 0.52438	kommandeur 0.72549	spam 0.45644
schnittholz 0.36016	unsicherheitsfaktor 0.50833	brigadegeneral 0.70832	absender 0.44231
stahlerzeugung 0.35535	gesundheitsrisiko 0.49435	generalleutnant 0.68834	morddrohung 0.42403
familienserie:40	**klammerzusatz:40**	**kantonsschule:40**	**diskussionsstand:40**
familienserie 1.00000	klammerzusatz 1.00000	kantonsschule 1.00000	diskussionsstand 1.00000
kinderserie 0.61804	satzteil 0.48122	kantonsspital 0.44357	verhandlungsstand 0.63103
arztserie 0.52314	teilsatz 0.45767	werkrealschule 0.41158	forschungsstand 0.58662
jugendserie 0.50889	einleitungssatz 0.40032	volksschule 0.40813	fachdiskussion 0.49509
vorabendserie 0.49620	eingangssatz 0.36906	burggraben 0.40451	sachstand 0.49434
kindersendung 0.48800	regierungsbeschluss 0.34738	stadtbahnhof 0.39872	planungsstand 0.46088
...	...	...	...
familienserie_c 0.28299	klammerzusatz_c 0.26985	kantonsschule_c 0.31903	diskussionsstand_c 0.29451
zeitzeichen:1000	**betonwerk:1000**	**artensterben:1000**	**bergrutsch:1000**
zeitzeichen 1.00000	betonwerk 1.00000	artensterben 1.00000	bergrutsch 1.00000
zeitansage 0.40170	zementwerk 0.52908	klimaveränderung 0.67102	felssturz 0.65060
bürgerfunk 0.37417	kieswerk 0.50888	klimaerwärmung 0.64183	bergsturz 0.62149
börsenblatt 0.35123	ziegelwerk 0.50207	klimawandel 0.62865	meteoriteneinschlag 0.52390
deutschlandfunk 0.34056	chemiewerk 0.48084	umweltzerstörung 0.58204	erdrutsch 0.50994
deutschlandradio 0.33557	eisenwerk 0.45757	waldsterben 0.57910	chemieunfall 0.46229
...	...	...	...
zeitzeichen_c 0.10524	betonwerk_c 0.16580	artensterben_c 0.03732	bergrutsch_c 0.09169

Table C.5 Neighbours of example compounds from the `nn-only test` set, using the `addition` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (40) and the cut-off rank (1000).

kriegsgefahr:1	**brückenpfeiler:1**	**spiegelbild:1**	**erscheinungsform:1**
kriegsgefahr 1.00000	brückenpfeiler 1.00000	spiegelbild 1.00000	erscheinungsform 1.00000
kriegsgefahr_c 0.46155	brückenpfeiler_c 0.56540	spiegelbild_c 0.49959	erscheinungsform_c 0.51178
hochwassergefahr 0.45071	pfeiler 0.55069	spiegel 0.42648	erscheinung 0.49181
umweltzerstörung 0.43656	betonpfeiler 0.50483	spiegelung 0.41682	ausdrucksform 0.46719
unfallgefahr 0.41535	kaimauer 0.49538	wirklichkeit 0.41160	gestalt 0.44266
inflationsgefahr 0.40978	betonwand 0.49265	bild 0.40622	darstellungsform 0.40976
brandgefahr 0.40631	verkehrsinsel 0.45392	realität 0.39703	beschaffenheit 0.40637
westhafen:36	**lautsprecherbox:36**	**kirchstraße:36**	**informationsmedium:36**
westhafen 1.00000	lautsprecherbox 1.00000	kirchstraße 1.00000	informationsmedium 1.00000
osthafen 0.61135	deckenlampe 0.45522	schulstraße 0.76794	kommunikationsmedium 0.73968
westpark 0.51842	glasvitrine 0.44521	marktstraße 0.75821	informationsquelle 0.63858
nordbahnhof 0.51150	antennenanlage 0.44132	bahnhofstraße 0.72634	massenmedium 0.61141
gänsemarkt 0.45759	holzbaracke 0.43539	poststraße 0.72166	marketinginstrument 0.54893
opernplatz 0.44832	frequenzweiche 0.42759	dorfstraße 0.69395	kommunikationsmittel 0.54508
...	...	...	...
westhafen_c 0.35479	lautsprecherbox_c 0.37441	kirchstraße_c 0.46373	informationsmedium_c 0.35383
raumordnung:1000	**körperöffnung:1000**	**lindenblatt:1000**	**rechtsetzung:1000**
raumordnung 1.00000	körperöffnung 1.00000	lindenblatt 1.00000	rechtsetzung 1.00000
landesplanung 0.75758	muskelbewegung 0.42673	schulterblatt 0.30866	aufgabenerledigung 0.46158
raumplanung 0.63917	kriegshandlung 0.40181	autodach 0.30522	steuererhebung 0.42723
städtebau 0.60532	hautfalte 0.39828	mauerstück 0.30212	rechtsangleichung 0.41123
landschaftsplanung 0.56305	landschaftsform 0.39658	musikredakteur 0.29830	rechtsanwendung 0.40867
landesentwicklung 0.54507	zahnwurzel 0.39494	bandmitglied 0.29817	bürokratieabbau 0.40687
...	...	...	...
raumordnung_c 0.13748	körperöffnung_c 0.13487	lindenblatt_c 0.06087	rechtsetzung_c 0.08577

Table C.6 Neighbours of example compounds from the `nn-only test` set, using the `w_addition` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (36) and the cut-off rank (1000).

literaturliste:1	millionenbetrag:1	kammerchor:1	kirchentür:1
literaturliste 1.00000	millionenbetrag 1.00000	kammerchor 1.00000	kirchentür 1.00000
literaturliste_c 0.64613	millionenbetrag_c 0.64081	kammerchor_c 0.66772	kirchentür_c 0.50159
linksammlung 0.63926	millionenhöhe 0.58265	kammerorchester 0.64804	schlosskirche 0.48988
literaturverzeichnis 0.63369	millionensumme 0.55848	jugendchor 0.59876	gartentür 0.48343
linkliste 0.61794	milliardenhöhe 0.53395	sinfonieorchester 0.59030	stahltür 0.46855
materialsammlung 0.56403	prozentbereich 0.52627	kinderchor 0.58360	gittertor 0.45725
stichwortverzeichnis 0.52657	millionenschaden 0.41002	knabenchor 0.56670	metalltür 0.44895
dienstunfähigkeit:8	**sacharbeit:8**	**flachwasserbereich:8**	**nonnenkloster:8**
dienstunfähigkeit 1.00000	sacharbeit 1.00000	flachwasserbereich 1.00000	nonnenkloster 1.00000
erwerbsunfähigkeit 0.59667	parlamentsarbeit 0.49931	flachwasserzone 0.54073	frauenkloster 0.75747
berufsunfähigkeit 0.58996	übersetzungsarbeit 0.47327	schilfgürtel 0.48476	damenstift 0.56505
ruhestand 0.50842	regierungsarbeit 0.45699	flachwasser 0.47880	chorherrenstift 0.55773
erwerbsminderung 0.50664	basisarbeit 0.45547	fischteich 0.45389	benediktinerkloster 0.54998
altersrente 0.48509	parteiarbeit 0.43764	uferstreifen 0.45278	dominikanerkloster 0.53757
...	...	...	...
dienstunfähigkeit_c 0.43460	sacharbeit_c 0.41684	flachwasserbereich_c 0.44431	nonnenkloster_c 0.51559
adelssitz:1000	**lügengeschichte:1000**	**schildhaupt:1000**	**bankleitzahl:1000**
adelssitz 1.00000	lügengeschichte 1.00000	schildhaupt 1.00000	bankleitzahl 1.00000
rittersitz 0.75560	räuberpistole 0.59323	wellenbalken 0.58326	kontonummer 0.87574
herrensitz 0.74777	horrorgeschichte 0.57983	wappenschild 0.49654	kontoinhaber 0.52949
herrschaftssitz 0.65484	kindergeschichte 0.47823	stammwappen 0.42626	spendenkonto 0.52281
renaissancebau 0.56082	tirade 0.45841	brustschild 0.38956	verwendungszweck 0.47371
familiensitz 0.55827	dreiecksgeschichte 0.44989	wappenbild 0.36243	sparkasse 0.45302
...	...	...	...
adelssitz_c 0.16478	lügengeschichte_c 0.19059	schildhaupt_c 0.07827	bankleitzahl_c 0.03088

Table C.7 Neighbours of example compounds from the nn-only test set, using the lexfunc representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (8) and the cut-off rank (1000).

verlobungszeit:1	knochenhaut:1	schuhcreme:1	wasserspiel:1
verlobungszeit 1.00000	knochenhaut 1.00000	schuhcreme 1.00000	wasserspiel 1.00000
verlobungszeit_c 0.50116	knochenhaut_c 0.61244	schuhcreme_c 0.51476	wasserspiel_c 0.48561
wirkungszeit 0.48789	gelenkkapsel 0.60170	zahnpasta 0.49788	wasserbecken 0.47995
friedenszeit 0.46251	schleimbeutel 0.52687	acrylfarbe 0.47672	bachlauf 0.46841
trauerzeit 0.43890	hirnhaut 0.48695	melissengeist 0.47571	wasserbassin 0.46445
haftzeit 0.40281	nervengewebe 0.48656	haarspülung 0.47266	brunnenanlage 0.44638
entscheidungsphase 0.40213	knorpel 0.47114	terpentin 0.45167	fischteich 0.43950
bundestagsfraktion:3	**windjacke:3**	**firmenwebsite:3**	**kochsalzlösung:3**
bundestagsfraktion 1.00000	windjacke 1.00000	firmenwebsite 1.00000	kochsalzlösung 1.00000
landtagsfraktion 0.75129	regenjacke 0.63745	firmenhomepage 0.80601	salzlösung 0.70271
fraktion 0.65323	jeansjacke 0.62556	webpräsenz 0.57686	kochsalz 0.51558
bundestagsfraktion_c 0.64196	windjacke_c 0.59997	firmenwebsite_c 0.56662	kochsalzlösung_c 0.48947
bündnis 0.61563	jeanshose 0.57968	internetpräsenz 0.50130	nasenspray 0.48070
kreistagsfraktion 0.61310	motorradjacke 0.56738	firmenpräsentation 0.49971	elektrolytlösung 0.47519
bundespartei 0.60149	latzhose 0.56195	webauftritt 0.46307	natronlauge 0.46132
pechstein:1000	**elfenbeinturm:1000**	**stoßfänger:1000**	**wegwarte:1000**
pechstein 1.00000	elfenbeinturm 1.00000	stoßfänger 1.00000	wegwarte 1.00000
dopingfall 0.36403	wissenschaftsbetrieb 0.46285	kotflügel 0.74618	schafgarbe 0.55761
klee 0.32240	schneckenhaus 0.39820	kühlergrill 0.74419	nachtkerze 0.54511
parteivorsitzende 0.30054	kunstbetrieb 0.38185	wagenfarbe 0.67383	küchenschelle 0.53882
weltcup 0.29197	jargon 0.37867	frontpartie 0.66997	königskerze 0.52426
steiger 0.29133	orchestergraben 0.37317	motorhaube 0.65202	kornblume 0.50359
...	...	...	...
pechstein_c 0.05878	elfenbeinturm_c 0.13736	stoßfänger_c 0.16159	wegwarte_c 0.08229

Table C.8 Neighbours of example compounds from the nn-only test set, using the fulladd representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

kandidatenliste:1	**budgetdefizit:1**	**holzkohle:1**	**gürtelschnalle:1**
kandidatenliste 1.00000	budgetdefizit 1.00000	holzkohle 1.00000	gürtelschnalle 1.00000
kandidatenliste__c 0.63785	budgetdefizit__c 0.65943	holzkohle__c 0.59483	gürtelschnalle__c 0.65171
wahlliste 0.62527	haushaltsdefizit 0.64377	schwefel 0.52316	schnalle 0.48734
vorschlagsliste 0.57987	handelsdefizit 0.61463	brennstoff 0.51826	motorradjacke 0.47604
stadtratswahl 0.53681	handelsbilanzdefizit 0.61146	torf 0.50332	goldkette 0.45954
landesliste 0.46998	haushaltsloch 0.51802	kohle 0.49648	ritterrüstung 0.45376
wahlprogramm 0.44950	finanzvolumen 0.49855	steinkohle 0.49128	pfeilspitze 0.45077
kopfnicken:3	**mittlerfunktion:3**	**kostensteigerung:3**	**buchpreis:3**
kopfnicken 1.00000	mittlerfunktion 1.00000	kostensteigerung 1.00000	buchpreis 1.00000
nicken 0.78996	vermittlerrolle 0.71572	preissteigerung 0.67603	friedenspreis 0.49533
kopfschütteln 0.63359	brückenfunktion 0.59088	kostenexplosion 0.66794	literaturpreis 0.48484
kopfnicken__c 0.61911	mittlerfunktion__c 0.47955	kostensteigerung__c 0.62525	buchpreis__c 0.47997
schulterzucken 0.59916	führungsfunktion 0.46672	kostenentwicklung 0.58990	jugendliteraturpreis 0.46408
achselzucken 0.59458	schlüsselfunktion 0.45244	kostenreduktion 0.54452	kulturpreis 0.43849
stirnrunzeln 0.56648	schlüsselstellung 0.44211	gebührenerhöhung 0.54289	filmpreis 0.43703
elfenbeinturm:1000	**ohrhänger:1000**	**wiesenweihe:1000**	**jahrgang:1000**
elfenbeinturm 1.00000	ohrhänger 1.00000	wiesenweihe 1.00000	jahrgang 1.00000
wissenschaftsbetrieb 0.46285	ohrstecker 0.64256	rohrweihe 0.65333	altersklasse 0.52201
schneckenhaus 0.39820	kettenanhänger 0.50323	uferschnepfe 0.47826	klasse 0.47660
kunstbetrieb 0.38185	halskette 0.45218	wachtelkönig 0.47073	jahrgangsstufe 0.43802
jargon 0.37867	armband 0.42427	feldlerche 0.44601	heft 0.43289
orchestergraben 0.37317	armreif 0.41709	zauneidechse 0.42596	schuljahr 0.41897
...	...	...	...
elfenbeinturm__c 0.13119	ohrhänger__c 0.12061	wiesenweihe__c 0.06221	jahrgang__c 0.04815

Table C.9 Neighbours of example compounds from the `nn-only test` set, using the `matrix` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

notbetrieb:1	**kulturtag:1**	**rattengift:1**	**blumenkind:1**
notbetrieb 1.00000	kulturtag 1.00000	rattengift 1.00000	blumenkind 1.00000
notbetrieb__c 0.51975	kulturtag__c 0.51901	rattengift__c 0.44518	blumenkind__c 0.46442
notprogramm 0.51340	jugendtag 0.40774	vergiftung 0.43146	landei 0.44734
handbetrieb 0.50669	fortbildungskurs 0.40524	nervengift 0.39849	bürohengst 0.43652
museumsbetrieb 0.45225	ferienkurs 0.40035	gift 0.38821	mauerblümchen 0.41811
zugbetrieb 0.41359	umwelttag 0.39099	schlangengift 0.37262	schulaufsatz 0.40413
eisenbahnbetrieb 0.40772	eröffnungsabend 0.38778	rauschgift 0.37219	urlaubsfoto 0.40066
königsfamilie:3	**nutzungskonzept:3**	**unterlassungsanspruch:3**	**deckungslücke:3**
königsfamilie 1.00000	nutzungskonzept 1.00000	unterlassungsanspruch 1.00000	deckungslücke 1.00000
königshaus 0.65108	sanierungskonzept 0.68259	schadensersatzanspruch 0.70699	finanzlücke 0.78838
herrscherfamilie 0.55195	finanzierungskonzept 0.67772	schadenersatzanspruch 0.66762	finanzierungslücke 0.65170
königsfamilie__c 0.49213	nutzungskonzept__c 0.67664	unterlassungsanspruch__c 0.66600	deckungslücke__c 0.52685
sommerresidenz 0.48254	verkehrskonzept 0.55334	erstattungsanspruch 0.60777	haushaltsloch 0.52232
hofstaat 0.46093	marketingkonzept 0.52754	unterlassung 0.55488	wissenslücke 0.44906
königspaar 0.45777	betriebskonzept 0.52228	kläger 0.48540	zinslast 0.42906
jahrgang:1000	**besenreiser:1000**	**goldjunge:1000**	**teufelskreis:1000**
jahrgang 1.00000	besenreiser 1.00000	goldjunge 1.00000	teufelskreis 1.00000
altersklasse 0.52201	krampfader 0.43881	eidgenosse 0.41849	dilemma 0.47161
klasse 0.47660	sonnenallergie 0.43391	tischnachbar 0.38663	spirale 0.46089
jahrgangsstufe 0.43802	krähenfüße 0.43122	bundesgenosse 0.37257	isolation 0.45622
heft 0.43289	nagelpilz 0.41929	lausbub 0.36786	kreislauf 0.44356
schuljahr 0.41897	orangenhaut 0.38424	musterknabe 0.35993	armut 0.41119
...	...	...	...
jahrgang__c 0.15994	besenreiser__c 0.06124	goldjunge__c 0.14188	teufelskreis__c 0.16199

Table C.10 Neighbours of example compounds from the `nn-only test` set, using the `fulllex` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

kaiserhof:1	filmklassiker:1	gedichtsammlung:1	genossenschaftsbank:1
kaiserhof 1.00000	filmklassiker 1.00000	gedichtsammlung 1.00000	genossenschaftsbank 1.00000
kaiserhof_c 0.53337	filmklassiker_c 0.65672	gedichtsammlung_c 0.58220	genossenschaftsbank_c 0.49299
königshof 0.50505	kultfilm 0.56101	gedichtband 0.57083	kreditgenossenschaft 0.44359
westbahnhof 0.47239	stummfilm 0.53810	kurzgeschichtensammlung 0.55403	finanzinstitut 0.43849
südbahnhof 0.45833	klassiker 0.51386	lyrikband 0.54998	geschäftsbank 0.43381
kurhaus 0.41580	filmgeschichte 0.49563	gedichtzyklus 0.53842	autofirma 0.42633
hofburg 0.41576	filmreihe 0.46209	buchpublikation 0.51648	raiffeisenbank 0.41938
sportbecken:3	**spielleitung:3**	**meeresgott:3**	**bergfestung:3**
sportbecken 1.00000	spielleitung 1.00000	meeresgott 1.00000	bergfestung 1.00000
schwimmerbecken 0.79349	rennleitung 0.43770	kriegsgott 0.43602	schlossruine 0.51806
nichtschwimmerbecken 0.71534	schiedsrichter 0.42218	sonnengott 0.41660	eishöhle 0.45929
sportbecken_c 0.54605	spielleitung_c 0.42094	meeresgott_c 0.41562	bergfestung_c 0.45664
solebecken 0.53926	vereinsführung 0.38894	orakelspruch 0.36983	marinebasis 0.44476
sprungturm 0.53286	kampfrichter 0.38565	gottvater 0.36892	palastanlage 0.44079
wasserrutsche 0.50002	einsatzleitung 0.37552	ödipus 0.35179	felsenhöhle 0.43599
tierkreis:1000	**buchholz:1000**	**schürzenjäger:1000**	**pechstein:1000**
tierkreis 1.00000	buchholz 1.00000	schürzenjäger 1.00000	pechstein 1.00000
tierkreiszeichen 0.62106	nordheide 0.57025	frauenheld 0.57270	dopingfall 0.36403
frühlingspunkt 0.43581	vogt 0.44773	bürgerschreck 0.50525	klee 0.32240
steinbock 0.42101	horst 0.40577	trinker 0.44374	parteivorsitzende 0.30054
horoskop 0.41849	bergheim 0.38159	menschenfresser 0.42879	weltcup 0.29197
sternzeichen 0.41526	schäfer 0.37702	spaßvogel 0.41207	steiger 0.29133
...	...	...	...
tierkreis_c 0.12350	buchholz_c 0.00390	schürzenjäger_c 0.17954	pechstein_c 0.08661

Table C.11 Neighbours of example compounds from the `nn-only test` set, using the `addmask` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

fenstersturz:1	puppenbühne:1	goldmünze:1	treuhandgesellschaft:1
fenstersturz 1.00000	puppenbühne 1.00000	goldmünze 1.00000	treuhandgesellschaft 1.00000
fenstersturz_c 0.47161	puppenbühne_c 0.50230	goldmünze_c 0.68389	treuhandgesellschaft_c 0.60968
balkankrieg 0.37277	puppentheater 0.48456	silbermünze 0.65940	immobiliengesellschaft 0.53690
umsturzversuch 0.35756	theatertruppe 0.40536	goldbarren 0.52926	wirtschaftsprüfungsgesellschaft 0.50366
mauerbau 0.34484	laienspielgruppe 0.40371	münze 0.52391	kapitalanlagegesellschaft 0.43972
bauernaufstand 0.34026	kinderzirkus 0.37598	euromünze 0.40528	holdinggesellschaft 0.43395
grabenkrieg 0.33789	gesangsgruppe 0.37250	briefmarke 0.40451	treuhand 0.42412
wissenschaftssprache:2	**programmstart:2**	**felgenbremse:2**	**kristallgitter:2**
wissenschaftssprache 1.00000	programmstart 1.00000	felgenbremse 1.00000	kristallgitter 1.00000
verkehrssprache 0.60098	systemstart 0.71687	scheibenbremse 0.65614	kristallstruktur 0.53256
wissenschaftssprache_c 0.57802	programmstart_c 0.55642	felgenbremse_c 0.53318	kristallgitter_c 0.50674
arbeitssprache 0.53413	verbindungsaufbau 0.53179	trommelbremse 0.50331	gitterstruktur 0.49505
umgangssprache 0.48379	fehlermeldung 0.45016	federgabel 0.46576	elektronenhülle 0.43213
weltsprache 0.47488	erststart 0.43107	rücktrittbremse 0.45414	metallgitter 0.42672
alltagssprache 0.44798	installationsprogramm 0.42529	gangschaltung 0.45402	wassermolekül 0.40722
kreuzband:1000	**pechstein:1000**	**bruchteil:1000**	**glücksspirale:1000**
kreuzband 1.00000	pechstein 1.00000	bruchteil 1.00000	glücksspirale 1.00000
sprunggelenk 0.61506	dopingfall 0.36403	zehntel 0.55662	lotterie 0.49057
kreuzbandriss 0.58321	klee 0.32240	prozentsatz 0.55219	lotto 0.41623
kniegelenk 0.57542	parteivorsitzende 0.30054	drittel 0.49671	kulturfonds 0.32268
achillessehne 0.57250	weltcup 0.29197	hälfte 0.48631	klassenlotterie 0.31018
schultergelenk 0.52865	steiger 0.29133	summe 0.47587	stiftungskapital 0.30479
...	...	...	...
kreuzband_c 0.12019	pechstein_c 0.09469	bruchteil_c 0.06994	glücksspirale_c 0.09046

Table C.12 Neighbours of example compounds from the `nn-only test` set, using the `wmask` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (2) and the cut-off rank (1000).

kriegsgefangener:1	**kongresshalle:1**	**eigenschaftswort:1**	**gefangenenaustausch:1**
kriegsgefangener 1.00000	kongresshalle 1.00000	eigenschaftswort 1.00000	gefangenenaustausch 1.00000
kriegsgefangener_c 0.66733	kongresshalle_c 0.56026	eigenschaftswort_c 0.49944	gefangenenaustausch_c 0.47592
zwangsarbeiter 0.56735	kongresszentrum 0.50006	begriffspaar 0.41511	feuerpause 0.46399
kriegsgefangenschaft 0.56660	veranstaltungshalle 0.49029	zahlwort 0.41479	waffenstillstand 0.42153
gefangener 0.56521	stadthalle 0.47710	verb 0.37766	waffengang 0.40481
kriegsgefangene 0.53552	eissporthalle 0.47178	persönlichkeitsmerkmal 0.36437	handelsvertrag 0.40111
internierung 0.50939	konzerthalle 0.45966	fabeltier 0.36368	raketenangriff 0.39876
kirchbau:2	**schadenfall:2**	**hindernislauf:2**	**gemeindesaal:2**
kirchbau 1.00000	schadenfall 1.00000	hindernislauf 1.00000	gemeindesaal 1.00000
kirchenneubau 0.72069	schadensfall 0.88803	hürdenlauf 0.64442	gemeindehaus 0.82184
kirchbau_c 0.66617	schadenfall_c 0.80039	hindernislauf_c 0.62493	gemeindesaal_c 0.81033
schulneubau 0.60698	versicherungsfall 0.66332	geländelauf 0.55704	pfarrsaal 0.80820
dombau 0.59400	versicherungsnehmer 0.55142	vierkampf 0.55051	pfarrheim 0.76253
kirchenbau 0.58399	versicherer 0.51248	slalom 0.51264	gemeindezentrum 0.74898
museumsbau 0.50437	versicherungssumme 0.50689	staffellauf 0.50778	pfarrzentrum 0.67701
nachtschatten:1000	**besenreiser:1000**	**wertschätzung:1000**	**tierkreis:1000**
nachtschatten 1.00000	besenreiser 1.00000	wertschätzung 1.00000	tierkreis 1.00000
lotos 0.36870	krampfader 0.43881	anerkennung 0.64735	tierkreiszeichen 0.62106
herbstzeitlose 0.34442	sonnenallergie 0.43391	sympathie 0.58090	frühlingspunkt 0.43581
mondstein 0.31443	krähenfüße 0.43122	offenheit 0.54826	steinbock 0.42101
johannisbeere 0.31363	nagelpilz 0.41929	hilfsbereitschaft 0.53844	horoskop 0.41849
rabe 0.31309	orangenhaut 0.38424	solidarität 0.52666	sternzeichen 0.41526
...	...	...	...
nachtschatten_c 0.12018	besenreiser_c 0.08151	wertschätzung_c 0.09489	tierkreis_c 0.11723

Table C.13 Neighbours of example compounds from the `nn-only test` set, using the `multimatrix` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (2) and the cut-off rank (1000).

C.2 Composed Representations for English Compounds

bear__hug:1	**wheat__gluten:1**	**self__contradiction:1**	**trust__fund:1**
bear__hug 1.00000	wheat__gluten 1.00000	self__contradiction 1.00000	trust__fund 1.00000
bear__hug__c 0.52217	wheat__gluten__c 0.53082	self__contradiction__c 0.61882	trust__fund__c 0.53749
hug 0.51559	gluten 0.52487	contradiction 0.59714	fund 0.52810
thumbs-up 0.47484	soya 0.49889	imperfection 0.41212	funds 0.50985
thumbs-up__sign 0.43816	soya__milk 0.46335	self__abnegation 0.40703	reserve__fund 0.45344
gunny__sack 0.39569	dairy__product 0.45730	self__absorption 0.40170	endowment__fund 0.45103
goose__pimple 0.38553	soy__flour 0.45423	self__justification 0.38959	fund__program 0.44768
water__hole:449	**catering__service:450**	**movie__role:451**	**mountain__summit:454**
water__hole 1.00000	catering__service 1.00000	movie__role 1.00000	mountain__summit 1.00000
swimming__hole 0.46194	catering__company 0.64547	bit__part 0.56831	mountain__peak 0.61218
water__source 0.45894	catering 0.54893	television__appearance 0.54874	rock__outcrop 0.47954
creek__bed 0.45723	food__service 0.48734	screen__debut 0.50435	mountain__range 0.46001
camping__ground 0.42570	consultancy__service 0.45602	stage__work 0.45850	mountain__area 0.45847
watering__hole 0.42167	maintenance__service 0.43561	stage__appearance 0.44975	foot__mountain 0.43734
...	...	...	...
water__hole__c 0.20123	catering__service__c 0.21350	movie__role__c 0.23136	mountain__summit__c 0.23471
bath__product:1000	**alleviation__program:1000**	**caucus__meeting:1000**	**spring__scale:1000**
bath__product 1.00000	alleviation__program 1.00000	caucus__meeting 1.00000	spring__scale 1.00000
bath__soap 0.52700	poverty__alleviation 0.69215	party__caucus 0.61939	spring__balance 0.65253
bath__oil 0.51992	poverty__relief 0.63933	leadership__meeting 0.57863	vernier__caliper 0.59026
beauty__product 0.51054	stem__research 0.61667	party__meeting 0.56668	self__loader 0.54831
bubble__bath 0.47617	customer__conversation 0.58969	luncheon__meeting 0.56332	foot__lever 0.54624
body__lotion 0.46524	self__contemplation 0.58957	prayer__session 0.53306	vacuum__gauge 0.54413
...	...	...	...
bath__product__c 0.08969	alleviation__program__c -0.20364	caucus__meeting__c 0.17859	spring__scale__c 0.03085

Table C.14 Neighbours of example compounds from the `en-comcom` `test` set, using the `head` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (449, 450, 451, 454) and the cut-off rank (1000).

Examples of Neighbours of the Composed Representations

training__program:1	**tuition__cost:1**	**surveillance__system:1**	**chat__room:1**
training__program 1.00000	tuition__cost 1.00000	surveillance__system 1.00000	chat__room 1.00000
training__program__c 0.72370	tuition__cost__c 0.57784	surveillance__system__c 0.60239	chat__room__c 0.65380
training 0.71293	tuition 0.56557	surveillance 0.58624	chat 0.65279
training__course 0.65797	living__expense 0.56488	security__system 0.50898	bulletin__board 0.62545
education__program 0.64155	tuition__fee 0.52773	surveillance__technology 0.43867	discussion__group 0.59571
job__training 0.58638	program__cost 0.52208	radar__system 0.43394	chat__group 0.59259
training__session 0.57561	tuition__increase 0.50444	surveillance__equipment 0.43105	chat__session 0.59116
evaluation__purpose:470	**sun__deck:470**	**thigh__injury:472**	**phone__operator:472**
evaluation__purpose 1.00000	sun__deck 1.00000	thigh__injury 1.00000	phone__operator 1.00000
publicity__purpose 0.53992	promenade__deck 0.61175	groin__injury 0.83244	phone__network 0.66830
research__purpose 0.53319	dining__area 0.58598	calf__injury 0.82590	phone__carrier 0.66003
emergency__use 0.52510	seating__area 0.58310	hamstring__injury 0.82071	phone__company 0.65695
propaganda__purpose 0.52397	lido__deck 0.57928	hamstring__problem 0.80122	phone__giant 0.63925
defense__purpose 0.51303	sun__lounge 0.57856	ankle__injury 0.77489	telecom__operator 0.60251
...	...	...	...
evaluation__purpose__c 0.25187	sun__deck__c 0.24463	thigh__injury__c 0.21614	phone__operator__c 0.21241
surface__fire:1000	**sport__competition:1000**	**kidney__dialysis:1000**	**union__sympathizer:1000**
surface__fire 1.00000	sport__competition 1.00000	kidney__dialysis 1.00000	union__sympathizer 1.00000
crown__fire 0.61235	sport__event 0.59090	dialysis__machine 0.81977	loyalist__troop 0.63439
ground__fire 0.42630	sports__meeting 0.53559	dialysis__treatment 0.77314	draft__evader 0.61964
flier__program 0.39366	gymnastics__competition 0.52647	dialysis__patient 0.67373	slave__dealer 0.61583
gas__jet 0.39173	sport__league 0.48917	dialysis 0.57501	prison__employee 0.61396
grass__fire 0.38710	swim__meet 0.48791	breathing__machine 0.48935	riot__officer 0.60783
...	...	...	...
surface__fire__c -0.07355	sport__competition__c 0.18990	kidney__dialysis__c 0.23924	union__sympathizer__c -0.04632

Table C.15 Neighbours of example compounds from the `en-comcom` `test` set, using the `modifier` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (470, 472) and the cut-off rank (1000).

police__constable:117	**water__purification:242**	**listing__agent:246**	**electron__microscopy:290**
police__constable 1.00000	water__purification 1.00000	listing__agent 1.00000	electron__microscopy 1.00000
police__sergeant 0.65905	purification__system 0.78499	home__seller 0.50637	electron__microscope 0.75592
constable 0.58711	reverse__osmosis 0.56329	selling__agent 0.47405	microscopy 0.71015
police__officer 0.55419	water__treatment 0.55478	seller 0.47063	spectroscopy 0.62403
police__inspector 0.54284	desalination 0.51687	buyer 0.44725	fluorescence__microscopy 0.58475
sergeant 0.49282	water__filter 0.48864	sale__price 0.43591	diffraction 0.56332
...	...	...	...
police__constable__c 0.29209	water__purification__c 0.24130	listing__agent__c 0.23349	electron__microscopy__c 0.20866
consumer__item:1000	**bowler__hat:1000**	**magic__realism:1000**	**day__nursery:1000**
consumer__item 1.00000	bowler__hat 1.00000	magic__realism 1.00000	day__nursery 1.00000
luxury__item 0.61573	frock__coat 0.59376	realism 0.45515	nursery__school 0.68084
consumer__durables 0.60859	dinner__jacket 0.58674	film__noir 0.41939	play__group 0.63668
consumer__goods 0.57867	trench__coat 0.58418	science__fiction 0.40187	nursery 0.57191
durables 0.54177	bow__tie 0.57410	fiction 0.39948	day__care 0.50996
household__appliance 0.54068	business__suit 0.57016	prose__poem 0.39913	infant__school 0.46909
...	...	...	...
consumer__item__c -0.10813	bowler__hat__c 0.01606	magic__realism__c 0.07804	day__nursery__c 0.00905
rowing__club:1000	**construction__worker:1000**	**computer__address:1000**	**state__support:1000**
rowing__club 1.00000	construction__worker 1.00000	computer__address 1.00000	state__support 1.00000
boat__club 0.60395	factory__worker 0.65234	tickler__file 0.64206	government__support 0.67951
rowing 0.50167	construction__site 0.62500	garden__experience 0.61454	state__subsidy 0.65265
yacht__club 0.49730	maintenance__worker 0.61880	import__system 0.57978	government__assistance 0.60804
hockey__club 0.49090	truck__driver 0.58486	cheek__muscle 0.55158	government__subsidy 0.60186
boat__race 0.43238	construction__crew 0.57325	telephone__listing 0.54964	government__funding 0.58542
...	...	...	...
rowing__club__c -0.04987	construction__worker__c -0.12450	computer__address__c -0.06272	state__support__c -0.04262

Table C.16 Neighbours of example compounds from the `en-comcom` `test` set, using the `multiplication` representation of the compound (300 dim). Four examples each of compounds assigned the best ranks (117,242,246,290), the middle (1000) and the cut-off rank (1000).

customer_satisfaction:1	**trust_fund:1**	**malaria_parasite:1**	**resource_management:1**
customer_satisfaction 1.00000	trust_fund 1.00000	malaria_parasite 1.00000	resource_management 1.00000
customer_satisfaction_c 0.67144	trust_fund_c 0.54359	malaria_parasite_c 0.60471	resource_management_c 0.63429
service_quality 0.64684	fund 0.52810	parasite 0.55700	management_issue 0.60866
customer_loyalty 0.64278	funds 0.50985	malaria 0.43453	management 0.59507
product_quality 0.60494	reserve_fund 0.45344	nematode 0.43028	management_practice 0.58025
satisfaction 0.59875	endowment_fund 0.45103	blood_cell 0.41639	water_resource 0.57911
customer_service 0.57920	fund_program 0.44768	influenza_virus 0.40990	management_strategy 0.57805
shrimp_cocktail:187	**production_order:187**	**volunteer_group:187**	**birth_control:187**
shrimp_cocktail 1.00000	production_order 1.00000	volunteer_group 1.00000	birth_control 1.00000
crab_cake 0.63660	pilot_film 0.41666	volunteer_organization 0.64771	family_planning 0.64467
steak_tartare 0.55323	search_warrant 0.39754	community_group 0.53170	abortion 0.62805
lobster_thermidor 0.54750	closure_order 0.37481	neighborhood_group 0.53159	pill 0.52069
porterhouse_steak 0.52034	production_run 0.37274	conservation_group 0.51916	population_control 0.48797
tuna_salad 0.51132	government_order 0.36837	church_group 0.50220	hormone_replacement 0.46705
...	...	...	...
shrimp_cocktail_c 0.33032	production_order_c 0.24727	volunteer_group_c 0.27699	birth_control_c 0.25072
adjustment_problem:1000	**rock_performance:1000**	**accession_negotiation:1000**	**test_room:1000**
adjustment_problem 1.00000	rock_performance 1.00000	accession_negotiation 1.00000	test_room 1.00000
speech_defect 0.54151	rock_video 0.50055	accession_talk 0.81776	testing_room 0.66581
discipline_problem 0.52526	rock_album 0.47910	membership_talk 0.76279	china_closet 0.55543
morale_problem 0.52153	rock_singer 0.47319	membership_negotiation 0.71746	starting_stall 0.55484
billing_problem 0.51368	solo_performance 0.44505	candidate_country 0.53624	dice_box 0.54466
learning_disorder 0.51168	rap_album 0.41676	ratification_process 0.48585	chicken_yard 0.53640
...	...	...	...
adjustment_problem_c 0.05901	rock_performance_c 0.15311	accession_negotiation_c 0.17679	test_room_c 0.07315

Table C.17 Neighbours of example compounds from the `en-comcom test` set, using the `dilation` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (187) and the cut-off rank (1000).

initiation_rite:1	**sewing_machine:1**	**poverty_trap:1**	**dipole_antenna:1**
initiation_rite 1.00000	sewing_machine 1.00000	poverty_trap 1.00000	dipole_antenna 1.00000
initiation_rite_c 0.48302	sewing_machine_c 0.59397	poverty_trap_c 0.48964	dipole_antenna_c 0.58292
initiation 0.41953	sewing 0.57117	benefit_system 0.46336	dipole 0.51744
kickback_scheme 0.39792	quilting 0.48184	welfare_system 0.41871	antenna 0.46497
prison_program 0.38725	machine_stitch 0.47350	marriage_penalty 0.40658	radiation_pattern 0.44861
rite 0.38340	machine_operator 0.46966	means_test 0.38529	dish_antenna 0.44799
ritual_killing 0.37734	embroidery 0.46061	rat_race 0.38416	throwing_knife 0.42321
rescue_vehicle:100	**reaction_time:100**	**advance_copy:100**	**phase_space:100**
rescue_vehicle 1.00000	reaction_time 1.00000	advance_copy 1.00000	phase_space 1.00000
rescue_equipment 0.49602	working_memory 0.51014	review_copy 0.59946	space_time 0.44398
rescue_squad 0.45921	response_time 0.47491	draft_copy 0.56645	phase_transition 0.42528
rescue_personnel 0.45826	learning_ability 0.46824	paperback_copy 0.51407	ground_state 0.41160
rescue_crew 0.45364	processing_time 0.44939	sneak_preview 0.47378	field_theory 0.41095
rescue_vessel 0.42381	cycle_time 0.44638	galley_proof 0.41045	quantum 0.40099
...	...	...	...
rescue_vehicle_c 0.30007	reaction_time_c 0.27750	advance_copy_c 0.27408	phase_space_c 0.26103
paper_market:1000	**vacuum_aspiration:1000**	**showpiece_event:1000**	**cart_track:1000**
paper_market 1.00000	vacuum_aspiration 1.00000	showpiece_event 1.00000	cart_track 1.00000
property_lending 0.63559	self_stimulation 0.46542	surprise_team 0.55356	dirt_path 0.58237
property_share 0.50744	crystal_gazing 0.45362	budget_session 0.54997	go_cart 0.56240
aircraft_market 0.50299	garden_cart 0.45026	weekend_election 0.53930	dirt_track 0.52775
property_sector 0.48177	exchange_transfusion 0.44189	rating_period 0.53279	logging_road 0.49539
air_lane 0.47996	motorized_wheelchair 0.43895	year_area 0.52381	gravel_road 0.47347
...	...	...	...
paper_market_c 0.16682	vacuum_aspiration_c 0.07520	showpiece_event_c 0.28813	cart_track_c 0.23656

Table C.18 Neighbours of example compounds from the `en-comcom test` set, using the `addition` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (100) and the cut-off rank (1000).

election__campaign:1	**decree__nisi:1**	**dung__beetle:1**	**drinking__alcohol:1**
election__campaign 1.00000	decree__nisi 1.00000	dung__beetle 1.00000	drinking__alcohol 1.00000
election__campaign__c 0.70908	decree__nisi__c 0.51042	dung__beetle__c 0.53302	drinking__alcohol__c 0.64101
election 0.65634	decree 0.45393	stag__beetle 0.45368	smoking 0.61267
fall__election 0.58337	annulment 0.45290	beetle 0.42449	drinking 0.60360
election__cycle 0.57361	summary__judgement 0.44723	pill__bug 0.41598	alcohol__consumption 0.57452
campaign__period 0.57320	committal__order 0.39749	dung 0.39514	binge__drinking 0.54170
campaign__team 0.55469	divorce 0.39232	water__strider 0.38161	alcohol__use 0.53805
rescue__vehicle:100	**application__procedure:100**	**music__market:100**	**phase__space:100**
rescue__vehicle 1.00000	application__procedure 1.00000	music__market 1.00000	phase__space 1.00000
rescue__equipment 0.49602	application__process 0.67551	music__world 0.58263	space__time 0.44398
rescue__squad 0.45921	application__form 0.59632	music__chart 0.54964	phase__transition 0.42528
rescue__personnel 0.45826	job__application 0.54862	pop__music 0.52977	ground__state 0.41160
rescue__crew 0.45364	grant__application 0.50493	music__industry 0.51918	field__theory 0.41095
rescue__vessel 0.42381	visa__application 0.46575	music__sale 0.51078	quantum 0.40099
...	...	...	...
rescue__vehicle__c 0.29996	application__procedure__c 0.30313	music__market__c 0.34352	phase__space__c 0.26121
shoo__fly:1000	**tobacco__market:1000**	**paper__wasp:1000**	**sports__vehicle:1000**
shoo__fly 1.00000	tobacco__market 1.00000	paper__wasp 1.00000	sports__vehicle 1.00000
pepper__steak 0.52205	wine__market 0.59988	garden__spider 0.58116	dangling__modifier 0.65794
potato__bug 0.50899	cigarette__market 0.59143	mud__dauber 0.56821	cutting__implement 0.63771
tub__thumper 0.49676	tea__market 0.58791	cicada__killer 0.56456	four-wheel-drive__vehicle 0.62591
rum__baba 0.49114	arms__market 0.57372	mining__bee 0.52083	toy__vehicle 0.62213
dice__box 0.48834	sugar__market 0.55928	house__centipede 0.49953	import__system 0.62018
...	...	...	...
shoo__fly__c -0.02019	tobacco__market__c 0.14430	paper__wasp__c 0.19814	sports__vehicle__c -0.13707

Table C.19 Neighbours of example compounds from the `en-comcom` `test` set, using the `w_addition` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (100) and the cut-off rank (1000).

investment__proposal:1	**cartoon__strip:1**	**space__radiation:1**	**state__organization:1**
investment__proposal 1.00000	cartoon__strip 1.00000	space__radiation 1.00000	state__organization 1.00000
investment__proposal__c 0.58941	cartoon__strip__c 0.45421	space__radiation__c 0.49697	state__organization__c 0.63551
investment__project 0.55414	newspaper__column 0.44210	radio__noise 0.48180	state__entity 0.54423
acquisition__plan 0.52641	cartoon__character 0.42425	radiation__biology 0.43957	government__entity 0.53913
export__contract 0.49631	cartoon 0.40936	neutron__radiation 0.41427	state__body 0.50611
aircraft__purchase 0.49383	cartoon__show 0.37220	radiation__exposure 0.41260	volunteer__organization 0.50608
investment__target 0.48651	cartoon__series 0.36934	ionizing__radiation 0.39443	church__organization 0.50271
word__order:14	**organ__donation:14**	**productivity__growth:14**	**safety__zone:14**
word__order 1.00000	organ__donation 1.00000	productivity__growth 1.00000	safety__zone 1.00000
sentence__structure 0.57507	organ__donor 0.62903	labor__productivity 0.76833	school__safety 0.49528
verb 0.56939	transplant 0.48802	labour__productivity 0.76627	exclusion__zone 0.46739
noun__phrase 0.51117	organ__transplant 0.46082	output__growth 0.65773	security__zone 0.44723
syntax 0.50443	brain__death 0.45716	wage__growth 0.65529	perimeter__fence 0.39763
punctuation 0.47007	blood__donation 0.43004	growth__rate 0.64150	protection__zone 0.38599
...	...	...	...
word__order__c 0.38327	organ__donation__c 0.35721	productivity__growth__c 0.54700	safety__zone__c 0.32153
nose__drops:1000	**clock__watcher:1000**	**competition__organiser:1000**	**crown__fire:1000**
nose__drops 1.00000	clock__watcher 1.00000	competition__organiser 1.00000	crown__fire 1.00000
dusting__powder 0.57627	bug__hunter 0.62380	project__organiser 0.59008	surface__fire 0.61235
flea__powder 0.54977	factory__director 0.60452	rally__organizer 0.49819	grass__fire 0.44568
chemical__spray 0.53368	caffeine__addict 0.60010	environment__commissioner 0.49577	moorland__fire 0.44329
calcium__nitrate 0.53318	taxi__dancer 0.59381	coalition__official 0.47552	rollover__crash 0.41715
ammonium__carbonate 0.52881	guinea__gold 0.58602	delegation__head 0.47152	union__miner 0.39997
...	...	...	...
nose__drops__c 0.26893	clock__watcher__c 0.17868	competition__organiser__c 0.22115	crown__fire__c 0.19200

Table C.20 Neighbours of example compounds from the `en-comcom` `test` set, using the `lexfunc` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (14) and the cut-off rank (1000).

monitor__program:1	**party__chief:1**	**rogue__elephant:1**	**service__division:1**
monitor__program 1.00000	party__chief 1.00000	rogue__elephant 1.00000	service__division 1.00000
monitor__program__c 0.48685	party__chief__c 0.57671	rogue__elephant__c 0.66267	service__division__c 0.70384
import__system 0.44099	party__boss 0.45157	horse__doctor 0.66240	service__department 0.56786
program__quality 0.43015	party__official 0.43346	plow__horse 0.65587	service__facility 0.56749
traffic__mitigation 0.42661	party__leader 0.42197	chicken__yard 0.65222	service__operation 0.51915
personality__quiz 0.42372	politburo__member 0.41440	bank__guard 0.63444	service__program 0.51402
buffer__store 0.42320	party__branch 0.41280	government__man 0.62989	service__office 0.51309
mail__system:2	**vegetable__garden:2**	**shooting__lodge:2**	**guide__rope:2**
mail__system 1.00000	vegetable__garden 1.00000	shooting__lodge 1.00000	guide__rope 1.00000
email__system 0.64818	vegetable__patch 0.76619	hunting__lodge 0.61701	center__pole 0.68092
mail__system__c 0.60377	vegetable__garden__c 0.74822	shooting__lodge__c 0.51058	guide__rope__c 0.64363
voice__mail 0.55972	kitchen__garden 0.73704	shooting__box 0.50183	grappling__iron 0.64078
mail__service 0.54594	herb__garden 0.71074	cow__barn 0.47860	water__wagon 0.61621
mail__box 0.45023	flower__bed 0.67242	army__hut 0.46117	snatch__block 0.60451
snail__mail 0.42475	fruit__tree 0.64774	manor__house 0.45763	starting__stall 0.60083
custom__officer:1000	**sounding__board:1000**	**service__break:1000**	**home__plate:1000**
custom__officer 1.00000	sounding__board 1.00000	service__break 1.00000	home__plate 1.00000
custom__official 0.76468	rallying__point 0.42654	rating__period 0.60125	plate__umpire 0.68733
stretcher__party 0.71902	launching__pad 0.41923	chow__line 0.56250	center__field 0.49705
slip__coach 0.69349	testing__ground 0.41371	operation__officer 0.55321	umpire 0.49245
riot__officer 0.68981	reference__point 0.38087	pace__lap 0.55097	dugout 0.46930
poll__taker 0.68704	teaching__tool 0.37899	moving__staircase 0.54872	strike__zone 0.45223
...	...	...	...
custom__officer__c 0.43134	sounding__board__c 0.15510	service__break__c 0.35390	home__plate__c 0.09510

Table C.21 Neighbours of example compounds from the `en-comcom test` set, using the `fulllex` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (2) and the cut-off rank (1000).

health__budget:1	**grain__company:1**	**media__representative:1**	**embroidery__stitch:1**
health__budget 1.00000	grain__company 1.00000	media__representative 1.00000	embroidery__stitch 1.00000
health__budget__c 0.59752	grain__company__c 0.63130	media__representative__c 0.55949	embroidery__stitch__c 0.59637
health__spending 0.57206	grain__trader 0.51409	media__people 0.52056	chain__stitch 0.50486
school__budget 0.44190	seed__company 0.50425	news__media 0.46791	knitting__stitch 0.50301
research__budget 0.43714	grain__producer 0.49439	media__organization 0.46213	tent__stitch 0.49815
pension__account 0.43607	mining__house 0.46649	government__representative 0.44808	machine__stitch 0.49003
development__budget 0.42394	oil__refiner 0.45989	event__organizer 0.42218	running__stitch 0.48148
cement__block:3	**fishing__area:3**	**week__program:3**	**source__materials:3**
cement__block 1.00000	fishing__area 1.00000	week__program 1.00000	source__materials 1.00000
cinder__block 0.78761	trout__fishing 0.65462	month__program 0.67070	research__material 0.62474
concrete__block 0.67699	fishing__ground 0.62830	week__course 0.64740	source__material 0.60566
cement__block__c 0.62378	fishing__area__c 0.53453	week__program__c 0.64090	source__materials__c 0.51021
adobe__brick 0.60881	salmon__fishing 0.49095	hour__course 0.59869	reference__material 0.45619
breeze__block 0.59828	fishing__season 0.48074	week__session 0.59868	teaching__material 0.44815
mud__brick 0.55095	picnic__ground 0.45943	month__course 0.57624	government__document 0.42262
cluster__bomblet:1000	**benefit__album:1000**	**gear__box:1000**	**shoulder__flash:1000**
cluster__bomblet 1.00000	benefit__album 1.00000	gear__box 1.00000	shoulder__flash 1.00000
gasoline__inventory 0.59269	hurricane__relief 0.54283	clutch 0.68303	garrison__cap 0.64212
solidarity__member 0.57432	quake__victim 0.48921	engine 0.63005	sailor__cap 0.59788
weapon__sale 0.56407	flag__issue 0.48688	gear 0.55704	pea__jacket 0.59712
cashew__tree 0.56154	campaign__advertisement 0.48320	torque__converter 0.54977	ski__cap 0.59492
dollar__level 0.55709	tub__thumper 0.48038	axle 0.52728	watch__cap 0.58635
...	...	...	...
cluster__bomblet__c 0.33417	benefit__album__c 0.28759	gear__box__c 0.13559	shoulder__flash__c 0.30276

Table C.22 Neighbours of example compounds from the `en-comcom test` set, using the `fulladd` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

dog__collar:1	rap__artist:1	property__sale:1	hill__town:1
dog__collar 1.00000	rap__artist 1.00000	property__sale 1.00000	hill__town 1.00000
dog__collar__c 0.54199	rap__artist__c 0.66841	property__sale__c 0.58137	hill__town__c 0.49966
leash 0.48995	rap__star 0.64815	property__transaction 0.54844	mountain__village 0.44740
collar 0.44047	recording__artist 0.56339	property__management 0.52949	market__town 0.40680
dog__tag 0.41794	hip__hop 0.54667	home__sale 0.50604	beach__town 0.37561
shoulder__strap 0.38407	rap 0.54657	estate__agent 0.47905	university__town 0.37451
sports__jacket 0.38068	rap__music 0.52175	property__market 0.46321	market__square 0.37379
brand__manager:3	**time__period:3**	**lake__poets:3**	**cheese__dip:3**
brand__manager 1.00000	time__period 1.00000	lake__poets 1.00000	cheese__dip 1.00000
marketing__manager 0.70351	time__frame 0.75858	folk__poet 0.65036	cheese__sauce 0.55767
product__manager 0.65581	period 0.68544	sand__hopper 0.63909	taco__sauce 0.54454
brand__manager__c 0.64731	time__period__c 0.59635	lake__poets__c 0.59926	cheese__dip__c 0.51926
marketing__director 0.64690	time__interval 0.59194	forest__god 0.59435	bechamel__sauce 0.51546
product__marketing 0.62479	year__period 0.58240	sea__swallow 0.59369	bean__dip 0.50829
marketing__executive 0.59426	time__span 0.57384	horse__latitude 0.59048	wax__bean 0.50158
rose__hip:1000	**developing__cost:1000**	**snow__job:1000**	**walkie__talkie:1000**
rose__hip 1.00000	developing__cost 1.00000	snow__job 1.00000	walkie__talkie 1.00000
pomegranate 0.54706	morale__building 0.52340	squeeze__play 0.47097	police__radio 0.50134
dog__rose 0.51804	profit__projection 0.51079	snipe__hunt 0.46792	radio__set 0.46869
borage 0.50045	data__formatting 0.49961	tub__thumper 0.44253	cell__phone 0.46803
evening__primrose 0.48132	shareholder__litigation 0.49381	conversation__stopper 0.43753	crystal__set 0.45369
rose__oil 0.47833	inventory__adjustment 0.49292	ham__actor 0.42959	communication__device 0.45242
...	...	...	...
rose__hip__c 0.18460	developing__cost__c 0.31569	snow__job__c 0.27037	walkie__talkie__c 0.16783

Table C.23 Neighbours of example compounds from the `en-comcom` `test` set, using the `matrix` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

larch__tree:1	loan__package:1	week__tour:1	maintenance__crew:1
larch__tree 1.00000	loan__package 1.00000	week__tour 1.00000	maintenance__crew 1.00000
larch__tree__c 0.57835	loan__package__c 0.63349	week__tour__c 0.76339	maintenance__crew__c 0.73689
umbrella__pine 0.57058	financing__package 0.60122	week__trip 0.69807	construction__crew 0.58136
hemlock__tree 0.55664	bailout__loan 0.56430	month__tour 0.66554	maintenance__worker 0.57666
cashew__tree 0.53540	dollar__loan 0.56111	week__visit 0.60117	ground__crew 0.56853
flowering__cherry 0.50876	aid__package 0.49299	week__vacation 0.55865	maintenance__personnel 0.53650
balloon__bomb 0.49897	rescue__package 0.48661	whirlwind__tour 0.52659	maintenance__staff 0.52905
hansom__cab:3	**shirt__pocket:3**	**wage__gap:3**	**ganglion__cell:3**
hansom__cab 1.00000	shirt__pocket 1.00000	wage__gap 1.00000	ganglion__cell 1.00000
hackney__coach 0.55839	jacket__pocket 0.73998	pay__gap 0.77808	nerve__fiber 0.55720
hansom 0.55333	coat__pocket 0.66339	gender__gap 0.61318	nerve__fibre 0.52902
hansom__cab__c 0.52081	shirt__pocket__c 0.65127	wage__gap__c 0.58148	ganglion__cell__c 0.50069
post__chaise 0.46151	back__pocket 0.60407	income__gap 0.55840	nerve__cell 0.47682
cab__driver 0.45983	breast__pocket 0.58599	achievement__gap 0.50864	cone__cell 0.46574
police__bus 0.44039	vest__pocket 0.49053	disparity 0.49488	neuron 0.43365
incumbency__advantage:1000	**company__reporting:1000**	**powder__compact:1000**	**web__site:1000**
incumbency__advantage 1.00000	company__reporting 1.00000	powder__compact 1.00000	web__site 1.00000
animal__leg 0.72256	reporting__requirement 0.50687	bathing__cap 0.66197	web__page 0.76294
voter__interest 0.71831	reporting__rule 0.50151	sewing__basket 0.65122	site 0.71682
inventory__problem 0.69622	spending__target 0.50093	traveling__bag 0.64386	page 0.68968
abortion__supporter 0.69511	trading__unit 0.48637	paper__knife 0.63481	web 0.66939
animal__worship 0.69415	banana__export 0.48386	brandy__snifter 0.63411	home__page 0.66098
...	...	...	...
incumbency__advantage__c 0.24467	company__reporting__c 0.31848	powder__compact__c 0.18807	web__site__c 0.25015

Table C.24 Neighbours of example compounds from the `en-comcom` `test` set, using the `addmask` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

disk_error:1	**currency_value:1**	**employee_manual:1**	**metal_armor:1**
disk_error 1.00000	currency_value 1.00000	employee_manual 1.00000	metal_armor 1.00000
disk_error_c 0.63742	currency_value_c 0.68259	employee_manual_c 0.69876	metal_armor_c 0.68537
hardware_error 0.62881	currency_rate 0.61195	company_attorney 0.59591	chain_armour 0.67611
program_error 0.56904	exchange_rate 0.54930	apple_fritter 0.58195	face_guard 0.65524
system_error 0.54074	currency_depreciation 0.53445	redundancy_policy 0.57816	bicycle_clip 0.64259
head_crash 0.52071	equity_price 0.49077	intelligence_cell 0.57682	bathing_trunks 0.63836
computer_error 0.51923	currency_policy 0.49042	x-ray_scan 0.57464	bathing_cap 0.61547
church_leader:3	**majority_party:3**	**picnic_table:3**	**sweetheart_deal:3**
church_leader 1.00000	majority_party 1.00000	picnic_table 1.00000	sweetheart_deal 1.00000
christian_leader 0.74565	minority_party 0.73798	fire_pit 0.72293	backroom_deal 0.65540
church_member 0.69506	party_caucus 0.60063	picnic_area 0.69909	barter_deal 0.54495
church_leader_c 0.64688	majority_party_c 0.49051	picnic_table_c 0.60321	sweetheart_deal_c 0.54421
church_official 0.61824	opposition_party 0.47730	picnic 0.58632	oil_deal 0.51831
clergy 0.61303	party_leader 0.46938	barbecue_pit 0.55666	business_deal 0.51084
community_leader 0.56897	majority_coalition 0.46083	basketball_court 0.54764	marketing_deal 0.48358
screen_price:1000	**police_lineup:1000**	**fossil_oil:1000**	**deed_poll:1000**
screen_price 1.00000	police_lineup 1.00000	fossil_oil 1.00000	deed_poll 1.00000
coach_fare 0.56077	bank_guard 0.57622	import_system 0.68093	marriage_certificate 0.53045
one-off_charge 0.53977	mopping-up_operation 0.57314	banana_export 0.65954	family_name 0.46612
manufacturing_level 0.51958	police_sweep 0.57112	price_distribution 0.64716	company_name 0.44564
seat_price 0.51609	surprise_inspection 0.56619	self_loader 0.64239	birth_certificate 0.40850
carriage_charge 0.51505	trash_barrel 0.56103	trading_transaction 0.64030	given_name 0.39348
...	...	...	...
screen_price_c 0.31463	police_lineup_c 0.32903	fossil_oil_c 0.37813	deed_poll_c 0.10163

Table C.25 Neighbours of example compounds from the `en-comcom` `test` set, using the `wmask` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (3) and the cut-off rank (1000).

police_academy:1	**greyhound_stadium:1**	**futures_exchange:1**	**rugby_player:1**
police_academy 1.00000	greyhound_stadium 1.00000	futures_exchange 1.00000	rugby_player 1.00000
police_academy_c 0.53538	greyhound_stadium_c 0.57663	futures_exchange_c 0.52665	rugby_player_c 0.76360
police_training 0.46586	stadium_site 0.50737	commodities_exchange 0.51712	soccer_player 0.62837
city_police 0.44562	exhibition_park 0.44719	exchange_floor 0.46548	football_player 0.61485
state_police 0.42296	police_substation 0.42528	futures_market 0.43570	hockey_player 0.56512
training_academy 0.40455	wildlife_hospital 0.42236	commodities_market 0.43415	rugby_league 0.56443
police_headquarters 0.38855	bus_depot 0.40868	stock_exchange 0.42780	tennis_player 0.56172
tax_adviser:2	**transmission_line:2**	**inventory_management:2**	**embassy_bombing:2**
tax_adviser 1.00000	transmission_line 1.00000	inventory_management 1.00000	embassy_bombing 1.00000
tax_specialist 0.65725	power_line 0.61887	inventory_control 0.71857	embassy_attack 0.63572
tax_adviser_c 0.63871	transmission_line_c 0.60699	inventory_management_c 0.66367	embassy_bombing_c 0.49294
tax_lawyer 0.57734	substation 0.60328	production_planning 0.53626	bombing 0.40472
tax_professional 0.57411	transmission_system 0.56338	supply_chain 0.52206	navy_destroyer 0.39885
accountant 0.57257	transmission_network 0.54151	management_software 0.50386	terror_attack 0.37650
tax_attorney 0.56635	gas_pipeline 0.53170	management_system 0.49499	truck_bombing 0.36914
cow_chip:1000	**field_game:1000**	**whistle_blower:1000**	**hall_porter:1000**
cow_chip 1.00000	field_game 1.00000	whistle_blower 1.00000	hall_porter 1.00000
gasoline_bomb 0.57686	marching_music 0.44044	disclosure 0.43698	computer_manager 0.67320
snap_roll 0.57404	go_cart 0.40094	prosecution 0.42416	bank_guard 0.66160
test_copy 0.57085	concert_piano 0.39885	lawsuit 0.42026	forest_god 0.65477
coin_bank 0.56334	sport_field 0.39742	harassment 0.40594	manufacturing_level 0.65423
dice_box 0.56003	table_dance 0.39360	espionage 0.39463	property_man 0.64983
...	...	...	...
cow_chip_c 0.32028	field_game_c 0.12954	whistle_blower_c -0.05645	hall_porter_c 0.41934

Table C.26 Neighbours of example compounds from the `en-comcom` `test` set, using the `multimatrix` representation of the compound (300 dim). Four examples each of compounds assigned the best rank (1), the middle (2) and the cut-off rank (1000).