UNIVERSITEIT VAN AMSTERDAM

MASTER THESIS

Improving the Compositionality of Word Embeddings

Author:
Mathijs J. SCHEEPERS

Supervisors:
dr. Evangelos KANOULAS
dr. Efstratios GAVVES

Assessor:
prof.dr. Maarten DE RIJKE

A thesis submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of ***Master of Science*** *in* ***Artificial Intelligence****.*

November 29, 2017

Abstract

Mathijs J. SCHEEPERS

Improving the Compositionality of Word Embeddings

We present an in-depth analysis of four popular word embeddings (*Word2Vec*, *GloVe*, *fastText* and *Paragram*) [84, 99, 17, 135] in terms of their semantic compositionality. In addition, we propose a method to tune these embeddings towards better compositionality. We find that training the existing embeddings to compose lexicographic definitions improves their performance in this task significantly, while also getting similar or better performance in both word similarity and sentence embedding evaluations.

Our method tunes word embeddings using a simple neural network architecture with definitions and lemmas from *WordNet* [86]. Since dictionary definitions are semantically similar to their associated lemmas, they are the ideal candidate for our tuning method, as well as evaluating for compositionality. Our architecture allows for the embeddings to be composed using simple arithmetic operations, which makes these embeddings specifically suitable for production applications such as web search and data mining. We also explore more elaborate and involved compositional models, such as recurrent composition and convolutional composition. We call our architecture: *CompVec*.

In our analysis, we evaluate original embeddings, as well as tuned embeddings, using existing word similarity and sentence embedding evaluation methods [30, 38]. Aside from these evaluation methods used in related work, we also evaluate embeddings using a ranking method which tests composed vectors using the lexicographic definitions already mentioned. In contrast to other evaluation methods, ours is not invariant to the magnitude of the embedding vector—which we show is important for composition. We consider this new evaluation method (*CompVecEval*) to be a key contribution.

Finally, we expand our research by training on a significantly larger dataset we constructed ourselves. This dataset contains pairs of titles and articles introductions from Wikipedia. Since the introduction of an encyclopedic article is definitional, we found them to be a logical progression from dictionaries. The creation of this dataset, by extracting it through the Wikipedia API, is another important contribution.

Acknowledgements

I would first like to thank my thesis supervisors Efstatios Gavves and Evangelos Kanoulas. They sent me on this compositionality learning adventure which is something I could never have imagined on my own. The fruitful discussions with them where always interesting and insightful. They encouraged me to work on challenging but reachable goals. I am grateful that they challenged me to write and submit papers. When two of them got rejected they quickly changed my initial disappointment into a motivation to improve and do better.

I would also like to acknowledge Maarten de Rijke, as the accessors of this thesis. I am grateful to him for taking valuable time out of his very busy schedule to read and comment on this thesis.

I would like to thank everybody at Label305, and especially my business partners Olav Peuscher, Xander Peuscher and Joris Blaak for standing behind me and encouraging me to make the most out of my studies, even when the business also required a lot of work to be finished. They always were kind enough to take on some of my responsibilities while I had to work on my master and on this thesis.

I would also like to thank my fellow master students with whom I was involved in various study projects, and who helped me with this thesis through their valuable comments and advice. I would like to name Bas Veeling, Joost van Doorn, Jörg Sander, Maartje ter Hoeve, Maurits Bleeker and David Zomerdijk in particular.

Finally, I must express my profound gratitude to my parents Martin and Christine, my brother Luuk, my flatmate Nico and to my girlfriend Bertina for providing me with unfailing support and continuous encouragement throughout my education and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Contents

Chapter 1

Introduction

> *Theodore*
> So when did you gave a name to yourself?
>
> *Samantha*
> Well... right when you asked me if I had a name, I thought: "Yeah he is right, I do need a name!". But I wanted to pick a good one. So I read a book called: 'How to name your Baby', and out of a hundred and eighty thousand names Samantha was the one I liked the best.
>
> *Theodore*
> Wait—so you read a whole book in *the* second that I asked you what your name was?
>
> *Samantha*
> In two one-hundredths of a second, actually...
>
> from the screenplay Her
> by Spike Jonze (2013)

Samantha, or OS1, is a fictional artificial intelligence from the 2013 film Her. In the film she shows an effortless ability to speak and understand natural human language, even though she is actually a computer. Through language she acquires immense knowledge and, eventually, gets a profound emotional connection to the protagonist Theodor. She does all of this without having a physical appearance and while speaking to Theodor through a tiny ear-piece.

Instrumental to making an AI such as Samantha, will be giving computers the ability to understand or give meaning to natural language. Language is the messy and loosely structured means people use to transfer and store information. Computers have a very different approach to storing and transferring information. Ones and zeros are structured neatly according to a well defined protocols. So when Samantha wants to pick a name and reads the book 'How to name your baby' she, as a computer, would need to go over all the human language in that book. If our goal is to give computers the ability to understand the messy way people communicate,

there has to be a method for computers to store the meaning, which human language encodes, in their native tongue—binary.

Researchers in the field of artificial intelligence have uncovered interesting algorithms [85] that start to give computers the ability to encode the meaning of single words and sentences. This thesis explores a method to find better *encodings of meaning* a computer can work with. We specifically want to combine encodings of *word meanings* in such a way that a good encoding of their joint meaning is created. The act of combining multiple representations of meaning into a new representation of meaning is called *semantic composition*.

1.1 Compositionality and Lexical Semantics

The principle of *compositionality* as defined by Frege [41] states that the meaning of an expression comprises the meaning of its constituents as well as the rules to combine them. Frege introduced the principle in 1892 so he could explain the way humans understand and give meaning to language. Today, researchers use the same principle to model the way computers represent meaning [119].

In order to start combining representations of meaning, we first need to have some principle representations to combine. While some have theorized atomic units called *semantic primes* [133, 63], we turn to something a bit more pragmatic. Currently the artificial intelligence research community mostly uses large vocabularies of words[1], and each AI model has representations for each individual word in these vocabularies.

If we want computers to have representations of the meaning of single words, i.e. *lexical semantics*, we need to ask ourselves: "What exactly is word meaning in terms of human language?". The exact interpretation and implications of this word specific meaning has been the subject of debate amoung philosophers and linguists. They have yet to come up with a definitive answer [45]. This makes the task exceptionally challenging since we are trying to model something which we do not fully understand.

What is clear however, is that lexicography, i.e. the science of writing dictionaries, is important to illustrate the relationship between words and their meaning [45]. People have been using dictionaries to give words meaning as early as 2300 BCE [130]. In this work, we will therefore turn to dictionaries for some of our approaches. They are the pragmatic human solution to the problem of word meaning. The question whether computers could learn from these as well, naturally arises.

Words in dictionaries, or lemmas, are described in one or more short but exact definitions. These dictionary definitions are called lexicographic definitions. If we have such a definition, according to the principle of compositionality, it should be composable to the word it describes. For example we should be able to compose: *"A small*

[1] There is also a lot of research into the applicability of phrase, subword and character representations.

domesticated carnivorous mammal with soft fur, a short snout, and retractable claws." into the lexical semantic representation of *'cat'*.

1.2 Word Embeddings

How do we present word meaning in terms of ones and zeros? Since any number can be represented in binary[2], we really need to look at numbers in general. For some time, researchers represented the semantics of phrases, sentences and documents using an m-dimensional vector of integers ($\mathbb{Z}_m$), also called *bag of words* [89, 32]. In these vectors, m was the size of the vocabulary, and an entry was non-zero if the word corresponding to that entry occurred in the text.

After the rise in popularity of artificial neural networks in the early twenty-tens, a new approach to represent lexical semantics became the new default for novel models. Neural networks can be used to learn n-dimensional real valued vector representations of words in a latent space ($\mathbb{R}_n$). These real valued vectors are called *word embeddings*, and they have become an essential aspect of models for various application domains, e.g. document retrieval [43], automatic summarization [139], machine translation [123], sentiment analysis [35] and question answering [122].

Ever since the paper [85] which made these real valued embeddings popular, there has been interest in their compositional properties [84]. These properties are especially important in the context of deep neural networks where multiple representations often need to be composed into a single deeper representation.

Learning word embeddings is done by optimizing the prediction of either the context of the word, or the word from its context [84, 99, 17]. This approach is based on an intuition neatly put into words by linguist Firth [40]: *"You shall know a word by the company it keeps"*. One major advantage of directly learning from context is that all text can directly be used for unsupervised training of the embeddings. Having large amounts of training data is a necessity for most deep learning models. Because *all* text can be used, the amount of potential training data for word embeddings is enormous. For example, the popular GloVe embeddings [99] are trained on the Common Crawl dataset [3] which contains $\approx$ 200 TB of raw text crawled from all over the entire world wide web.

Training based on context results in representations which captures both syntax as well as semantics, since syntax is inherently present in context. While syntactic information is important for composing representations, it is not necessarily useful for applying semantics in a model. We will explore an approach to tune context-learned word embeddings towards embeddings which better represent semantics.

[2] Real, natural and imaginary numbers are represented in computers as floating point numbers [49], which are not always exact but more often really close estimations of these numbers.

[3] The Common Crawl dataset can be found at: `http://commoncrawl.org`.

In fact, improving the lexical semantics of word embeddings as well as semantic compositionality of word embeddings is the focus of this thesis. Hence our title: *Improving the Compositionality of Word Embeddings*.

1.3 Contribution

The thesis starts off with background and related work in Chapter 2. It will touch upon research from linguistics, artificial intelligence and deep learning specifically.

In order to improve the semantic compositionality of word embeddings we first introduce an evaluation method, a benchmark if you will, in Chapter 3. This new evaluation method is called *CompVecEval*. It is meant to measure whether a set of embeddings is able to compose lexicographic definitions into the words these definitions describe. We compose the embeddings using simple algebraic composition functions such as addition, multiplication or averaging. The test differs from other phrase and sentence embedding tests because it uses a balltree ranking [94] algorithm—which is an exact nearest neighbor algorithm. It therefore considers the relation to all other lexical representations and is not invariant to the embeddings magnitude, i.e. the Euclidean norm. With the new evaluation method we analyze four popular and widely used word embeddings: *Word2Vec* [84], *GloVe* [99], *fastText* [17] and *Paragram* [135]. These embeddings are often used directly as static features for specific models, or used for *transfer learning* to kick start a model's training operation and consequently improve its final performance. In the chapter (3) we will present new insights into the compositionality of these embeddings under simple algebraic operations. Our results show that summing embeddings can be just as effective or even better than averaging them. Even though averaging happens a lot in popular algorithms and has a theoretical framework backing it up[7].

We use a subset of lexicographic definitions and lemmas from *WordNet* [86] as data for our evaluation method. Because words in a dictionary can have multiple definitions, our test will specifically evaluate for the various senses of ambiguous words. If we take the example of *"cat"* again, it does not only refer to the furry animal but also to: *"A method of examining body organs by scanning them with X-rays and using a computer to construct a series of cross-sectional scans along a single axis"*. Our tests makes sure the embeddings of all lexicographic descriptions of *"cat"* are able to compose into its lexical representation. Not just the most frequent, as that could be the case when learning embeddings from large corpora.

Chapter 4 introduces a model that is able to update and tune word embeddings using a different subset of the data from *WordNet*. The word embeddings are tuned towards better compositionality under a specific algebraic composition function. During tuning we test the embeddings not only according to the new evaluation method *CompVecEval*, but also to fifteen different sentence embedding evaluations and thirteen different word similarity evaluations. All these methods combined provide a clear picture of the overall quality of the embeddings, including their compositionality.

The results show that we are able to improve, sometimes by a large margin, on the four popular embeddings. We call the method of tuning word embeddings towards better compositionality: *CompVec*.

One of the disadvantages of commutative algebraic composition functions is that they are unable to model more complex aspects of compositionality. For example, these functions are not able to take word order into account. Chapter 5 turns to learned composition functions. The chapter starts with a simple projection layer, but quickly turns to recurrent and convolutional composition functions. Using these learned composition functions we are able to improve on the results from Chapter 4, albeit only slightly.

In Chapter 6 we depart from using *WordNet* based training data, simply because the dictionary is limited in size. We follow an intuition that is true for many machine learning tasks: "The more data we have, the better the models will preform". Instead of using *WordNet* we turn to data from the online encyclopedia *Wikipedia*. Since we want our data to be definitional, we created a custom dataset of title and summary pairs for every English entry from the website. The lemma and lexicographic definition, that are used in other chapters, are now substituted for the title and summary from the encyclopedia. The results from tuning embeddings using this dataset are mixed.

Finally Chapter 7 concludes with a retrospective on the entire thesis. The chapter discusses the results, various short comings in our work and it gives recommendations for future research.

Replication and Open Source

All the code written to conduct the various experiments in this thesis is open source and available online at `https://github.com/tscheepers/`. We included a separate repository for *CompVec* and *CompVecEval* at `https://thijs.ai/CompVec/`. With this code repository everyone can use our new tuned embeddings, but also evaluate their own using the introduced evaluation method. In addition, we open sourced our Wikipedia dataset at `https://thijs.ai/Wikipedia-Summary-Dataset/` since no dataset that was solely based on article introductions was ever published.

Chapter 2

Background

Before we present our new work on the compositionality of word embeddings, we first look into the work that has already been done on this and related subjects. Starting with early work on word meaning, compositionality from the perspective of philosophers and linguists. Then continuing, with work on distributional semantics and early word embeddings. Finally, this background chapter will finish with compositional models in deep learning.

2.1 Philosophy of Language

This thesis starts with the assumption that certain semantics can be captured in individual words. Because of this, questions arise such as: *"What is word meaning?"* and *"What is a word?"*. These questions are actually hard to answer. Therefore, they are a topic of debate among philosophers and linguists [45]. The notion of *a word* can be used as a means of explaining concepts, e.g. morphology. Or one could think of the notion of a word metaphysically, by asking questions such as: *"What are the conditions that two different utterances are actually the same word?"* [19].

In this thesis, we choose to define words more pragmatically. We see them as tokens in a body of written text, typically separated by spaces. More specially, they can be split in a body of text using a *tokenizer* such as the one found in the natural language toolkit (NLTK) [15].

2.1.1 Lexicography

While it is difficult to answer questions around word meaning in general, this is not true for the meaning of a specific words. There is already a quite obvious and old solution to this problem—a dictionary. Lexicography, or the practice of writing dictionaries, plays an important role in systematizing the word descriptions, on which a lot of linguistic work relies. Linguists use dictionaries to shed light on the relationships between words and their meaning [12, 64, 56]. Putnam [101] even goes so far as stating that the phenomenon of writing dictionaries gave rise to the entire idea of *semantic theory*.

An important aspect of lexicography is the process of *lemmatization*. It comes down to the grouping together of inflected forms of a word so they can be analyzed as a single item. For example, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', is the *lemma* that ends up in a dictionary. The grouping of semantically similar elements provided a paradigm for much subsequent research on lexical semantics, such as *decompositional theories* of word meaning. In our research, we will use these lemmas as targets for our evaluation method on compositionality as well as a target for tuning embeddings towards better compositionality.

2.1.2 Contextualism

Word embeddings are often learned by looking at directly adjacent context words, therefore it is interesting to discuss the philosophical work on word meaning, specifically in regard to *contextualism*.

Grice's theory of conversation and implicatures [54] is a standard work among linguists and philosophers. He marginalizes the importance of context in regard to semantics. Basically, he believes that context is only import for the semantics of indexical words (such as 'I', 'now', 'here', etc.) and of context-sensitive structures, e.g. tense [45].

Travis [127] and Searle [115, 114] argue that the semantic relevance of context is much more important, if not essential. If one looks at the sentence: *"I'm in the car."*, it can be interpreted as *"Hurry up!"*. On the other hand, if two people are leaving for a meeting, or it could mean *"I'm almost there."* if it is stated to someone who has an appointment with him or her.

It has to be noted that context can be defined as linguistic context in terms of close expression or as the real-world context a statement made in—i.e. the environment in which, and with what intentions a statement is made. If we take context as meaning the former, and consider the remarkable progress in *Natural Language Processing* (NLP) since the advent of word embeddings learned through context, this progress could provide some empirical arguments for contextualism. In other words, results from NLP research can be used as arguments for the ideas of Travis and Searle.

2.1.3 The Principle of Compositionality

Compositionality in linguistics was first defined back in 1892 by Frege [41]. Later, it was placed neatly into the present context by Janssen [66]. In 2010 the mathematical foundations of compositional semantics were described by Coecke, Sadrzadeh, and Clark [28].

The concept of compositionality is defined as: *the meaning of a complex expression is fully determined by its structure, the meanings of its constituents as well as the rules to combine them*. The principle is not necessarily accepted or proven. Rather, it provides a framework on how one could think about semantics and language.

Compositionality is a framework to think about the semantics of entirely new complex expressions in relation to complex expressions we already understand. Is it not great, that we can almost immediately understand an infinity large collection of complex expressions the first time we hear or read them? Compositionality is the best explanation we have for these phenomena [125].

Critics of compositionality point to the fact that it does not explain cases where the meaning of a larger expression depends on the intention of the speaker. Neither can it explain meaning influenced by the setting in which an expression is made, without the expression itself containing information on this setting. A great example of this is: sarcasm. Sarcasm can not be inferred purely on the basis of the words and their composition, even though a statement made sarcastically can mean something entirely different from the same statement made without sarcasm.

Nevertheless, the principle provides the artificial intelligence research community a great tool for thinking about semantics and their representations in NLP models.

2.1.4 Semantic Primes

One last philosophic principle to discuss is a *decompositional approach to lexical semantics* [133, 63, 48]. By following this idea we see that composition is not only important to understand the meaning of a phrase or sentence, but also the meaning of single words themselves.

Both Wierzbicka [133] and Jackendoff [63] theorized that word meaning could emerge by compositing multiple so called *semantic primes*. They go so far as to define some of these primes and describing them in *Natural Semantic Metalanguage*, which they themselves define. While in general, artificial intelligence researches will prefer such an exercise to be driven by data, this way of thinking could lead to an interesting direction for future research—as will be discussed in Chapter [**chap:conclusion**].

2.2 Compositional models before Deep Learning

Now we turn from philosophy to actual linguistic models. Vector space models have been popular since the 1990s, specifically in *distributional semantics*. In this time period, the first models for estimating continuous representations from a text corpus were developed, including *Latent Semantic Analysis* (LSA) [32] and *Latent Dirichlet Allocation* (LDA) [16].

Mitchell and Lapata [89, 88] were the first to semantically compose meaning using a simple element-wise algebraic operation on word vector representations. This work does not use the real valued word embeddings which we use in this thesis, and are popular today. However, the work does compare various operations on word embeddings and how it affects their composition, similar to this work. They come to different conclusions. In their results, multiplicative models are superior to the additive models, which is not the case in our analysis.

Erk and Padó [37] stated that complex semantic relations in phrases are difficult to model by using simple element-wise algebraic composition. Specifically, commutative operations—such as addition or multiplication cannot take order into account. Therefore, it is difficult to model the difference between for example: *"I have to read this book."* and *"I have this book to read"*. In this study, we are able to improve upon existing word embeddings, despite still using a order invariant algebraic composition functions. Some of our experiments with learnable composition functions, that do take order into account, do not perform better by a large margin.

Erk and Padó [37] also discussed the problem of limited capacity. Encoding long phrases into fixed size vector representations often requires compression, and this could mean the loss of information. Recent advancements in *neural machine translation* (NMT) [25] have shown that semantic information of a phrase, or an entire sentence, can in fact be captured in a real-valued vector to the extend needed for the fairly complicated task of machine translation. For longer sentences, the compressed encoding vector can be concatenated with an attention vector [8, 79]. These are usually created using a weighed average. The performance gains yielded from adding an attention mechanism also illustrate that a simple element-wise arithmetic composition can contribute to semantics.

Before the popularity of deep learning approaches increased, there has been progress with using more sophisticated distributional approaches. These distributional approaches can suffer from data sparsity problems due to large matrices that contain co-occurrence frequencies. Baroni and Zamparelli [11] composed adjective-noun phrases using an adjective matrix and a noun vector. Grefenstette and Sadrzadeh [53] did something similar but they use a matrix for relational words and vectors for argument words. Yessenalina and Cardie [137] used matrices instead of vectors to model each word and compose them using matrix multiplication. Matrix multiplication is not commutative; and can, to some extent, take order in to account.

2.3 Research on Embeddings

Bengio et al. [14] first coined the term *word embedding*, in the context of training a neural language model. Collobert and Weston [29] showed that word embeddings are actually useful for downstream tasks and are great candidates for pretraining.

The popularization of word embeddings can be attributed to Mikolov et al. [84, 85] with *Word2Vec* and their *skip-gram* algorithm. In their work, they discuss composition for word embeddings in terms of analogy tasks. They give a clear picture of the additive compositional properties of word embeddings, however the analogy tasks are still somewhat selective. We will introduce four word embeddings and their algorithms used in this thesis, including Word2Vec, in Section 3.1.

A popular method for creating paragraph representations is called *Paragraph2Vec* or *Doc2Vec* [75], in which word vectors are averaged, as well as combined, with a separate paragraph representation. Such a combined representation can then be used

in document retrieval. This method makes an implicit assumption that averaging is a good method for composition. While averaging is a simple operation, our results show that another simple operation will likely perform better on most embeddings.

Kiros et al. [71] presented the *skip-through* algorithm. Inspired by *skip-gram*, it predicts a context of composed sentence representations given the current composed sentence representation. Which could be described as being a decompositional approach to creating sentence representations.

Wieting et al. [135] showed that word embeddings, such as Word2Vec and GloVe, could be further enhanced by training them to compose sentence embeddings using averaging for the purpose of paraphrasing. Using their embeddings for composition through averaging has shown significant improvements on *semantic textual similarity* (STS) tasks. The structure of their model is similar to ours, but it differs in the loss function and the training objective. Their loss function is magnitude invariant, and this explains why they prefer averaging since averaging and summing are essentially exactly the same if you normalize the embeddings magnitude. Our task involves direct composition to lexicographic lemmas while their training task was a paraphrasing task. The resulting embeddings from this research are the *Paragram* embeddings.

Arora, Liang, and Ma [7] improved on Wieting et al. [135] using a simple weighted average using the function $\frac{a}{a+p(w)}$, where a is a parameter and $p(w)$ is the estimated word frequency. They give a theoretical explanation why this works and why it is in line with empirical results from models such as Word2Vec. We do not apply this weighting mechanism to this work, but we do think experimenting with it could improve our results even further.

Kenter, Borisov, and de Rijke [68] combined approaches from Kiros et al. [71] with the approach from Wieting et al. [135] to create an unsupervised method for learning sentence embeddings using a siamese neural network which tries to predict a sentence from context (CBOW). Kenter, Borisov, and de Rijke also average word embeddings to create a semantic representation of sentences.

Finally, it has to be mentioned that fMRI-imaging results suggest that word embeddings are related to how the human brain encodes meaning [90].

2.4 Compositional models in Deep Learning

In this thesis we will use both simple and complex models for composition. In Chapter 3 we will start by introducing four algebraic composition functions we will use. In Chapter 5 we will expand to more elaborate neural models. Here we will first discuss some related work on these composition functions.

2.4.1 Algebraic composition

Aside from work by Mitchell and Lapata, there are a lot of applications where algebraic composition is applied as part of a model's architecture. Examples are weighted averaging in attention mechanisms [8, 79], or in memory networks [131].

Paperno and Baroni [97] provided some mathematical explanations for why algebraic composition is performing well. Arora, Liang, and Ma [7] introduced a mathematical framework which attempts a rigorous theoretical understanding for the performance of averaging skip-gram vectors. Gittens, Achlioptas, and Mahoney [47] built on this and proofed that the skip-gram algorithm actually ensures additive compositionality in terms of analogy tasks. There is one caveat, they assume a uniform word distribution. However, it is widely known that words are distributed according to Zipf's law [140].

2.4.2 Convolutional composition

Work on more elaborate neural network composition can be divided into two categories: *convolutional approaches* and *recurrent approaches*. Convolutional approaches use a *convolutional neural network* (CNN) to compose word representations into n-gram representations. Kim [69] composed embeddings using a single layer CNN to perform topic categorization and sentiment analysis. Kalchbrenner, Grefenstette, and Blunsom [67] presented a new pooling layer to apply CNNs to variable length input sentences. Liu et al. [78] later improved this model by reversing its architecture. In Chapter 5 we will also use two convolutional composition models. We did not find it to be the most effective method, but this could be due to our straightforward architecture.

2.4.3 Recurrent composition

Models utilizing a *recurrent neural network* (RNN) can read input sentences of varying length. They have been used for NMT [25] and *neural question answering* (NQA) [131, 52] as well as other model classes. Cho et al. [25] introduced the encoder-decoder architecture as well as the *gated recurrent unit* (GRU) to be a more efficient alternative to the *long short-term memory* (LSTM) [59]. Sutskever, Vinyals, and Le [123] improved upon the encoder-decoder model by stacking recurrent layers and reversing the input. The GRU unit is empirically evaluated by Chung et al. [26] and they found that the GRU is comparable in performance with less computational cost. We use the GRU to create an order dependent composition function in Chapter 5.

In most deep learning models word embeddings are trained jointly with the model in a supervised manner. The embedding-matrix in these models are good candidates for transfer learning from the unsupervised context-driven approach to jump start training. When applying transfer learning it is important to consider the compositional properties of the used embeddings.

We would like to make a remark on the encoder-decoder architecture, since we could not find a similar remark anywhere in the literature. In the case of such recurrent models, one should consider the effect its architecture has on the compositionality of representations at various points in the model. For example, the encoder in a NMT model uses semantic and syntactic information to generate a good sentence representation through an RNN. However, the decoder is only interested in a representation of the semantics of the encoded sentence. But within the encoder the hidden state should still contain syntactic information to allow for the composition to happen properly for each encoding step. So inherently the model is not optimized for pure semantics at the start of decoding. This remark is unrelated to our work on tuning for better compositionality but still interesting to consider when creating an encoder-decoder architecture.

Now we turn to attention mechanisms [8, 79], which are important components of NMT and NQA systems. In such a mechanisms, creating an attention vector boils down to using a different method for composition, as opposed to RNN-encoding. In a traditional attention architecture, multiple assumptions are made on how they compose representations, e.g. using the hidden states from the encoder as input and using a weighted average over all source words or a specific window.

2.4.4 Recursive composition

Socher et al. [119, 117, 118] introduce a *matrix-vector recursive neural networks* (MVRNN), which uses the syntactic tree structure from constituency parse to compose embeddings. Their models are not end-to-end because of the required constituency parser. The model relies on a correct parse to make good compositions, this is not always the case. But it is one of the first models that tries to separate syntactic information from the word embedding to focus solely on the semantics.

The same work also introduces the SST dataset described in Section 4.4.

2.5 Evaluation of Word and Sentence Embeddings

Evaluating word representations in general is a difficult task. This usually happens in terms of the cosine similarity between two words and is handcrafted for specific examples. The method by Faruqui and Dyer [38][1] is a popular way do such an evaluation. Their evaluation combines thirteen different word pair similarity sets [58, 46, 39], with a total of 11,212 word pairs, and they use the *Spearman*'s rank correlation coefficient as a metric. Because their method focuses on word pairs they can capture the semantic similarity between words, but cannot necessarily say something about their compositionality. In Section 4.3 we will discuss how we use these evaluation methods in this work.

[1] More commonly known as `https://wordvectors.org`.

There are analogy tasks which you could use to evaluate embeddings in terms of both semantics as well as compositionality [84, 99], instead of just only looking at semantics. These tasks are limited and specific in scope, however. We do not use word analogy evaluations in this thesis.

Conneau et al. [30] created grouping of sentence evaluation methods[2], including some on compositionality. Downstream performance in applications such as sentiment classification [117, 111] or question answering [77] provide an extrinsic evaluation of compositionality, but results may suffer from other confounding effects that affect the performance of the classifier. The *Microsoft Research Paraphrasing Corpus* (MRPC) [34] and STS [4, 5], from SemEval challenges, are evaluation tasks which can be used to determine sentence similarity and are also good candidates to test composition indirectly. Marelli et al. [83] created the *Sentences Involving Compositional Knowledge* (SICK) dataset which tests two important aspects of composition specifically: textual relatedness and textual entailment. We use all of these evaluation methods in this work. We discuss them in Section 4.4.

None of these works seem to evaluate broad compositional semantics directly. Our evaluation method, called *CompVecEval*, fills this gap. We will introduce it in the next chapter.

[2]The set of evaluation methods is provided in the SentEval software library `https://github.com/facebookresearch/SentEval`.

Chapter 3

Evaluating the compositionality of existing word embeddings

Throughout this thesis, four different sets of pretrained word embeddings will be compared. This chapter will first introduce these embeddings and their associated algorithms. Next, we will explain four simple algebraic composition functions that can be applied to the embeddings in order to create a single representation of meaning from a chain of multiple words, e.g. for a phrase or sentence.

When there is a set of word embeddings and a means of composition, there has to be a method to check if the new composed embeddings are good semantic representations. To this end, we will introduce an evaluation method for measuring the quality of the semantic composition of word embeddings. This new evaluation method is called: *CompVecEval*.

The evaluation method will use a subset of lexicographic definitions and lemmas from *WordNet* [86]. If we have such a definition, according to the principle of compositionality, it should be composable to the word it describes. For example, if we have all the single word embeddings for the words in the lexicographic definition from WordNet: *"A small domesticated carnivorous mammal..."* when put through a particular composition function, the result should be close to the lexical semantic representation of the lemma *'cat'*. Because words in a dictionary can have multiple definitions, our test will specifically evaluate for the various senses of ambiguous words. So the test makes sure the embeddings of all lexicographic descriptions of *"cat"* are able to compose into its lexical representation.

The experiments in this chapter will show results of our four different pretrained word embeddings in combination with our four algebraic composition functions on our new evaluation metric.

Contributions presented in this chapter:

Technical The CompVecEval method to evaluate semantic compositionality through lexicographic data;

Literature an overview of four popular algorithms to create word embeddings;

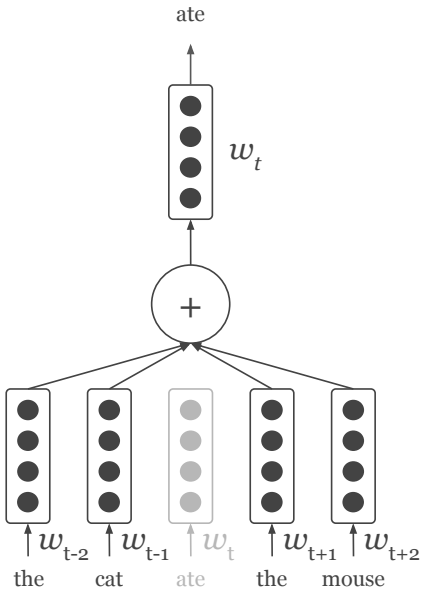


FIGURE 3.1: This figure shows the CBOW architecture. We have the sentence "The cat ate the mouse." and the model tries to predict the word 'ate' from the left context 'the cat' and the right 'the mouse'.

Literature an overview of four commutative algebraic composition functions and their applications with references to literature; and

Scientific a comparison of the four algebraic composition methods on the four popular word embedding algorithms through our evaluation method CompVecEval.

3.1 Four popular pretrained Word Embeddings

Shortly after the rise in popularity of artificial neural networks the unsupervisedly learned *word embedding* became a popular, if not the most popular, method to represent the lexical semantics of a word in latent space. Generally speaking, the method is the embedding from a mathematical space with one dimension per word in the vocabulary, to a real-valued vector space with a lower dimensionality ($\mathbb{R}_n$).

Word embeddings are an essential part of modern model architectures for various application domains, e.g. document retrieval [43], automatic summarization [139], machine translation [123], sentiment analysis [35] and question answering [122]. Many architectures for tasks in NLP have in fact, completely replaced traditional distributional features, such as LSA [31] features and Brown clusters [21], with word embeddings [106]. Goth [51] even hails word embeddings as the primary reason for NLP's breakout.

Most architectures for application specific purposes start their training procedure with pretrained embeddings. These pretrained embeddings can come from anywhere. In this thesis, will use four popular publicly available embeddings which many researchers have used to kick start their model's training procedure.

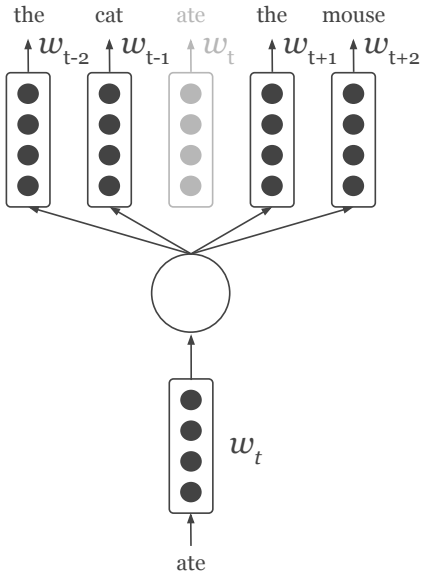


FIGURE 3.2: This figure shows the Skip-gram architecture. We have the sentence "The cat ate the mouse." and the model tries to predict the left context 'the cat' and the right 'the mouse' from the center word 'ate'.

3.1.1 Word2Vec

Arguably the most popular word embedding model is the one that ushered in their rise in popularity and resulted in hundreds of papers [106]. Word2vec by Mikolov et al. [85, 84] was introduced in 2013. The model is, in contrast to other deep learning models, a shallow model of only one layer without non-linearities. Mikolov et al. approached the problem differently than Bengio et al. [14], who originally coined the term *word embedding*, using suggestions from Mnih and Hinton [91]. The model left out the multiple layers and the non-linearities so it could gain in scalability and use very largex corpora.

The paper by Mikolov et al. [85] introduced two architectures for unsupervisedly learning word embeddings from a large corpus of text. The first architecture is called CBOW and is depicted in Figure 3.1, it tries to predict the center word from the summation of the context vectors within a specific window. The second, and more successful architecture, is called *skip-gram* and is depicted in Figure 3.2. This architecture does the exact opposite, it tries to each of the context words directly from the center word. The used pretrained word2vec embeddings are trained using the Skip-gram algorithm. This algorithm is also the inspiration for the algorithms behind the GloVe and fastText embeddings.

In the skip-gram algorithm, each word is assigned both a context (u_w) and a target vector (v_w). These vectors are used to predict the context words (c) that appear around word (w) within a window of M tokens. The probability is expressed using a softmax function.

$$p(w|c) = \frac{e^{u_c^T v_w}}{\sum_{i=1}^{n} e^{u_i^T v_w}} \tag{3.1}$$

In practice one could use methods to speed up the training procedure by using either a hierarchical softmax or negative sampling [84].

The algorithm assumes that the conditional probability of each context window around the word w factorizes as the product of the conditional probabilities.

$$p(w_{-M}, ..., w_M|c) = \prod_{\substack{m=-M \\ m\neq 0}}^{M} p(w_m|c) \tag{3.2}$$

Now in order to find the embeddings, we can maximize the likelihood of the entire training corpus by going over all words using equations 3.1 and 3.2. Which entails in maximizing:

$$\frac{1}{T}\sum_{i=1}^{T}\sum_{\substack{m=-M \\ m\neq 0}}^{M} \log p(w_{i+m}|w_i). \tag{3.3}$$

In equation 3.3, T denotes the total number of tokens in the training corpus.

Mikolov et al. published pretrained embeddings alongside their work[1]. These embeddings were the first to be trained on a significantly large corpus of 100 billion tokens from English news articles and have a dimensionality of 300. These articles originated from different media outlets and were bundled together a news search engine from Google, called: Google News. In our work, we will use these publicly available pretrained embeddings.

3.1.2 GloVe

Global Vector for Word Representation by Pennington, Socher, and Manning [99] (GloVe) was inspired by the skip-gram algorithm and tries to approach the problem from a different direction. Pennington, Socher, and Manning show that the ratio of co-occurrence probabilities of two specific words contains semantic information. The idea is similar to TF-IDF [108] but for weighing the importance of a context word during the training of word embeddings.

Their algorithm works by gathering all co-occurrence statistics in a large sparse matrix X, wherein each element represents the times word i co-occurs with j within a window similar to skip-gram. After which the word embeddings are defined in terms of this co-occurrence matrix:

[1]The pretrained Word2vec embeddings can be found at: `https://code.google.com/archive/p/word2vec/`.

$$w_i^T w_j + b_i + b_j = \log(X_{ij}). \tag{3.4}$$

In order to find the optimal embeddings w_i and w_j for all words in the vocabulary V the following least squares cost function should be minimized.

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2 \tag{3.5}$$

In the cost function f is a weighting function which helps to prevent learning only from extremely common word pairs. The authors fixed X^{max} to 100 and found that the hyperparameter $\alpha = 3/4$ produced the best empirical results.

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{X^{max}})^{\alpha} & \text{if } X_{ij} < X^{max} \\ 1 & \text{otherwise} \end{cases} \tag{3.6}$$

The authors published various different embeddings alongside the paper[2]. There are embeddings with varying dimensionalities trained on Wikipedia articles and tweets from Twitter. Besides these embeddings Pennington, Socher, and Manning also published embeddings trained on a dataset from Common Crawl[3]. This dataset contains 840 billion tokens, which is significantly more than the 100 billion tokens the Word2vec embeddings were trained on. The published Common Crawl trained embeddings have a dimensionality of 300 and a original vocabulary of 2.2 million. We use them in this thesis to compare GloVe to the other pretrained embeddings.

3.1.3 fastText

Bojanowski et al. [17] introduced the fastText embeddings by extending the skip-gram algorithm to not consider words as atomic but as bags of character n-grams. Their idea was inspired by a the work from Schütze [113] in 1993, who learned representations of character four-grams through singular value decomposition (SVD). An example might be nice to illustrate the bags of character n-grams. For instance, the word *'lions'* with $n = 3$ will be represented by the character n-grams:

```
<li, lio, ion, ons, ns>
```

and an additional sequence, which is treated as separate from the n-grams:

```
<lions>.
```

In practice, the authors extract all n-grams for $3 \leq n \leq 6$. One of the main advantages of this approach is that word meaning can now be transferred between words, and thus embeddings of new words can be extrapolated from embeddings of the n-grams already learned. An obvious example is word morphology, e.g. perhaps you have

[2]The pretrained GloVe embeddings can be found at: `https://nlp.stanford.edu/projects/glove/`.
[3]The Common Crawl dataset can be found at: `http://commoncrawl.org`.

seen *'lion'* in the training data but have not seen *'lionesque'*. Now with the bags of character n-grams we know what the suffix <esque> means and the root <lion> and can thus extrapolate the meaning of *'lionesque'*.

The training objective for skip-gram *with negative sampling* is denoted as minimizing:

$$\sum_{i=1}^{T}\Big(\sum_{\substack{m=-M\\ m\neq 0}}^{M}\ell(s(w_i,w_m))+\sum_{n\in\mathcal{N}}\ell(-s(w_i,w_n))\Big). \tag{3.7}$$

Where $\ell(x)=\log(1+e^{-x})$ and w_n is the negative sample. If we wanted to employ Mikolov's method [84] we would have $s(w,c)=u_w^T v_c$. Instead Bojanowski et al. represent the scoring function s as a sum over the bag of character n-grams in the word:

$$s(w,c)=\sum_{g\in\mathcal{G}_w} z_g^t v_c. \tag{3.8}$$

In equation 3.8, $\mathcal{G}_w$ is the collection of all the n-grams for the considered word w, and z are the embeddings learned for these n-grams. If we wanted to obtain the embedding for a specific word, we would simply sum the associated character n-grams.

The authors published pretrained word vectors for 294 different languages[4], all trained from the Wikipedia dumps of the different languages. All the pretrained word embeddings have a dimensionality of 300. In this work, we use the word embeddings extracted from the English Wikipedia which contains a mere 3.4 billion words. This is orders of magnitude smaller than the size of the corpora GloVe and Word2Vec are trained on.

3.1.4 Paragram

Wieting et al. [135] introduced a method to tune existing word embeddings using paraphrasing data. The focus of their paper is not on creating entirely new word embeddings from a large corpus. Instead, the authors are taking existing pretrained GloVe embeddings and tune them so words in similar sentences are able to compose in the same manner [134]. Their technique is therefor somewhat similar to the technique we will introduce in Chapter 4.

Their training data consists of a set of P phrase pairs (p_1, p_2), where p_1 and p_2 are assumed to be the paraphrases. The objective function they use focuses to increase cosine similarity, i.e. the similarity of the angles between the composed semantic representations of two paraphrases. To make sure the magnitude of the embeddings does not diverge by exploding or imploding, a regularization term is added to keep the embedding matrix W_w similar to the original GloVe embedding matrix $W_{w_{GloVe}}$.

[4]The pretrained fastText embeddings can be found at: `https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md`.

$$\frac{1}{|P|}\Big(\sum_{\substack{(p_1,p_2)\in P \\ (n_1,n_2)\in \mathcal{N}}} \max(0, \delta - \cos(f^c(p_1), f^c(p_2)) + \cos(f^c(p_1), f^c(n_1))) + \max(0, \delta - \cos(f^c(p_1), f^c(p_2)) + \cos(f^c(p_2), f^c(n_2)))\Big) + \lambda_w ||W_{w_{GloVe}} - W_w||^2 \tag{3.9}$$

In the objective function a composition function $f^c(x)$ is included. This composition function is similar to the composition function we will introduce in the next section of this chapter. n_1 and n_2 are carefully-selected negative examples taken from a mini-batch during optimization. The intuition is that we want the two phrases to be more similar to each other $cos(f^c(p_1), f^c(p_2))$ than either is to their respective negative examples n_1 and n_2, by a margin of at least δ.

Important to mention is that Wieting et al. expresses similarity in terms of angle and not in terms of actual distance. Our evaluation method, as well as our optimization function from Chapter 4, will do this differently. Additionally, Wieting et al. only explored one algebraic composition function, namely: averaging of the word vectors. In our work, we will explore not only averaging but four different algebraic composition functions.

The data for tuning the embeddings used by the authors is the PPDB[5] or *the Paraphrase Database* by Ganitkevitch, Van Durme, and Callison-Burch [44]. Specifically, they used version XXL which contains 86 million paraphrase pairs. An example of a short paraphrase is: *"thrown into jail"* which is semantically similar to *"taken into custody"*.

Wieting et al. [135] published their pretrained embeddings called Paragram-Phrase XXL[6], which are in fact tuned GloVe embeddings, alongside their paper. These embeddings also have a dimensionality of 300 and have a limited vocabulary of 50.000. In order to apply the embeddings, according to Wieting et al., they should be combined with Paragram-SL999 [134] which are tuned on the SimLex dataset [58]. We therefore use a combination of these embeddings in our work.

3.2 Algebraic Composition Functions

Now that all the pretrained embeddings and their associated algorithms are introduced, it is time to see how these word embeddings can be combined, i.e. composed, into semantic representations of a set of words. Figure 3.3 shows how this can be done. The compositional function f^c can be anything from a very complicated neural network to simple element-wise addition. In the evaluation in this chapter, we will

[5]The paraphrase database can be found at: `http://www.cis.upenn.edu/~ccb/ppdb/`.

[6]The pretrained Paragram-Phrase XXL embeddings and Paragram-SL999 embeddings can be found at: `http://ttic.uchicago.edu/~wieting/`.

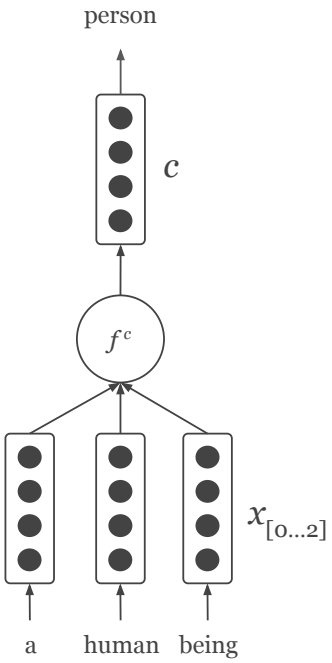


FIGURE 3.3: This figure shows the composition of word embeddings as a function of f^c. The input embeddings go into the composition function f^c and get composed into a single composed embedding c. "*A human being*" composed to the lemma "*person*" is an example from our dataset.

test four simple algebraic composition functions. Eventually, in Chapter 5 we will also introduce six learnable and more elaborate composition functions.

Combining multiple intermediate representations into one simple representation is something which happens in a lot of deep learning architectures. But in itself, it is not often studied in detail. In our case, we look at the compositionality of word embeddings but these approaches could be extended to other types of representations. Similar evaluations on compositional functions could take place.

First, we will try to create a composition by applying element-wise operations: $+$, $\times$, $\max(p)$ and $\text{average}(p)$. Composing by simple commutative mathematical operation is not ideal, since the act of composing considers neither non-linear relationships between individual words nor the order of the words. Instead such relationships should already be present in the linear space of all words under the operation. By analyzing the results from simple operations, we could have new insights into the word embedding space itself, how it already has compositional properties and how it can be used.

3.2.1 Compose by Averaging

The function f^c using averaging for embedding composition:

$$f^c(p) = \text{average}(p) = \frac{1}{|p|} \sum_{w \in p} w. \tag{3.10}$$

Averaging representations is a popular composition function for word embeddings, and is used in a lot of architectures. For example, the popular doc2vec algorithm [75] uses it to combine word representations in order to generate a representation of an entire paragraph or document. The work by Wieting et al. [135], that resulted in the Paragram embeddings, also found that tuning embeddings using simple averaging results in better sentence embeddings. Although, they did not try other algebraic composition function. Kenter, Borisov, and de Rijke [68] use averaging in their unsupervised method to train sentence embeddings from large bodies of text.

Arora, Liang, and Ma [7] provide a theoretical justification for composition through averaging word embeddings. They show that algebraic operations on word embeddings can be used to solve analogy tests, but only under certain conditions. Specifically, the embeddings have to be generated by randomly scaling uniformly sampled vectors from the unit sphere, and the ith word in the corpus must have been selected with probability proportional to $e^{u_w^T c_i}$. Here is c_i the so-called discourse vector, which describes the topic of the corpus on the ith word. This discourse vector changes gradually as the algorithm traverses the corpus. Another specific condition is that the discourse vector should change according to a random walk on the unit sphere. Because their argument relies on vectors sampled from the unit sphere it could be strongly tied to composition through addition as we will show in the next section.

A lot of evaluation measures for semantic representations do comparison based on the angle of vectors and not on their actual distance. For example, word similarity and word analogy tasks compare vectors based on their cosine similarity. Obviously if one takes the average of vectors or just adds them and only looks at the angle for comparison the result will be exactly the same.

It should be noted that the magnitude, i.e. norm, of the pretrained embeddings is sometimes assumed to be 1 [106]. If one makes this assumption, averaging is a logical composition function over summation. However, the magnitude of all pretrained embeddings is not 1. Instead, it varies from embedding to embedding. If one wants the embeddings to have this property they would have to normalize the embeddings, with which they could lose valuable information.

3.2.2 Additive Composition

The function f^c using summation for embedding composition:

$$f^c(p) = \sum_{w \in p} w. \tag{3.11}$$

As discussed in the previous section, for word similarity and word analogy tasks, adding word vectors will yield the same result as averaging them. This is due to the fact that these evaluation measures only look at the difference between the angles of word vectors. Interestingly, Arora, Liang, and Ma [7] show that you can improve

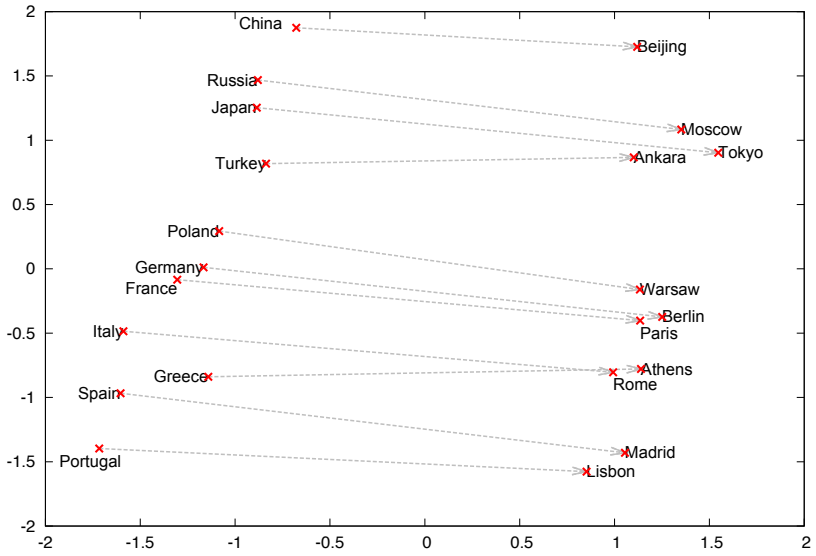


FIGURE 3.4: This figure from Mikolov et al. [84] shows a two-dimensional PCA projection of embeddings of countries and their capital cities. With this visualization, we can clearly see how word embeddings are able to solve word analogy tasks.

upon sentence embedding evaluations from averaging word embeddings by weighting them according to:

$$\frac{a}{a + f(w)} \tag{3.12}$$

where a is a hyper parameter and $f(w)$ is the estimated word frequency. In essence, this would mean that summing word embeddings is a better composition function as long as the norm of the embedding associated with word w is equal to equation 3.12.

To add more weight to this assertion, Gittens, Achlioptas, and Mahoney [47] provide theoretical justification for the presence of additive compositionality in word embeddings learned using the skip-gram model. This theoretical justification does not have the strict conditions of the work by Arora, Liang, and Ma [7] and is therefore more general. Gittens, Achlioptas, and Mahoney [47] still make the following assumption: *"Capturing the compositionality of words is to say that the set of context words $c_1, ..., c_m$ has the same meaning as the single word c for every other word w, $p(w|c_1, ..., c_m) = p(w|c)$."*, which is still a debatable assumption since general semantic compositionality is not defined as such [124].

Summing word embeddings to compose words is done by various architectures, examples are the original CBOW algorithm [84] and the combination of n-grams in the fastText algorithm [17]. Additive composition is used less than averaging if one looks at architectures for combining word embeddings to create phrase or sentence embeddings.

Talking about the additive composition of word embeddings without discussing the famous royalty example could be considered heresy in the research community. Seemingly, half of the papers on word embeddings mention this example.

Mikolov et al. [84] found that word embeddings could easily solve word analogy tasks, e.g. *'king' - 'man' + 'woman' = 'queen'*. Figure 3.4 shows how one could create the average transformation from a country one would need to make to arrive at the embedding of that countries capital. This shows additive composition in possible the form of *'man'* + `royalty` = *'king'* or *'Netherlands'* + `capital` = *'Amsterdam'*.

3.2.3 Multiplicative Composition

The function f^c using multiplication for embedding composition:

$$f^c(p) = \prod_{w \in p} w. \tag{3.13}$$

The work by Mitchell and Lapata [89, 88] shows that multiplicative models are superior to additive models on specific types of word vectors. This work does not use the real valued word embeddings that are compared in this thesis, rather they use frequency based embeddings.

One problem with multiplicative models is that it can suffer from vanishing of exploding scalars within the vector. Since the initialization of word vectors happens using uniformly sampling from the unit sphere, the scalars within a word vector are likely to be lower than or close to 1. Which means that if multiple vectors are multiplied the resulting scalars can vanish. Floating point numbers within computers have limited precision, in our experiments we use 32 bit precision floating points. Which makes it hard for computers to handle very small, or very large, values with sufficient precision.

3.2.4 Compose by Max-pooling

The function f^c using max-pooling for embedding composition:

$$f^c(p) = \max(p) = \max_{w \in p} w. \tag{3.14}$$

Outside of NLP, the max-pooling operation has been successful in improving results for image classification tasks in Computer Vision [73]. In essence, max-pooling composes outputs from a convolutional filter into a single output. This was found to be beneficial for final image classification performance.

We do not expect max-pooling to yield the best results of our four algebraic composition functions. However, it could be an interesting composition function to compare to. We see it as a baseline.

Max-pooling cannot combine information from a specific embedding dimension but it makes a discrete choice to use one of the scalar values per embedding, i.e. the maximum, which we think will limit its ability to compose.

3.3 WordNet as lexicon for evaluating compositionality

Now, we will continue with our contribution and introduce the dataset for CompVecEval. In order to evaluate compositional semantics, i.e. the composition of word embeddings using a specific composition function, we turn to a dictionary for our data. Lexicography is important to illustrate the relationship between words and their meaning [45]. They are the pragmatic human solution to the problem of word meaning. The words or lemmas in dictionaries all have compact descriptive definitions, which can be composed semantically into the meaning of that word, and are thus ideal for our task.

We choose to use WordNet [86] as the basis for our dataset. The synonym set in WordNet allows for the creation of pairs $x = (d, l_d)$ of definitions $d \in \mathcal{D}$ with one, or many lemmas $l_d \subset \mathcal{L}$ associated with that definition. A definition is a list of words where $d = \{w^d \in \mathcal{W} | w_1^d, w_2^d \dots w_n^d\}$. These words are tokenized from a string using the NLTK [15] software library. For our evaluation method, we only consider single word, i.e. unigram, lemmas which are also in $\mathcal{W}$ for $\mathcal{L}$, which makes $\mathcal{L} \subset \mathcal{W}$. We added this constraint because this makes the evaluation method more usable because word embedding does not necessary have to be applied on the target side, even though that is still possible.

If we find that the lemma is actually one of the definition words, we do not add that lemma to l_d for that particular x. Basically we do not allow lemmas to be explained by using the exact same word. We end up with $|\mathcal{X}| = 76,441$ unique data points with a vocabulary of $|\mathcal{D}| = 48,944$ unique words and a target vocabulary of $|\mathcal{L}| = 33,040$ unique lemmas.

3.3.1 Train-test split

Since our objective is to create a tuning method and an evaluation method for existing embeddings, using this new dataset, we have to split the dataset into train and test portions. The structure of the data in $\mathcal{X}$ is such that we cannot randomly split anywhere. When splitting we make sure that a lemma l with multiple definitions d are all in the same set. Otherwise, the training algorithm would be able to train on lemmas that are also in the test set, which would make for unbalanced results. Additionally, we make sure that both training and test datasets contain at least one definition word w which is the same as a lemma l from the other set, to prevent diverging embeddings.

We end up with a train dataset of $72,322$ data points and a test dataset of $4,119$ data points. The test dataset contains $1,658$ unique lemmas.

We made the dataset and the code to create the entire dataset freely available.[7].

[7] You can find the dataset and scripts to create it on: `https://github.com/tscheepers/CompVec`.

3.3.2 Ngram Lemmas

When we start tuning embeddings in Chapter 4 we also consider lemmas which consist of multiple tokens. One could expect that a lexicon such as WordNet does only contain lemmas that exist of one word, i.e. unigrams. However, this is not the case. Multi-gram lemmas do exist within WordNet. When we expand to ngram lemmas with $n > 1$ we can create a train set with $|\mathcal{X}| = 126,926$ instead of $72,322$, which is an increase of $54,604$ data points. The target vocabulary then increases to $|\mathcal{L}| = 35,040$ unique lemma words. Our primary motivation to extend the dataset has to do with the tuning model described in Chapter 4. We would like the model to learn from as much data as it can.

3.3.3 Vocabulary overlap with the pretrained embeddings

Not all words in the vocabularies of the four pretrained embeddings overlap with our WordNet based vocabulary. In Appendix C.1 you can see the exact overlap for each of them. Embeddings for words that are not in the vocabulary of pretrained embeddings will be initialized randomly by sampling scalar values from a uniform distribution between -1 and 1.

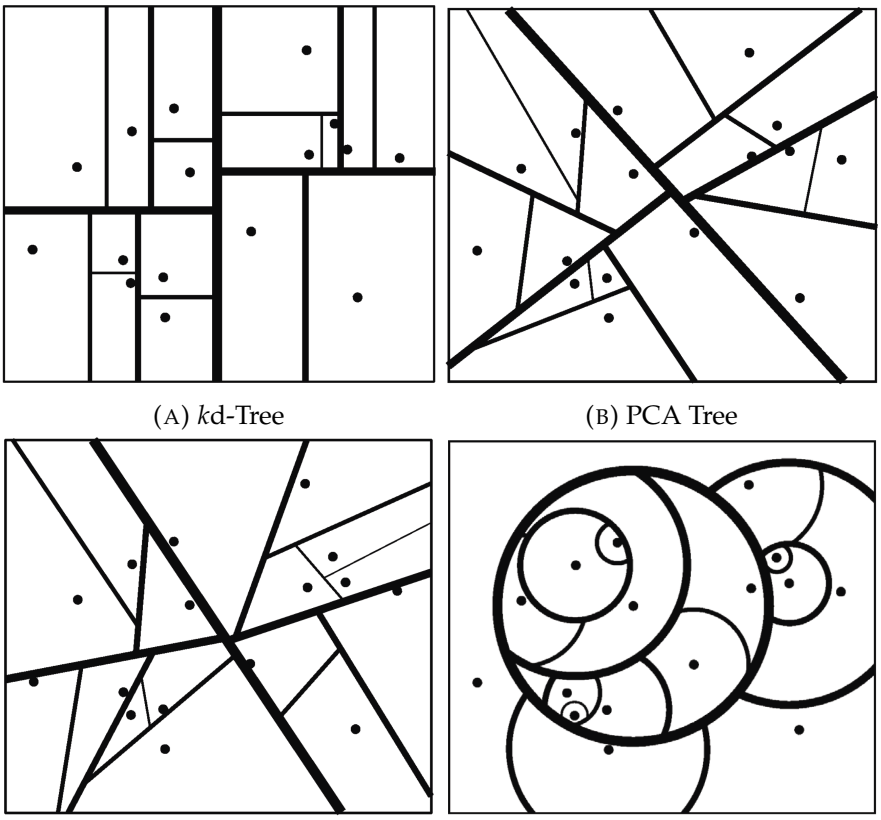
We realize that this makes it difficult to directly compare the pretrained embeddings. However, our goal was not to compare the four pretrained embeddings. Instead, we wanted to focus on creating an evaluation measure which could be applied to any word embedding with any vocabulary. If we would have filtered the vocabulary according to the vocabularies of the pretrained embeddings it would have biased our evaluation method towards these. Besides, penalizing word embeddings for not including a vocabulary word which is present in a dictionary could also benefit our evaluation method.

3.3.4 Data input practicalities

In our experiments, we truncate the input definitions tokens to a maximum of 32, and pad the unused tokens. In order to determine this cut point, we looked at the length of all definitions. We found that the mean definition length was: 10 tokens, the 95th percentile was: 22 tokens and the 99th percentile: 31 tokens. Additionally, we create a mask for the padding tokens to correctly handle the composition functions for variable input length.

We mask padding tokens and make sure the composition in our model is handled correctly. For example, padding tokens should have a vector filled with zero values as an embedding for the $+$ operation while they should have a vector filled with one values as embedding for the $\times$ operation.

For ngram lemmas, the mean length is 1.75, the 95th percentile is 4 tokens and 99th percentile is 7 tokens. We choose to truncate the ngram target lemmas at 8 tokens. But, this is only true for the expanded dataset used in Chapters 4 and 5.



(A) kd-Tree (B) PCA Tree

(C) Ball Tree (D) vp-Tree

FIGURE 3.5: This figure from Kumar, Zhang, and Nayar [74] illustrates the working of various nearest neighbor ranking algorithms.

3.4 Compositional Vector Evaluation

Testing semantic compositionality using data from WordNet has to be done in a specific manner. It has to be the case that the evaluation method can distinguish between embeddings with different magnitudes, as stated in Section 3.2.2. In general, representations of semantics in neural architectures are effected by a vector's magnitude. One does not normalize each intermediate representation, not only because this pragmatic, it can also lead to a loss of information and can influence a neural network's performance.

3.4.1 Nearest Neighbor Ranking

Instead of using a magnitude invariant distance metric such cosine similarity, we compare our composed representation to our target word representation using absolute Euclidean distance. To achieve this, we employ a nearest neighbor ranking through the ball tree algorithm [94].

The ball tree algorithm produces a complete ranking of all the $1,658$ possible target lemmas from our WordNet based test set[8]. In the subsequent ranking, we mark all correct target words from l_d as equally relevant, and all other words as irrelevant.

[8]The amount of possible target lemmas will be the same for both unigram target and ngram target datasets, since both datasets use the same test set.

We believe that ranking is superior to other evaluation methods because it is independent of the provided embedding space. Additionally, there are several metrics you can compute out of a ranking which could be interpreted in different ways, especially regarding compositionality. Also, ranking allows us to find structure in the very noisy compositional representations. On the other hand, one mayor disadvantage of using ball tree ranking is the time complexity involved in creating a ranking tree, which is $O(n(\log n)^2)$ [94, 74].

Kumar, Zhang, and Nayar [74] compared various ranking algorithms and found that the Ball Tree algorithm was the only one of these three options that had "Excellent" nearest neighbor search performance but low construction performance. Search performance is important because creating a ranking efficiently requires this to be high. Figure 3.5 shows various other options, kd-Tree and PCA Tree have inferior search performance, whereas vp-Tree has comparable search performance to Ball Tree. In future work, one could investigate performance improvements which might be possible by switching to a vantage point algorithm (vp-Tree) [138].

3.4.2 Ranking measures

Now that a ranking of the relevant results is obtained, we can apply several well-known ranking measures to it. We will look at: *Mean Reciprocal Rank* (MRR) [24], *Mean Average Precision* (MAP) as well as *Mean Precision@10* (MP@10). These ranking measures are generally used in *Information Retrieval* (IR). We also introduce *Mean Normalized Rank* (MNR), which is a metric that is not top-heavy. For all metrics we can state: "higher is better".

$$\text{precision} := \frac{|\{\text{correct target lemmas}\} \cap \{\text{nearest target lemmas}\}|}{|\{\text{nearest target lemmas}\}|} \tag{3.15}$$

Precision is one of the oldest metrics for ranking. Since it is possible that a lexicographic definition can describe multiple lemmas, precision can measure whether all these relevant lemmas are listed within the ranking.

$$\text{Precision@k} := P(d,k) = \frac{\sum_{r_d=1}^{k} \text{rel}(r_d)}{k} \qquad \text{MP@k} := \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} P(d,k) \tag{3.16}$$

In IR, one is often only interested in the k top most results, and not necessarily in all the results. Therefore, Precision@k is often used. This variation on regular precision has a cut-off point at k, as we see in equation 3.16. In this equation $\text{rel}(r_d)$ is either 1 or 0 depending on whether the result r for definition d is relevant or not. In order to calculate the precision at k for a number of examples, we turn to MP@k which just averages the results for different test examples.

$$\text{MAP} := \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{\sum_{r_d=1}^{|\mathcal{L}|} P(d, r_d) \times \text{rel}(r_d)}{|l_d|} \tag{3.17}$$

Calculating precision@k does not give a higher value when a relevant result is higher in the ranking, as long as the relevant result is higher than k the value does not change. In order to capture higher relevant lemmas, we look to average precision—or to MAP for multiple rankings.

$$\text{MRR} := \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{1}{\text{rank}_d} \tag{3.18}$$

An alternative to MAP is MRR. This measure only uses the *highest relevant rank*, denoted by rank_d, and is therefore simpler. However, it can not distinguish between a ranking with more than two relevant results where the first relevant result is at the exact same index but the ranking quality difference can be found in comparing the rank of the second or third relevant results.

Both MAP and MRR are so called *top heavy* metrics, where a difference in ranking higher in the list makes a larger difference in the eventual value, than when the difference happens lower in the ranking.

$$\text{MNR} := 1 - \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{\text{rank}_d}{|\mathcal{L}|} \tag{3.19}$$

We also evaluate using a new metric we introduce in this thesis: MNR. It describes the fraction of the total dataset that does not need to be viewed when encountering a relevant result. In contrast to MAP and MRR, this metric is not top heavy. All of these metrics are so-called recall based metrics. A nice aspect of MNR, is that it is really easy to interpret. For example, an MNR of 0.9 means we have seen 0.1 of the all the lemmas before we found the first relevant one, which one could consider not very good but still better than for example: 0.8. The ability that MNR allows to make such a simple statement, makes it an intuitive metric in our opinion.

3.4.3 Defining CompVecEval

The broader goal of introducing an evaluation method for semantic composition through lexicographic definitions, is not only to compare four pretrained embeddings and some composition functions. Rather, the goal is: giving researchers an empirical tool to reason about the compositionality of their search for new word embeddings and new composition models. It is desirable that the method is both simple in its technique as its interpretability.

We choose to define CompVecEval as computing the MRR measure on rankings provided by a Euclidean nearest neighbor algorithm calculated from the test set of $4,119$ **data points**[9]**.**

We only use a subset of the data points because we use the rest for tuning the embeddings in Chapter 4. We chose MRR because it is simple in its application, but top

[9]These exact data points are available in our code repository.

heavy and thus has a nice distribution. If we would have taken MNR as our primary metric it would always have been a result between 0.5 and 1.0, since 0.5 can be obtained through a random ranking. It is our opinion that MRR could be a better indicator because of its top heavy nature, and since a random ranking will result in a close to 0.0 value.

If ones task does not require the training data, one could use all the available data points (76,441 or 131,926, depending on using unigram or ngram lemmas) for an even more accurate evaluation method. However, this would yield results that are not comparable to the results in this thesis. Whenever one wants more insight into the compositionality of word embeddings, one can also look at the other ranking methods considered: MNR, MAP or MP@10. The results from the experiments in this chapter will include all these metrics. However, in subsequent chapters we will stick to one metric according to the CompVecEval definition and thus stick to MRR.

3.5 Experimental results

In our experiments we will now evaluate the combinations of our four composition functions with our four pretrained embeddings on all metrics discussed in Section 3.4.2. Next, we will do some introspection, and look at some qualitative results.

3.5.1 Quantitative results

Table 3.1 shows the results for our CompVecEval evaluation. We can clearly see that additive composition, i.e. using the $+$ operation, is far superior to all other composition methods. Keep in mind that this new evaluation method uses distance to compare embeddings instead of angle, as seen with other approaches. Our empirical results for word2vec specifically is in line with the mathematical proof Gittens, Achlioptas, and Mahoney [47] provide. This proof states that skip-gram results in additive compositionality, with some small caveats. For the other algorithms of the pretrained embeddings, no such mathematical proof has been presented. However, our empirical results show similar behavior.

Because the random initializations for out of vocabulary words, we should be cautious when comparing the pretrained word embeddings. Additionally, this means that the evaluation methods constitutes some stochasticity. In Appendix A.5 on statistical significance, we show that for additive composition this stochasticity negligible. When we do compare the various pretrained word embeddings, we assume the penalization out of vocabulary words by randomly initializing them.

In our original short paper [109], which presented the method described in this chapter we evaluated the various word embeddings only with a conjunctive, i.e. overlapping, vocabulary. This eliminated the stochasticity but also made the evaluation method less general. The results presented in this paper are totally in line with the results reported here, and similar conclusions were drawn.

TABLE 3.1: The results from evaluating ranking. All results for MRR, MAP, and MP@10 are denoted as $\times 100$ in percentages. We also included results from random rankings on the dataset, this is not specific to any of the word embeddings.

		Word2Vec	GloVe	fastText	Paragram
MRR	random	0.7 %			
	$+$	16.8 %	11.9 %	20.7 %	**26.5 %**
	avg(d)	2.0 %	3.3 %	3.0 %	*3.8 %*
	$\times$	0.6 %	0.9 %	0.9 %	*1.0 %*
	max(d)	6.6 %	13.7 %	14.6 %	*20.5 %*
MNR	random	54.2 %			
	$+$	83.9 %	83.5 %	86.3 %	**90.3 %**
	avg(d)	71.7 %	*75.5 %*	71.2 %	71.2 %
	$\times$	62.8 %	*65.2 %*	59.0 %	54.6 %
	max(d)	63.1 %	83.7 %	78.5 %	*85.7 %*
MAP	random	0.6 %			
	$+$	15.3 %	10.8 %	18.9 %	**24.8 %**
	avg(d)	1.8 %	2.9 %	2.6 %	*3.4 %*
	$\times$	0.6 %	0.8 %	0.8 %	*0.9 %*
	max(d)	6.0 %	12.4 %	13.3 %	*18.9 %*
MP@10	random	0.1 %			
	$+$	2.9 %	2.2 %	3.6 %	**5.2 %**
	avg(d)	0.3 %	0.6 %	0.5 %	*0.8 %*
	$\times$	0.1 %	0.1 %	0.1 %	*0.3 %*
	max(d)	1.1 %	2.4 %	2.4 %	*3.8 %*

When comparing pretrained embeddings, Paragram is the clear winner. It is also the only embedding which is already tuned from an original embedding (namely GloVe). In the original paper by [135] they already showed significant improvements over GloVe on sentence evaluation tasks, so this is in line with our results here.

We also see a huge margin between the performance of composing by averaging and additive composition. Clearly averaging is not necessarily the best approach when evaluating embeddings using not just their angle. We also see that multiplicative composition clearly fails to amount to a meaningful representation as the results are relatively close to the random baseline. This is in direct contradiction of the statements made by Mitchell and Lapata [89].

Composition by max-pooling also seems to work surprisingly well. Especially, when considering the loss of data inherit in the operation. The operation will discretely choose the maximum value on the embedding dimension. Clearly this seems to have important semantic value.

Word2Vec performs relatively poorly on this compositional dataset, but this can be explained by the nature of the test data. The dataset where Word2Vec was trained on used news data, where fastText and GloVe use more definitional data, Wikipedia and Common Crawl respectively. Still lots of researchers use Word2Vec as a starting point for their training procedure, our evaluation here shows that there are better options.

fastText is able to perform well on additive composition. When comparing it to GloVe,

TABLE 3.2: These are some examples from the top-5 ranked results from the fastText embeddings. Other examples follow the same discussed patterns. This table also shows the excluded stop words, we italicized the actual definition tokens from d.

	Target Words l_d	Definition	$\|d\|$	RR
$+$	drydock	A *large dock* from which *water* can be *pumped* out; *used* for *building ships* or for *repairing* a *ship* below its *waterline*	10	1.0
	iatrogenic	A *induced* by a *physician's words* or *therapy* (*used especially* of a *complication resulting* from *treatment*)	10	1.0
$\times$	kitten	Have *kittens*	1	1.0
$\max(d)$	fruitlessly, unproductively, unprofitably	In an *unproductive manner*	2	1.0
	precociously	*precocious manner*	2	1.0
	imperialist, imperialistic	Of or *relating* to *imperialism*	2	1.0

it performs comparable overall. But, when looking at additive composition specifically, fastText has a clear advantage. We think this can be attributed to the nature of the fastText algorithm and its use of additive composition of character ngrams. We can see that Paragram and GloVe perform better across compositional operations. This makes Pragram and GloVe more flexible alternatives.

The overall trend is that, truly being good at composition is still hard for embeddings under these simple operations. To highlight this, we added a randomly generated ranking for all data points. In this random baseline, we see a MNR of 54.2 % and not of 50.0 % because there are various data points with more than one relevant target words. When we look at the best MNR score for Paragram with addition, 90.3 % comes down to seeing 400 of the total 4,119 target words on average, before encountering a relevant one. This is way better than the random baseline, but still very bad. Which is good news, since it means there is a lot of room for improvement in making embeddings more composable.

If we look at the various ranking measures, i.e. MRR, MNR, MAP and MP@10, we see similar patterns across all of them. There are no real discrepancies between top heavy and uniform ranking measures; or between precision based and recall based ranking measures. Because of this lack of discrepancies we feel confident in defining CompVecEval as using the simple top heavy measure MRR.

3.5.2 Qualitative results

Table 3.2 shows some of the best ranking examples of the fastText embeddings. The examples in this table illustrate patterns that can be attributed to the mathematical operations applied. As we already saw in the quantitative analysis the multiplicative operation performs poorly. And, we see this back in our example since the best ranking examples are not composed at all, i.e. their definition only exists of one word. The $\max(d)$ operation also shows an interesting pattern where we see definitions containing two words appear in the top ranked results, and one of these words is closely related to the target word. This is behavior you would expect of the $\max(d)$ operation. The additive operation shows some sophisticated composition examples to be

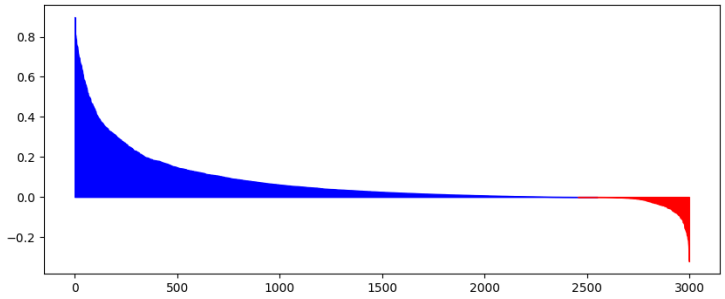


FIGURE 3.6: This figure shows the difference in MNR between + and average(d) for the GloVe embeddings. In blue we see the embeddings with a higher MNR for + and in red we see embeddings with a higher MNR for average(d).

its best ranking results, which is certainly interesting and requires further investigation. It is surprising that long definitions with 10 tokens end up at the top of the list for addition.

To see at what types of sentences particular operations are actually good at we looked at the difference in metrics for specific definitions. In figure 3.6 we see the difference in MNR for + and average(d). When looking at the definitions which perform best with average(d) compared to + we see specific definitions with a high number of tokens of rare target words. An example: *'CF': "The most common congenital disease; the child's lungs and intestines and pancreas become clogged with thick mucus; caused by defect in a single gene; no cure is known. "*. When looking at the opposite, i.e. the best performing under + we see more general definitions but still for a high number of definition words. An example: *'Receptor': "An organ having nerve endings (in the skin or viscera or eye or ear or nose or mouth) that respond to stimulation."*.

In Figure 3.7 we see a t-SNE [82] projection where we compare a composed embedding to all possible target embeddings. We can see that for this particular example the composed definition is very close to the associated lemma. Which is what we expect.

3.6 Conclusion

In this chapter, we introduced a new take on an existing dataset combined with combinatorial methods to evaluate word embeddings on their semantic compositionality. With this evaluation method, one can gain new insights in the applicability of word embeddings in neural networks and how one should approach defining a neural network architecture to best cope with semantic composition.

The main finding of the work from this chapter is that the best composition function one could use is additive composition. We provide empirical evidence for the mathematical claims made by [47] on skip-gram. We find similar empirical evidence for the other algorithms.

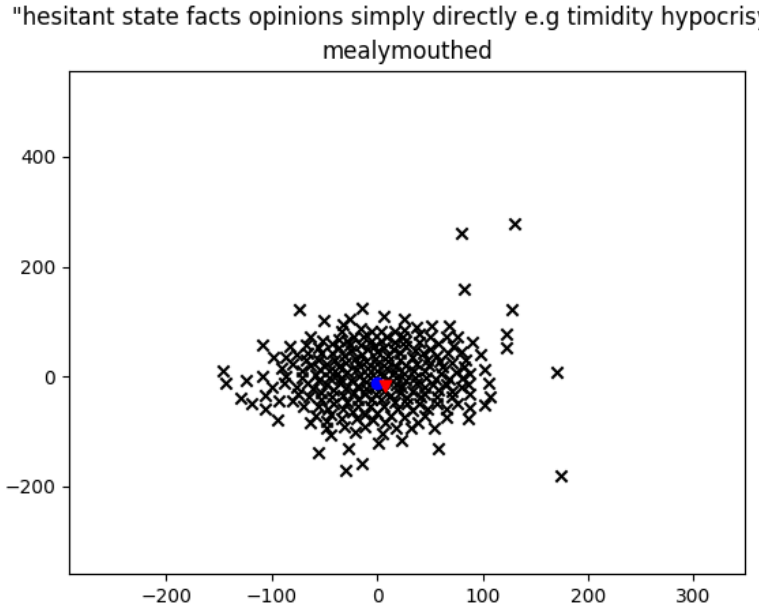


FIGURE 3.7: The t-SNE [82] projection of a good composition for from the + model with word2vec embeddings. The blue dot is the representation of the target words, and the red triangle the composed representation.

From the four popular pretrained vectors investigated, we found that Paragram embeddings by Wieting et al. [135] are probably the best embeddings one can use for pretraining or transfer learning when using them in a model that wants to use semantic representations of words and wants them to be composable.

In the next chapter we will see if we can tune the existing embeddings to see if we are able to improve their performance on both CompVecEval and other evaluation methods.

The contributions described in this chapter were also presented in the short paper: 'Analyzing the compositional properties of word embeddings' [109]. In this paper we used a slightly different dataset, one which only included data comprised of words from the conjunction of four vocabularies (WordNet, word2vec, GloVe and fastText). This allowed us to compare pretrained embeddings more directly.

Chapter 4

Training word embeddings for algebraic composition

In the last chapter we introduced a method to evaluate word embeddings on their compositionality under a specific algebraic composition function. In this chapter, we will explore if we are able to improve the compositionality of word embeddings for these specific algebraic composition functions. First, we will introduce a tuning model. This model is inspired by negative sampling [84] and siamese network architectures [20]. Second, we make sure the general quality of the word embeddings does not degrade by tuning. When we tune embeddings we should make sure the embeddings are still generally applicable. They should not only be tuned to the task specifically, but rather have comparable or better performance across a wide range of tasks. Therefore, we will check our tuned embeddings not only against CompVecEval, but also against fifteen sentence embedding evaluation methods and thirteen word representation evaluation methods. Each of these evaluation methods was created to check a different aspect of either the sentence or the word embeddings space.

Contributions presented in this chapter:

Technical The CompVec model architecture to train word embeddings by optimizing them for semantic composition under a specific composition function;

Literature a more detailed overview of thirteen different word representation evaluation methods (earlier briefly described by Faruqui and Dyer [38]);

Literature a more detailed overview of fifteen different sentence representation evaluation methods (earlier briefly described by Conneau et al. [30]);

Scientific the experimental results from tuning the four popular word embedding algorithms using the four different *algebraic* composition function through all discussed evaluation methods; and

Technical publicly available tuned variants of the four pretrained word embeddings.

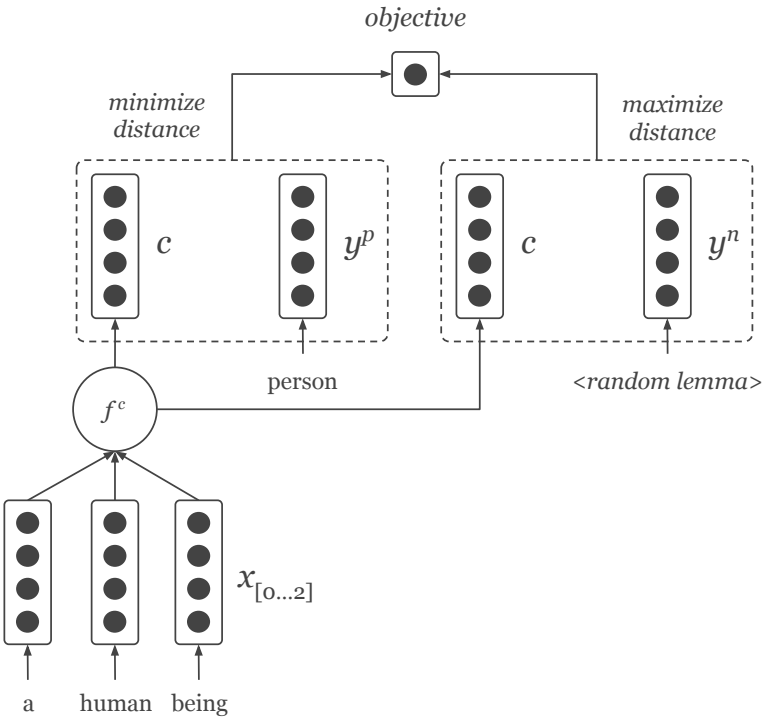


FIGURE 4.1: This figure illustrates the CompVec model structure. The input embeddings go into the composition function f^c and get composed into a single composed embedding c. We see how this composed vector c gets compared against the positive example y^p and the negative example y^n in the loss function.

4.1 Tune embeddings for compositionality

We will first introduce our model structure: *CompVec*. In order to tune embeddings, we must consider the n input embeddings $x_{[0...n]}$ which result in a composed vector c trough composition function f^c as depicted in Figure 4.1. Additionally, there are two target vectors, one for the correct lemma y^p and one random negative example y^n. The goal of our model is to minimize the distance between the vectors c and y^p and maximize the distance between c and y^n.

4.1.1 Triplet loss

The triplet loss function [57] is used as the objective function for our training algorithm. Using the negative example ensures the embedding values do not converge to 0. In the previous chapter, we discussed the importance of evaluating on Euclidean distance instead of evaluating on cosine similarity. In the experiments in this chapter, we use two different loss functions: one using Euclidean distance and one using cosine similarity, see Equation 4.1 and 4.2 respectively.

$$\text{triplet loss} := \sum_{i=1}^{N} \max\Big(||c_i - y_i^p||^2 - ||c_i - y_i^n||^2 + \alpha, 0\Big) \tag{4.1}$$

Wieting et al. [135] use a similar, albeit a slightly more elaborate, objective function for tuning the Paragram embeddings. We explained this in Section A.2. Where Wieting et al. use two different negative examples and only compares on cosine similarity, we use just one negative example and compare mainly on Euclidean distance. In their research, they focus on compositing by averaging embeddings, where we found that summation could be considered superior from our initial results. Wieting et al. also use a regularization term where they try to keep the final embedding matrix W as close to the initial embedding matrix $W_{initial}$ as possible. We omit this regularization term since we believe this unnecessarily constraints our tuning procedure.

$$\text{cosine triplet loss} := \sum_{i=1}^{N} \max\Big(\cos(c_i, y_i^p) - \cos(c_i, y_i^n) + \alpha, 0 \Big) \tag{4.2}$$

In the triplet loss function, we want to minimize the distance from c and y^p and maximize the distance between c and y^n. However, we take a margin α into account. If the joint distance is within this margin we consider c and y^p close enough and c and y^n far enough. In our experiments, we choose to use the margin value $\alpha = 5$ for the additive composition and $\alpha = 0.25$ for all other models. We choose these values based on the difference in magnitude of composed embeddings.

4.1.2 Optimization

In order to tune the embeddings and optimize the objective function we use an algorithm reminiscent of Stochastic Gradient Decent [103]. In stochastic mini-batch gradient descent, the algorithm computes the gradient of the objective function $\mathcal{L}(\theta)$ with respect to to the parameters θ. The new parameters take on the value:

$$\theta_{t+1} = \theta_t - \eta \Delta_\theta \mathcal{L}(\theta). \tag{4.3}$$

Here the step η the algorithm takes each training iteration is called the *learning rate*. Each training iteration we compute the objective using a different set of training examples, i.e. the mini-batch. This mini batch is sampled stochastically from the entire distribution of training examples.

In our experiments, we will use an optimization algorithm called Adaptive Moment Estimation (Adam) [70]. We choose not to use the newer Nadam [36] optimizer nor the newer Eve [72] optimizer due to a lack of implementation availability. Adam is an optimization algorithm introduced by Kingma and Ba in 2014 and expands on stochastic mini-batch gradient decent. It computes adaptive learning rates for each parameter. Additionally, it stores exponentially decaying gradients v_t and keeps an exponentially decaying average of past gradients m_t, similar to the concept of *momentum* [105].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)(\Delta_\theta \mathcal{L}(\theta)) \tag{4.4}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\Delta_\theta \mathcal{L}(\theta))^2 \tag{4.5}$$

In order to update the parameters, the algorithm uses $\hat{m}_t$ and $\hat{v}_t$ which are derived from the first momentum m_t and the second momentum v_2 respectively. There is an adjustment to make sure these parameters do not go to zero at the start of training.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.6}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.7}$$

The update function for Adam comes down to:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t. \tag{4.8}$$

By using this optimization function on the triplet loss, it will ensure that the composed embedding c moves closer to the reference lemma embedding y^p and at the same time moves away from a random other reference lemma embedding y^n.

In our experiments we use a learning rate of $\eta = 1e^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 1e^{-8}$.

4.1.3 Regularization

While most word embedding algorithms do not include a form of regularization during the training procedure, we experimented with adding it to ours. Regularization is a method that is generally used to prevent over-fitting.

We have empirically validated that adding dropout [121] improved the final results considerably. Dropout is a technique where single neurons or scalers in the input embeddings are "dropped out" during the training procedure. This happens with a dropout probability of p, i.e. scalars will be included with a probability of $p - 1$. On a side note, one interesting aspect of dropout is that it has been shown to have a full probabilistic interpretation [42].

Through light hyper-parameter tuning we have found that a dropout probability of $P_{dropout} = 0.25$ on the input embeddings resulted in good tuning performance. All our final models are trained with this specific dropout configuration and probability.

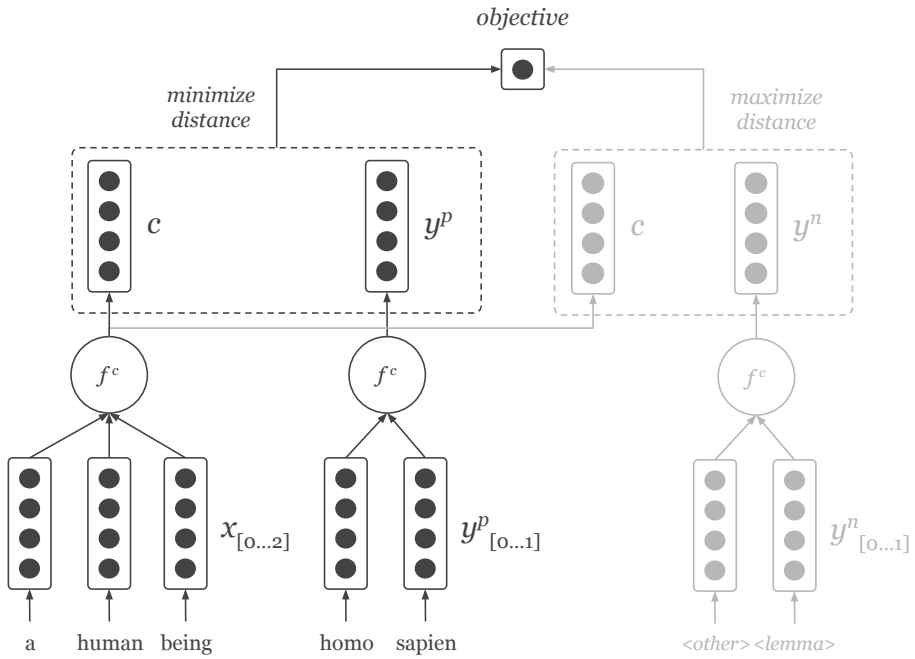


FIGURE 4.2: This figure shows CompVec model structure with targets that consist of multiple tokens. In this case the targets will also be composed.

4.1.4 Training data and multi token targets

As described in Section 3.3, we have separate train and test datasets. We evaluate our models using two different training datasets. First, we experimented with the 72,322 data points which only has unigram target lemmas. With this dataset, we can optimize the embeddings using the model structure depicted in Figure 4.1.

Additionally, we explore the effect of the larger dataset consisting of 126,926 data points, which contains ngram target lemmas with $1 \leq n \leq 8$. When using multi token targets for y^p and y^n, we need to make sure that these are also composed into a single vector. Therefore, this dataset requires both the definition (source) and the lemma (target) to be composed by composition function f^c. This is depicted in Figure 4.2.

The maximizing effect of the distance to the negative example makes sure that f^c does not allow for the composed vectors to go to zero.

During training the loss is calculate on both the train and the test set, respectively they will be the train loss and test loss.

We train the model for 20,000 steps using batches of 512 data points with the described Adam optimizer. This means the models using unigram lemma dataset will be trained for 142 epochs and the models using ngram lemma dataset will be trained for 81 epochs.

4.2 Measuring improvement of the overall quality

Our proposed evaluation method, CompVecEval is an evaluation method that uses similar lexicographic data as the data our models are trained on. While it is a method for evaluating the compositionality of word embeddings, it is not necessarily a measure for the general overall quality of word embeddings. Especially, because we are training for the task of composition specifically.

While it is difficult to measure the general quality of word embeddings, we can evaluate the embeddings using various additional methods. These methods should test the word embeddings on tasks our models are not directly optimized for. This way we can present a broad view of the quality and applicability of the resulting word embeddings.

Some of these methods will report ranking correlations calculated using the embeddings and rankings produced by human judges. The metrics for these types of evaluation methods are usually Pearson's rank correlation coefficient [98] or Spearman's rank correlation coefficient [120]. First, we will discuss these two correlation coefficients briefly in the following sections, before diving into the evaluation methods for word and sentence representations.

4.2.1 Pearson's rank correlation coefficient

Pearson's correlation coefficient (r) [98] is also sometimes called the sample correlation coefficient and was introduced by Pearson in 1896. It is calculated on values $x_1, ..., x_n$ and $y_1, ..., y_n$ from two lists, which are tested for correlations. Person's r is obtained through the following formula:

$$\text{Pearson's } r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{x})^2}}. \tag{4.9}$$

Here $\bar{x}$ is the mean, i.e. $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Analogously for $\bar{y}$.

We will use Pearson's r for our sentence embedding evaluations described in Section 4.4. It is customary to report Pearson's r for these methods. Pearson's correlation coefficient formula is also the foundation for the the next coefficient—Spearman's r.

4.2.2 Spearman's rank correlation coefficient

Spearman [120] introduced a variation on Pearson's r in 1904 called: Spearman's rank correlation coefficient. Instead of using the actual values x_i and y_i, it instead uses the rank indices from two ranked lists of data points, $r_1, ..., r_n$ and $s_1, ..., s_n$. Spearman's r can be obtained through the following formula:

$$\text{Spearman's } r = 1 - \frac{6\sum_{i=1}^{n}(r_i - s_i)^2}{n(n^2 - 1)}. \tag{4.10}$$

TABLE 4.1: All the different datasets we use for word evaluation.

Dataset	*n* word pairs	Paper	Year
RG-65	65	Rubenstein and Goodenough [104]	1965
MC-30	30	Miller and Charles [87]	1991
WS-353	353	Finkelstein et al. [39]	2001
YP-130	130	Yang and Powers [136]	2006
WS-353-SIM	203	Agirre et al. [1]	2009
WS-353-REL	252	Agirre et al. [1]	2009
MTurk-287	287	Radinsky et al. [102]	2011
MTurk-771	771	Halawi et al. [55]	2012
MTurk-3000	3000	Bruni et al. [22]	2012
RW-STANFORD	2034	Luong, Socher, and Manning [80]	2013
VERB-143	143	Baker, Reichart, and Korhonen [10]	2014
SimLex-999	999	Hill, Reichart, and Korhonen [58]	2016
SimVerb-3500	3500	Gerz et al. [46]	2016

We will use Spearman's *r* for our word similarity evaluations in Section 4.3 as this is customary for these types of evaluation methods.

4.3 Evaluating Word Representations

"How is the general quality of individual word embeddings affected by the tuning of the embeddings for composition under a specific operation?" This is the question we will answer by evaluating on thirteen different word similarity and relatedness datasets. These thirteen datasets were compiled by Faruqui and Dyer [38] for the website `wordvectors.org`, which is used widely to measure embedding quality. These thirteen, include datasets which are widely used among researchers to evaluate word embeddings.

We incorporated the code behind `wordvectors.org` into our training procedure. This allowed us to evaluate before, during and after training. We will now introduce each of the datasets individually. We will report the results of all datasets (Table 4.1) but will primarily report, WS-353, SimLex-999 and SimVerb-3500.

It is easy to confuse semantic similarity with semantic relatedness. Semantic relatedness describes any relation between two words, while semantic similarity only describes the "is a" relation, e.g. a 'car' is similar to a 'truck' but is related to 'road' and 'driving'.

We will report Spearman's *r* for all of these methods. It will compare word pair values contained in the dataset, which are the result from human annotations, with the *cosine similarity* between two of our word embeddings. Keep in mind that the metrics do not use absolute distance but rather only the angle between two words.

The vocabulary we use for tuning is limited. Therefore, it does not contain all words used in the evaluation measures. In Appendix C.1 we list the amount pairs we are able to use for each of the datasets. Because the evaluation omits some word pairs one has to be careful when comparing the results reported in this chapter to the results

TABLE 4.2: Here we see examples from the SimLex-999 dataset example values compared to values from WordSim-353, we clearly see that SimLex only captures similarity independent of relatedness.

Pair	SimLex-999 rating	WS-353 rating
coast - shore	9.00	9.10
clothes - closet	1.96	8.00

from other works, which are most likely obtained using entirely different vocabularies. All this also has to be considered when comparing the results reported in this chapter to the results presented in Chapter 6.

While researching word similarity evaluation measures for this thesis, we contributed open source code to the repository of `wordvectors.org`*. This can be seen at:* `https://github.com/mfaruqui/eval-word-vectors/graphs/contributors`*. We mainly expanded the wordvectors.org code with the SimVerb-3500 dataset.*

4.3.1 WS-353

WS-353[1] by Finkelstein et al. [39] from 2001 is a widely used WordNet based word similarity measure. Naturally, we should improve on this measure since it is based on similar data. We also use WordNet as a basis for the dataset used for tuning the embeddings. This should be noted and might make the results on this dataset biased. This word similarity set is not very elaborate either, with merely 353 word pairs. However, it is one of the most widely used. It was originally introduced to evaluate search ranking algorithms.

Similarity and Relatedness Split

Agirre et al. [1] provided a more detailed study on the similarity and relatedness of the dataset by Finkelstein et al. [39]. They split the original dataset into two two separate subsets[2], one only containing word pairs regarding similarity and one only containing word pairs regarding relatedness. These are called *WS-353-SIM* and *WS-353-REL* respectively. The word pairs in these sets are not necessarily mutually exclusive. This results in a similarity dataset containing 203 word pairs and a relatedness dataset containing 252 word pairs.

4.3.2 SimLex-999

SimLex-999 by Hill, Reichart, and Korhonen [58] is a fairly large word similarity dataset with 999 word pairs[3]. These 999 word pairs can be divided up into 666 noun pairs, 222 verb pairs and 111 adjective pairs.

[1]The WS-353 dataset can be downloaded at: `http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/`.

[2]The WS-353-SIM and WS-353-REL datasets can be downloaded at: `http://alfonseca.org/eng/research/wordsim353.html`.

[3]The SimLex-999 dataset can be downloaded at: `http://www.cl.cam.ac.uk/~fh295/simlex.html`.

The dataset was specifically created to capture similarity independent of relatedness and association. In Table 4.2 we see an example provided by the authors. It shows that the example pair *'coast'* and *'shore'* are similar and thus rank high on both datasets. However, we see that *'clothes'* and *'closet'* are related but not similar.

The experiments by Hill, Reichart, and Korhonen [58] show that it was challenging for NLP models to replicate the results from the human annotators. They attribute this difficulty to the similarity - relatedness split. Most models, including the ones described in this thesis, infer connections between concepts from their co-occurrence in corpora which primarily reflects relatedness.

It is our intuition that training using lexicographic data will improve on similarity specifically, since dictionary definitions could be generally more focused on similarity.

4.3.3 SimVerb-3500

SimVerb-3500 by Gerz et al. [46] focuses on the similarity between 3500 verb pairs[4]. The way these verb pairs are captured makes for a very balanced dataset

The USF norms data set [93] was used in the creation of SimVerb-3500. USF is the largest dataset of free association norms collected for the English language. Human subjects were asked to give the first word that came up in their head when provided with one of 5000 concepts. Each of these concepts was presented to 10 participants. The resulting dataset contained 72,000 concept, word associations. The authors of SimVerb-3500 sampled from this dataset to create both similar and dissimilar word pairs.

4.3.4 Early word similarity datasets

We also include two small datasets *RG-65* by Rubenstein and Goodenough [104] and *MC-30* by Miller and Charles [87]. These contain 65 and 30 pairs of nouns respectively and have been given similarity rankings by humans directly. They are the first of these types of datasets described in the literature, with *RG-65* being introduced in 1965 and *MC-30* in 1991.

4.3.5 Stanford's rare words

RW-STANFORD by Luong, Socher, and Manning [80] from Stanford, is a dataset which solely contains pairs of rare words—2034 pairs to be exact[5]. An example of such a word pairs is: *'villainous'* - *'wicked'*. The dataset was introduced to show how an RNN model using morphemes, was better at producing representations of rare

[4]The SimVerb-3500 dataset can be downloaded at: `http://people.ds.cam.ac.uk/dsg40/simverb.html`.

[5]The RW-STANFORD dataset can be downloaded at: `https://nlp.stanford.edu/~lmthang/morphoNLM/`.

words than earlier models. The dataset contains nouns, verbs and adjectives and also provides scores between word pairs of with different parts-of-speech.

We expect our tuned embeddings to perform better on this rare word dataset, since a dictionary such as ours can provide good definitions of rare words. Whereas, corpora could have too few occurrences of a rare word to create a good semantic representation from context.

4.3.6 YP-130 and VERB-143

YP-130 by Yang and Powers [136] is a dataset from 2006 containing 130 verb pairs with human similarity judgments. It was created before the SimVerb-3500 dataset, and was up to 2016 the method to measure verb similarity. Later in 2014, Baker, Reichart, and Korhonen [10] introduced *VERB-143*, which was constructed slightly different but is in many ways comparable to YP-130. Since SimVerb-3500 contains an order of magnitude more pairs than both YP-130 and VERB-143, it is now the go-to measure for this specific task.

4.3.7 Miscellaneous datasets created using Mechanical Turk

Radinsky et al. [102] created a dataset to capture semantic relatedness using Amazon Mechanical Turk in 2011. It is software which employs crowd sourcing to create datasets using the input of people from all over the world. For this specific dataset, the authors focused on the temporal aspect of 287 word pairs and how this affected relatedness. Now their dataset is more commonly known as *MTurk-287*.

One year later Halawi et al. [55] created an additional dataset using Mechanical Turk, this dataset contains 771 word pairs[6]. Their work introduced a large-scale unsupervised approach to learning word-word relatedness. In order to test this, they created the dataset, which is now known as *MTurk-771*.

That same year (2012) Bruni et al. [22] introduced a significantly larger dataset that consists of 3000 word pairs—*MTurk-3000*[7]. These word pairs were randomly selected from words that occur at least 700 times in a large encyclopedic corpus. They were also judged by people using Amazon Mechanical Turk. They originally used it to measure their model's performance, which used computer vision techniques for NLP applications.

[6]The MTurk-771 dataset can be downloaded at: `http://www2.mta.ac.il/~gideon/mturk771.html`.

[7]The MTurk-3000 dataset can be downloaded at: `http://clic.cimec.unitn.it/~elia.bruni/MEN.html`.

TABLE 4.3: The sentence evaluation methods we will employ.

Dataset	Task	Paper	Year
TREC	Question-Type Classification	Li and Roth [77]	2002
CR	Product Review	Hu and Liu [62]	2004
SUBJ	Subjectivity Status	Pang and Lee [96]	2004
MRPC	Paraphrase Detection	Dolan and Brockett [34]	2005
MPQA	Opinion Polarity	Wiebe, Wilson, and Cardie [132]	2005
MR	Movie Review	Maas et al. [81]	2011
STS12	Semantic Similarity	Agirre et al. [3]	2012
STS13	Semantic Similarity	Agirre et al. [2]	2013
SST	Sentiment Analysis	Socher et al. [117]	2013
STS14	Semantic Similarity	Agirre et al. [4]	2014
SICK-E	Compositional Entailment	Marelli et al. [83]	2014
SICK-R	Semantic Relatedness	Marelli et al. [83]	2014
STS15	Semantic Similarity	Agirre et al. [5]	2015
STS16	Semantic Similarity	Agirre et al. [6]	2016
STS-B	Semantic Similarity Benchmark	Cer et al. [23]	2017

4.4 Evaluating Sentence Representations

In the previous section, we discussed evaluating the quality of individual word embeddings. In this section, we will discuss the evaluation of *sentence embeddings*. This entails an indirect evaluation of the compositionality of word embeddings, i.e. we will evaluate how well specific word embeddings can be combined into a usable sentence representation using a specific composition function f^c. After the creation of a sentence embedding, it will be used for a specific task.

We utilize a software library titled: *SentEval* developed by Conneau et al. [30] to evaluate our composed sentence embeddings. The library contains a collection of various sentence evaluation measures. It allows the code that consumes the library to compose a sentence representation given a specific sentence through a callback. The resulting sentence representation is then used to perform the evaluation method specific to the task.

Through the callback our model is tasked with composing a sentence representation from a provided string. We tokenize the string using NLTK [15], similar to how we did it with our CompVecEval dataset. We should note that we do not use a special *unknown* token for composition, which some models do. So, if an embedding for a specific token is not in our vocabulary and not in the vocabulary of the pretrained embeddings we ignore it for composition. So similar to the word embedding evaluation, the vocabulary used can play a important role. In Appendix C.1, we list the exact vocabulary of each individual sentence evaluation method, and how it overlaps with our vocabulary. There we also show, how many of the vocabulary words are transfered from the pretrained embeddings and how many are randomly initialized.

When an evaluation method requires the training of a specific classification model the library will train logistic regression classifier. A linear classifier requires less parameters than for example a simple neural network, this makes them suitable for small

datasets. The L2 penalty of the logistic regression is tuned using grid-search. When no test set is available, the library uses k-folds cross validation. We choose to set $k = 10$.

Additionally, when a method produces a ranked list or ranked pairs we will report Pearson's r as a metric, since this is done in related work as well. For classifiers, we will simply report accuracy between 0.0 and 1.0.

So, to see if embeddings combined through a specific composition function result in sentence embeddings that are usable across a wide range of tasks, we will test before, during and after tuning using the fifteen different methods provided by SentEval (see Table 4.3). In the next sections, we will discuss each method in detail so we can interpret their specific results more effectively.

Aside from the sentence evaluation methods grouped in the work by Conneau et al. [30], we also ran a selection of experiments on Information Retrieval tasks. Specifically, on the Robust04 and WSJ document retrieval datasets. We elaborate on this septate evaluation in Appendix A.4.

While researching sentence embedding evaluation measures for this thesis, we contributed open source code to the SentEval repository. This can be seen at: `https://github.com/facebookresearch/SentEval/graphs/contributors`. *We made the code Python 3.x compatible and allowed it to be loaded through PIP*[8] *as a library. In addition, we fixed some issues and bugs.*

4.4.1 TREC: Question-Type Classification

The TREC question-type dataset[9] is a dataset that was created by Li and Roth [77] for the Text Retrieval Conference (TREC) back in 2002. It contains 5500 data points for training. The task is to classify each one as one of six classes: Abbreviation, Entity, Description, Human, Location and Numeric. Each class has multiple leaf classes. The data points are simple strings of text, all of which are questions. For our experiments, we will report classification accuracy on 500 test data points using the six main classes learned using the composed representations from the questions. An example from the TREC dataset: "What are the twin cities?" and the questions has the corresponding main class: *Location* and leaf class: *City*.

4.4.2 Microsoft Research Paraphrasing Corpus

Our second dataset is the MRPC by Dolan and Brockett [34], which contains 5800 pairs of sentences which have been extracted from news websites[10]. They are annotated by humans who indicated if the sentence pair is a paraphrase, i.e. captures

[8] PIP is a Python package manager, used to include open source libraries in a software project.

[9] The TREC dataset can be downloaded at: `http://cogcomp.cs.illinois.edu/Data/QA/QC/`.

[10] The MRPC dataset can be downloaded at: `https://www.microsoft.com/en-us/download/details.aspx?id=52398`.

semantic equivalence. In contrast to word similarity datasets, the MRPC only contains two binary classes which indicate whether a pair is either a paraphrase of not. Our method will train a binary logistic regression classifier on 4077 data points and test on 1726 data points. We report accuracy on these test data points.

4.4.3 Stanford: Simple, Sentiment and Topic Classification

Wang and Manning [129] gathered several datasets for their work on sentiment and topic classifications[11]. Four of these datasets are now more widely used as baselines, and are therefore included in our sentence evaluation. Each of the datasets contain examples of two classes. The task is to classify a specific sentence as either one or the other. The datasets do not have a predefined train-test split. Therefore, we train a classifier using k-folds cross validation.

Movie Review

The first dataset (*MR*), is a large balanced dataset of 50.000 full length movie reviews from IMDB. The dataset was originally created by Maas et al. [81]. A review can be either negative or positive. An example data point is: *"Too slow for a younger crowd, too shallow for an older one."*, which is labeled as *negative*.

Product Review

The next dataset contains product reviews by customers (*CR*), and was created from data originally compiled by Hu and Liu [62] using the processing technique employed by [92]. A review can be either negative or positive, of which there are 1366 and 2406 respectively. An example data point is: *"We tried it out Christmas night and it worked great."*, which is labeled as *positive*.

Subjectivity Status

Rather than classifying if a data point is positive or negative, this next dataset focuses on subjectivity (*SUBJ*). It was originally created by Pang and Lee [96] and contains 10.000 balanced data points which are essentially with reviews and objective plot summaries. A data point can be either subjective or objective. An example data point is: *"A movie that doesn't aim too high, but doesn't need to."*, which is labeled as *subjective*.

Opinion Polarity

The last of these datasets is created from a subset of the Multi Perspective Question Answering corpus (*MPQA*). Specifically, the subset on opinion polarity which was

[11] The Stanford classification datasets can be downloaded at: `https://nlp.stanford.edu/~sidaw/home/projects:nbsvm`.

TABLE 4.4: The various subsets of the annual Semantic Textual Similarity tasks.

Year	Dataset	Pairs	Source
2012	MSRpar	1500	Newswire
	MSRvid	1500	Video descriptions
	OnWN	750	Glosses [12]
	SMTeuroparl	750	Machine Translation Evaluation
	SMTnews	750	Machine Translation Evaluation
2013	FNWN	189	Glosses [12]
	OnWN	561	Glosses [12]
	HDL	750	Newswire Headlines
2014	OnWN	750	Glosses [12]
	Deft Forum	450	Forum Posts
	Deft News	300	News Summaries
	HDL	750	Newswire Headlines
	Images	750	Image Descriptions
	Tweet-News	750	Tweet-News pairs
2015	Ans. Forums	750	Q&A Forum Answers
	Ans. Students	375	Student Answers
	belief	375	Committed Belief
	HDL	750	Newswire Headlines
	Images	750	Image descriptions
2016	HDL	249	Newswire headlines
	Plagiarism	230	Short Answer Plagiarism
	Post Editing	244	Machine Translation Post Edits
	Ans.-Ans.	254	Q&A Forum Answers
	Quest.-Quest.	209	Q&A Forum Questions

originally introduced by Wiebe, Wilson, and Cardie [132]. Polarity is also sometimes called semantic orientation and requires an example to explain.

If we have the sentence *"African observers **generally approved** (positive) of his victory while Western governments **denounced** (negative) it."* we can describe parts of it with a specific semantic orientation. This is what the dataset seeks to capture.

The subset contains 3316 positive data points and 7308 negative ones. An example data points is: *"don't want"*, which is labeled as *negative* and another data point is: *"would like to tell"*, which is labeled as *positive*.

4.4.4 SemEval: Semantic Textual Similarity

Where SimLex-999 and WS-353-SIM provides a tool to measure semantic similarity on word embeddings, the STS datasets from the annual SemEval workshop is such a tool for sentence embeddings. Each year, the Special Interests Group on the Lexicon of the Association for Computational Linguistics (ACL) publishes such a dataset. For the workshop, researchers compete to obtain the best scores on that year's dataset using supervised and unsupervised methods. For supervised methods, the participants are allowed to use datasets from previous years as a train set.

TABLE 4.5: This is the contents of the compiled STS Benchmark dataset by Cer et al. [23].

Type	Original Dataset	STS Year	Train pairs	Dev pairs	Test pairs
News	MSRpar	2012	1000	250	250
News	HDL	2013-16	1999	250	250
News	Deft News	2014	300	0	0
Captions	MSRvid	2012	1000	250	250
Captions	Images	2014-15	1000	250	250
Captions	En-En	2017	0	125	125
Forum	Deft Forum	2014	450	0	0
Forum	Ans. Forum	2015	0	375	0
Forum	Ans.-Ans.	2016	0	0	254
			5749	1500	1379

In Table 4.4 we see how these sets are structured. The data comes from a variety of sources. It must be noted that the data from the sources described with 'Glosses' contains text extracted from WordNet[12]. This could make the results of STS tasks including these subsets slightly biased since we also train on data from WordNet.

In 2017 Cer et al. [23] published a compiled benchmark dataset which contains data from all previous STS tasks. It has a predefined train, test and development split. Table 4.5 shows how this dataset called: STS-Benchmark[13] is structured.

STS data contains sentence pairs which are judged on their semantic similarity by human annotators on a scale from 0 to 5. When one compares sentence embeddings using the STS datasets, it is standard practice to calculate cosine similarity between each pair of sentence representations and report Pearson's r. This is what we report as well.

Each dataset has several subsets. We calculate Pearson's r on each subset and then calculate the weighted mean of this r for the entire dataset. We weight r values using the amount of pairs in the subset, see table 4.4. On the STS-Benchmark we report this weighted Pearson's r on the test set.

We test statistical significance on Pearson's r for each subset. Runs are considered statistically significantly different for p-values < 0.001. See Tables A.6 and B.5 in the Appendix for all the p-values.

One of the most popular STS datasets is the one from 2014, which is used in a lot of work on unsupervised sentence representations [68, 135, 30]. We will report results on all datasets but will primarily focus on STS14.

4.4.5 Stanford Sentiment Treebank

In addition to the earlier discussed movie review and product review datasets, there is an additional sentiment analysis dataset from Stanford. It was introduced by Socher

[12]Glosses from lexical resources including WordNet [86], OntoNotes [61] and FrameNet [9].

[13]The latest SemEval STS benchmark dataset can be downloaded at: `http://ixa2.si.ehu.es/stswiki/index.php/STSbenchmark`.

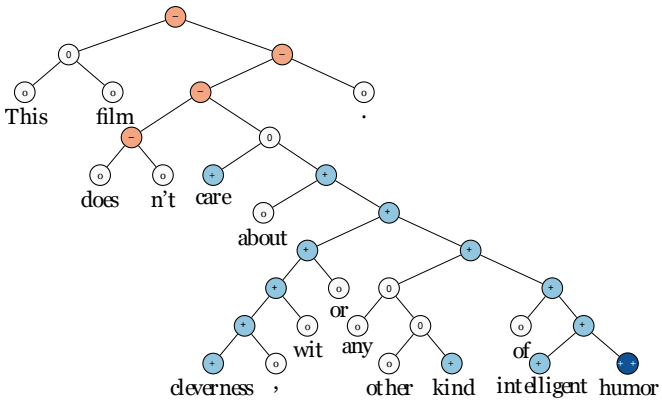


FIGURE 4.3: This figure from Socher et al. [117] shows an example data point from the Stanford Sentiment Treebank, we clearly see the parse tree and how it affects sentiment.

et al. [117] and is called the Stanford Sentiment Treeset (SST)[14]. It is the first corpus with fully labeled parse trees, an example is shown in Figure 4.3. The authors argue that the parse trees allow for a complete analysis of the compositional effects of sentiment in language.

For our method, we only use binary classification examples, as described in the original paper. Our method does not go into the specifics, such as contrastive conjunctions or high-level negation. We train a binary linear regression classifier on the provided 67,349 train examples and report test accuracy on the 1,821 test data points.

4.4.6 Sentences Involving Compositional Knowledge

The last dataset we discuss is the SICK dataset by [83], i.e. Sentences Involving Compositional Knowledge[15]. Originally it was part of the SemEval 2014 task set, just as STS14. The SICK dataset consists of two separate aspects: Entailment and Semantic Relatedness.

SICK Entailment

The entailment subset contains 9,840 sentence pairs for testing. These pairs are combined with a label which can be: Neutral, Contradiction or Entailment. We use three matching methods to extract the relations between the two composed sentences u and v: concatenation of the two representations (u, v); element-wise product $u \cdot v$; and absolute element-wise difference $|u - v|$. The total resulting vector is fed into a logistic regression classifier, just as the other described methods which use classification. We train the classifier on a separate dataset containing 10,000 data points.

[14]The Stanford Sentiment Treeset can be downloaded at: `https://nlp.stanford.edu/sentiment/index.html`.

[15]The SICK datasets can be downloaded at: `http://clic.cimec.unitn.it/composes/sick.html`.

The entire test dataset contains 5,595 neutral pairs, 1,424 contradiction pairs, and 2,821 entailment pairs. An example of a *contradicting* pair of sentences is: "A man is typing on a machine used for stenography." and "The man isn't operating a stenograph.".

SICK Relatedness

Where the entailment required a classifier, the relatedness dataset is treated more or less the same as the STS evaluations methods. This means that we learn to predict the probability distribution of the relatedness scores [126]. Which is then reported as Pearson's r. Instead of using cosine similarity between the different $\hat{y}$ is predicted using a neural network. With this we follow the same approach as Tai, Socher, and Manning [126], which is common for SICK Relatedness [30]. The neural network is trained on a separate training dataset containing 10,000 examples. The test is executed on the same sentence pairs as used with SICK Entailment, it contains 923 pairs with scores between 1 and 2; 1,373 with scores between 2 and 3; 3,872 with scores between 3 and 4; and finally 3,672 with scores between 4 and 5.

4.5 Experimental results

Now that our tuning model and all our evaluation methods are introduced it is due time to present our experimental results.

Our model will tune the four popular word embeddings *Word2Vec* [84], *GloVe* [99], *fastText* [17] and *Paragram* [135] to be used for composition under an algebraic operation. In addition to tuning the four popular embeddings, we also train new embeddings from random initializations. Since we have four operations and five different initializations, in total we run the train procedure twenty times for our final result.

We will present our results on the experiments which use the Euclidean distance loss function, and multi word target tokens. In addition, we executed the twenty experiments multiple times using other parameters.

First, we run the experiments with the smaller dataset containing only unigram target lemmas. It resulted in slightly poorer performance across all metrics, which is why we choose to present our results on ngrams tokens in this chapter. The results from this experiment can be found in Appendix A.1.

Second, we run the experiments with the cosine similarity loss function resulting in truly abysmal results. The results from this experiment can be found in Appendix A.2.

For additive composition with ngram tokens, we ran the four experiments for the different pretrained embeddings, multiple times. This was done in order to determine statistical significance for all evaluation methods on their values before and after training. See Appendix A.5 for the details regarding this significance testing

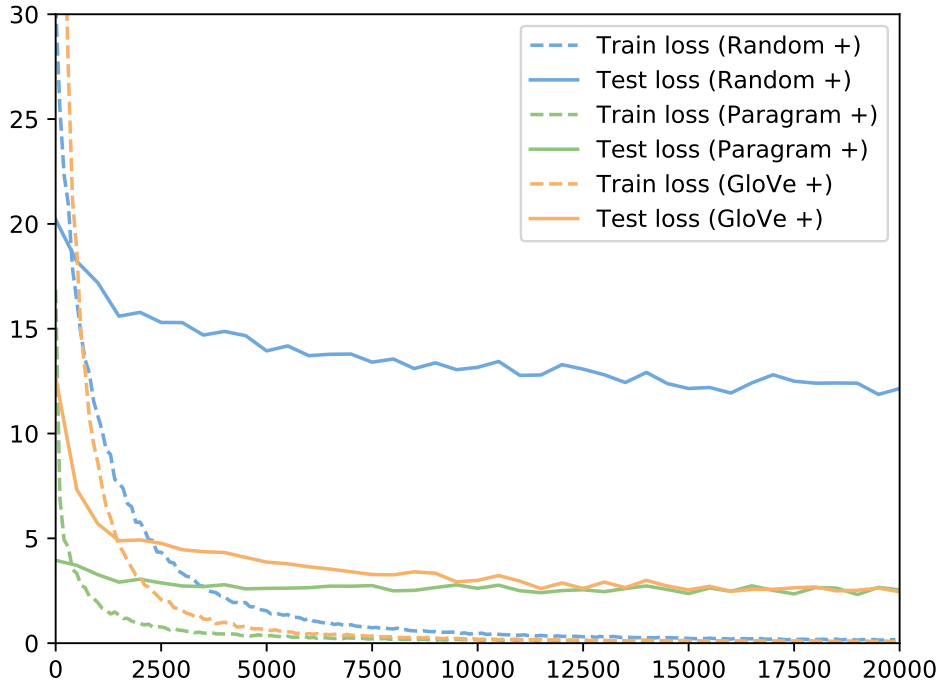


FIGURE 4.4: This figure shows the loss progression during tuning for Random, Paragram and Glove with additive composition.

using Student's t-test [50]. All reported evaluation metrics for additive composition are reported as the mean, which is demerited over several runs.

In Figure 4.4, we can see the progression of the Euclidean loss function during training for additive composition. The procedure is comprised of 20.000 steps, and in each step we train using a mini-batch of 512 training examples. We choose to show the loss progression of three initializations with a specific composition function to illustrate the models learning capability.

Clearly, the loss on the training examples converges to 0, albeit that each one does this at a different rate. The figure shows that test loss also decreases for the three different initializations depicted. This is an indication that the model is learning its task correctly.

We see the random initialization is unable to obtain the low test loss that the other two pretrained embeddings are able to. It must be noted again that Paragram embeddings are themselves tuned GloVe embeddings. The Paragram embeddings start the lowest, which is consistent with our results from the previous chapter. But the GloVe and Paragram test loss eventually converge to a similar number.

Next, we will discuss the effect of tuning using various evaluation functions. First CompVecEval, which we introduced in Chapter 3; second the word representation evaluations; and third the sentence representation evaluations.

TABLE 4.6: **CompVecEval** MRR scores ×100, for our tuned embeddings under a specific operation. We also trained embeddings without pretraining (*None*).

	None	Word2Vec		GloVe		fastText		Paragram	
	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
$+$	10.6	16.8	23.0 (+6.2)	11.9	26.5 (+14.5)	20.7	26.3 (+5.6)	26.5	**29.9** (+3.3)
average(d)	3.5	2.0	2.0 (-0.0)	3.3	4.1 (+0.8)	3.0	3.4 (+0.5)	3.8	*4.1* (+0.3)
$\times$	0.3	0.6	0.9 (+0.2)	0.9	0.9 (-0.0)	0.9	1.0 (+0.1)	*1.0*	0.5 (-0.5)
max(d)	8.8	6.6	15.6 (+9.0)	13.7	20.1 (+6.4)	14.6	18.6 (+4.0)	20.5	*23.3* (+2.8)

This section on the experimental results will finish with showing a t-SNE [82] projection of the most successful tuned embeddings in Figure 4.8.

4.5.1 Results on CompVecEval

Table 4.6 shows the CompVecEval results from the twenty different experiments. We see the CompVecEval results from before tuning under 'Pre.' and after under 'Tuned'.

Across the board we see that the embeddings can tuned to be able to compose better using a specific composition function. The exception is the multiplicative composition function, but this is no surprise since before tuning it already performed poorly. We have a similar explanation as in Section 3.5.1, with it having to do with exploding and vanishing representations.

Embeddings for additive composition and composition using max pooling have the most to gain from tuning. In Figure 4.5 we see the progression of the results on CompVecEval during training. GloVe has the most to gain by tuning, while additively tuned Paragram embeddings came out on top.

The difference in CompVecEval score, before and after training, is statistically significant on all additive composition experiments (see Appendix A.5).

4.5.2 Results on Word Representation Evaluations

Being able to improve upon a task which you are training for specifically is one thing, but being able to improve upon generally or on another task is something else entirely.

In Table 4.7, we see the results on all the word representation datasets described in Section 4.3. Specifically, we choose to highlight SimLex and SimVerb since these are the latest and largest datasets, and additionally WS-353 because it is the most widely used.

In contrast to the results from CompVecEval, here we see that averaging performs comparable to summation. The word evaluation methods use cosine similarity, therefore it only looks at the angle of each embedding. With these results, we can state that our averaged embeddings do still have the right angle, even after tuning, and it might even be the case that this improves. However, this totally leaves out the magnitude of an embedding which could be an important piece of information.

TABLE 4.7: **Word representation evaluation** scores in Spearman's r $\times 100$, for our tuned embeddings under a specific operation. We also trained embeddings without pretraining (*None*).

		None	Word2Vec		GloVe		fastText		Paragram	
		Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
WS-353	$+$	27.4	70.4	70.1 (-0.3)	71.9	*76.5* (+4.6)	74.5	70.1 (-4.3)	73.1	72.1 (-1.1)
	average(d)	42.7	70.4	67.8 (-2.5)	71.9	73.2 (+1.3)	74.5	69.8 (-4.6)	73.1	**76.6** (+3.5)
	$\times$	7.4	70.4	13.6 (-56.8)	71.9	56.1 (-15.8)	*74.5*	14.1 (-60.3)	73.1	56.9 (-16.3)
	max(d)	14.6	70.4	72.0 (+1.6)	71.9	74.7 (+2.8)	74.5	*75.0* (+0.6)	73.1	74.7 (+1.6)
WS-353 (Rel.)	$+$	21.5	63.5	59.8 (-3.7)	65.9	**72.4** (+6.5)	69.5	61.4 (-8.1)	67.0	64.7 (-2.3)
	average(d)	35.8	63.4	58.7 (-4.7)	65.9	67.8 (+1.9)	*69.5*	61.4 (-8.0)	67.0	69.1 (+2.1)
	$\times$	8.8	63.4	3.4 (-60.0)	65.9	45.4 (-20.5)	*69.5*	4.9 (-64.5)	67.0	47.6 (-19.4)
	max(d)	13.8	63.4	64.5 (+1.1)	65.9	69.0 (+3.1)	69.5	*70.3* (+0.8)	67.0	68.5 (+1.5)
WS-353 (Sim.)	$+$	24.9	80.3	79.5 (-0.8)	81.1	*82.4* (+1.3)	78.7	75.2 (-3.6)	80.6	79.2 (-1.4)
	average(d)	48.3	80.3	78.3 (-2.0)	81.1	81.3 (+0.2)	78.7	75.3 (-3.4)	80.6	**84.1** (+3.5)
	$\times$	10.0	80.3	21.3 (-59.0)	*81.1*	69.9 (-11.2)	78.7	22.3 (-56.4)	80.6	72.0 (-8.7)
	max(d)	19.0	80.3	*82.6* (+2.3)	81.1	82.3 (+1.3)	78.7	79.1 (+0.4)	80.6	81.7 (+1.0)
SimLex	$+$	18.5	44.0	51.4 (+7.4)	40.2	50.1 (+9.8)	37.3	46.5 (+9.1)	66.2	**67.0** (+0.8)
	average(d)	18.3	44.0	37.5 (-6.5)	40.2	45.0 (+4.8)	37.3	35.5 (-1.8)	*66.2*	62.1 (-4.1)
	$\times$	-0.8	44.0	7.8 (-36.2)	40.2	33.7 (-6.6)	37.3	6.2 (-31.1)	*66.2*	49.1 (-17.1)
	max(d)	8.6	44.0	48.2 (+4.2)	40.2	43.0 (+2.7)	37.3	41.1 (+3.8)	*66.2*	66.0 (-0.2)
SimVerb	$+$	12.0	34.2	39.3 (+5.2)	25.4	34.2 (+8.7)	23.0	34.1 (+11.1)	56.8	**58.3** (+1.5)
	average(d)	15.5	34.2	30.7 (-3.4)	25.4	32.2 (+6.8)	23.0	29.2 (+6.2)	*56.8*	55.7 (-1.1)
	$\times$	2.4	34.2	6.3 (-27.9)	25.4	26.4 (+1.0)	23.0	10.0 (-13.0)	*56.8*	48.6 (-8.2)
	max(d)	5.2	34.2	36.7 (+2.5)	25.4	28.9 (+3.5)	23.0	27.5 (+4.5)	56.8	*57.6* (+0.8)
RG-65	$+$	36.6	76.1	78.1 (+2.0)	76.8	**83.7** (+6.9)	79.4	75.5 (-3.9)	73.6	78.1 (+4.5)
	average(d)	63.5	76.1	81.9 (+5.8)	76.8	*83.2* (+6.4)	79.4	79.7 (+0.4)	73.6	82.4 (+8.8)
	$\times$	-3.3	76.1	14.1 (-62.0)	76.8	65.5 (-11.4)	*79.4*	26.9 (-52.5)	73.6	68.4 (-5.1)
	max(d)	20.7	76.1	77.9 (+1.8)	76.8	81.1 (+4.3)	79.4	*83.7* (+4.4)	73.6	77.7 (+4.2)
MC-30	$+$	39.9	83.8	81.5 (-2.3)	82.6	**89.9** (+7.2)	85.5	74.3 (-11.1)	74.1	81.0 (+6.9)
	average(d)	55.2	83.8	78.3 (-5.5)	82.6	79.3 (-3.3)	*85.5*	75.6 (-9.8)	74.1	79.5 (+5.3)
	$\times$	-19.4	83.8	23.2 (-60.6)	82.6	65.2 (-17.4)	*85.5*	18.3 (-67.1)	74.1	73.2 (-0.9)
	max(d)	25.4	83.8	86.4 (+2.6)	82.6	84.9 (+2.2)	85.5	*87.6* (+2.2)	74.1	78.2 (+4.1)
Rare Word	$+$	8.3	58.0	56.5 (-1.5)	52.5	57.4 (+4.9)	54.3	54.0 (-0.2)	63.5	**64.1** (+0.6)
	average(d)	5.2	58.2	42.4 (-15.8)	52.5	50.1 (-2.4)	54.3	43.5 (-10.8)	*63.5*	58.9 (-4.6)
	$\times$	0.2	57.7	17.0 (-40.7)	52.5	50.5 (-2.0)	54.3	20.7 (-33.5)	*63.5*	57.4 (-6.1)
	max(d)	-1.4	58.1	56.9 (-1.2)	52.5	54.1 (+1.6)	54.3	53.6 (-0.6)	*63.4*	63.2 (-0.2)
YP-130	$+$	16.2	49.8	66.8 (+17.0)	51.9	62.2 (+10.3)	49.9	60.2 (+10.3)	65.5	**69.7** (+4.2)
	average(d)	12.4	49.8	58.6 (+8.7)	51.9	62.3 (+10.5)	49.9	53.6 (+3.7)	65.5	*68.2* (+2.7)
	$\times$	5.1	49.8	29.7 (-20.1)	51.9	39.5 (-12.3)	49.9	15.8 (-34.1)	*65.5*	62.6 (-2.9)
	max(d)	12.7	49.8	54.1 (+4.2)	51.9	52.1 (+0.3)	49.9	55.3 (+5.4)	65.5	*66.8* (+1.3)
Verb-143	$+$	3.7	49.7	41.3 (-8.5)	35.8	28.2 (-7.5)	38.9	29.5 (-9.4)	*58.2*	51.8 (-6.4)
	average(d)	9.2	49.7	15.9 (-33.8)	35.8	18.9 (-16.9)	38.9	22.0 (-16.8)	*58.2*	42.7 (-15.5)
	$\times$	-1.7	49.7	21.0 (-28.7)	35.8	18.0 (-17.8)	38.9	9.9 (-29.0)	*58.2*	44.3 (-13.9)
	max(d)	9.7	49.7	48.7 (-1.0)	35.8	41.9 (+6.1)	38.9	38.0 (-0.9)	58.2	**59.5** (+1.3)
MTurk-3k	$+$	21.0	72.9	69.7 (-3.2)	*80.0*	79.1 (-0.9)	75.8	68.7 (-7.1)	75.9	74.8 (-1.1)
	average(d)	38.8	72.7	66.9 (-5.9)	*80.0*	79.2 (-0.8)	75.8	70.9 (-4.9)	75.9	79.2 (+3.4)
	$\times$	1.9	73.1	15.2 (-57.9)	*80.0*	67.4 (-12.6)	75.8	23.8 (-52.0)	75.9	54.7 (-21.2)
	max(d)	16.0	73.0	74.3 (+1.3)	80.0	**81.3** (+1.4)	75.8	77.6 (+1.8)	75.9	78.1 (+2.3)
MTurk-287	$+$	11.6	66.5	67.3 (+0.8)	72.3	*73.7* (+1.4)	68.2	59.7 (-8.5)	67.0	66.2 (-0.7)
	average(d)	39.6	67.2	67.6 (+0.4)	72.3	**74.4** (+2.1)	68.2	70.3 (+2.1)	67.0	73.5 (+6.5)
	$\times$	8.3	66.5	5.8 (-60.6)	*72.3*	60.6 (-11.7)	68.2	26.4 (-41.8)	67.0	57.7 (-9.3)
	max(d)	13.7	65.8	68.4 (+2.6)	*72.3*	71.2 (-1.2)	68.2	67.0 (-1.2)	67.0	69.2 (+2.2)
MTurk-771	$+$	26.0	66.8	68.8 (+1.9)	70.9	*72.6* (+1.8)	66.9	65.4 (-1.5)	70.1	71.6 (+1.4)
	average(d)	40.8	66.8	62.1 (-4.8)	70.9	70.5 (-0.4)	66.9	62.2 (-4.6)	70.1	**75.3** (+5.2)
	$\times$	2.3	66.8	13.7 (-53.1)	*70.9*	52.9 (-18.0)	66.9	20.3 (-46.6)	70.1	55.5 (-14.6)
	max(d)	16.9	66.8	70.5 (+3.6)	70.9	72.4 (+1.5)	66.9	69.5 (+2.6)	70.1	*72.6* (+2.5)

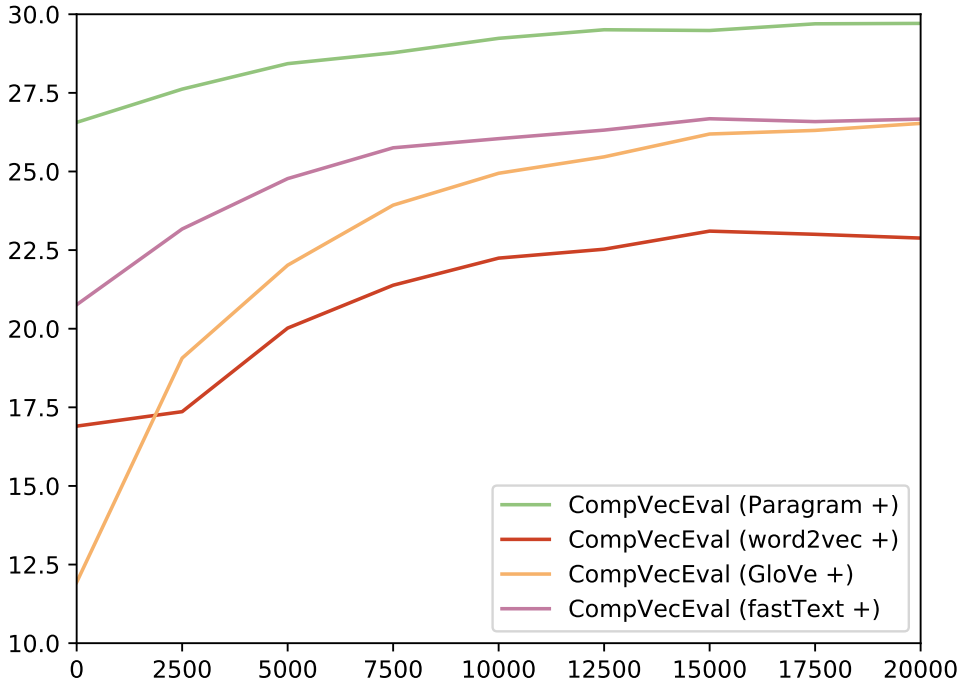


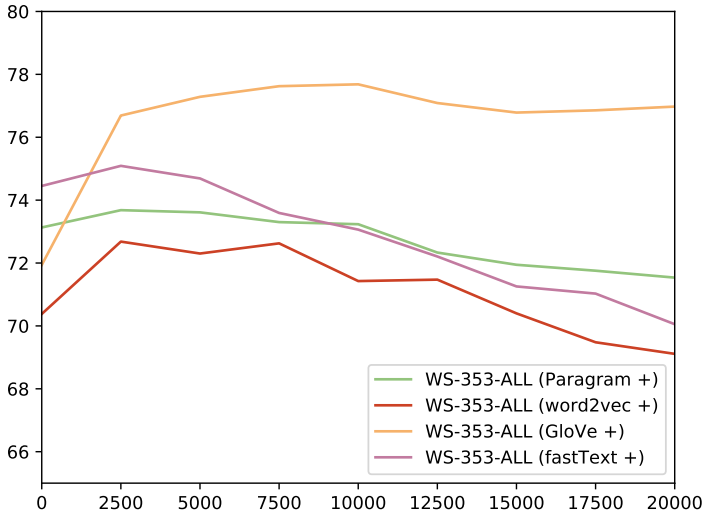
FIGURE 4.5: This figure shows the progression of CompVecEval score during tuning for word2vec, fastText, Paragram and Glove with additive composition.

In Figure 4.6 we see the progression of Spearman's *r* correlation coefficient while testing at different times during the training procedure. Specifically, on the training procedure with additive composition.

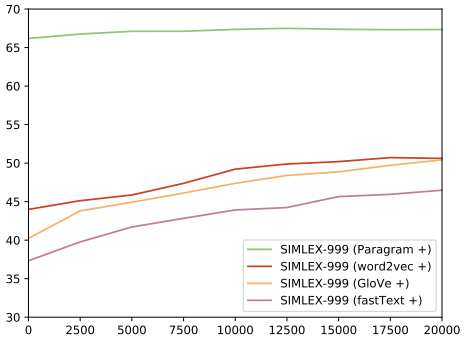
Both SimLex and SimVerb show a nice upward trend with all different initialization functions. Even for the initial top performer: Paragram, even though the improvement is minor. We zoomed into the data for the tuned Paragram embeddings in Figures 4.6c and 4.6e. It has to be mentioned that some of the Paragram embeddings [134] were tuned on the SimLex [58] dataset. So naturally, they will score high on SimLex word similarity. This might be why we do not see a big improvement. What is remarkable is that we are able to improve at all, even when the embeddings are already tuned on the SimLex dataset.

Interestingly, we do not see the same behavior across all the different datasets. Most of the are able to improve by tuning under additive composition, however not all of them. For example, for tuning the Paragram embeddings 7 out of the 11 are able to show improvement. However, when we look at the GloVe embeddings this is 9 out of the 11. Figure 4.6a shows this trend for WS-353 specifically. This is in-line with the large CompVecEval improvement we were able to make on the GloVe embeddings using tuning for additive composition.

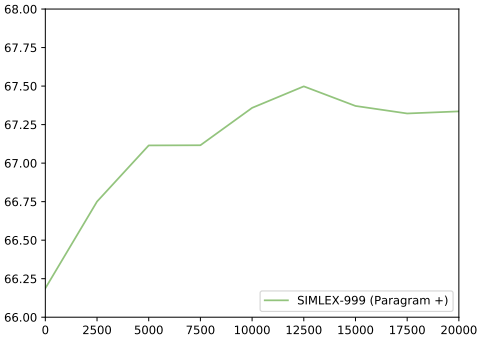
When looking at WS-353 scores for the first 2500 iterations, we see an upward trend for all pretrained models. After which it is downhill for all of them, except GloVe.



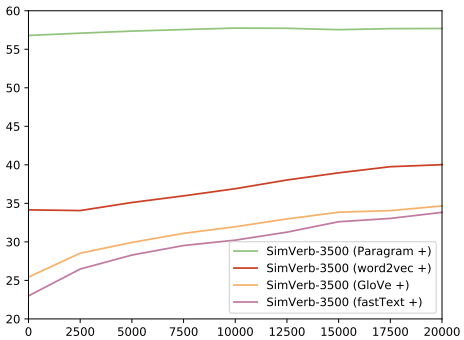
(A) WS-353 +



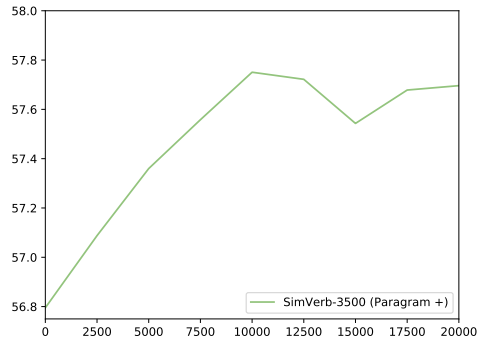
(B) SimLex +



(C) SimLex Paragram +



(D) SimVerb +



(E) SimVerb Paragram +

FIGURE 4.6: This figure shows the progression of word representation evaluations scores during tuning for word2vec, fastText, Paragram and Glove with additive composition. We display Spearman's $r \times 100$.

These kinds of patterns were seen across several other tests. This could indicate that the embeddings are over fitting on the specific task, and is losing general performance. Interestingly, we see different turning points for different evaluation methods, and this begs questions such as: "When should we stop training exactly so as not to lose general performance?" and "When we employ a method such as early stopping using a validation set, should we do this using the task specific test set or using a general evaluation method?". These questions are difficult to answer since there is no *one* evaluation method for general performance.

The difference in word representation scores, before and after training, are statistically significant most additive composition experiments (see Appendix A.5). MC-30 has difficulty attaining statistical significance for Word2Vec and Paragram, which could be due to the low number of word pairs. Furthermore, on Word2Vec we were not able to find statistical significant results for WS-353 and MTurk-287. All other results on experiments with additive composition are statistically significant.

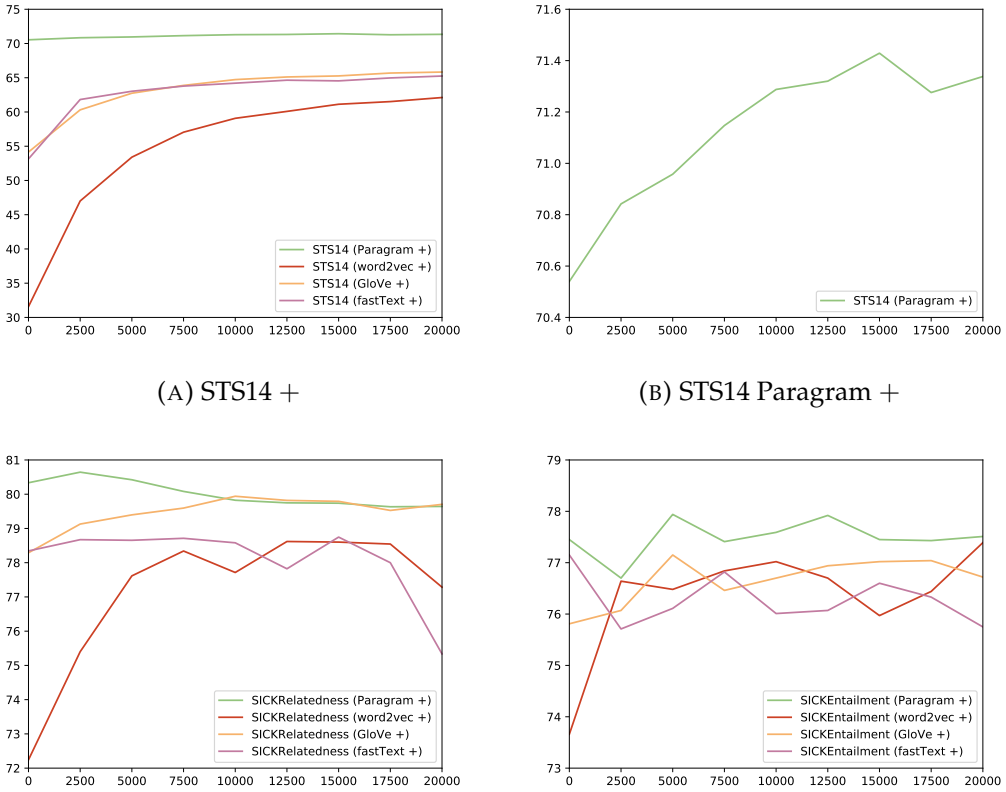
4.5.3 Results on Sentence Representation Evaluations

Now we turn to the sentence embeddings, here we are not only testing the embedding quality itself but also in relation with its corresponding composition function.

Table 4.8 shows the result of eleven different sentence embedding evaluations. Here we only show STS14 and STS15 for the STS metrics. For the results on the subsets specifically, and for STS12, 13, 16 and STS-Benchmark we refer to Tables A.4 and A.5 in the appendix.

When we look at the results for MRPC we manage to improve on the results from Wieting et al. [135] with their Paragram embeddings. Which is interesting because their embeddings are specifically trained to compose into paraphrasing sentence embeddings, and the MRPC is a paraphrasing evaluation metric. It must be stated that start and end scores for MRPC with additive composition are *not* statistically different for Word2Vec, GloVe and fastText (see Appendix A.5). Therefore, we will ignore MRPC in further comparisons.

When looking at other classification tasks: TREC, MR, CR, SUBJ, MPQA and SST we see that all embeddings (tuned and untuned) have roughly similar performance when composed using averaging or summation. We do not see a big difference anywhere. Therefore, we can *not* always observe a significantly different result for additive composition. Specifically, for TREC and the Stanford datasets (see Appendix A.5). Additionally, when we compare pretrained embeddings with the embeddings trained from random initialization, we see a difference but not a very big one. This could be due to the fact that these classification methods are really dependent on specific words (with specific representations) to be present in the sentence, and that all embeddings are already very good at capturing these words. Also, we could have limited our results by using the linear regression classifier.



(A) STS14 +

(B) STS14 Paragram +

(C) SICK Relatedness +

(D) SICK Entailment +

FIGURE 4.7: This figure shows the progression of sentence representation evaluations scores during tuning for word2vec, fastText, Paragram and Glove with additive composition. We display Pearsons's $r \times 100$ for STS14 and SICK-R and accuracy $\times 100$ for SICK-E.

We see that tuning for composition has a very big impact in the STS metrics for Word2Vec, GloVe and fastText, as can be seen in Figure 4.7a. The improvements for the Paragram embeddings are also there but less apparent (Figure 4.7b). These improvements still lead to the best result because the Paragram embeddings already score high on the STS metrics to begin with.

It is interesting that tuning for composition improves the STS metrics significantly for the first three embeddings, even though that the task of composing definitions into short lemmas is not directly related to sentence representations. When the original STS14 challenge was held the contestants trained models on STS data specifically and the 75th percentile of the score was 71.4 which means our best score is better or comparable to 75 % of contestants while we did not use task specific training data.

Keep in mind that we are also using cosine similarity to compare two sentence representations for the STS metrics. When comparing word representations, it seemed not to matter whether we were optimizing for composition through averaging or for additive composition. But when looking at the STS results it seems it does. STS results plummet when the model is optimizing for averaging, but when optimizing for summation it does not. Because averaging normalizes the embeddings, it could be difficult for the model to set a correct magnitude for each embedding while tuning.

This could explain the discrepancy between averaging and summation on the STS results.

All results on the STS14 and 15 tasks with additive composition are statistically significant (see Appendix A.5).

For the SICK Relatedness and Entailment tests we see comparable results for most composition functions, and are not able to improve on them. This is reflected in figures 4.7d and 4.7c. Interestingly we still see an improvement for the first 2500 steps for SICK relatedness, as we did for WS-353. The way we evaluate SICK data points is quite involved, by following the method from Tai, Socher, and Manning [126] we train a neural network to determine relatedness, and the concatenated representation for entailment, the evaluation models themselves could have a big impact on the SICK performance. For example, when we look at the embeddings trained from random initialization they already have very good SICK-R and SICK-E scores compared to for example the Paragram embeddings.

As mentioned, we also ran a experiments on Information Retrieval tasks with untuned and tuned Paragram embeddings for additive composition. Specifically, on the Robust04 and WSJ document retrieval datasets. We see an improvement in performance when using our tuned embeddings. See Appendix A.4 for more details.

4.5.4 Impact of randomly initialized embeddings

As mentioned in Section 3.3.3, some embeddings were randomly initialized. Specifically, if a word from our WordNet vocabulary did not have a corresponding embedding in the pretrained dataset. We want to use a vocabulary which is not biased towards the four word embeddings we use, this is why we choose to include these randomly initialized embeddings. One could see the random initialized embeddings as penalization of the pretrained embeddings for not including the embedding for a specific dictionary word.

This vocabulary consideration has also some effect on the tuning. For example, on the surface one could think the gains in overall performance on the different metrics were simply due to tuning of the randomly initialized embeddings. Luckily, we can rebut this by looking at the vocabulary overlap information in Table C.2 in the appendix. We see that the vocabulary of the various evaluation methods has a negligible amount of words which are randomly initialized. Specifically for Glove this is the case. Interestingly, GloVe is also one of the pretrained embeddings which had the most to gain from tuning for composition.

4.5.5 Qualitative results on embedding magnitude

When composing using additive composition one could argue that the magnitude of a single word embedding can be seen as its semantic significance to the resulting composed embedding. After all, an embedding with relatively a small magnitude

TABLE 4.8: **Sentence representation evaluation** scores for STS and SICK-R in Pearson's $r \times 100$, all other scores in accuracies $\times 100$. These evaluations are done for before and after tuning the embeddings for a specific composition operation. We also trained embeddings without pretraining (*None*). Results for other STS datasets can be found in Appendix A.3.

		None	Word2Vec		GloVe		fastText		Paragram	
		Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
TREC	$+$	73.6	75.1	78.6 (+3.5)	81.2	81.4 (+0.2)	79.5	79.2 (-0.3)	81.0	*82.0* (+1.0)
	average(d)	79.0	79.0	81.4 (+2.4)	82.6	**83.0** (+0.4)	82.2	82.8 (+0.6)	82.8	82.0 (-0.8)
	$\times$	25.6	25.2	25.0 (-0.2)	19.8	16.8 (-3.0)	*37.6*	27.2 (-10.4)	35.6	34.4 (-1.2)
	max(d)	52.2	66.8	64.0 (-2.8)	76.2	66.0 (-10.2)	71.8	66.6 (-5.2)	*82.6*	71.0 (-11.6)
MRPC	$+$	67.2	68.7	69.0 (+0.3)	69.6	69.2 (-0.4)	68.4	69.3 (+1.0)	69.9	*71.9* (+2.0)
	average(d)	69.3	71.8	71.1 (-0.8)	72.5	72.1 (-0.4)	71.0	72.1 (+1.2)	72.5	**72.8** (+0.3)
	$\times$	*67.2*	66.5	66.5 (-0.0)	66.0	65.6 (-0.5)	66.5	66.5 (-0.0)	66.3	66.6 (+0.3)
	max(d)	66.8	69.7	70.4 (+0.7)	*71.7*	68.6 (-3.0)	67.9	69.2 (+1.2)	70.5	69.3 (-1.2)
MR	$+$	63.4	75.3	74.4 (-0.9)	**76.2**	75.7 (-0.5)	74.0	73.5 (-0.5)	74.2	74.5 (+0.3)
	average(d)	65.9	75.4	73.9 (-1.5)	*76.0*	75.4 (-0.6)	73.9	72.8 (-1.1)	74.5	74.2 (-0.2)
	$\times$	49.5	50.1	50.3 (+0.2)	50.9	50.6 (-0.3)	50.9	*51.9* (+1.0)	49.2	50.2 (+1.0)
	max(d)	60.9	66.2	70.4 (+4.1)	72.8	*72.9* (+0.0)	65.6	68.9 (+3.3)	69.4	70.4 (+0.9)
CR	$+$	71.4	77.2	76.7 (-0.5)	*78.9*	78.7 (-0.2)	76.9	75.9 (-1.0)	78.0	78.6 (+0.6)
	average(d)	73.6	77.9	77.1 (-0.8)	**79.5**	78.9 (-0.6)	77.5	77.4 (-0.1)	78.5	78.4 (-0.1)
	$\times$	62.9	63.7	*63.8* (+0.0)	52.2	45.5 (-6.7)	*63.8*	63.7 (-0.0)	63.7	63.5 (-0.2)
	max(d)	67.5	70.9	74.5 (+3.5)	75.2	75.5 (+0.3)	72.2	73.7 (+1.5)	*75.9*	74.5 (-1.4)
SUBJ	$+$	81.1	89.6	89.1 (-0.5)	*90.4*	90.0 (-0.4)	89.8	88.8 (-1.0)	88.1	87.9 (-0.2)
	average(d)	83.8	89.8	88.7 (-1.0)	**90.8**	90.1 (-0.7)	90.2	89.3 (-0.9)	88.5	89.0 (+0.5)
	$\times$	51.0	50.1	50.1 (+0.0)	49.8	49.1 (-0.7)	50.1	*52.0* (+1.9)	49.9	50.1 (+0.2)
	max(d)	72.2	78.7	84.4 (+5.8)	*87.0*	85.8 (-1.2)	83.7	85.3 (+1.6)	83.3	82.6 (-0.7)
MPQA	$+$	71.9	*86.2*	85.1 (-1.2)	86.0	85.9 (-0.1)	85.6	85.4 (-0.2)	86.0	86.1 (+0.0)
	average(d)	75.4	86.8	85.2 (-1.6)	86.5	85.5 (-1.0)	86.0	85.0 (-1.0)	**86.9**	86.0 (-0.9)
	$\times$	71.3	76.7	71.7 (-5.0)	68.1	68.8 (+0.7)	*77.6*	73.2 (-4.4)	76.7	76.0 (-0.7)
	max(d)	73.4	84.9	84.0 (-0.9)	85.3	85.4 (+0.1)	84.0	83.6 (-0.4)	85.3	*85.4* (+0.1)
SST	$+$	62.9	79.3	77.8 (-1.5)	**79.4**	79.2 (-0.2)	77.2	76.6 (-0.6)	79.1	78.8 (-0.3)
	average(d)	65.7	78.1	76.3 (-1.9)	*78.4*	77.4 (-0.9)	76.5	76.0 (-0.5)	76.7	76.4 (-0.3)
	$\times$	50.1	50.2	50.1 (-0.2)	50.3	49.5 (-0.8)	*50.5*	50.1 (-0.3)	50.3	*50.5* (+0.2)
	max(d)	64.0	68.4	72.6 (+4.2)	*76.0*	74.6 (-1.4)	68.3	70.6 (+2.3)	74.1	75.4 (+1.3)
STS14	$+$	64.9	31.9	61.8 (+29.9)	54.1	65.7 (+11.5)	53.0	65.2 (+12.2)	70.5	**71.1** (+0.5)
	average(d)	40.0	32.0	41.5 (+9.6)	54.1	48.0 (-6.1)	53.2	46.3 (-6.9)	*70.5*	49.9 (-20.6)
	$\times$	0.9	4.6	1.9 (-2.7)	5.9	7.4 (+1.5)	8.6	*21.2* (+12.6)	1.6	4.7 (+3.1)
	max(d)	57.8	22.4	62.4 (+39.9)	60.6	*67.1* (+6.5)	42.6	65.1 (+22.5)	61.6	66.3 (+4.7)
STS15	$+$	67.4	36.9	68.7 (+31.9)	58.1	66.6 (+8.5)	58.1	68.8 (+10.7)	75.0	**75.2** (+0.2)
	average(d)	45.5	36.3	47.3 (+10.9)	58.1	51.3 (-6.8)	58.2	52.2 (-6.0)	*75.0*	54.6 (-20.3)
	$\times$	3.3	1.6	2.9 (+1.3)	14.3	*14.7* (+0.4)	3.5	14.1 (+10.6)	2.0	5.2 (+3.1)
	max(d)	55.9	25.9	62.8 (+36.9)	64.3	*66.8* (+2.5)	47.5	65.2 (+17.7)	65.5	66.7 (+1.1)
SICK-E	$+$	74.2	73.8	77.3 (+3.5)	75.8	76.7 (+0.9)	76.8	76.1 (-0.7)	77.5	*78.0* (+0.6)
	average(d)	76.0	77.1	76.9 (-0.2)	79.0	78.2 (-0.8)	78.3	76.6 (-1.7)	**81.1**	78.9 (-2.1)
	$\times$	*58.3*	56.7	56.7 (-0.0)	28.2	56.7 (+28.5)	56.7	56.7 (+0.0)	56.7	56.9 (+0.2)
	max(d)	72.7	76.6	*80.0* (+3.3)	78.1	79.2 (+1.1)	79.8	78.1 (-1.6)	79.1	77.9 (-1.2)
SICK-R	$+$	72.3	73.4	78.4 (+5.0)	78.3	80.0 (+1.7)	78.2	78.0 (-0.2)	*80.3*	79.9 (-0.4)
	average(d)	69.9	71.4	72.2 (+0.8)	79.8	75.5 (-4.3)	79.1	74.0 (-5.0)	**81.5**	77.0 (-4.5)
	$\times$	20.1	5.1	1.4 (-3.7)	19.2	25.3 (+6.1)	11.6	*26.1* (+14.5)	16.6	19.2 (+2.6)
	max(d)	68.6	75.2	78.8 (+3.7)	79.8	*80.2* (+0.4)	78.7	78.1 (-0.6)	78.7	79.0 (+0.3)

TABLE 4.9: This table shows a selection of words from the tuned GloVe embeddings and their magnitude. These words are selected from top fifties.

Highest		Lowest		Fallers		Risers	
	$\|\|e_{tuned}\|\|$		$\|\|e_{tuned}\|\|$		$\|\|e_{tuned}\| - \|\|e_{pre}\|\|$		$\|\|e_{tuned}\| - \|\|e_{pre}\|\|$
sporophores	15.88	the	3.23	of	-1.62	repeatedly	+2.69
agarics	14.32	of	3.35	the	-1.47	german	+2.07
phrenitis	13.90	hand	4.53	wood	-1.35	irritate	+1.81
screen-oriented	12.13	versa	4.55	yellow	-1.15	tense	+1.69
oblong-elliptic	11.93	especially	4.71	strawberry	-1.05	unlikely	+1.63

makes a small difference when being composed. Consequently, an embedding with a large magnitude can make a big difference.

In Table 4.9, we see how our tuning procedure for additive composition affects embedding magnitudes, i.e. norms. You would expect that words that have a small impact on semantics have a small magnitude, if we look at for example 'the' and 'of' we can see that this is indeed the case. Furthermore, our training procedure makes sure these embeddings have an even smaller impact as they are among the greatest fallers. Interestingly, we see that some descriptive words fall w.r.t. their original embedding magnitude, specifically the colors. Not only yellow, but almost all of them are among the greatest fallers. By contrast, other words which could have a more ambiguous descriptive effect, e.g. 'german' and 'irritate', are among the greatest risers. As expected, words with a very specific and unambiguous lexical semantics are among the ones with the highest magnitudes, e.g. 'oblong-elliptic'.

4.6 Conclusion

In this chapter, we introduced a method to tune embeddings for algebraic composition. Additionally, we introduced different methods to test the general performance and applicability of embeddings.

While it is difficult to interpret general performance based on 29 different metrics, we can confidently say that our tuning method improves performance in the task of composing lexicographic definitions for additive composition. Additionally, these tuned embeddings do not suffer greatly in their applicability for different tasks, moreover they are even better for when looking at word similarity (SimLex and SimVerb) or at sentence similarity (STS14).

In the next chapter, we will look at more elaborate composition functions, some of these learnable composition functions are not commutative and can therefore take order into account.

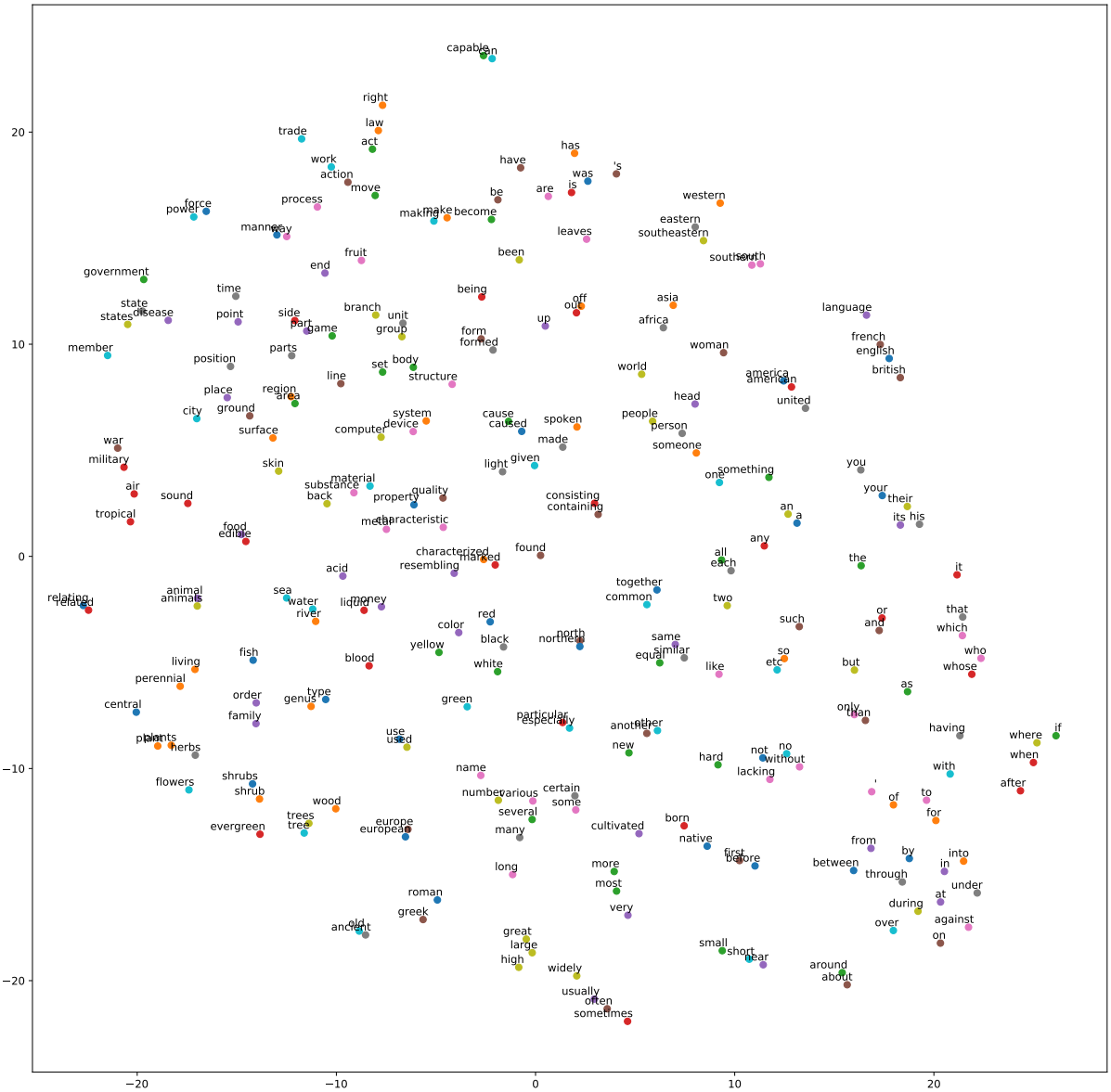


FIGURE 4.8: This figure shows t-SNE [82] projection of some selected words after the Paragram embeddings are tuned for additive composition.

Chapter 5

Neural models for composition

In the last chapter, we tuned our embeddings for algebraic composition functions. These algebraic composition functions do not have parameters that need to be learned themselves. In this chapter, we turn to more elaborate, learnable composition functions. All our learnable composition functions are a specific type of artificial neural network, hence our chapter title: *'Neural models for composition'*.

We will introduce six different learnable composition functions in this chapter. The first being a simple projection of the composed vector after summation, i.e. a single layer perceptron. Then we turn to three different recurrent composition functions, which are all based on the idea of an RNN [60]. Lastly, we explore two different convolutional composition functions, these use a CNN [76]. Our hypothesis is that these functions are better able to compose vectors, since the functions have more flexibility.

We evaluate our models using the same methods described in the previous chapter. Instead of comparing tuned embeddings with untuned embeddings used for direct composition, here we compare a learned composition functions with fixed embeddings with the same learned composition functions which were able to tune embeddings during training.

The rest of our training procedure is exactly the same as described in Chapter 4. We use multiple target tokens and the Euclidean distance loss function. Training happens in 20.000 batches of 512 training examples.

Contributions presented in this chapter:

Literature An overview of learnable perceptron, recurrent and convolutional composition architectures;

Technical implementations of six different variants of learnable *neural* composition functions on our CompVec architecture; and

Scientific the experimental results from tuning the four popular word embedding algorithms using the six different *neural* composition function through all twenty-nine evaluation methods.

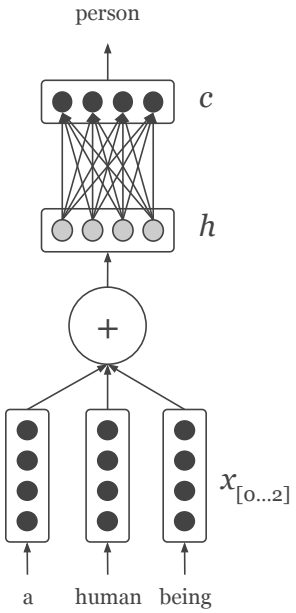


FIGURE 5.1: This figure shows how the learned projection layer is appended to the model.

5.1 Projecting algebraic compositions

Our first and simplest learned composition function was inspired by the work of Wieting et al. [135] who also introduced the Paragram embeddings. It involves projecting the summed embeddings through a single layer neural network. This single layer has 300 hidden neurons, which is similar to the dimensionality of the embedding.

In Equation 5.1, the exact function is shown. During the training procedure we will learn the weight matrix W_h, aside from tuning the embedding matrix.

$$h = \sum_{x_i \in X} x_i c = \tanh(hW_h + b) \tag{5.1}$$

Figure 5.1 illustrates the equation more visually.

It has to be noted that this specific composition function is still commutative, and thus cannot take order into account. It *does* have a nonlinear *tanh* function, which makes the composition function more expressive.

Wieting et al. [135] shows that his projected vectors are comparable to the algebraic average he uses. One important thing we should mention is that the *tanh* limits the magnitude of the output representation. This is because its upper limit for each individual output scalar goes to 1, and is therefore bound to ≈ 17.32 for representation with a dimensionality of 300.

If our results from the previous chapter are any indication of the importance of the magnitude of c, we should not expect the best performance from this composition function because the magnitude is bound by *tanh*.

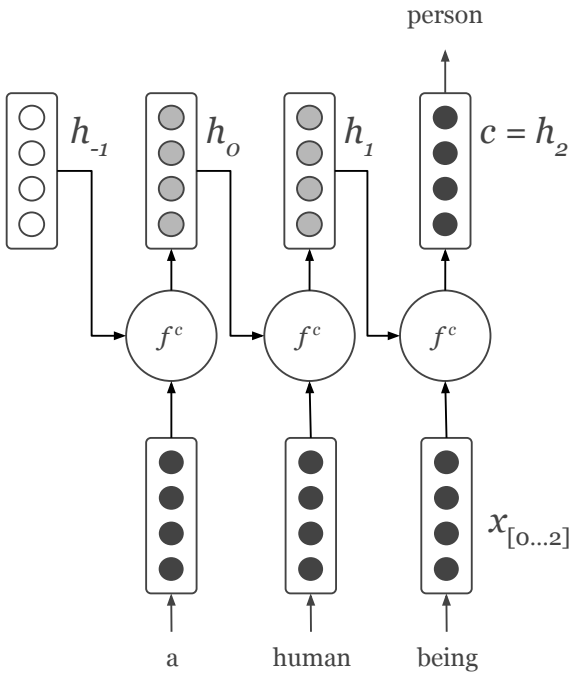


FIGURE 5.2: This figure shows the inner workings of a RNN for composing word embeddings. We see how in each time step the RNN function is fed with a new word embedding and the previous hidden state.

5.2 Recurrent Composition

Recurrent Neural Networks (RNNs) are a particular kind of neural network, which is able to model sequences of data. As discussed in Chapter 2, it has shown great promise in many NLP tasks.

In a traditional neural network, we assume that all inputs and outputs are independent of each other. For many tasks, this is not the right way to think about the. If one wants to predict the next word in a sentence, you should really know which word came before it.

The most important attribute a RNN is able to take into account which other models were not, is word order. With an RNN we are able to model the semantic difference between for example: *"I have to read this book."* and *"I have this book to read"*. We already discussed some recurrent models used for direct and indirect composition in the background chapter (Section 2.4.3).

We will use three different types of recurrent neural networks. First, we start with a traditional RNN as described by [60]. Next, we turn to the Gated Recurrent Unit (GRU) [27] as to combat a problem which is often referred to as the Vanishing Gradient Problem [13]. Lastly, we try to model a sequence of words in both directions with the Bi-directional GRU [112].

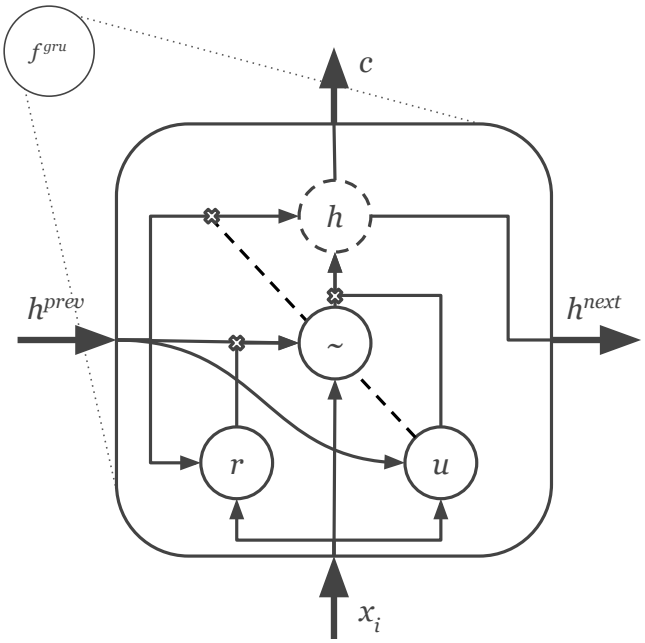


FIGURE 5.3: This figure shows the inner workings of a GRU. We see the update and reset gate with their input and output. We see $\tilde{h}$ and how it influences the final output c or h.

5.2.1 Vanilla RNN

Our first recurrent model is something called a vanilla or plain RNN. It has one layer of recurrent units which take embeddings as input. Our hidden units have a dimensionality of 300, similar to our word embeddings. The function considers both the output from the previous hidden state and the current state. This is described in the Equation 5.2.

$$h = \tanh(xW_x + h_{t-1}W_h + b) \tag{5.2}$$

Figure 5.2 shows the RNN schematically. We use the last output as our composed vector. In the first time-step, we consider only the input embedding and consider the previous hidden state as being a zero-vector.

When we input a training batch into our composition model utilizing an RNN we make sure to correctly interpret the masked input tokens. This entails checking weather an input token is a padding token, and if so just passing the h state along to the next time step instead of passing it through the recurrent model again.

The model's parameters (W_x) are initialized by drawing samples from a normal distribution with a standard deviation of $\sigma = 0.01$. The biases are initialized at 0.

5.2.2 Gated Recurrent Unit

Our second recurrent model uses a GRU instead of plain RNN units. It has been shown that GRU is well suited for language related tasks, and performs on par with

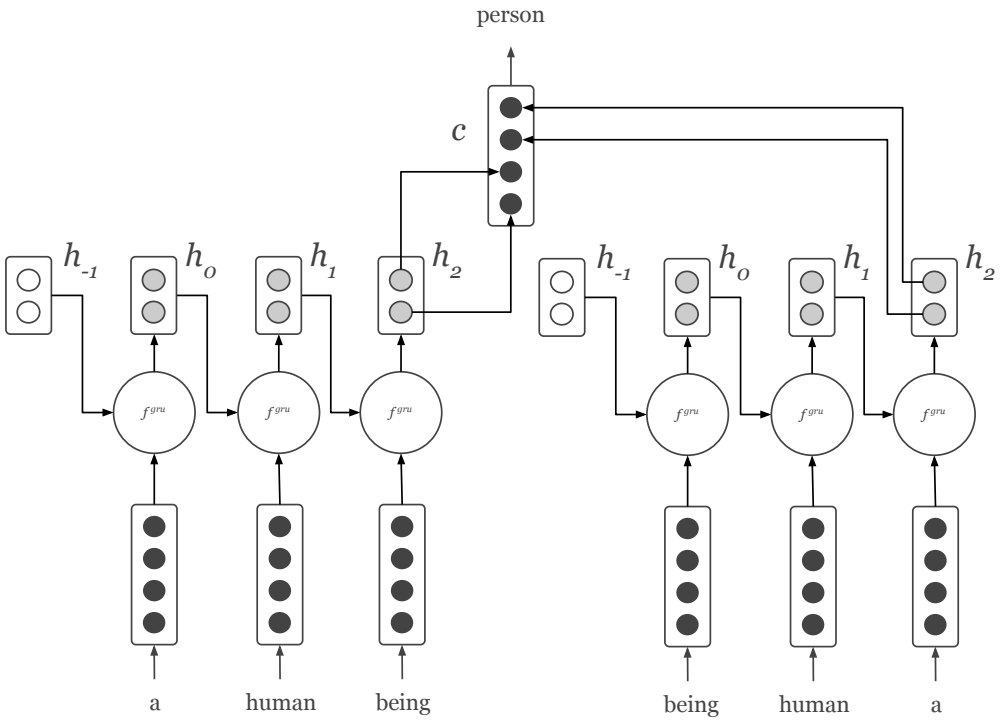


FIGURE 5.4: This figure shows the inner workings of a bi-directional GRU for composing word embeddings. We see how in each time step the GRU function is fed with a new word embedding and the previous hidden state.

or better than the well-known LSTM [26]. Using an LSTM or GRU combats the Vanishing Gradient Problem [13], which basically describes why it is difficult for a vanilla RNN to model long sequences.

$$\begin{aligned} r &= \sigma(xW_{xr} + h_{t-1}W_{hr} + b_r) && \text{reset gate} \\ u &= \sigma(xW_{xu} + h_{t-1}W_{hu} + b_u) && \text{update gate} \\ \tilde{h} &= \tanh(xW_{x\tilde{h}} + r \cdot h_{t-1}W_{h\tilde{h}} + b_{\tilde{h}}) && \text{candidate update} \\ h &= 1.0 - u \cdot h_{t-1} + u \cdot \tilde{h} && \text{final update} \end{aligned} \tag{5.3}$$

This model has three times the parameters of our plain RNN model. Equation 5.3 gives us the mathematical definition of a GRU. It contains three so-called gates: a reset gate, an update gate and a candidate gate. Figure 5.3 illustrates the internals of a GRU, and these gates schematically. The gates allow the model to consider a specific part of the 'memory', i.e. input from the previous time step, and a specific part from the input embedding. With gating the model is able to remember information over longer periods, i.e. more time steps.

Some of the model's parameters (W_{xr}, W_{xu} and $W_{x\tilde{h}}$) are initialized by drawing samples from a normal distribution with a standard deviation of $\sigma = 0.01$. The parameters W_{hr}, W_{hu} and $W_{h\tilde{h}}$ are initialized using an initializer that generates an orthogonal matrix, which is standard practice for GRU models. All the biases are initialized at 0.

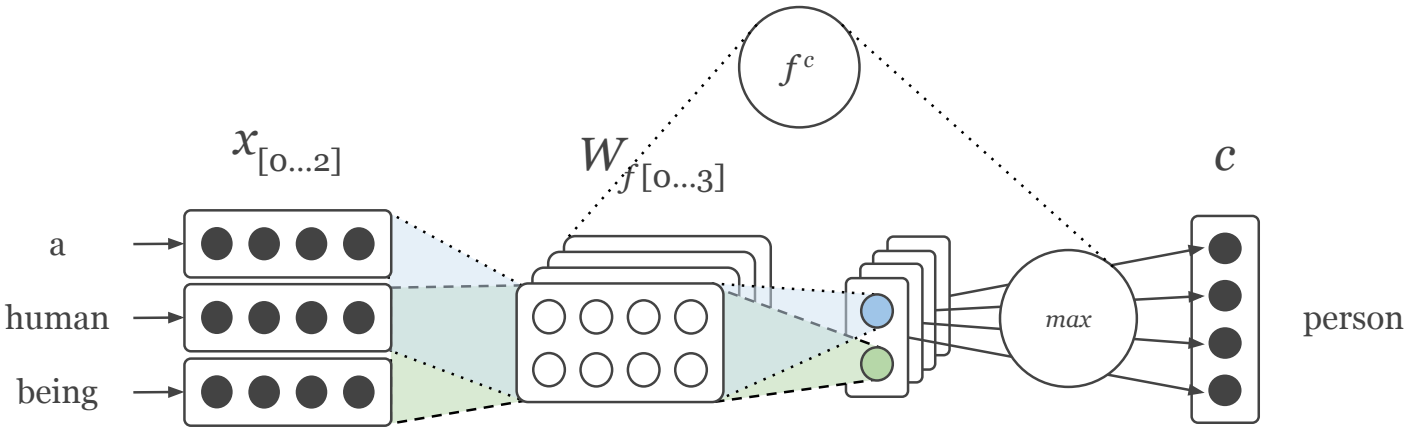


FIGURE 5.5: This figure shows the inner workings of a CNN for composing word embeddings. We see the weight matrix for the filters, the max pooling operation as well as c the final output.

5.2.3 Bi-directional GRU

Our last recurrent model is a *bi-directional recurrent neural network* [112] *using GRU units*. The network has the input sequence in regular order project to a hidden state of 150 units. In addition, we have a separate model with its own parameters doing this for the input sequence in reverse order. We combine the two outputs using concatenation, the entire model is illustrated in Figure 5.4.

Often times a word's meaning can conditioned on a word which come after it, this can already be modeled by a regular GRU using the memory, but a bi-directional GRU can make these relationships more explicit.

5.3 Convolutional Composition

Convolutional Neural Networks (CNNs) are typically used for computer vision. They were the main reason for several major breakthroughs in image classification and are at the center of most computer vision algorithms today, from face detection in photos to self-driving cars [18]. CNNs also have interesting applications in NLP. For example, we discussed some convolutional composition functions in the background chapter (Section 2.4.2).

CNNs for Computer Vision convolve over an image using a collection of square filters. These filters slide over the image to detect features in the image. When applying CNNs to NLP we should *not* think of these filters as traversing an image of pixels, instead we should take the matrix of word embeddings of an entire sentence and traverse this using a sliding window.

Figure 5.5 illustrates how this works in our model. We convolve over the matrix of word embeddings using filters which have the same width as the dimensionality of the embeddings (300 in our case). The height of the filter is also an important attribute of the model. In the figure, we see a height of 2. This height determines how

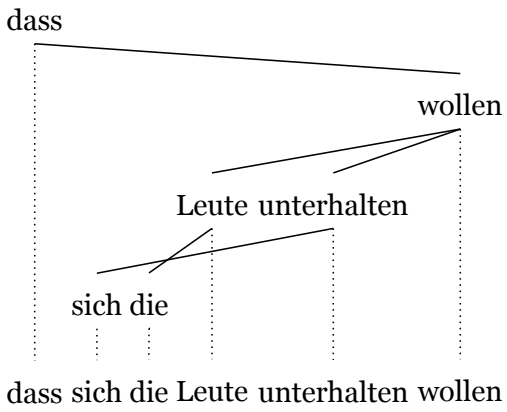


FIGURE 5.6: This figure shows a dependency parse tree for a German sentence with a long distance dependency [33].

many word representations a filter can take into account in one step. Such a step will produce a single scaler, which is a weighted sum according to the filter's weights and the values of the embeddings. Stepping over the entire sentence produces $s_{length} - f_{height} + 1$ results. This we see in Figure 5.5 for two steps, the first producing the blue value and the second producing the green value. These $s_{length} - f_{height} + 1$ values will go through a pooling function to come out as a single scalar. In most popular CNNs this is the max-pool function, which our CNN will use as well. Max-pooling has been criticized for being an operation which leads to the loss of information [107], however it has been shown to be very effective.

In our implementation of a CNN for NLP we append zero vectors in front and at the end of the sentence so we are still able to convolve over sentences that have the same or a lower number of tokens as our filter width.

Recurrent neural networks are able to take word order into account this is also the case for CNNs. However, there is a small caveat. RNNs are able to model long distance dependencies, like in the German sentence: *"Dass sich die Leute unterhalten wollen."*[1] which contains a relationship between 'dass' and 'wollen' which are 6 words apart (See Figure 5.6). Modeling such a relationship correctly would require a CNN with at least a filter with a width of 6. However, in German, which is notorious for its long-distance dependencies, these distances can be over 20 words. This language property does not allow convolution models with small filters to compose sentences effectively.

5.3.1 CNN with single filter width for balanced output

We will explore two different CNN architectures for composition. The first model has only a single filter width for a balanced representation of c. We will use a filter where the dimensions correspond to the embedding size ($E = 300$) and the width of the sliding window ($F = 3$). We ensure that each filter predicts a single dimension of the target embedding. This means we also will have 300 different filters. Again, schematically this is clearly shown in Figure 5.5.

[1] Which translates to: *"That people want to converse."*

TABLE 5.1: Table showing the filter widths and the associated number of filters for our more elaborate CNN.

Filter dimensions	Number of filters
3×300	150
5×300	75
7×300	50
9×300	25

The mathematical definition of such a CNN is as follows:

$$\begin{array}{ll} \tilde{h}_{ij} = \sum_{a=0}^{F} \sum_{b=0}^{E} W_{ab}^{f} x_{(i+a)(j+b)} & \text{convolution} \\ h = \text{ReLU}(\tilde{h} + b) & \text{nonlinearity.} \\ c = \text{maxpool}(h) & \text{pooling} \end{array} \tag{5.4}$$

Here W^f is the weight matrix containing the filter weights. We use a so called rectified linear unit as an alternative activation function, $\text{ReLU}(x) = \begin{cases} x & if\ x > 0 \\ 0 & otherwise \end{cases}$. This activation function *does* have a lower bound, it *does not* have an upper bound.

The model's parameters (W^f) are initialized by drawing samples from a normal distribution with a standard deviation of $\sigma = 0.1$. The biases are initialized at 0. We do not apply strides in our model.

5.3.2 More elaborate CNN width different filter widths

For the second more complex CNN we employ different filters with different sliding window widths. This CNN should be able to better capture long distance dependencies. We choose filters with a sliding window size of: 3, 5, 7 and 9 (See Table 5.1). These differences in filter width will produce a composed representation (c) which could contain dimensions which are in imbalance with one another. When our compositional function is applied at both the source and the target side this should not be a problem, however when only applied at the source side it could be.

The approach with multiple filter widths is not uncommon for convolutional models used for sentiment classification [69].

5.4 Experimental results

In our experiment, we trained the six learnable composition models we just described all two times. We applied transfer learning strategies [116]. The first time, we fixed the embedding weights and only trained the parameters of the composition functions for 20,000 steps. In the result tables these results are displayed under 'Org.' short for 'original embeddings'. The second time, we fixed the embeddings only for the first 2,500 steps after which we also tuned the embeddings for the next 17,500 steps.

TABLE 5.2: **CompVecEval** MRR scores ×100, for our tuned embeddings under a learned composition function. It is important to note that under *Original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

	Word2Vec		GloVe		fastText		Paragram	
	Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
$+$	16.8	23.0 (+6.2)	11.9	26.5 (+14.5)	20.7	26.3 (+5.6)	26.5	*29.9* (+3.3)
average(d)	2.0	2.0 (-0.0)	3.3	4.1 (+0.8)	3.0	3.4 (+0.5)	3.8	*4.1* (+0.3)
$+$ Proj.	9.7	14.3 (+4.6)	17.5	22.7 (+5.2)	16.0	19.1 (+3.1)	20.3	*24.8* (+4.6)
RNN	8.5	7.3 (-1.2)	15.7	14.7 (-0.9)	14.2	12.6 (-1.7)	*16.3*	15.6 (-0.7)
GRU	23.4	20.7 (-2.7)	28.9	28.9 (+0.0)	27.8	26.1 (-1.6)	29.2	*29.8* (+0.6)
Bi-GRU	23.6	20.4 (-3.2)	30.2	30.1 (-0.1)	29.0	26.3 (-2.7)	29.7	**30.3** (+0.5)
CNN-3	11.4	14.8 (+3.3)	21.9	22.6 (+0.7)	22.2	22.4 (+0.2)	*24.0*	23.8 (-0.1)
CNN-3,5,7,9	12.0	14.8 (+2.8)	23.4	23.7 (+0.4)	23.3	22.6 (-0.7)	22.4	*24.2* (+1.8)

The rest of the training procedure is similar to the procedure described in the previous chapter (Chapter 4).

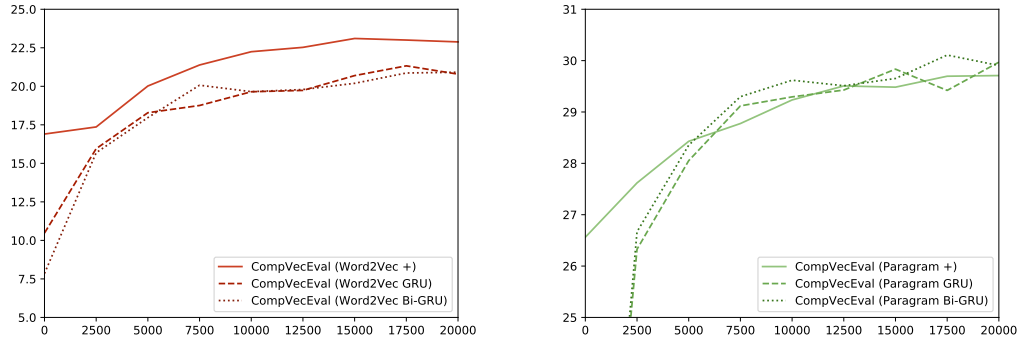
5.4.1 Results on CompVecEval

In Table 5.2, we see the results for the six learned composition functions. They are compared against the algebraic composition functions $+$ and average(d) from the results of the methods described in the previous chapter. We see that, for GloVe and Paragram, the Bi-directional GRU is able to beat additive composition by a small margin. Overall, we see that the models which allow the embeddings to be tuned are slightly better but also only by a small margin.

On the projection function, we have results that are not in line with Wieting et al. [135]. Where they found that using a projection performed comparable to the algebraic composition function we find that for our limited dataset this is not the case. As stated this could be due to the non-linearity (*tanh*) which limits the create an upper bound for the composed representation (c).

For the learned composition functions, we see that the recurrent models outperform the convolutional models. The more elaborate convolutional model with larger window sizes (CNN-3,5,7,9) did outperform the simpler CNN, as well as the RNN. However, it was surpassed by both the GRU models. Both GRU models consistently beat all other learned composition functions. We see that the bi-directional GRU was sometimes better but overall, they perform comparable. This could be due to the limited token length of our definitions. It could be that with longer tokens the addition of the bi-directional GRU could have a larger impact.

In Figure 5.7, we see the progression of the CompVecEval scores during the training of additive composition, the GRU and the bi-directional GRU. The left graph shows the results for Word2Vec in red, and the right graph shows the results for Paragram in green. In this case we show the progression of the learned composition functions which were able to tune the embeddings after 2,500 training steps. For the Word2Vec



(A) CompVecEval Word2Vec (B) CompVecEval Paragram

FIGURE 5.7: This figure shows the progression of CompVecEval score during training and tuning for word2vec; and Paragram with additive, GRU and Bi-GRU composition. We display CompVecEval MRR $\times 100$.

results, we see that the learned composition functions are not able to reach the same performance as the tuned embeddings for additive composition (+). When we turn to the results for the Paragram embeddings, we see that around training step 5,000 the bi-directional GRU has caught up with the additive composition function and is able to pass it. It could be the case that the Paragram embeddings can benefit more from the expressive power of learned composition functions, such as non-linearities and taking order in to account. Where, for example, Word2Vec still has much to gain from a better linear embedding space.

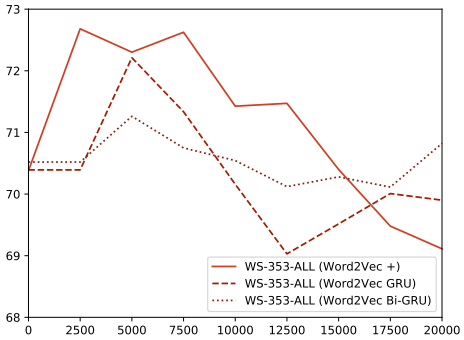
5.4.2 Results on Word Representation Evaluations

In Table 5.3, we present the results on the word representation evaluation methods: WS-353, SimLex and SimVerb. We see results for both models using original embeddings and models allowed to tune embeddings while training. Table B.1 in the appendix shows the results for *all* thirteen word representation methods described in Section 4.3. In the results table we can clearly see that for the models with fixed embedding they are not changed at all, since the word representation scores are all the same[2].

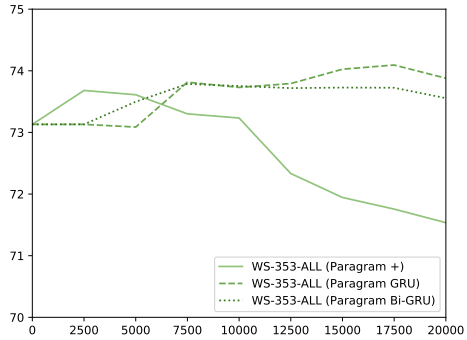
We see that training has an overall positive effect on the word representations themselves. Thinking about this, one could consider it to be counterintuitive. Simply because one could expect the embeddings to be tuned to be used specifically in conjunction with the model (and its parameters), and therefore lose general performance. This does not appear to be the case. If we look at the bi-directional GRU trained for composing the tuned Paragram embeddings, it improves performance on *all* word representation metrics. Not only WS-353, SimLex and SimVerb but also the ten others only reported in the appendix (Table B.1).

Figure 5.8 shows the progression of the scores during training using Word2Vec and Paragram embeddings. Again, we show additive composition (+), the GRU model

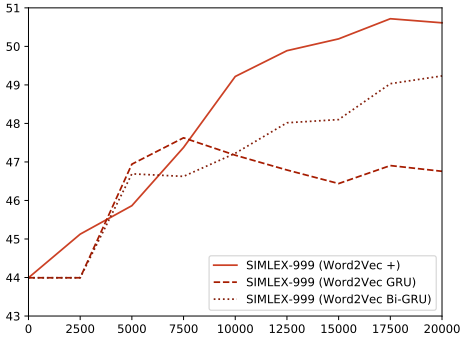
[2]Aside from some small fluxuations due to the random initialization of out of vocabulary words.



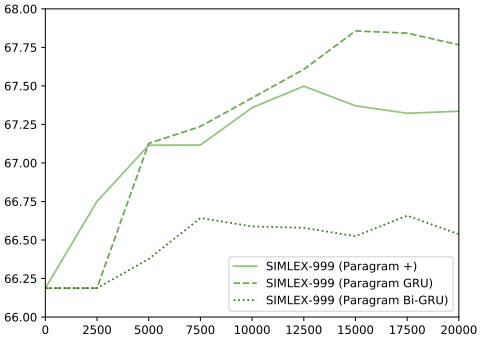
(A) WS-353 Word2Vec



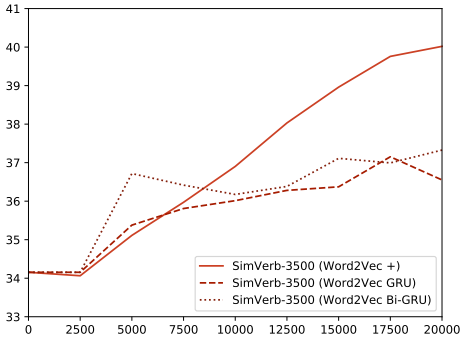
(B) WS-353 Paragram



(C) SimLex Word2Vec



(D) SimLex Paragram



(E) SimVerb Word2Vec

(F) SimVerb Paragram

FIGURE 5.8: This figure shows the progression of word representation evaluations scores during tuning of word2vec, and Paragram with additive, GRU and Bi-GRU composition. We display Spearman's $r \times 100$.

TABLE 5.3: **Word representation evaluation** scores in Spearman's r $\times 100$, for our learned composition functions. It is important to note that under *original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

		Word2Vec		GloVe		fastText		Paragram	
		Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
WS-353	+	70.4	70.1 (-0.3)	71.9	*76.5* (+4.6)	74.5	70.1 (-4.3)	73.1	72.1 (-1.1)
	average(d)	70.4	67.8 (-2.5)	71.9	73.2 (+1.3)	74.5	69.8 (-4.6)	73.1	**76.6** (+3.5)
	+ Proj.	70.4	70.8 (+0.4)	71.9	73.1 (+1.2)	*74.5*	72.0 (-2.4)	73.1	73.3 (+0.2)
	RNN	70.4	66.1 (-4.3)	71.9	73.2 (+1.2)	74.5	74.8 (+0.4)	73.1	*75.1* (+2.0)
	GRU	70.4	70.3 (-0.1)	71.9	72.9 (+0.9)	74.5	*76.5* (+2.1)	73.1	74.4 (+1.2)
	Bi-GRU	70.7	68.5 (-2.3)	71.9	73.2 (+1.2)	74.5	73.9 (-0.6)	73.1	*74.8* (+1.6)
	CNN-3	70.4	67.4 (-3.0)	71.9	72.5 (+0.5)	*74.5*	73.4 (-1.1)	73.1	73.6 (+0.4)
	CNN-3,5,7,9	70.4	68.6 (-1.7)	71.9	71.8 (-0.1)	*74.5*	72.7 (-1.7)	73.1	72.6 (-0.5)
SimLex	+	44.0	51.4 (+7.4)	40.2	50.1 (+9.8)	37.3	46.5 (+9.1)	66.2	*67.0* (+0.8)
	average(d)	44.0	37.5 (-6.5)	40.2	45.0 (+4.8)	37.3	35.5 (-1.8)	*66.2*	62.1 (-4.1)
	+ Proj.	44.0	46.6 (+2.6)	40.2	44.4 (+4.2)	37.3	43.6 (+6.3)	66.2	*66.7* (+0.6)
	RNN	44.0	44.3 (+0.3)	40.2	43.5 (+3.2)	37.3	39.7 (+2.4)	*66.2*	66.1 (-0.1)
	GRU	44.0	47.3 (+3.3)	40.2	44.6 (+4.3)	37.3	43.8 (+6.4)	66.2	**67.2** (+1.0)
	Bi-GRU	44.0	51.1 (+7.1)	40.2	45.4 (+5.2)	37.3	43.3 (+6.0)	66.2	*67.2* (+1.0)
	CNN-3	44.0	46.2 (+2.2)	40.2	44.0 (+3.7)	37.3	41.6 (+4.3)	66.2	*66.9* (+0.7)
	CNN-3,5,7,9	44.0	47.0 (+3.0)	40.2	44.3 (+4.1)	37.3	40.3 (+2.9)	66.2	*67.2* (+1.0)
SimVerb	+	34.2	39.3 (+5.2)	25.4	34.2 (+8.7)	23.0	34.1 (+11.1)	56.8	**58.3** (+1.5)
	average(d)	34.2	30.7 (-3.4)	25.4	32.2 (+6.8)	23.0	29.2 (+6.2)	*56.8*	55.7 (-1.1)
	+ Proj.	34.2	35.7 (+1.5)	25.4	28.5 (+3.0)	23.0	29.4 (+6.4)	56.8	*57.3* (+0.5)
	RNN	34.2	29.4 (-4.8)	25.4	27.1 (+1.7)	23.0	24.3 (+1.3)	*56.8*	56.0 (-0.8)
	GRU	34.2	37.2 (+3.1)	25.4	29.7 (+4.2)	23.0	29.5 (+6.5)	56.8	*57.9* (+1.1)
	Bi-GRU	34.2	36.9 (+2.8)	25.4	29.5 (+4.1)	23.0	29.9 (+6.9)	56.8	*58.1* (+1.3)
	CNN-3	34.2	33.1 (-1.1)	25.4	28.3 (+2.9)	23.0	26.2 (+3.2)	56.8	*57.6* (+0.8)
	CNN-3,5,7,9	34.2	33.0 (-1.1)	25.4	27.8 (+2.4)	23.0	26.6 (+3.6)	56.8	*57.3* (+0.5)

and the Bi-directional GRU. For both the GRU and the Bi-directional GRU we can see that the embeddings remain fixed for the first 2,500 steps. Across all graphs we see their lines remaining flat for the first part of training.

From the figures and the data in the table it is difficult to point out a specific model which is best to be used for improving the word representations. Overall, we can point to the Paragram embeddings combined with additive composition, GRU or the bi-directional GRU as producing the best word representations. But among these three approaches there is no clear winner.

5.4.3 Results on Sentence Representation Evaluations

Now we turn to sentence representation evaluation for our learned composition functions. This is where the composition functions are actually applied in a task different than they were trained for. Table 5.4 shows the result of our learned composition functions on STS14, SICK Relatedness and SICK Entailment.

For the results on all different sentence evaluations discussed in Section 4.4 we refer to Table B.2 in the Appendix.

TABLE 5.4: **Sentence representation evaluation** scores for STS and SICK-R in Pearson's $r \times 100$, all other scores in accuracies $\times 100$. It is important to note that under *Original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

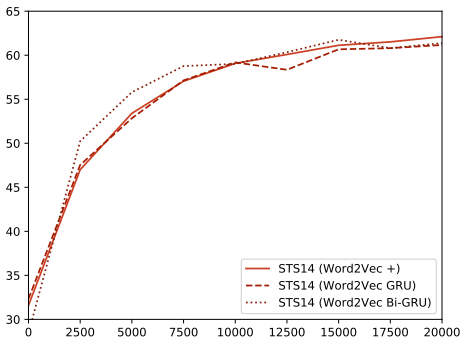
		Word2Vec		GloVe		fastText		Paragram	
		Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
STS14	+	31.9	61.8 (+29.9)	54.1	65.7 (+11.5)	53.0	65.2 (+12.2)	70.5	**71.1** (+0.5)
	average(d)	32.0	41.5 (+9.6)	54.1	48.0 (-6.1)	53.2	46.3 (-6.9)	*70.5*	49.9 (-20.6)
	+ Proj.	10.1	27.1 (+16.9)	25.2	49.7 (+24.5)	22.6	39.3 (+16.8)	27.0	*55.5* (+28.5)
	RNN	41.4	46.5 (+5.2)	52.4	48.0 (-4.4)	46.8	49.8 (+3.0)	*55.7*	47.6 (-8.1)
	GRU	48.2	62.5 (+14.2)	62.9	65.5 (+2.6)	57.9	63.9 (+6.0)	65.5	*67.4* (+1.9)
	Bi-GRU	53.5	62.3 (+8.8)	66.1	67.5 (+1.4)	62.1	64.6 (+2.5)	66.8	*67.9* (+1.1)
	CNN-3	36.0	52.9 (+16.8)	55.8	59.2 (+3.4)	51.0	57.4 (+6.4)	*63.6*	63.5 (-0.1)
	CNN-3,5,7,9	35.5	54.4 (+18.9)	59.2	61.6 (+2.4)	54.0	59.1 (+5.1)	63.7	*64.2* (+0.5)
SICK-E	+	73.8	77.3 (+3.5)	75.8	76.7 (+0.9)	76.8	76.1 (-0.7)	77.5	*78.0* (+0.6)
	average(d)	77.1	76.9 (-0.2)	79.0	78.2 (-0.8)	78.3	76.6 (-1.7)	*81.1*	78.9 (-2.1)
	+ Proj.	71.0	76.6 (+5.6)	74.0	75.4 (+1.4)	73.8	76.0 (+2.2)	75.9	*78.2* (+2.3)
	RNN	67.7	69.9 (+2.2)	69.3	68.4 (-0.9)	72.0	69.5 (-2.6)	*74.3*	67.3 (-7.0)
	GRU	75.6	78.2 (+2.6)	80.6	80.5 (-0.0)	77.7	78.7 (+1.0)	**81.5**	81.3 (-0.2)
	Bi-GRU	74.7	78.9 (+4.3)	79.7	78.7 (-1.0)	79.0	79.5 (+0.5)	81.1	*81.1* (+0.0)
	CNN-3	69.6	73.2 (+3.7)	72.9	73.8 (+0.9)	73.4	74.4 (+1.0)	74.7	*75.6* (+0.9)
	CNN-3,5,7,9	74.2	74.9 (+0.7)	74.1	*76.4* (+2.3)	75.6	75.1 (-0.6)	76.0	74.2 (-1.8)
SICK-R	+	73.4	78.4 (+5.0)	78.3	80.0 (+1.7)	78.2	78.0 (-0.2)	*80.3*	79.9 (-0.4)
	average(d)	71.4	72.2 (+0.8)	79.8	75.5 (-4.3)	79.1	74.0 (-5.0)	*81.5*	77.0 (-4.5)
	+ Proj.	66.5	70.2 (+3.7)	68.0	72.3 (+4.2)	67.0	70.4 (+3.4)	63.8	*73.4* (+9.6)
	RNN	63.8	62.4 (-1.4)	68.1	62.4 (-5.6)	68.3	63.6 (-4.8)	*69.6*	63.3 (-6.3)
	GRU	74.8	77.5 (+2.8)	*81.6*	81.2 (-0.4)	79.4	79.2 (-0.2)	81.3	81.1 (-0.2)
	Bi-GRU	76.9	78.3 (+1.3)	**81.8**	80.1 (-1.6)	80.1	79.1 (-1.0)	81.3	80.4 (-0.9)
	CNN-3	66.1	72.5 (+6.4)	73.4	73.3 (-0.1)	72.6	73.0 (+0.4)	*75.2*	74.4 (-0.8)
	CNN-3,5,7,9	73.9	75.6 (+1.7)	75.9	77.9 (+2.0)	77.5	75.9 (-1.6)	*78.4*	77.4 (-0.9)

When we look at the results for STS14 for Paragram we sadly see the weak point of learned composition functions. In learning to compose lexicographic definitions the models are unable to reach the performance of our simple additive composition function. We can clearly see this in Figures 5.9b. However, when we look at the less performing Word2Vec model in Figure 5.9a, we can see that the GRU and Bi-directional GRU are able to attain similar performance. However, this could be due to the worse initial embedding space of word2vec.

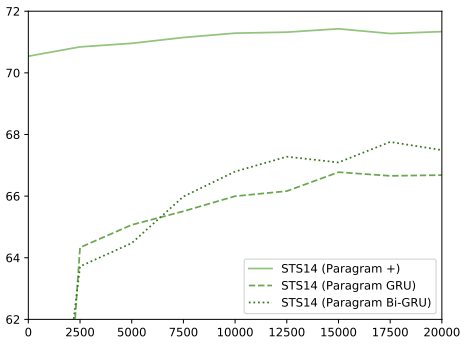
The Tables B.3 and B.4 in the appendix show all results on the different STS sets. Overall they are in line with the results from STS14.

Interestingly, when we look at SICK Entailment and Relatedness we *do* see an increased performance for the learned models. Where tuning for additive composition could not yield better results for these two tasks, which are specifically focused on the compositional properties of sentence representations. Clearly Figures 5.9d and 5.9f illustrate the progression made during training using the Paragram embeddings.

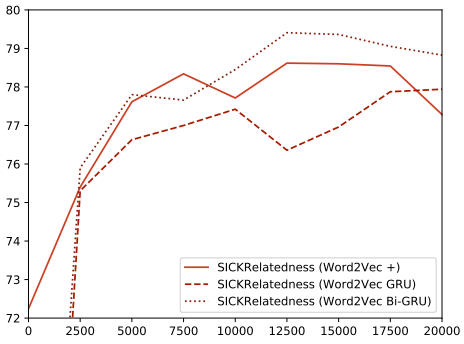
Also across the board we see that the bi-directional GRU outperforms the GRU most of the times. This could indicate an increase in expressiveness from the more involved architecture of the bi-directional GRU. Especially in applications for different tasks than where the model is trained for.



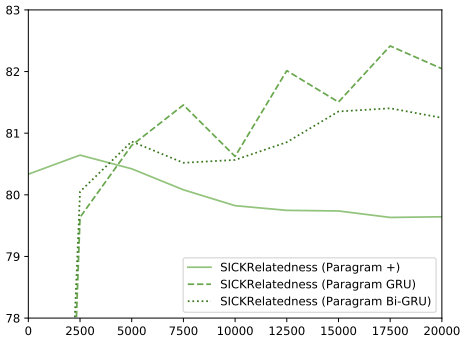
(A) STS14 Word2Vec



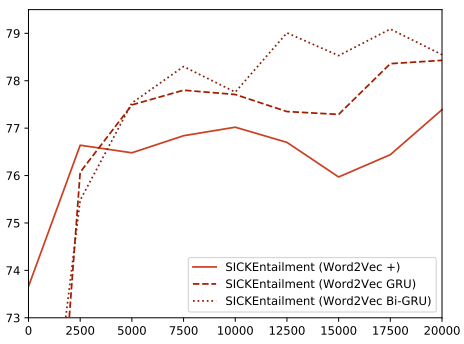
(B) STS14 Paragram



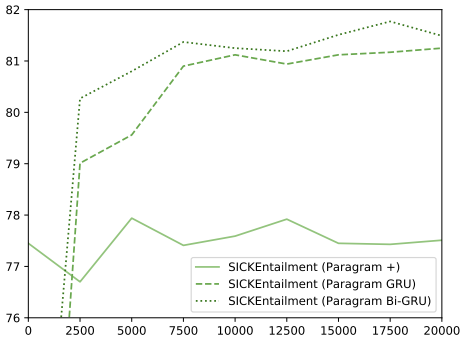
(C) SICK Relatedness Word2Vec



(D) SICK Relatedness Paragram



(E) SICK Entailment Word2Vec



(F) SICK Entailment Paragram

FIGURE 5.9: This figure shows the progression of sentence representation evaluations scores during tuning of word2vec, and Paragram with additive, GRU and Bi-GRU composition. We display Pearsons's $r \times 100$ for STS14 and SICK-R and accuracy $\times 100$ for SICK-E.

5.5 Conclusion

To conclude, our results for the learnable composition functions indicate that we are could improve upon the algebraic composition functions in the task they are trained for. Specifically, the GRU and the bi-directional GRU are candidate models for semantically composing lexicographic definitions. This improvement does not translate to every task and we did not test for statistical significance due to resource constraints. Interestingly, we see word representations generally improve rather than only being tuned to be used in conjunction with the model and lose general performance. However, when we look at the STS14 results the learned composition function cannot attain the same performance as simple additive composition.

Aside from better or worse results on evaluation metrics, learnable composition models have other downsides. For instance, they are far more computationally expensive when used for inference, i.e. summing is simply faster than pumping embeddings through a complicated multi-step neural network.

Upsides are that learnable composition functions, specifically RNNs, are able to capture long distance dependencies and are able to take word order into account. This is simply impossible for algebraic composition functions. Additionally, they allow for more complex composition with their utilization of non-linear operations.

The findings from this and the previous chapter were submitted to the 2018 edition of The Web Conference (27th edition of the former WWW conference) which will take place 23-27 April 2018 in Lyon. It was submitted as a full paper titled: "Improving Word Embedding Compositionality using Lexicographic Definitions" [110].

Chapter 6

Semantic composition of encyclopedia entries

Thus far we have trained and tuned both embeddings and composition functions with data from WordNet. This dataset is rather small with a limited size of 126,926 data points. In this chapter, we follow an intuition that is true for many machine learning tasks: "The more data we have, the better the models will perform". Instead of using *WordNet* we turn to data from the online encyclopedia *Wikipedia*. Since we want our data to be definitional, we created a custom dataset of title and summary pairs for every English entry from the website. The lemma and lexicographic definition, that are used in other chapters, are now substituted for the title and summary from the encyclopedia.

In this chapter, we will first discuss how we created the dataset from a Computer Science perspective. We will continue with discussing the results from our experiments using our novel dataset for tuning for additive composition and training GRU models for composing encyclopedic introductions. We will report the same metrics as the ones reported in the previous two chapters.

Contributions presented in this chapter:

Technical A novel dataset containing titles and *article introductions* of all five million English Wikipedia articles; and

Scientific the experimental results from tuning the four popular word embedding algorithms using this new dataset through all twenty-nine evaluation methods.

6.1 Wikipedia article introductions

Figure 6.1 shows a regular Wikipedia article. In red, the introduction paragraph is shown. This is the data our new dataset will focus on. In combination with the article title, we believe encyclopedic article introductions to be a great substitute for

FIGURE 6.1: This figure shows which section of a Wikipedia article we consider to be the 'Introduction' or 'Summary'.

lexicographic definitions. Generally speaking, they describe the exactly what is in the title. Which fueled our intuition to use this data.

We will take these introductions from 5,315,384 different Wikipedia articles, which is an almost 42-fold increase in data pairs. Additionally, both the title and the article introduction contain far more tokens on average when compared to the WordNet lemmas and definitions.

6.2 Scraping

Wikipedia distributes so-called dumps monthly, containing the entire text contents of the encyclopedia. A lot of models in artificial intelligence use these data dumps to train unsupervised text models. Among them are the fastText [17] and GloVe [99] embeddings. However, these dumps were not suitable for extracting the article introductions. The dumps contain large XML files for each article, however these XML files do not contain a clear point for the separation of the introduction and the rest of the article. It should be noted that one cannot simply extract the first paragraph. Since, the first paragraph is not always the introduction. A lot of Wikipedia articles contain warning labels, references to spoken audio, or side content boxes as their first piece or article data.

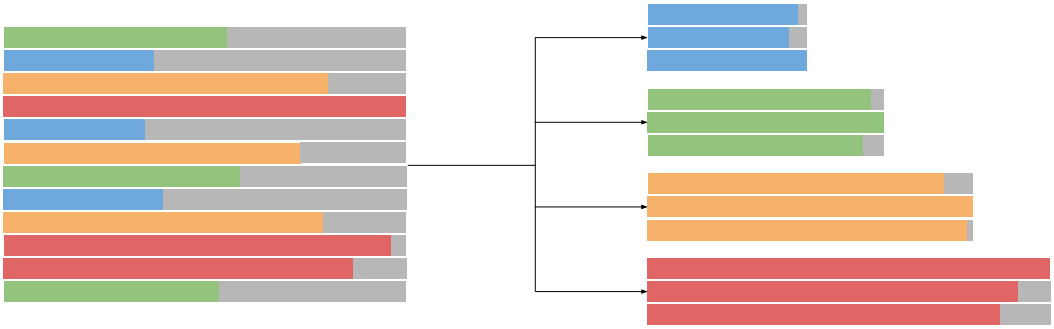


FIGURE 6.2: An example of how bucketing improves performance by eliminating the need for padding tokens. Padding tokens are here displayed in gray.

Luckily, the Wikipedia web service API does contain a method to retrieve article introductions. This API is called TextExtracts[1]. In order to use the API we setup a local version of Wikipedia and wrote a script that would call the API for *every individual English Wikipedia article*. We created a list of document identifiers for all these articles by directly querying the Wikipedia MySQL database. Using this list, we took each document identifier and called the API for each document separately. This we did because the API only allowed us to retrieve a limited number of document introduction at a time.

We quickly found that calling our local Wikipedia installation was too slow due to cold page caches. We therefore turned to the production or live Wikipedia version (at `wikipedia.org`). By specifying parameters such as `maxlag` and a scraping friendly user-agent, we where able to scrape *all* live English Wikipedia articles and extract their introductions.

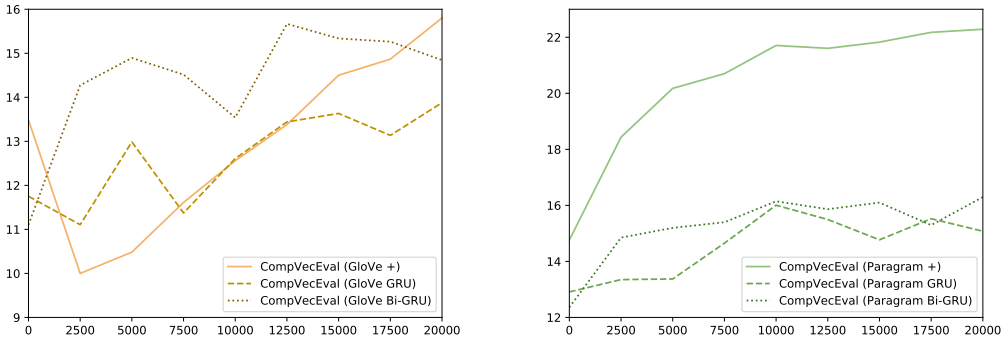
6.3 Dataset preparation

After saving all the data from our scraping procedure in September of 2017 we processed it in several steps. Similar to how we processed the WordNet data we tokenized, lowercased and removed punctuation. In addition to using the data for our experiments, we made both the raw and the processed data available for use in other models and applications[2]. For this purpose, we also created a version of the dataset containing stemmed tokens, this data was not used in our training procedure.

We finally ended up with $|X| = 5,315,384$ different title and introduction pairs. In contrast to WordNet we do not have ambiguous article introductions. So, each pair contains a unique title and description. The vocabulary has a size of $|V| = 5,171,164$ unique words. The entire dataset contains $296,210,530$ tokens, i.e. words. Because of GPU memory limitations we only consider words with at least a frequency of 18 to be a part of our vocabulary, this shrinks our final vocabulary to $|V| = 378,950$ unique

[1] Documentation for this API can be found here: `https://www.mediawiki.org/wiki/Extension:TextExtracts`.

[2] The complete dataset can be downloaded at: `https://thijs.ai/Wikipedia-Summary-Dataset/`.



(A) CompVecEval GloVe (B) CompVecEval Paragram

FIGURE 6.3: This figure shows the progression of CompVecEval score during training and tuning using Wikipedia data for GloVe; and Paragram with additive, GRU and Bi-GRU composition. We display CompVecEval MRR $\times 100$.

words, for all other words we consider them to be represented a special '*<unknown>*' token. This is still considerably larger than the vocabulary from our WordNet dataset.

The mean token length of an introduction is 52.1 tokens with a 99th percentile of 269. For the article title the mean token length is 2.61 with a 99th percentile of 7.

6.4 Streaming batches and bucketing

One advantage of the smaller WordNet dataset was that it could be kept in RAM memory during the training procedure. This is not the case for the 5 million data points of the Wikipedia dataset. Especially, because our training procedure is implemented in Python which has adds memory cost to the storing and handling of numbers. Therefore, we implemented a streaming approach to loading our data from disk during training, at the end of a batch we call the Unix utility `shuf`, to shuffle the data in place on disk with an efficient program outside of the Python process.

Because the length of encyclopedic introductions is larger and more irregular than the length of lexicographic definitions, we employ a technique called bucketing to speed up our training procedure. Bucketing reduces the need for padding tokens in batches. We show an example of how this works in Figure 6.2. We create four so called buckets, each containing introductions of different token lengths. The first bucket contained introductions with lengths between $1 \leq |i| < 19$, the second $19 \leq |i| < 34$, the third $34 \leq |i| < 64$, and the forth $64 \leq |i| < 266$. These buckets are based on the token length percentiles (25th, 50th, 75th and 99th percentiles). We loop through the four different batches during training, each next batch is taken from a different bucket.

TABLE 6.1: This table displays the results of tuning the word embeddings using **data from Wikipedia (20,000 batches)**. For all measures counts: "Higher is better". The results for CompVecEval, MRPC and SICK-E scores are denoted in accuracies $\times 100$. All other sentence evaluation measures are denoted in Pearson's $r \times 100$, whereas the word similarity measures are denoted in Spearman's $r \times 100$.

		Word2Vec		GloVe		fastText		Paragram	
Measure	Composition	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
CompVecEval	+	8.1	16.0 (+7.9)	13.5	15.8 (+2.3)	11.2	16.1 (+5.0)	14.8	**22.3** (+7.5)
	GRU	10.5	9.1 (-1.3)	14.4	13.9 (-0.5)	12.2	10.4 (-1.7)	*16.8*	15.1 (-1.8)
	Bi-GRU	13.5	11.2 (-2.3)	15.2	14.8 (-0.4)	14.3	12.3 (-2.0)	15.7	*16.3* (+0.6)
STS14	+	59.9	66.0 (+6.0)	63.7	67.1 (+3.4)	61.8	67.3 (+5.5)	**74.0**	73.6 (-0.3)
	GRU	51.9	52.9 (+1.0)	62.9	63.2 (+0.2)	55.6	54.4 (-1.1)	67.0	*67.3* (+0.3)
	Bi-GRU	58.6	58.2 (-0.3)	64.7	62.9 (-1.8)	57.1	58.5 (+1.4)	*68.5*	68.4 (-0.0)
SICK-E	+	68.4	*69.6* (+1.2)	69.2	68.8 (-0.4)	68.7	68.8 (+0.1)	69.4	69.5 (+0.1)
	GRU	72.9	71.8 (-1.0)	73.3	*73.6* (+0.3)	71.5	71.8 (+0.3)	72.5	73.3 (+0.9)
	Bi-GRU	72.2	72.5 (+0.2)	74.3	73.2 (-1.1)	71.8	70.7 (-1.2)	**74.3**	74.2 (-0.1)
SICK-R	+	75.7	75.1 (-0.7)	76.0	75.2 (-0.7)	76.0	75.2 (-0.8)	76.2	*76.3* (+0.1)
	GRU	75.5	72.0 (-3.5)	76.0	76.5 (+0.5)	73.3	72.7 (-0.6)	76.6	*76.8* (+0.2)
	Bi-GRU	76.2	73.0 (-3.2)	**77.3**	76.9 (-0.4)	75.6	73.4 (-2.2)	76.6	76.7 (+0.1)
SimLex	+	44.2	42.5 (-1.8)	40.7	42.9 (+2.2)	37.3	35.5 (-1.9)	**63.3**	62.7 (-0.6)
	GRU	44.2	41.5 (-2.7)	40.7	41.1 (+0.4)	37.3	35.9 (-1.5)	*63.2*	62.6 (-0.6)
	Bi-GRU	44.2	40.8 (-3.4)	40.7	41.5 (+0.8)	37.3	37.4 (+0.0)	*63.2*	62.8 (-0.4)
SimVerb	+	36.7	33.7 (-3.0)	28.3	30.8 (+2.6)	25.1	23.6 (-1.5)	50.1	**50.2** (+0.1)
	GRU	36.7	33.6 (-3.1)	28.3	28.8 (+0.5)	25.1	24.9 (-0.2)	*50.1*	50.0 (-0.1)
	Bi-GRU	36.7	32.5 (-4.2)	28.3	28.4 (+0.1)	25.1	24.2 (-0.9)	*50.1*	49.8 (-0.3)
WS-353	+	68.3	61.8 (-6.5)	71.0	73.6 (+2.6)	**73.9**	67.9 (-5.9)	70.2	64.4 (-5.8)
	GRU	68.1	64.1 (-4.0)	71.0	71.7 (+0.7)	**73.9**	69.7 (-4.1)	70.0	69.9 (-0.1)
	Bi-GRU	68.1	64.0 (-4.1)	71.0	71.1 (+0.1)	**73.9**	72.7 (-1.2)	69.3	68.5 (-0.7)

6.5 Experimental results

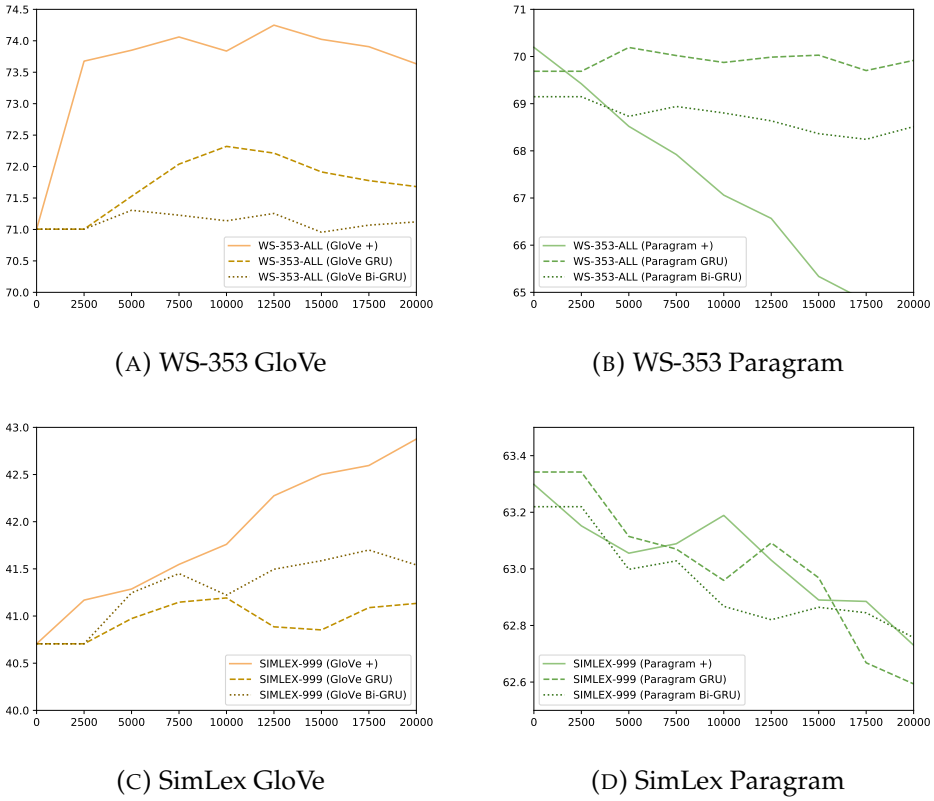
For the first set of experiments, we train on 20.000 batches of 512 training examples, which roughly equates to 2 epochs on the entire Wikipedia dataset. We train using all four pretrained embeddings combined with the additive composition function (+), GRU and bi-directional GRU.

Since we wanted to see if our results could improve further when training longer, we also ran experiments for 100.000 batches of 512 training examples ($\approx$ 10 epochs). Since these experiments take a long time to run we only executed them for additive composition and GRU composition with the Paragram embeddings.

6.5.1 Broad training procedure

Table 6.1 shows the results for the experiments on 20.000 batches. Here we only report the most important metrics. Appendix C has all the results for all the metrics described in Sections 4.3 and 4.4.

If we compare the results under the *'Pre'* columns with the results from the previous chapters we see that not all numbers correspond exactly. This is because we use a entirely different vocabulary for training the with the Wikipedia dataset (see Appendix C.1). Because of this entirely different vocabulary some evaluation method's scores



(A) WS-353 GloVe

(B) WS-353 Paragram

(C) SimLex GloVe

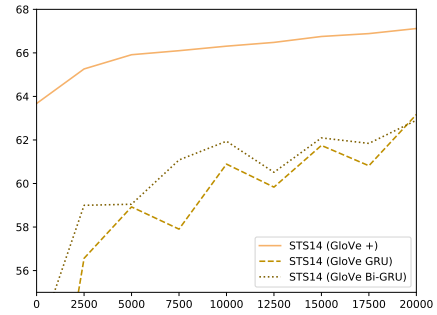
(D) SimLex Paragram

FIGURE 6.4: This figure shows the progression of word representation evaluations scores during tuning using Wikipedia. We display Spearman's $r \times 100$.

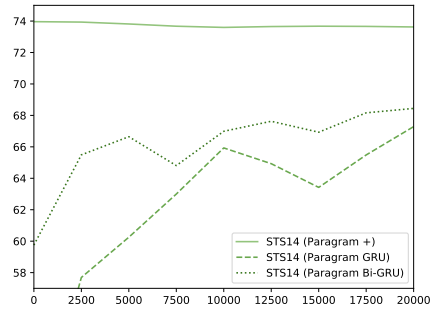
have large differences for their initial starting positions. The random initialization described in Section 3.3.3 and 4.5.4 can also have an influence. Keep in mind that, when looking at the data in the *'Pre'* columns, we did not want to evaluate existing word embeddings on their own. They are evaluated in conjunction with their tuned counter parts to see whether tuning has an impact on semantic compositionality and general performance. Therefore, the word representation and sentence representation metrics reported in the *'Pre'* columns should not be directly compared to ones reported in other works, since these probably also use different vocabularies.

In Figure 6.3, we see the progression of CompVecEval during the training procedure for the GloVe and the Paragram embeddings (Figure 6.3b). For Paragram we see an overall upward trend. However, it is not reaching the same level as training on our WordNet dataset. This could be expected, since we are now training for a related but still different task. Keep in mind that we are still using the WordNet data for CompVecEval itself.

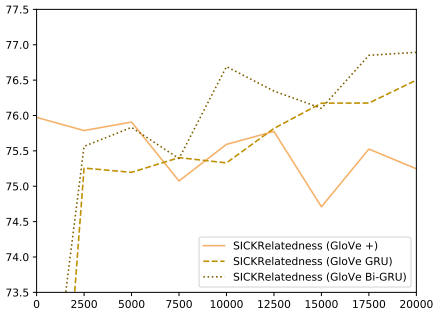
The GloVe embeddings also trend upwards, but training for additive composition results in this large drop in performance after the first 2.500 steps (Figure 6.3a). This is interesting, because it is a pattern we have not seen in any of the previous results. The sharp increase in GloVe performance from Chapter 4, and the small increase here, could be explained by the nature of the data GloVe is trained on. GloVe embeddings are already trained on data from Common Crawl (which contains Wikipedia data).



(A) STS14 GloVe



(B) STS14 Paragram



(C) SICK Relatedness GloVe

(D) SICK Relatedness Paragram

FIGURE 6.5: This figure shows the progression of sentence representation evaluations scores during tuning using Wikipedia. We display Pearsons's $r \times 100$ for STS14 and SICK-R.

This could mean that it is already structured in such a way so it does not have much to gain from composing Wikipedia introduction. However, this is a mere speculative statement.

When we look at Figure 6.6, which displays the results for word representation evaluations, we clearly see that the training procedure has a negative effect on the Paragram embeddings. However, when we look at the GloVe embeddings, we see a slight upward trend. Keep in mind that the Spearman's r for SimLex on GloVe is a lot lower to start with.

For the sentence embedding evaluations we also see mixed results (Figure 6.5). While we do see an upward trend in performance on SICK relatedness and STS14, this is nowhere near the performance gains we have seen in the results form previous chapters. Across the board, we are not able to replicate the results from our training procedure using WordNet data.

6.5.2 Long training procedure

20.000 steps still only represent 2 epochs on the Wikipedia dataset. However, training all combinations of composition functions with pretrained embeddings was for us infeasible due to resource constraints. We therefore trained for 100,000 steps ($\approx$ 10 epochs) with limited configurations. We start with the Paragram embeddings

TABLE 6.2: This table displays the results of training using Wikipedia data **for 100,000 steps**.

		Paragram	
		Pre.	Tuned
CompVecEval	+	14.8	**21.8** (+7.0)
	GRU		*15.5*
STS14	+	**74.0**	73.5 (-0.5)
	GRU		*65.7*
SICK-E	+	*69.4*	68.9 (-0.4)
	GRU		**73.0**
SICK-R	+	**76.2**	76.0 (-0.3)
	GRU		*75.7*
SimLex	+	**63.3**	62.2 (-1.1)
	GRU		*62.9*
SimVerb	+	**50.3**	50.1 (-0.2)
	GRU		*49.9*
WS-353	+	**69.4**	64.2 (-5.2)
	GRU		*69.3*

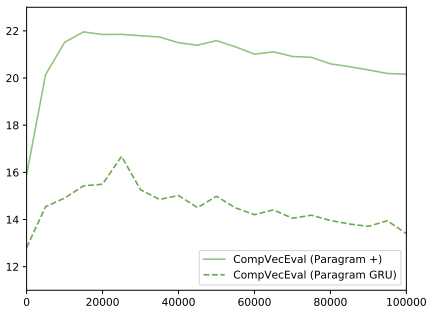
and train tuned embeddings using the additive composition. Additionally, we also trained a GRU composition function, where we allowed the embeddings to change after 2,500 training steps.

The results are shown in Table 6.2 and in Figure 6.6. These results are in line with the results from the shorter training procedure. Interestingly, we see behavior reminiscent of over fitting on CompVecEval after 20,000 steps. Overall, we see that training and tuning using Wikipedia data results in worse embeddings and worse semantic composition.

6.6 Conclusion

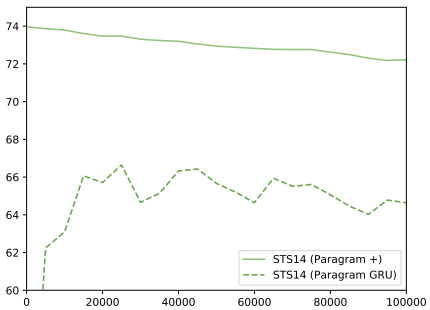
We introduced a new Wikipedia based dataset containing pairs of title and article introductions. Using this data to improve the semantic composition of word embeddings was our goal. However, our results show that this training data is sadly not always beneficial for this task. Especially, when compared against the results from the previous chapters (4 and 5).

We published the dataset itself, since its creation was not strait forward. We hope that it can be used by other researcher for other NLP tasks. The complete dataset can be downloaded at: `https://thijs.ai/Wikipedia-Summary-Dataset/`.

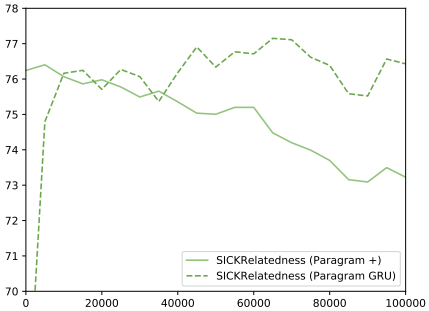


(A) CompVecEval

(B) SimLex



(C) STS14



(D) SICK Relatedness

FIGURE 6.6: This figure shows the progression of some evaluation during a **long 100,000 step** run for additive composition (+) and GRU on the Wikipedia dataset.

Chapter 7

Conclusion

This thesis started out with a reference to a fictional AI character from the film Her. We stated that creating such an AI would require a computer to understand the meaning of language. One of the future milestones in AI research will be: finding great representations of meaning a computer can work with. We hope that our work, in providing empirical results on functions for semantic compositions and our tuned embeddings, brings us a tiny step closer to this goal, which still seems so inconceivably far away.

Summarizing, in this work we introduced a new method to tune word embeddings for ten composition functions. Four of which are algebraic and easily applicable in large scale industrial systems, and six of which are learned. Additionally, we presented a new method to evaluate word embeddings called: *CompVecEval*. This method is different from existing methods, since it directly tries to evaluate compositional semantics from lexicographic definitions to lemmas. In addition, it relies on ranking instead of accuracy and is not invariant the magnitude of the embedding vector.

We analyzed four popular word embeddings and found that the Paragram embeddings are the most versatile for various forms of composition[1]. Our tuned CompVec Paragram embeddings are a good choice if your model uses them for additive composition. Using GRU composition is also a candidate which seems to be a more elaborate but effective composition function for specific tasks. In addition, for various evaluation measures, we found that tuning the embeddings on our lexicon based dataset performed better or comparable. When selecting word embeddings for transfer learning, so they can be the starting point for training a neural network, our results can give insights into which embeddings you could choose. We published and open sourced both the dataset as well as the tuned embeddings.

Additionally, we tried extending our approach from lexicographic to encyclopedic data. In order to do so we created a new dataset containing pairs of Wikipedia titles and article introductions. The results from experiments using this dataset were mixed and do not yield even better embeddings over the ones obtained using our lexicographic dataset.

[1] Keeping in mind, the caveat on the random initializations as discussed in Section 3.3.3.

7.1 Retrospective and discussion

Looking back on the research that has gone into the creation of this thesis, we have several critiques on our own work. As is true for much in life, things can always be better and this thesis is no exception.

We make the assumption in defining CompVecEval that composing lexicographic definitions is a good task to measure overall semantic composition. However, this is an assumption which still needs theoretical backing from philosophers, linguists and computer scientists. Sadly, thus far there is nothing definitive [45]. Our empirical results show some promise, but are insufficient to make such broad conclusions.

During our experiments for tuning and training models, we arbitrarily choose 20.000 steps to be entirety of the training procedure. We could optimize the training procedure by splitting off a validation set[2] and employ established methods such as early stopping [100].

The same is true for all the hyper-parameters we use. Tuning the margin in the loss function, the parameters on the Adam optimization function [70], the initializations of model parameters and dropout rate could all result in better overall performance. Even though we experimented to find good parameters, we think we can still make big improvements. Sadly, we did not have the time nor the computational resources to make these adjustments.

Because we use randomized initialization for embeddings, which are associated with a word not in the vocabulary of the four pretrained embeddings, and because we use similar randomized initialization for parameters in our neural models all results are somewhat stochastic. Although we did manage to run statistical significance testing on some specific experiments (See Appendix A.5), we did not manage these tests for all our experiments. It would have been nice to repeat all other experiments a couple of times and determine statistical significance for all the results. Sadly again, we did not have the time nor the computational resources.

In Section A.2, we described the loss function used by Wieting et al. [135] to tune the Paragram embeddings. In Chapter 4 we used our own cosine based loss function but experiments using this loss function yielded bad results. We omitted a regularization term which Wieting et al. introduced to keep the embedding matrix close to the original matrix. It would have been interesting to also run our experiments with this term added. We believe it could have an impact, and improve our performance on composition through averaging.

Some papers use analogy tasks to evaluate the quality of single word embeddings [99, 84]. While we included quite extensive word embedding evaluation with thirteen different methods, it would have been nice to test our embeddings on these analogy datasets as well.

[2]Our implementation is already prepared for splitting the dataset in new ways, and creating a validation set is such an example. For our WordNet dataset this is not necessarily trivial without our implementation, because of the constraints explained in section 3.3.

This leads us to a somewhat contrastive self-criticism. The thirteen different word evaluation methods, the fifteen different sentence evaluation methods, our own CompVecEval evaluation method and the separate IR search evaluation, results in pretty jumbled overall results which are hard to interpret. It would be nice if the research community finds a good way to test overall word and sentence quality using a single method instead of relying on all these different methods, which results do not always agree. Understandably, it is difficult to create such a method. Especially, because human judgment on language meaning changes over time.

7.2 Recommendations for future work

For future research, we want to look into multi-layered composition functions. One neural model we would like to explore is a convolutional model containing so called: 'dilated convolutions' [95]. These could provide a solution to the problems with long distance relationships described in Section 5.3.

Also, one limitation of our lexicographic dataset is its size. Other dictionaries sometimes contain more definitions and words. For example, the Oxford Dictionary is a lot larger and the data from this dictionary can be extracted using their open API [3]. Our intuition is that using this data could increase the performance, not only because there is more data in the Oxford Dictionary but also because its vocabulary is larger.

Results by Arora, Liang, and Ma [7] also sparked an idea on weighing embeddings. Their research found that if one weights the embeddings according to a function containing the word frequency they could increase performance on various tasks using the Paragram embeddings. We would like to explore how this relates to the initial embedding magnitude and see if we can adjust the norm of an embedding according to this weighing function and see if it improves the results for additive composition.

Published results on the theoretical understanding of word embeddings space is currently limited to the work by Arora, Liang, and Ma [7] and Gittens, Achlioptas, and Mahoney [47]. They only go into the theoretical understanding of the original skip-gram algorithm. Further work could look at the theoretical and mathematical properties of the embedding spaces created using various different algorithms.

In Chapter 5 we discussed the bounding effect of some non-linear activation functions (*tanh / sigmoid / ReLU*). Since we and Arora, Liang, and Ma [7] found that the magnitude of embeddings could play a role in correct semantic composition one could explore learnable compositional models that compose words into unbound sentence embeddings.

In this thesis we used words as a basic unit of representing semantics, while other works use phrases [84], character n-grams [17], characters or sentences. When researching the idea of *semantic primes* [133, 63] for our background chapter, we considered them as a model for semantic representations. Not in the sense that Wierzbicka

[3] The Oxford Dictionary API can be accessed at: `https://developer.oxforddictionaries.com`

[133] and Jackendoff [63] described them. But in the sense that we would like to see a model that is allowed to choose a semantic prime in latent space. This decision should rely on a specific context representation using a discrete choice, e.g. through the Gumble-softmax function [65]. How this would work exactly still needs to be thought out, but it could be an interesting research direction.

Appendix A

Miscellaneous results on tuning for algebraic composition

In this appendix we will present miscellaneous results from the experiments described in Chapter 4 on tuning word embeddings towards better compositionality.

A.1 Tuning embeddings with single token targets

In Table A.1 we see the tuning results for single token targets. They are inferior to tuning using multi token targets.

Interesting to notice is that fastText is able to compose better according to CompVecEval when trained on data using a single token. Also we see that almost all models have similar performance before tuning, compared to the multi token results. One exception is Paragram in this case, while it is able to climb 3.0 points on CompVecEval it started significantly lower at 23.6 (where the multi token model started at 26.6). The only difference is that both models used a slightly different initialization for the words that were our WordNet vocabulary but were not in the (expanded[1]) Paragram vocabulary.

A.2 Tuning using the cosine similarity loss function

Table A.2 shows the results obtained from tuning embeddings using the cosine loss function described in Equation 4.2. As discussed in Section Wieting et al. [135] has a similar loss function but uses a regularization term which makes sure the new embedding matrix does not deviate significantly from the original embedding matrix. We did not add such a term and this could be the reason for the poor performance using the cosine similarity loss function.

[1]We expanded Paragram according to the suggestions by Wieting et al. [134].

TABLE A.1: This table displays the results of tuning the word embeddings using **single token lemmas as targets**. For all measures counts: "Higher is better". The results for CompVecEval, MRPC and SICK-E scores are denoted in accuracies $\times 100$. All other sentence evaluation measures are denoted in Pearson's $r \times 100$, whereas the word similarity measures are denoted in Spearman's $r \times 100$.

		Word2Vec		GloVe		fastText		Paragram	
Measure	Composition	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
CompVecEval	$+$	17.5	25.2 (+7.7)	11.9	28.8 (+16.9)	20.7	**29.4** (+8.7)	23.6	26.7 (+3.0)
	average(d)	2.1	2.0 (-0.0)	3.3	*6.8* (+3.5)	3.0	4.8 (+1.8)	3.5	5.1 (+1.5)
	$\times$	0.6	0.5 (-0.1)	0.9	*1.4* (+0.4)	0.9	0.9 (+0.0)	1.1	0.7 (-0.4)
	max(d)	1.3	3.3 (+2.0)	3.4	7.4 (+4.0)	2.3	*9.1* (+6.8)	3.4	7.6 (+4.2)
STS14	$+$	32.7	54.3 (+21.7)	54.1	62.3 (+8.2)	52.8	61.5 (+8.6)	**70.5**	70.2 (-0.3)
	average(d)	32.1	32.2 (+0.0)	54.1	57.3 (+3.2)	52.8	49.5 (-3.3)	**70.5**	65.5 (-5.0)
	$\times$	5.1	12.3 (+7.1)	5.9	8.3 (+2.4)	7.9	*30.2* (+22.2)	1.6	6.9 (+5.3)
	max(d)	21.7	45.5 (+23.8)	60.6	*65.3* (+4.7)	42.5	61.4 (+18.9)	61.6	63.5 (+1.9)
MRPC	$+$	*70.2*	*70.2* (-0.0)	69.7	68.9 (-0.8)	69.7	67.7 (-2.0)	69.7	70.0 (+0.3)
	average(d)	71.0	72.0 (+1.0)	72.3	72.1 (-0.3)	71.7	**73.1** (+1.4)	72.5	71.9 (-0.6)
	$\times$	66.5	66.5 (+0.1)	66.0	64.9 (-1.1)	66.5	66.6 (+0.1)	66.2	*67.4* (+1.2)
	max(d)	69.0	*71.3* (+2.3)	*71.3*	70.3 (-1.0)	68.3	70.5 (+2.3)	70.5	70.8 (+0.3)
SICK-E	$+$	74.1	75.8 (+1.8)	75.8	76.4 (+0.6)	76.9	76.9 (-0.0)	*77.5*	*77.7* (+0.2)
	average(d)	77.1	76.1 (-1.0)	79.0	78.4 (-0.6)	79.4	77.8 (-1.6)	**81.1**	80.9 (-0.2)
	$\times$	56.7	56.8 (+0.1)	28.2	56.5 (+28.3)	56.7	57.9 (+1.2)	56.7	*58.3* (+1.6)
	max(d)	75.4	79.4 (+3.9)	78.1	79.7 (+1.6)	*79.8*	78.7 (-1.1)	79.1	79.6 (+0.5)
SICK-R	$+$	73.6	76.9 (+3.4)	78.3	78.4 (+0.1)	78.1	78.0 (-0.1)	*80.3*	80.0 (-0.4)
	average(d)	71.7	73.5 (+1.8)	79.8	78.3 (-1.5)	79.1	76.6 (-2.6)	**81.5**	80.7 (-0.8)
	$\times$	8.0	20.4 (+12.4)	19.2	30.6 (+11.3)	11.6	*40.5* (+28.9)	16.6	24.7 (+8.2)
	max(d)	74.5	79.1 (+4.7)	79.8	*79.9* (+0.1)	78.6	78.8 (+0.3)	78.7	78.8 (+0.1)
SimLex	$+$	44.0	51.6 (+7.6)	40.2	47.6 (+7.4)	37.3	46.4 (+9.1)	64.2	*65.0* (+0.9)
	average(d)	44.0	49.6 (+5.6)	40.2	48.1 (+7.8)	37.3	45.5 (+8.1)	64.2	**66.1** (+2.0)
	$\times$	44.0	44.4 (+0.4)	40.2	42.7 (+2.5)	37.3	37.5 (+0.2)	*64.2*	63.7 (-0.5)
	max(d)	44.0	49.8 (+5.8)	40.2	45.0 (+4.8)	37.3	43.2 (+5.9)	64.1	*65.8* (+1.7)
SimVerb	$+$	34.2	42.9 (+8.8)	25.4	36.8 (+11.4)	23.0	35.8 (+12.8)	53.3	*56.0* (+2.6)
	average(d)	34.2	40.9 (+6.8)	25.4	35.7 (+10.3)	23.0	34.3 (+11.3)	53.4	**56.4** (+3.0)
	$\times$	34.2	34.4 (+0.2)	25.4	30.5 (+5.1)	23.0	25.7 (+2.7)	53.4	*53.5* (+0.2)
	max(d)	34.2	42.4 (+8.3)	25.4	32.2 (+6.8)	23.0	31.4 (+8.4)	53.2	*56.1* (+2.9)
WS-353	$+$	70.8	69.7 (-1.0)	71.9	*76.0* (+4.1)	74.5	73.7 (-0.7)	71.6	70.8 (-0.8)
	average(d)	70.6	70.3 (-0.3)	71.9	**77.8** (+5.9)	74.5	74.3 (-0.2)	71.4	72.1 (+0.7)
	$\times$	70.4	68.1 (-2.3)	71.9	*75.2* (+3.2)	74.5	74.7 (+0.3)	71.4	70.2 (-1.2)
	max(d)	70.3	72.8 (+2.5)	71.9	74.8 (+2.9)	74.5	*75.6* (+1.1)	71.7	72.0 (+0.3)

TABLE A.2: This table displays the results of tuning the word embeddings using the **cosine similarity loss function**. For all measures counts: "Higher is better". The results for CompVecEval, MRPC and SICK-E scores are denoted in accuracies $\times 100$. All other sentence evaluation measures are denoted in Pearson's $r \times 100$, whereas the word similarity measures are denoted in Spearman's $r \times 100$.

		Word2Vec		GloVe		fastText		Paragram	
Measure	Composition	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
CompVecEval	$+$	16.5	1.3 (-15.2)	11.9	1.1 (-10.8)	20.7	1.1 (-19.6)	**23.5**	1.4 (-22.1)
	average(d)	2.0	1.3 (-0.8)	3.3	1.2 (-2.1)	2.9	1.3 (-1.7)	*3.5*	1.4 (-2.1)
	$\times$	0.6	0.4 (-0.2)	0.0	0.0 (-0.0)	0.9	0.5 (-0.4)	*1.1*	0.6 (-0.5)
	max(d)	1.3	0.6 (-0.6)	3.4	0.4 (-3.0)	2.3	0.4 (-1.9)	*3.4*	0.5 (-2.9)
STS14	$+$	32.1	24.4 (-7.7)	54.1	22.4 (-31.8)	53.1	23.9 (-29.2)	**70.5**	27.7 (-42.8)
	average(d)	31.9	46.4 (+14.5)	54.1	41.0 (-13.2)	53.4	46.6 (-6.8)	*70.5*	48.3 (-22.2)
	$\times$	3.9	1.2 (-2.7)	0.0	0.0 (-0.0)	*10.3*	3.2 (-7.1)	1.6	6.1 (+4.4)
	max(d)	21.6	33.6 (+12.0)	60.6	28.9 (-31.7)	42.7	32.4 (-10.3)	*61.6*	39.6 (-22.0)
MRPC	$+$	68.6	64.6 (-4.0)	*69.4*	63.6 (-5.8)	68.8	66.4 (-2.4)	69.2	64.8 (-4.3)
	average(d)	70.8	69.5 (-1.3)	72.3	71.1 (-1.2)	71.3	68.9 (-2.4)	**72.5**	69.7 (-2.8)
	$\times$	*66.5*	*66.5* (-0.0)	0.0	0.0 (-0.0)	*66.5*	*66.5* (-0.0)	66.2	*66.5* (+0.3)
	max(d)	71.5	70.5 (-1.1)	*71.7*	69.6 (-2.1)	68.2	67.9 (-0.2)	69.8	68.8 (-1.0)
SICK-E	$+$	74.4	68.6 (-5.8)	75.8	70.4 (-5.4)	*77.2*	70.5 (-6.7)	*77.5*	70.8 (-6.7)
	average(d)	76.0	76.3 (+0.3)	79.0	75.9 (-3.1)	79.2	76.1 (-3.1)	**81.1**	77.3 (-3.8)
	$\times$	*56.7*	*56.7* (-0.0)	0.0	0.0 (-0.0)	*56.7*	*56.7* (-0.0)	*56.7*	*56.7* (-0.0)
	max(d)	75.9	77.6 (+1.7)	78.1	77.1 (-1.0)	*79.5*	76.7 (-2.8)	79.1	76.1 (-3.1)
SICK-R	$+$	73.6	56.3 (-17.3)	78.3	56.5 (-21.8)	78.2	55.2 (-23.0)	*80.3*	59.5 (-20.9)
	average(d)	72.6	68.4 (-4.2)	79.8	67.0 (-12.8)	78.9	68.5 (-10.5)	**81.5**	70.8 (-10.7)
	$\times$	5.4	9.9 (+4.5)	0.0	0.0 (-0.0)	11.6	10.7 (-0.9)	*16.6*	15.5 (-1.1)
	max(d)	74.3	73.2 (-1.1)	*79.8*	70.7 (-9.1)	78.6	71.1 (-7.5)	78.7	73.1 (-5.6)
SimLex	$+$	44.0	-4.9 (-48.8)	40.2	-6.6 (-46.9)	37.3	-5.4 (-42.7)	*64.2*	-7.2 (-71.4)
	average(d)	44.0	-6.3 (-50.3)	40.2	-6.1 (-46.3)	37.3	-6.8 (-44.1)	**64.3**	7.4 (-56.9)
	$\times$	44.0	40.6 (-3.4)	0.0	0.0 (-0.0)	37.3	37.5 (+0.2)	*64.3*	60.6 (-3.7)
	max(d)	44.0	4.4 (-39.6)	40.2	11.2 (-29.0)	37.3	4.8 (-32.5)	*64.1*	36.0 (-28.1)
SimVerb	$+$	34.2	-2.2 (-36.4)	25.4	-4.1 (-29.5)	23.0	-3.6 (-26.6)	*53.3*	-3.9 (-57.2)
	average(d)	34.2	3.9 (-30.3)	25.4	1.6 (-23.9)	23.0	1.3 (-21.7)	*53.4*	6.9 (-46.5)
	$\times$	34.2	34.3 (+0.1)	0.0	0.0 (-0.0)	23.0	29.5 (+6.5)	53.3	**53.7** (+0.5)
	max(d)	34.2	7.5 (-26.7)	25.4	7.9 (-17.6)	23.0	3.1 (-19.9)	*53.5*	25.0 (-28.4)
WS-353	$+$	70.7	-6.1 (-76.8)	71.9	-4.8 (-76.8)	**74.5**	-3.8 (-78.2)	71.6	-6.3 (-77.9)
	average(d)	70.4	2.1 (-68.4)	71.9	7.1 (-64.8)	**74.5**	0.9 (-73.6)	71.3	9.8 (-61.6)
	$\times$	70.8	59.1 (-11.7)	0.0	0.0 (-0.0)	**74.5**	68.5 (-5.9)	71.7	68.5 (-3.2)
	max(d)	70.4	11.9 (-58.5)	71.9	40.0 (-31.9)	**74.5**	24.6 (-49.8)	72.1	33.8 (-38.3)

TABLE A.3: In this table we show results for an Information Retrieval task, and how our tuned embeddings can improve on this task. Results are displayed as MAP@1000 $\times 100$, just as in the work by Van Gysel, de Rijke, and Kanoulas.

	Paragram +		NVSM [128]
	Pre.	Tuned	
Robust04	8.12	9.08 (+ 0.96)	15.0
WSJ	13.45	14.20 (+ 0.75)	20.8

A.3 Results on semantic textual similarity

In table A.4 and A.5 we show the results of the evaluations on all subsets for the STS tasks from 2012 to 2016, in addition we show the results of the STS Benchmark. Not all results are statistically significant, however most of them are. All corresponding p-vales can be found in A.6.

These are just the results for the algebraic composition functions, for the learned composition function we refer to the next appendix (B.3). We left out the $\times$ composition function, since it is clear from all earlier results that it was not a composition function worth considering.

A.4 Using tuned embeddings for Information Retrieval

Together with Van Gysel, de Rijke, and Kanoulas, we tested the original and our tuned Paragram embeddings (for additive composition) in Information Retrieval tasks.

We used the same code and methods that were used by Van Gysel, de Rijke, and Kanoulas [128] for their new neural vector spaces for unsupervised information retrieval. In their work, the new and more elaborate model performs better on several information retrieval tasks. They compare their model to an additive embedding baseline. We repeated the baseline experiments with the original and our tuned Paragram embeddings. In Table A.3 we see an improvement in the MAP@1000 results, however not an improvement which outperforms their model. We refer to the paper [128] for the exact evaluation procedure. It must be mentioned that these experiments were executed once, and no statistical significance has been established.

TABLE A.4: **STS12-14** scores in Pearson's $r \times 100$. These evaluations are done for before and after tuning the embeddings for a specific algebraic composition operation. We also trained embeddings without pretraining (*None*).

		None	Word2Vec		GloVe		fastText		Paragram	
		Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
STS12	$+$	52.5	26.2	53.2 (+27.1)	49.7	55.8 (+6.1)	48.9	56.4 (+7.5)	**60.6**	60.2 (-0.5)
	average(d)	27.9	26.3	28.7 (+2.4)	49.7	35.3 (-14.4)	49.1	32.7 (-16.4)	*60.6*	35.6 (-25.1)
	max(d)	49.6	19.6	55.6 (+36.0)	52.7	56.1 (+3.4)	40.4	55.2 (+14.9)	55.0	*56.6* (+1.6)
MSRpar	$+$	41.3	34.2	43.8 (+9.6)	40.4	35.6 (-4.8)	**47.4**	46.0 (-1.4)	42.1	40.0 (-2.1)
	average(d)	27.3	34.2	28.1 (-6.1)	40.4	31.8 (-8.6)	*46.3*	30.3 (-16.0)	42.1	29.5 (-12.6)
	max(d)	31.8	25.3	*40.3* (+15.1)	34.5	31.4 (-3.1)	36.4	37.1 (+0.6)	39.8	35.2 (-4.6)
MSRvid	$+$	59.4	-6.3	64.0 (+70.3)	64.3	71.9 (+7.6)	55.8	68.4 (+12.5)	77.9	*78.4* (+0.5)
	average(d)	-7.1	-7.3	-1.6 (+5.7)	64.3	12.0 (-52.3)	55.7	8.4 (-47.3)	*77.9*	13.8 (-64.0)
	max(d)	75.1	-10.4	80.6 (+91.0)	76.8	82.6 (+5.7)	43.9	81.5 (+37.6)	78.9	**84.0** (+5.1)
SMTeuroparl	$+$	**48.6**	27.0	39.0 (+12.0)	26.4	43.2 (+16.8)	24.4	43.8 (+19.5)	44.0	46.4 (+2.3)
	average(d)	43.1	28.1	33.9 (+5.8)	26.4	37.4 (+11.0)	25.7	35.1 (+9.4)	*44.0*	38.1 (-5.9)
	max(d)	30.9	19.7	29.4 (+9.7)	22.5	*32.4* (+9.9)	15.2	28.9 (+13.7)	21.4	30.3 (+8.8)
OnWN	$+$	65.4	47.1	69.2 (+22.1)	61.6	71.5 (+9.9)	64.5	68.8 (+4.3)	72.0	**72.6** (+0.6)
	average(d)	52.6	47.1	55.1 (+7.9)	61.6	59.3 (-2.3)	64.7	58.1 (-6.6)	*72.0*	60.5 (-11.5)
	max(d)	57.4	39.5	66.7 (+27.1)	66.0	69.2 (+3.1)	58.5	67.4 (+9.0)	64.9	*70.2* (+5.3)
SMTnews	$+$	40.7	31.9	37.1 (+5.2)	44.2	48.3 (+4.1)	37.7	44.8 (+7.1)	**60.8**	56.3 (-4.6)
	average(d)	30.8	33.2	31.3 (-1.8)	44.2	38.1 (-6.1)	39.7	32.7 (-7.0)	**60.8**	37.9 (-22.9)
	max(d)	42.1	27.9	46.5 (+18.7)	51.2	55.6 (+4.3)	36.0	47.3 (+11.3)	*59.0*	49.9 (-9.0)
STS13	$+$	61.5	27.9	59.7 (+31.7)	52.7	62.2 (+9.5)	45.9	60.7 (+14.8)	66.4	**67.2** (+0.8)
	average(d)	32.2	28.3	37.5 (+9.2)	52.7	42.2 (-10.5)	46.1	41.6 (-4.5)	*66.4*	43.5 (-22.9)
	max(d)	55.5	19.8	60.7 (+41.0)	57.4	*63.4* (+6.1)	35.3	62.5 (+27.1)	57.1	62.9 (+5.7)
FNWN	$+$	42.8	8.2	45.9 (+37.8)	39.7	44.9 (+5.3)	30.7	47.6 (+16.9)	44.6	**50.9** (+6.3)
	average(d)	23.3	9.0	35.1 (+26.0)	39.7	29.0 (-10.7)	30.8	27.1 (-3.7)	*44.6*	33.1 (-11.5)
	max(d)	3.8	3.2	15.0 (+11.8)	*31.4*	24.6 (-6.8)	20.9	21.5 (+0.7)	22.3	21.2 (-1.1)
OnWN	$+$	65.1	8.5	59.1 (+50.6)	53.1	70.1 (+17.0)	42.4	62.5 (+20.1)	*75.4*	74.4 (-1.0)
	average(d)	15.7	9.2	25.3 (+16.1)	53.1	36.5 (-16.6)	42.3	32.5 (-9.8)	*75.4*	34.0 (-41.3)
	max(d)	79.7	1.3	79.2 (+78.0)	68.6	82.0 (+13.4)	30.0	81.0 (+51.0)	71.3	**84.6** (+13.3)
HDL	$+$	63.5	47.5	63.5 (+16.1)	55.8	60.6 (+4.9)	52.4	62.7 (+10.3)	65.1	**65.9** (+0.8)
	average(d)	46.7	47.5	47.3 (-0.2)	55.8	49.8 (-6.0)	52.8	52.2 (-0.6)	*65.1*	53.2 (-11.9)
	max(d)	50.5	37.8	58.4 (+20.6)	55.5	*59.3* (+3.8)	43.0	58.9 (+15.9)	55.3	57.1 (+1.8)
STS14	$+$	64.9	31.9	61.8 (+29.9)	54.1	65.7 (+11.5)	53.0	65.2 (+12.2)	70.5	**71.1** (+0.5)
	average(d)	40.0	32.0	41.5 (+9.6)	54.1	48.0 (-6.1)	53.2	46.3 (-6.9)	*70.5*	49.9 (-20.6)
	max(d)	57.8	22.4	62.4 (+39.9)	60.6	*67.1* (+6.5)	42.6	65.1 (+22.5)	61.6	66.3 (+4.7)
Deft Forum	$+$	43.0	22.3	32.3 (+10.0)	22.9	37.4 (+14.5)	25.7	35.5 (+9.8)	*48.0*	47.5 (-0.5)
	average(d)	25.7	22.4	19.8 (-2.6)	22.9	24.1 (+1.3)	25.1	19.7 (-5.3)	**48.0**	28.1 (-19.9)
	max(d)	38.4	17.3	38.4 (+21.1)	34.4	43.2 (+8.8)	22.0	41.0 (+19.0)	39.7	*43.5* (+3.8)
Deft News	$+$	59.9	50.0	65.2 (+15.2)	59.5	67.0 (+7.5)	56.8	66.9 (+10.1)	67.2	**69.9** (+2.7)
	average(d)	37.9	49.4	39.2 (-10.2)	59.5	46.9 (-12.6)	57.0	44.9 (-12.1)	*67.2*	46.3 (-20.9)
	max(d)	65.7	44.7	66.3 (+21.6)	66.7	*69.5* (+2.7)	44.4	67.8 (+23.5)	65.1	67.7 (+2.6)
HDL	$+$	62.5	41.2	60.7 (+19.5)	52.2	56.3 (+4.1)	51.5	60.2 (+8.7)	63.0	**63.5** (+0.6)
	average(d)	45.3	41.0	43.3 (+2.3)	52.2	45.7 (-6.5)	52.0	45.8 (-6.2)	*63.0*	50.7 (-12.3)
	max(d)	45.8	30.8	54.0 (+23.2)	50.7	56.0 (+5.3)	41.6	*56.5* (+14.9)	53.9	54.2 (+0.3)
Images	$+$	72.9	12.3	73.5 (+61.3)	60.6	75.8 (+15.1)	59.8	75.4 (+15.6)	80.0	**81.5** (+1.5)
	average(d)	28.5	12.0	34.7 (+22.6)	60.6	43.7 (-17.0)	60.1	43.0 (-17.1)	*80.0*	45.1 (-34.9)
	max(d)	64.7	7.0	75.5 (+68.5)	73.5	*78.5* (+5.0)	49.9	76.2 (+26.3)	73.6	78.2 (+4.6)
OnWN	$+$	70.6	27.9	70.2 (+42.3)	61.8	77.5 (+15.7)	58.3	74.1 (+15.8)	*81.6*	80.7 (-0.9)
	average(d)	34.4	28.8	43.1 (+14.3)	61.8	52.4 (-9.4)	58.2	49.0 (-9.2)	*81.6*	51.2 (-30.4)
	max(d)	79.6	13.6	80.9 (+67.3)	73.0	85.0 (+12.0)	44.9	84.1 (+39.3)	75.5	**86.4** (+10.9)
Tweet-News	$+$	68.7	44.7	59.0 (+14.3)	58.6	69.5 (+10.9)	57.3	68.3 (+10.9)	72.5	**73.2** (+0.7)
	average(d)	61.4	44.8	59.1 (+14.2)	58.6	65.1 (+6.5)	58.0	63.9 (+5.9)	*72.5*	67.3 (-5.2)
	max(d)	49.6	32.5	51.8 (+19.4)	58.4	*62.4* (+4.0)	45.9	57.0 (+11.1)	55.1	59.4 (+4.2)

TABLE A.5: **STS15-16 and Benchmark** scores in Pearson's $r \times 100$. These evaluations are done for before and after tuning the embeddings for a specific algebraic composition operation. We also trained embeddings without pretraining (*None*).

		None	Word2Vec		GloVe		fastText		Paragram	
		Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
STS15	$+$	67.4	36.9	68.7 (+31.9)	58.1	66.6 (+8.5)	58.1	68.8 (+10.7)	75.0	**75.2** (+0.2)
	average(d)	45.5	36.3	47.3 (+10.9)	58.1	51.3 (-6.8)	58.2	52.2 (-6.0)	*75.0*	54.6 (-20.3)
	max(d)	55.9	25.9	62.8 (+36.9)	64.3	*66.8* (+2.5)	47.5	65.2 (+17.7)	65.5	66.7 (+1.1)
Ans. Forums	$+$	48.8	19.7	56.2 (+36.6)	36.5	42.4 (+5.9)	35.0	48.1 (+13.0)	64.0	**65.3** (+1.3)
	average(d)	30.4	18.5	32.1 (+13.6)	36.5	33.0 (-3.5)	35.3	35.0 (-0.2)	*64.0*	40.2 (-23.8)
	max(d)	32.3	9.9	42.4 (+32.5)	*55.0*	50.8 (-4.2)	22.5	46.1 (+23.6)	52.9	50.2 (-2.7)
Ans. Students	$+$	71.2	48.4	76.5 (+28.1)	65.4	74.8 (+9.4)	68.9	76.1 (+7.2)	**78.0**	77.7 (-0.3)
	average(d)	52.4	48.1	54.6 (+6.5)	65.4	57.9 (-7.5)	68.7	59.3 (-9.4)	**78.0**	58.7 (-19.3)
	max(d)	65.4	32.1	70.0 (+37.9)	67.4	69.8 (+2.4)	63.0	72.8 (+9.8)	*73.8*	72.8 (-1.0)
Belief	$+$	66.2	35.5	60.1 (+24.6)	43.5	60.4 (+16.9)	43.9	64.0 (+20.1)	**75.9**	75.6 (-0.3)
	average(d)	45.4	35.6	46.1 (+10.6)	43.5	48.6 (+5.1)	44.2	49.9 (+5.7)	**75.9**	53.3 (-22.5)
	max(d)	45.3	28.4	49.1 (+20.8)	57.4	*62.7* (+5.2)	31.1	51.3 (+20.2)	55.3	56.2 (+0.9)
HDL	$+$	67.7	49.5	65.2 (+15.7)	59.5	63.2 (+3.7)	58.1	66.5 (+8.5)	69.7	**70.2** (+0.5)
	average(d)	53.2	48.7	51.7 (+3.0)	59.5	53.8 (-5.7)	58.1	56.0 (-2.1)	*69.7*	58.4 (-11.3)
	max(d)	53.8	36.2	58.5 (+22.3)	57.8	61.4 (+3.6)	46.0	*62.8* (+16.8)	59.5	62.8 (+3.3)
Images	$+$	73.3	22.0	75.1 (+53.1)	67.4	76.8 (+9.4)	66.1	76.5 (+10.4)	82.3	**82.5** (+0.3)
	average(d)	38.5	21.5	43.6 (+22.1)	67.4	52.8 (-14.6)	66.1	51.0 (-15.1)	*82.3*	54.6 (-27.6)
	max(d)	65.6	16.1	77.0 (+60.9)	75.6	*79.3* (+3.7)	54.2	76.6 (+22.4)	74.7	77.8 (+3.1)
STS16	$+$	61.4	33.2	58.8 (+25.6)	49.8	61.5 (+11.7)	44.0	58.5 (+14.5)	**69.3**	68.9 (-0.4)
	average(d)	33.9	32.4	35.6 (+3.3)	49.7	39.5 (-10.3)	44.1	39.0 (-5.1)	**69.3**	42.2 (-27.2)
	max(d)	52.0	25.7	57.3 (+31.5)	*62.1*	61.0 (-1.1)	34.8	56.1 (+21.3)	61.1	60.5 (-0.6)
Ans.-Ans.	$+$	51.8	18.8	41.1 (+22.3)	35.9	41.7 (+5.8)	16.3	36.9 (+20.6)	**65.2**	62.7 (-2.6)
	average(d)	22.7	17.6	19.9 (+2.3)	35.9	23.3 (-12.6)	15.7	18.3 (+2.5)	**65.2**	27.7 (-37.6)
	max(d)	34.9	17.3	43.6 (+26.2)	54.2	51.4 (-2.8)	11.3	37.4 (+26.1)	*57.7*	49.9 (-7.8)
HDL	$+$	64.6	48.7	**65.3** (+16.7)	57.3	62.7 (+5.4)	53.2	60.7 (+7.5)	64.8	64.5 (-0.3)
	average(d)	51.3	48.4	53.2 (+4.8)	57.2	55.8 (-1.5)	53.5	56.1 (+2.6)	*64.8*	58.5 (-6.3)
	max(d)	52.3	44.3	*59.9* (+15.6)	57.0	58.2 (+1.2)	46.4	55.3 (+8.9)	57.2	55.8 (-1.4)
Plagiarism	$+$	73.1	38.1	76.6 (+38.5)	55.8	77.7 (+21.9)	60.4	76.6 (+16.1)	79.3	**80.3** (+1.0)
	average(d)	44.4	36.2	49.5 (+13.3)	55.8	56.0 (+0.2)	60.4	58.8 (-1.6)	*79.3*	59.2 (-20.1)
	max(d)	54.5	18.8	64.2 (+45.5)	65.8	64.1 (-1.8)	40.2	65.2 (+25.1)	63.4	*67.1* (+3.7)
Post Editing	$+$	81.4	47.7	78.3 (+30.6)	54.2	72.8 (+18.6)	59.5	78.6 (+19.1)	82.2	**82.8** (+0.6)
	average(d)	60.5	47.6	60.1 (+12.5)	54.2	57.4 (+3.2)	59.7	62.1 (+2.4)	*82.2*	64.5 (-17.7)
	max(d)	66.1	44.9	70.0 (+25.1)	72.1	72.4 (+0.3)	55.1	71.1 (+16.0)	*75.0*	71.8 (-3.1)
Quest.-Quest.	$+$	32.7	9.8	30.2 (+20.3)	45.8	53.2 (+7.4)	30.6	39.0 (+8.4)	*53.8*	53.1 (-0.7)
	average(d)	-16.1	9.1	-10.1 (-19.2)	45.8	0.7 (-45.1)	31.0	-5.1 (-36.1)	*53.8*	-4.4 (-58.2)
	max(d)	53.5	-0.9	48.4 (+49.3)	**61.8**	59.2 (-2.6)	20.2	52.4 (+32.2)	51.4	58.7 (+7.3)
STS-Benchmark	$+$	52.7	44.2	58.4 (+14.3)	54.2	55.2 (+1.0)	51.1	57.4 (+6.3)	*63.9*	61.3 (-2.6)
	average(d)	45.7	44.8	50.5 (+5.6)	61.7	55.3 (-6.5)	54.8	54.6 (-0.2)	**69.6**	54.5 (-15.1)
	max(d)	60.0	58.7	64.4 (+5.8)	*68.5*	66.9 (-1.6)	61.7	65.0 (+3.3)	67.8	65.9 (-1.9)

TABLE A.6: **STS12-16** p-values for the Pearson correlations. Results that are not statistically significant are displayed in bold ($p > 0.001$). For additive composition (+) we display results for a single run.

		None	Word2Vec		GloVe		fastText		Paragram	
		Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
MSRpar	+	2.7e-32	4.5e-23	1.4e-36	8.7e-31	8.5e-24	9.0e-42	1.3e-40	1.4e-33	3.6e-30
	average(d)	3.0e-14	5.5e-22	4.9e-15	8.7e-31	4.6e-19	5.0e-41	2.2e-17	1.4e-33	1.4e-16
	max(d)	4.5e-19	2.1e-12	1.0e-30	2.3e-22	1.2e-18	6.1e-25	8.0e-26	7.1e-30	2.6e-23
MSRvid	+	1.2e-72	**7.4e-02**	1.7e-87	1.2e-88	4.4e-120	6.8e-62	2.1e-104	1.1e-153	9.5e-157
	average(d)	**5.3e-02**	**4.6e-02**	**6.5e-01**	1.2e-88	**1.0e-03**	2.6e-62	**2.2e-02**	1.1e-153	1.4e-04
	max(d)	6.5e-137	**4.5e-03**	7.1e-173	3.6e-147	4.6e-188	1.2e-36	3.4e-179	3.5e-160	7.7e-201
SMTeuroparl	+	1.4e-28	3.2e-08	3.9e-18	9.8e-09	2.7e-22	5.7e-08	5.9e-23	3.3e-23	7.8e-26
	average(d)	3.3e-22	8.8e-10	8.3e-14	9.8e-09	1.2e-16	2.5e-08	1.0e-14	3.3e-23	2.6e-17
	max(d)	1.3e-11	2.1e-05	1.3e-10	1.1e-06	1.2e-12	**1.1e-03**	2.9e-10	3.6e-06	3.5e-11
OnWN	+	1.3e-92	6.9e-43	6.6e-108	1.6e-79	2.6e-118	4.1e-88	2.0e-106	1.4e-120	7.6e-124
	average(d)	1.7e-54	9.5e-43	1.1e-60	1.6e-79	2.4e-72	3.2e-90	5.4e-69	1.2e-120	6.4e-76
	max(d)	6.3e-67	2.0e-29	1.6e-97	3.7e-95	6.2e-108	5.2e-70	1.2e-100	8.0e-91	1.8e-112
SMTnews	+	2.5e-17	1.0e-10	1.9e-14	1.6e-20	1.0e-24	4.0e-14	4.4e-21	9.9e-42	1.1e-34
	average(d)	3.4e-10	1.1e-11	1.6e-10	1.6e-20	3.0e-15	1.7e-16	2.1e-11	9.9e-42	4.1e-15
	max(d)	1.3e-18	1.5e-08	7.8e-23	4.5e-28	1.1e-33	1.2e-13	1.1e-23	1.0e-38	1.4e-26
FNWN	+	8.1e-10	**2.7e-01**	3.0e-11	1.6e-08	8.8e-11	1.6e-05	4.4e-12	1.2e-10	7.4e-14
	average(d)	**1.2e-03**	**2.2e-01**	7.6e-07	1.6e-08	5.1e-05	1.6e-05	1.6e-04	1.2e-10	3.2e-06
	max(d)	**6.0e-01**	**6.6e-01**	**4.0e-02**	1.0e-05	6.3e-04	**4.0e-03**	**2.9e-03**	**2.0e-03**	**3.4e-03**
OnWN	+	7.7e-69	**5.6e-02**	4.1e-54	4.4e-42	4.2e-84	8.1e-26	5.1e-62	5.0e-104	6.3e-100
	average(d)	1.9e-04	**2.9e-02**	1.2e-09	4.4e-42	4.3e-19	9.7e-26	3.2e-15	5.0e-104	1.1e-16
	max(d)	2.7e-124	**7.7e-01**	3.9e-122	2.3e-79	7.9e-138	3.9e-13	6.6e-132	2.1e-88	5.2e-155
HDL	+	5.8e-86	1.5e-43	4.6e-86	1.7e-62	1.9e-76	2.1e-53	3.8e-83	9.5e-92	1.3e-94
	average(d)	5.5e-42	1.6e-43	4.3e-43	1.7e-62	3.0e-48	5.5e-55	1.5e-53	9.5e-92	4.9e-56
	max(d)	8.5e-50	7.3e-27	9.6e-70	1.1e-61	2.6e-72	4.3e-35	4.1e-71	2.7e-61	4.3e-66
Deft Forum	+	1.2e-21	1.3e-06	2.2e-12	9.3e-07	2.4e-16	4.3e-08	8.2e-15	2.6e-27	1.0e-26
	average(d)	3.2e-08	1.6e-06	2.2e-05	9.2e-07	2.2e-07	7.0e-08	2.5e-05	2.6e-27	1.4e-09
	max(d)	2.7e-17	2.3e-04	3.1e-17	5.8e-14	7.1e-22	2.4e-06	1.1e-19	1.9e-18	3.3e-22
Deft News	+	1.2e-30	5.3e-20	1.2e-37	4.1e-30	2.1e-40	5.1e-27	2.5e-40	7.9e-41	2.7e-45
	average(d)	1.2e-11	7.3e-20	1.8e-12	4.1e-30	8.5e-18	3.0e-27	2.6e-16	7.9e-41	2.4e-17
	max(d)	1.7e-38	3.7e-16	2.2e-39	4.7e-40	1.5e-44	6.7e-16	8.6e-42	1.6e-37	1.4e-41
HDL	+	1.9e-82	6.1e-33	1.0e-76	1.1e-53	4.8e-64	1.7e-51	3.5e-75	4.4e-84	5.7e-86
	average(d)	3.9e-39	1.1e-31	1.3e-35	1.1e-53	6.7e-40	2.8e-53	3.2e-40	4.4e-84	4.0e-50
	max(d)	3.2e-40	6.5e-18	5.5e-58	3.8e-50	5.1e-63	9.2e-33	1.7e-64	8.5e-58	1.7e-58
Images	+	4.7e-125	**2.1e-03**	1.8e-128	1.6e-76	7.5e-141	7.0e-73	1.5e-138	9.0e-168	1.9e-179
	average(d)	1.6e-15	9.5e-04	1.3e-22	1.6e-76	2.7e-36	8.7e-75	4.4e-35	9.0e-168	8.7e-39
	max(d)	2.7e-90	**5.6e-02**	5.3e-139	4.5e-128	2.2e-157	1.7e-48	4.1e-143	6.3e-129	7.8e-156
OnWN	+	3.2e-114	5.7e-14	2.6e-112	4.4e-80	3.8e-151	2.4e-69	2.9e-131	4.1e-180	2.9e-173
	average(d)	2.9e-22	8.8e-16	3.3e-35	4.4e-80	4.5e-54	3.4e-69	1.5e-46	4.1e-180	2.7e-51
	max(d)	1.4e-165	1.8e-04	5.1e-175	1.3e-125	3.0e-210	2.1e-38	3.5e-202	5.5e-139	4.3e-225
Tweet-News	+	7.3e-106	4.4e-36	1.4e-71	2.5e-70	4.1e-109	4.3e-65	5.7e-104	3.0e-123	9.1e-127
	average(d)	4.8e-79	2.3e-38	1.0e-71	2.5e-70	1.5e-91	1.2e-68	3.1e-87	3.0e-123	5.3e-100
	max(d)	8.7e-48	7.4e-20	8.5e-53	7.0e-70	2.6e-82	2.6e-40	1.0e-65	7.8e-61	1.3e-72
Ans. Forums	+	8.2e-24	8.0e-05	1.2e-32	2.9e-13	8.5e-18	2.5e-11	4.5e-23	1.3e-44	6.4e-47
	average(d)	1.8e-09	3.1e-04	2.0e-10	2.9e-13	5.7e-11	2.0e-12	2.9e-12	1.3e-44	5.2e-16
	max(d)	1.6e-10	**5.5e-02**	8.0e-18	4.6e-31	5.3e-26	1.1e-05	4.1e-21	2.2e-28	2.6e-25
Ans. Students	+	5.0e-117	2.4e-46	8.5e-145	8.2e-93	2.0e-135	2.5e-109	1.6e-142	2.0e-154	2.6e-152
	average(d)	3.9e-54	1.1e-44	1.3e-59	8.2e-93	1.9e-68	7.7e-106	1.5e-72	2.0e-154	9.5e-71
	max(d)	8.2e-93	1.9e-19	1.6e-111	1.8e-100	2.2e-110	3.3e-84	1.4e-124	5.0e-130	5.5e-125
Belief	+	9.8e-49	2.5e-12	3.5e-38	9.0e-19	1.0e-38	1.1e-18	1.4e-44	2.2e-71	1.3e-70
	average(d)	1.8e-20	1.3e-12	3.7e-21	9.0e-19	1.3e-23	2.3e-19	4.9e-25	2.2e-71	6.3e-29
	max(d)	2.4e-20	2.3e-08	3.6e-24	2.8e-34	2.6e-42	7.6e-10	1.6e-26	1.9e-31	1.2e-32
HDL	+	1.7e-101	5.3e-48	5.5e-92	4.3e-73	8.4e-85	2.3e-69	4.4e-97	4.5e-110	2.5e-112
	average(d)	4.2e-56	6.0e-46	1.5e-52	4.3e-73	1.8e-57	6.0e-69	4.1e-63	4.5e-110	1.2e-69
	max(d)	2.1e-57	1.1e-24	3.9e-70	3.7e-68	5.5e-79	1.6e-40	1.4e-83	5.5e-73	1.5e-83
Images	+	2.5e-127	6.5e-09	5.7e-137	1.7e-100	8.4e-147	2.9e-95	4.1e-145	1.5e-185	6.1e-188
	average(d)	6.4e-28	2.7e-09	3.7e-36	1.7e-100	4.4e-55	2.4e-95	7.3e-51	1.5e-185	1.4e-59
	max(d)	1.2e-93	9.8e-06	4.6e-148	8.3e-140	5.4e-163	1.8e-58	9.2e-146	7.4e-135	3.5e-153
Ans.-Ans.	+	7.5e-19	**2.0e-03**	9.1e-12	4.0e-09	4.3e-12	**1.2e-02**	1.3e-09	3.4e-32	3.8e-29
	average(d)	2.7e-04	**4.8e-03**	**1.4e-03**	4.0e-09	1.8e-04	**1.2e-02**	**3.5e-03**	3.4e-32	7.5e-06
	max(d)	1.1e-08	**5.7e-03**	3.5e-13	8.4e-21	1.6e-18	**7.3e-02**	7.9e-10	6.0e-24	2.3e-17
HDL	+	7.6e-31	1.1e-16	1.1e-31	3.7e-23	1.4e-28	3.2e-20	1.9e-26	5.2e-31	1.1e-30
	average(d)	3.9e-18	4.7e-16	1.3e-19	4.6e-23	9.6e-22	7.2e-20	4.4e-22	5.2e-31	2.9e-24
	max(d)	7.0e-19	2.2e-13	1.3e-25	7.3e-23	5.2e-24	1.1e-14	2.6e-21	5.3e-23	9.5e-22
Plagiarism	+	9.6e-40	2.1e-08	1.2e-45	3.1e-20	1.1e-47	8.3e-25	1.3e-45	5.2e-51	3.7e-53
	average(d)	1.5e-12	1.6e-08	1.2e-15	3.1e-20	2.1e-20	2.6e-24	8.4e-23	5.2e-51	4.1e-23
	max(d)	3.4e-19	**4.3e-03**	4.0e-28	5.8e-30	5.4e-28	2.5e-10	2.9e-29	3.3e-27	2.1e-31
Post Editing	+	4.6e-59	1.1e-14	6.6e-52	5.0e-20	1.8e-41	8.2e-25	1.9e-52	3.3e-61	7.8e-63
	average(d)	8.8e-26	3.4e-15	2.3e-25	5.0e-20	8.8e-23	5.9e-25	1.9e-27	3.3e-61	4.1e-30
	max(d)	4.9e-32	1.6e-13	3.1e-37	1.7e-40	6.5e-41	9.7e-21	8.1e-39	3.0e-45	5.6e-40
Quest.-Quest.	+	1.4e-06	**1.5e-01**	9.1e-06	3.0e-12	1.2e-16	6.3e-06	5.2e-09	4.8e-17	1.3e-16
	average(d)	**1.9e-02**	**1.9e-01**	**1.4e-01**	3.0e-12	**9.2e-01**	4.8e-06	**4.7e-01**	4.8e-17	**5.3e-01**
	max(d)	7.1e-17	**9.0e-01**	1.1e-13	2.1e-23	3.8e-21	**3.4e-03**	4.1e-16	1.7e-15	1.0e-20

TABLE A.7: Table on the statistical significance of the various evaluation measures on word2vec and GloVe. We executed the tests for additive composition (+) multiple times to determine statistical significance. Results that are **not significant** are displayed in bold ($p > 0.001$).

	Word2Vec +							GloVe +						
	n	μ_{pre}	σ_{pre}	μ_{tuned}	σ_{tuned}	p	t	n	μ_{pre}	σ_{pre}	μ_{tuned}	σ_{tuned}	p	t
CompVecEval (MRR)	12	16.8	0.43	23.0	0.29	4.5e-22	-40.1	12	11.9	0.01	26.6	0.14	3.0e-42	-333.5
MNR	12	83.9	0.59	87.9	0.12	1.7e-16	-22.1	12	83.5	0.01	92.2	0.07	2.6e-44	-414.0
MAP	12	15.3	0.39	21.2	0.23	9.3e-23	-43.2	12	10.8	0.01	24.6	0.14	8.5e-42	-318.2
MP@10	12	2.9	0.06	4.4	0.05	4.1e-26	-61.5	12	2.2	0.00	4.9	0.03	6.6e-40	-261.2
WS-353	*12*	*70.4*	*0.09*	*70.6*	*1.25*	**7.5e-01**	**-0.3**	12	71.9	0.00	76.2	0.50	9.4e-19	-28.2
WS-353 (Sim.)	*12*	*80.3*	*0.01*	*79.6*	*1.08*	**4.0e-02**	**2.2**	12	81.1	0.00	82.6	0.61	3.6e-08	-8.2
WS-353 (Rel.)	12	63.5	0.18	60.8	2.05	2.3e-04	4.4	12	65.9	0.00	71.2	0.94	4.4e-15	-18.9
MC-30	*12*	*83.8*	*0.00*	*84.8*	*2.97*	**2.7e-01**	**-1.1**	12	82.6	0.00	86.2	2.88	4.3e-04	-4.1
RG-65	12	76.1	0.00	80.2	1.81	1.4e-07	-7.6	12	76.8	0.00	83.0	1.34	2.9e-13	-15.4
Rare Word	12	58.0	0.11	55.1	0.85	1.5e-10	11.2	12	52.5	0.03	57.1	0.49	1.5e-19	-30.7
MTurk-3k	12	72.9	0.24	69.3	0.48	9.2e-17	22.7	12	80.0	0.00	79.0	0.31	2.9e-10	10.8
MTurk-287	*12*	*66.5*	*0.77*	*66.6*	*1.86*	**9.1e-01**	**-0.1**	12	72.3	0.00	73.5	0.53	3.0e-07	-7.2
MTurk-771	12	66.8	0.00	69.3	0.78	3.3e-10	-10.7	12	70.9	0.00	72.2	0.50	1.2e-08	-8.8
YP-130	12	49.8	0.00	64.0	1.88	1.2e-17	-25.0	12	51.9	0.00	60.8	1.03	5.1e-19	-29.0
SimLex	12	44.0	0.00	51.2	0.64	2.0e-21	-37.5	12	40.2	0.00	50.4	0.54	3.2e-26	-62.2
Verb-143	12	49.7	0.00	36.0	3.35	3.5e-12	13.6	12	35.8	0.00	29.1	1.28	2.5e-14	17.4
SimVerb	12	34.2	0.00	40.0	0.69	1.2e-18	-27.9	12	25.4	0.00	34.6	0.28	2.0e-31	-107.5
MR	12	75.3	0.15	74.3	0.29	1.0e-09	10.1	*12*	*76.2*	*0.05*	*76.0*	*0.20*	**2.1e-02**	**2.5**
CR	*12*	*77.2*	*0.29*	*77.0*	*0.53*	**3.3e-01**	**1.0**	*12*	*78.9*	*0.07*	*78.7*	*0.33*	**1.0e-02**	**2.8**
SUBJ	12	89.6	0.15	88.9	0.27	7.1e-08	7.9	12	90.4	0.03	90.0	0.18	6.6e-07	6.9
MPQA	12	86.2	0.11	85.3	0.20	1.3e-12	14.3	*12*	*86.0*	*0.00*	*85.8*	*0.19*	**3.1e-02**	**2.3**
SST	12	79.3	0.36	78.1	0.45	4.2e-07	7.1	12	79.4	0.08	78.7	0.42	3.1e-05	5.2
TREC	12	75.1	1.55	78.2	1.06	1.7e-05	-5.5	*12*	*81.2*	*0.27*	*80.1*	*1.60*	**3.5e-02**	**2.3**
MRPC	*12*	*68.7*	*0.80*	*69.3*	*0.86*	**1.3e-01**	**-1.6**	*12*	*69.6*	*0.14*	*69.1*	*0.80*	**9.4e-02**	**1.7**
SICK-E	12	73.8	0.63	77.0	0.49	7.5e-12	-13.1	12	75.8	0.00	76.8	0.71	2.2e-04	-4.4
SICK-R	12	73.4	0.73	78.2	0.53	1.8e-14	-17.6	12	78.3	0.00	79.8	0.29	3.0e-14	-17.2
STS14	12	31.9	0.23	62.1	0.25	5.9e-41	-291.5	12	54.1	0.00	65.7	0.11	4.4e-43	-364.0
STS15	12	36.9	0.49	68.6	0.35	5.6e-36	-173.1	12	58.1	0.00	66.6	0.14	3.4e-37	-196.7

A.5 Determining statistical significance

We were unable to run each specific experiment multiple times, but we did manage to run the experiments with additive composition and ngram targets multiple times. Using these experiments we determined statistical significance for the difference before and after training. We did this using Student's t-test [50]. All μ-values, σ-values, p-values and t-statistics are shown in Tables A.7 and A.8.

We see that we were unable to determine statistical significance for some sentence representation evaluation methods, as well as some word representation evaluation methods. Specifically MC-30; the Stanford Simple Sentiment and Topic, Classification; TREC; and MRPC do not show a significantly different result before and after tuning for some pretrained embeddings.

TABLE A.8: Table on the statistical significance of the various evaluation measures on fastText and Paragram. We executed the tests for additive composition (+) multiple times to determine statistical significance. Results that are **not significant** are displayed in bold ($p > 0.001$).

	fastText +							Paragram +						
	n	μ_{pre}	σ_{pre}	μ_{tuned}	σ_{tuned}	p	t	n	μ_{pre}	σ_{pre}	μ_{tuned}	σ_{tuned}	p	t
CompVecEval (MRR)	12	20.7	0.04	26.6	0.20	1.4e-30	-98.2	17	26.5	0.02	29.8	0.10	2.9e-44	-121.9
MNR	12	86.3	0.03	90.9	0.07	3.1e-37	-197.5	17	90.3	0.02	92.8	0.04	4.0e-51	-199.8
MAP	12	18.9	0.04	24.6	0.17	1.3e-31	-109.5	17	24.8	0.02	27.8	0.10	2.5e-44	-122.4
MP@10	12	3.6	0.01	4.8	0.05	1.7e-28	-79.0	17	5.2	0.00	5.7	0.05	6.5e-32	-49.9
WS-353	12	74.5	0.00	69.7	0.86	8.7e-15	18.3	17	73.1	0.00	70.3	1.01	1.5e-12	11.1
WS-353 (Sim.)	12	78.7	0.00	73.8	1.02	1.3e-13	16.0	17	80.6	0.00	78.5	1.03	1.6e-09	8.3
WS-353 (Rel.)	12	69.5	0.00	61.8	1.21	4.6e-16	21.0	17	67.0	0.00	62.4	1.15	1.1e-16	15.8
MC-30	12	85.5	0.00	76.6	1.95	4.1e-13	15.1	*17*	*74.1*	*0.00*	*76.9*	*3.21*	**1.4e-03**	**-3.5**
RG-65	12	79.4	0.00	76.2	1.71	4.0e-06	6.1	17	73.6	0.00	77.8	1.29	2.1e-14	-13.1
Rare Word	12	54.3	0.00	53.7	0.28	3.9e-06	6.1	17	63.5	0.05	63.8	0.29	7.8e-04	-3.7
MTurk-3k	12	75.8	0.00	68.6	0.55	8.9e-23	43.2	17	75.9	0.00	74.6	0.23	6.9e-21	22.0
MTurk-287	12	68.2	0.00	61.7	1.39	2.5e-13	15.5	17	67.0	0.00	64.7	0.72	9.4e-14	12.4
MTurk-771	12	66.9	0.00	66.1	0.70	7.8e-04	3.9	17	70.1	0.00	71.5	0.40	1.8e-14	-13.2
YP-130	12	49.9	0.00	58.2	2.12	7.6e-12	-13.1	17	65.5	0.00	68.6	1.17	5.8e-12	-10.6
SimLex	12	37.3	0.00	46.1	0.53	4.7e-25	-55.0	17	66.2	0.00	67.2	0.28	1.7e-15	-14.4
Verb-143	12	38.9	0.00	29.7	3.51	1.4e-08	8.7	17	58.2	0.00	54.4	1.44	5.0e-12	10.6
SimVerb	12	23.0	0.00	33.2	0.81	1.8e-22	-41.9	17	56.8	0.00	57.9	0.27	5.4e-17	-16.2
MR	12	74.0	0.20	73.5	0.20	1.8e-05	5.5	17	74.2	0.00	74.5	0.25	1.9e-06	-5.8
CR	*12*	*76.9*	*0.32*	*76.9*	*0.59*	**9.4e-01**	**0.1**	*17*	*78.0*	*0.07*	*77.8*	*0.39*	**5.7e-02**	**2.0**
SUBJ	12	89.8	0.16	89.1	0.19	1.5e-09	9.9	17	88.1	0.01	88.3	0.19	8.3e-04	-3.7
MPQA	12	85.6	0.08	85.0	0.26	7.4e-07	6.8	*17*	*86.0*	*0.00*	*86.0*	*0.17*	**2.1e-01**	**1.3**
SST	*12*	*77.2*	*0.35*	*77.2*	*0.60*	**8.6e-01**	**0.2**	17	79.1	0.00	78.7	0.33	7.1e-05	4.6
TREC	*12*	*79.5*	*0.87*	*78.8*	*1.17*	**1.1e-01**	**1.7**	*17*	*81.0*	*0.11*	*80.9*	*0.78*	**4.4e-01**	**0.8**
MRPC	*12*	*68.4*	*0.66*	*68.5*	*0.72*	**5.8e-01**	**-0.6**	17	69.9	0.00	70.9	0.69	1.4e-06	-5.9
SICK-E	*12*	*76.8*	*0.41*	*76.7*	*0.46*	**6.0e-01**	**0.5**	17	77.5	0.00	77.8	0.22	1.2e-07	-6.8
SICK-R	*12*	*78.2*	*0.12*	*78.1*	*0.90*	**6.1e-01**	**0.5**	*17*	*80.3*	*0.00*	*80.2*	*0.20*	**4.7e-03**	**3.0**
STS14	12	53.0	0.13	65.1	0.23	6.6e-35	-154.7	17	70.5	0.00	71.3	0.13	4.2e-21	-22.4
STS15	12	58.1	0.17	68.6	0.23	9.0e-33	-123.7	17	75.0	0.00	75.1	0.09	2.0e-07	-6.6

Appendix B

Miscellaneous results on learning to compose

In this appendix we will present miscellaneous results from the experiments described in Chapter 5 on learning composition functions using artificial neural networks.

B.1 Results on word representation evaluations

In Table B.1 we see the results for all word representation evaluation methods described in Section 4.3. We omitted the results for + projections and the RNN from some of the evaluation methods since they where not the best performing composition functions.

B.2 Results on sentence representation evaluations

In Table B.2 we see the results for all sentence representation evaluation methods described in Section 4.4. We omitted the results for + projections and the RNN from some of the evaluation methods since they where not the best performing composition functions.

B.3 Results on semantic textual similarity

In table B.3 and B.4 we show the results of the evaluations on all subsets for the STS tasks from 2012 to 2016, in addition we show the results of the STS Benchmark. Not all results are statistically significant, however most of them are. All corresponding p-vales can be found in B.5.

These are just the results for the algebraic composition functions, for the learned composition function we refer to the next appendix (B.3). We left out the + projection, RNN, CNN-3 composition functions, since it is clear from all earlier results that these were clearly underperforming.

TABLE B.1: **Word representation evaluation** scores in Spearman's r $\times 100$, for our learned composition functions. It is important to note that under *original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

		Word2Vec		GloVe		fastText		Paragram	
		Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
WS-353	+ Proj.	70.4	70.8 (+0.4)	71.9	73.1 (+1.2)	*74.5*	72.0 (-2.4)	73.1	73.3 (+0.2)
	RNN	70.4	66.1 (-4.3)	71.9	73.2 (+1.2)	74.5	74.8 (+0.4)	73.1	*75.1* (+2.0)
	GRU	70.4	70.3 (-0.1)	71.9	72.9 (+0.9)	74.5	**76.5** (+2.1)	73.1	74.4 (+1.2)
	Bi-GRU	70.7	68.5 (-2.3)	71.9	73.2 (+1.2)	74.5	73.9 (-0.6)	73.1	*74.8* (+1.6)
	CNN-3	70.4	67.4 (-3.0)	71.9	72.5 (+0.5)	*74.5*	73.4 (-1.1)	73.1	73.6 (+0.4)
	CNN-3,5,7,9	70.4	68.6 (-1.7)	71.9	71.8 (-0.1)	*74.5*	72.7 (-1.7)	73.1	72.6 (-0.5)
WS-353 (Rel.)	GRU	63.4	61.6 (-1.8)	65.9	66.9 (+1.0)	69.5	**72.4** (+3.0)	67.0	68.2 (+1.2)
	Bi-GRU	64.0	59.0 (-5.0)	65.9	66.6 (+0.7)	*69.5*	67.3 (-2.2)	67.0	69.0 (+2.0)
	CNN-3	63.4	56.9 (-6.6)	65.9	67.0 (+1.1)	*69.5*	67.2 (-2.2)	67.0	68.1 (+1.1)
	CNN-3,5,7,9	63.4	58.0 (-5.4)	65.9	66.0 (+0.1)	*69.5*	65.2 (-4.3)	67.0	65.2 (-1.7)
WS-353 (Sim.)	GRU	80.3	80.9 (+0.6)	81.1	**83.3** (+2.2)	78.7	80.5 (+1.8)	80.6	81.8 (+1.1)
	Bi-GRU	80.3	78.7 (-1.5)	81.1	*83.0* (+2.0)	78.7	80.3 (+1.6)	80.6	82.1 (+1.4)
	CNN-3	80.3	79.3 (-1.0)	81.1	*82.1* (+1.1)	78.7	78.6 (-0.1)	80.6	80.5 (-0.1)
	CNN-3,5,7,9	80.3	81.1 (+0.8)	*81.1*	80.9 (-0.1)	78.7	78.6 (-0.1)	80.6	80.7 (+0.0)
SimLex	+ Proj.	44.0	46.6 (+2.6)	40.2	44.4 (+4.2)	37.3	43.6 (+6.3)	66.2	*66.7* (+0.6)
	RNN	44.0	44.3 (+0.3)	40.2	43.5 (+3.2)	37.3	39.7 (+2.4)	*66.2*	66.1 (-0.1)
	GRU	44.0	47.3 (+3.3)	40.2	44.6 (+4.3)	37.3	43.8 (+6.4)	66.2	**67.2** (+1.0)
	Bi-GRU	44.0	51.1 (+7.1)	40.2	45.4 (+5.2)	37.3	43.3 (+6.0)	66.2	*67.2* (+1.0)
	CNN-3	44.0	46.2 (+2.2)	40.2	44.0 (+3.7)	37.3	41.6 (+4.3)	66.2	*66.9* (+0.7)
	CNN-3,5,7,9	44.0	47.0 (+3.0)	40.2	44.3 (+4.1)	37.3	40.3 (+2.9)	66.2	*67.2* (+1.0)
SimVerb	+ Proj.	34.2	35.7 (+1.5)	25.4	28.5 (+3.0)	23.0	29.4 (+6.4)	56.8	*57.3* (+0.5)
	RNN	34.2	29.4 (-4.8)	25.4	27.1 (+1.7)	23.0	24.3 (+1.3)	*56.8*	56.0 (-0.8)
	GRU	34.2	37.2 (+3.1)	25.4	29.7 (+4.2)	23.0	29.5 (+6.5)	56.8	*57.9* (+1.1)
	Bi-GRU	34.2	36.9 (+2.8)	25.4	29.5 (+4.1)	23.0	29.9 (+6.9)	56.8	**58.1** (+1.3)
	CNN-3	34.2	33.1 (-1.1)	25.4	28.3 (+2.9)	23.0	26.2 (+3.2)	56.8	*57.6* (+0.8)
	CNN-3,5,7,9	34.2	33.0 (-1.1)	25.4	27.8 (+2.4)	23.0	26.6 (+3.6)	56.8	*57.3* (+0.5)
RG-65	GRU	76.1	*81.3* (+5.2)	76.8	79.4 (+2.6)	79.4	79.6 (+0.3)	73.6	74.0 (+0.4)
	Bi-GRU	76.1	80.4 (+4.3)	76.8	80.6 (+3.8)	79.4	*81.3* (+2.0)	73.6	78.8 (+5.2)
	CNN-3	76.1	79.3 (+3.2)	76.8	78.6 (+1.7)	79.4	*81.1* (+1.8)	73.6	78.3 (+4.8)
	CNN-3,5,7,9	76.1	79.1 (+3.0)	76.8	77.7 (+0.9)	79.4	*79.9* (+0.5)	73.6	73.2 (-0.3)
MC-30	GRU	83.8	*85.5* (+1.7)	82.6	84.2 (+1.5)	*85.5*	77.3 (-8.1)	74.1	74.1 (-0.0)
	Bi-GRU	83.8	*85.7* (+1.9)	82.6	84.1 (+1.5)	85.5	85.4 (-0.1)	74.1	80.0 (+5.9)
	CNN-3	83.8	*86.5* (+2.7)	82.6	84.9 (+2.2)	85.5	80.5 (-5.0)	74.1	77.8 (+3.7)
	CNN-3,5,7,9	83.8	85.7 (+1.9)	82.6	81.3 (-1.4)	85.5	**87.3** (+1.9)	74.1	74.5 (+0.4)
Rare Word	GRU	57.8	54.0 (-3.8)	52.5	53.5 (+1.0)	54.3	52.2 (-2.1)	*63.5*	63.4 (-0.1)
	Bi-GRU	57.9	54.2 (-3.7)	52.5	53.3 (+0.8)	54.3	52.7 (-1.5)	63.6	*63.8* (+0.2)
	CNN-3	58.1	53.3 (-4.8)	52.5	53.4 (+0.9)	54.3	52.8 (-1.5)	63.4	*63.9* (+0.5)
	CNN-3,5,7,9	58.0	54.2 (-3.8)	52.5	52.7 (+0.3)	54.3	53.4 (-0.8)	*63.5*	63.2 (-0.3)
YP-130	GRU	49.8	53.1 (+3.3)	51.9	57.4 (+5.5)	49.9	58.1 (+8.2)	65.5	*67.1* (+1.6)
	Bi-GRU	49.8	60.4 (+10.6)	51.9	55.4 (+3.5)	49.9	53.7 (+3.9)	65.5	*65.7* (+0.2)
	CNN-3	49.8	56.1 (+6.2)	51.9	54.3 (+2.5)	49.9	53.7 (+3.8)	*65.5*	63.2 (-2.3)
	CNN-3,5,7,9	49.8	60.4 (+10.5)	51.9	55.5 (+3.6)	49.9	52.3 (+2.4)	65.5	**69.4** (+3.9)
Verb-143	GRU	49.7	39.5 (-10.3)	35.8	33.8 (-1.9)	38.9	38.7 (-0.2)	58.2	*59.4* (+1.2)
	Bi-GRU	49.7	39.8 (-9.9)	35.8	38.3 (+2.6)	38.9	30.6 (-8.3)	58.2	**59.4** (+1.2)
	CNN-3	49.7	41.8 (-8.0)	35.8	37.4 (+1.6)	38.9	37.7 (-1.2)	*58.2*	55.3 (-2.9)
	CNN-3,5,7,9	49.7	41.9 (-7.9)	35.8	39.2 (+3.4)	38.9	29.1 (-9.8)	58.2	*58.2* (+0.0)
MTurk-3k	GRU	72.7	70.6 (-2.2)	80.0	*80.4* (+0.4)	75.8	75.1 (-0.7)	75.9	77.6 (+1.7)
	Bi-GRU	73.1	70.3 (-2.8)	80.0	**80.9** (+1.0)	75.8	74.5 (-1.3)	75.9	*77.7* (+1.9)
	CNN-3	72.9	71.5 (-1.4)	80.0	*80.1* (+0.1)	75.8	74.8 (-1.1)	75.9	76.4 (+0.5)
	CNN-3,5,7,9	72.8	71.0 (-1.8)	80.0	*80.8* (+0.8)	75.8	75.0 (-0.8)	75.9	77.3 (+1.4)
MTurk-287	GRU	66.2	66.5 (+0.3)	72.3	*72.5* (+0.2)	68.2	67.3 (-0.9)	67.0	68.0 (+1.0)
	Bi-GRU	66.2	68.5 (+2.3)	72.3	*72.9* (+0.6)	68.2	66.1 (-2.1)	67.0	67.5 (+0.6)
	CNN-3	67.5	64.9 (-2.6)	*72.3*	71.3 (-1.0)	68.2	67.1 (-1.1)	67.0	67.5 (+0.6)
	CNN-3,5,7,9	66.4	61.7 (-4.7)	*72.3*	71.8 (-0.6)	68.2	68.4 (+0.2)	67.0	66.3 (-0.7)
MTurk-771	GRU	66.8	66.8 (+0.0)	70.9	72.0 (+1.2)	66.9	67.3 (+0.4)	70.1	**72.7** (+2.5)
	Bi-GRU	66.8	68.2 (+1.4)	70.9	*72.1* (+1.3)	66.9	69.0 (+2.1)	70.1	71.9 (+1.7)
	CNN-3	66.8	66.9 (+0.1)	70.9	*72.2* (+1.3)	66.9	66.7 (-0.2)	70.1	71.9 (+1.8)
	CNN-3,5,7,9	66.8	65.2 (-1.6)	70.9	*71.4* (+0.6)	66.9	66.1 (-0.8)	70.1	70.9 (+0.7)

TABLE B.2: **Sentence representation evaluation** scores for STS and SICK-R in Pearson's $r \times 100$, all other scores in accuracies $\times 100$. It is important to note that under *Original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

		Word2Vec		GloVe		fastText		Paragram	
		Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
TREC	GRU	81.0	82.0 (+1.0)	81.8	79.2 (-2.6)	**84.8**	79.0 (-5.8)	83.4	80.4 (-3.0)
	Bi-GRU	84.4	83.4 (-1.0)	79.0	81.0 (+2.0)	**84.8**	81.8 (-3.0)	84.4	78.6 (-5.8)
	CNN-3	63.6	71.0 (+7.4)	*77.4*	77.0 (-0.4)	79.0	*80.6* (+1.6)	63.8	*77.4* (+13.6)
	CNN-3,5,7,9	70.6	73.0 (+2.4)	79.6	80.4 (+0.8)	81.2	*82.2* (+1.0)	71.4	76.6 (+5.2)
MRPC	+ Proj.	70.0	68.8 (-1.2)	70.0	67.0 (-3.0)	*70.1*	69.1 (-1.0)	68.9	68.2 (-0.8)
	RNN	69.4	68.5 (-0.9)	69.6	68.3 (-1.3)	*70.8*	68.3 (-2.5)	67.4	67.1 (-0.3)
	GRU	70.1	72.7 (+2.6)	71.2	**73.2** (+2.0)	72.7	72.1 (-0.6)	72.9	72.3 (-0.6)
	Bi-GRU	71.5	72.6 (+1.0)	72.6	*72.7* (+0.1)	72.0	72.6 (+0.6)	72.1	72.5 (+0.5)
	CNN-3	*70.3*	66.1 (-4.2)	66.1	67.0 (+0.9)	65.8	66.0 (+0.2)	67.7	66.6 (-1.0)
	CNN-3,5,7,9	*69.7*	67.4 (-2.3)	68.4	68.2 (-0.2)	67.2	67.0 (-0.2)	68.3	69.5 (+1.2)
MR	GRU	71.6	70.7 (-0.8)	**74.3**	73.7 (-0.6)	72.2	71.2 (-1.0)	72.0	72.1 (+0.0)
	Bi-GRU	72.2	71.3 (-0.9)	*73.8*	*73.8* (-0.0)	72.5	71.3 (-1.2)	70.8	71.1 (+0.3)
	CNN-3	64.8	64.7 (-0.1)	69.3	*69.5* (+0.3)	66.6	67.2 (+0.6)	69.4	66.6 (-2.8)
	CNN-3,5,7,9	66.5	64.6 (-1.9)	*70.5*	70.0 (-0.6)	68.0	67.8 (-0.2)	69.2	67.8 (-1.4)
CR	GRU	75.1	73.8 (-1.2)	*77.2*	*77.9* (+0.6)	74.9	75.1 (+0.2)	77.6	76.9 (-0.7)
	Bi-GRU	76.6	75.4 (-1.2)	**78.0**	77.5 (-0.5)	75.4	74.0 (-1.3)	76.2	75.4 (-0.8)
	CNN-3	70.1	69.9 (-0.1)	71.8	73.3 (+1.6)	70.2	69.9 (-0.3)	*74.9*	74.1 (-0.9)
	CNN-3,5,7,9	73.7	71.6 (-2.1)	74.5	*75.4* (+0.9)	72.9	71.6 (-1.3)	73.2	72.8 (-0.3)
SUBJ	GRU	87.3	85.8 (-1.5)	*88.7*	88.1 (-0.6)	87.8	88.0 (+0.2)	85.6	86.9 (+1.2)
	Bi-GRU	88.0	87.5 (-0.5)	**88.9**	88.4 (-0.6)	88.7	87.7 (-1.0)	86.0	86.9 (+0.9)
	CNN-3	80.1	81.4 (+1.3)	*83.8*	83.2 (-0.5)	83.5	82.4 (-1.2)	83.5	82.0 (-1.5)
	CNN-3,5,7,9	82.0	80.7 (-1.3)	85.2	84.0 (-1.2)	*85.7*	83.2 (-2.5)	82.7	82.5 (-0.2)
MPQA	GRU	85.2	84.5 (-0.7)	85.3	84.9 (-0.4)	85.1	84.8 (-0.4)	*86.1*	85.7 (-0.4)
	Bi-GRU	85.0	84.2 (-0.8)	85.4	84.9 (-0.5)	85.2	84.7 (-0.6)	85.9	**86.2** (+0.3)
	CNN-3	79.5	79.2 (-0.3)	84.2	83.8 (-0.5)	83.6	83.5 (-0.0)	*85.0*	83.7 (-1.2)
	CNN-3,5,7,9	82.7	80.7 (-2.0)	*84.2*	83.4 (-0.7)	83.5	83.1 (-0.4)	84.1	84.0 (-0.1)
STS14	+ Proj.	10.1	27.1 (+16.9)	25.2	49.7 (+24.5)	22.6	39.3 (+16.8)	27.0	*55.5* (+28.5)
	RNN	41.4	46.5 (+5.2)	52.4	48.0 (-4.4)	46.8	49.8 (+3.0)	*55.7*	47.6 (-8.1)
	GRU	48.2	62.5 (+14.2)	62.9	65.5 (+2.6)	57.9	63.9 (+6.0)	65.5	*67.4* (+1.9)
	Bi-GRU	53.5	62.3 (+8.8)	66.1	67.5 (+1.4)	62.1	64.6 (+2.5)	66.8	**67.9** (+1.1)
	CNN-3	36.0	52.9 (+16.8)	55.8	59.2 (+3.4)	51.0	57.4 (+6.4)	*63.6*	63.5 (-0.1)
	CNN-3,5,7,9	35.5	54.4 (+18.9)	59.2	61.6 (+2.4)	54.0	59.1 (+5.1)	63.7	*64.2* (+0.5)
STS15	GRU	54.3	65.0 (+10.7)	64.6	65.8 (+1.2)	57.4	63.9 (+6.5)	*69.4*	68.9 (-0.5)
	Bi-GRU	59.8	63.6 (+3.9)	67.6	69.6 (+2.0)	63.3	65.5 (+2.2)	**70.4**	70.0 (-0.3)
	CNN-3	43.6	57.8 (+14.2)	59.9	62.6 (+2.7)	58.0	60.7 (+2.7)	*68.6*	67.4 (-1.2)
	CNN-3,5,7,9	44.0	59.8 (+15.8)	64.2	65.2 (+0.9)	60.4	61.5 (+1.1)	69.3	*69.5* (+0.2)
SST	GRU	74.6	74.8 (+0.2)	76.9	*77.2* (+0.3)	74.7	74.8 (+0.2)	76.5	76.4 (-0.2)
	Bi-GRU	75.8	74.7 (-1.1)	77.8	**77.9** (+0.1)	74.1	74.3 (+0.3)	75.5	76.6 (+1.1)
	CNN-3	66.9	65.6 (-1.3)	72.9	72.2 (-0.7)	68.8	71.1 (+2.3)	*73.6*	71.4 (-2.2)
	CNN-3,5,7,9	68.2	66.7 (-1.5)	*74.7*	73.5 (-1.2)	71.5	70.2 (-1.3)	72.5	71.0 (-1.5)
SICK-E	+ Proj.	71.0	76.6 (+5.6)	74.0	75.4 (+1.4)	73.8	76.0 (+2.2)	75.9	*78.2* (+2.3)
	RNN	67.7	69.9 (+2.2)	69.3	68.4 (-0.9)	72.0	69.5 (-2.6)	*74.3*	67.3 (-7.0)
	GRU	75.6	78.2 (+2.6)	80.6	80.5 (-0.0)	77.7	78.7 (+1.0)	**81.5**	81.3 (-0.2)
	Bi-GRU	74.7	78.9 (+4.3)	79.7	78.7 (-1.0)	79.0	79.5 (+0.5)	81.1	*81.1* (+0.0)
	CNN-3	69.6	73.2 (+3.7)	72.9	73.8 (+0.9)	73.4	74.4 (+1.0)	74.7	*75.6* (+0.9)
	CNN-3,5,7,9	74.2	74.9 (+0.7)	74.1	*76.4* (+2.3)	75.6	75.1 (-0.6)	76.0	74.2 (-1.8)
SICK-R	+ Proj.	66.5	70.2 (+3.7)	68.0	72.3 (+4.2)	67.0	70.4 (+3.4)	63.8	*73.4* (+9.6)
	RNN	63.8	62.4 (-1.4)	68.1	62.4 (-5.6)	68.3	63.6 (-4.8)	*69.6*	63.3 (-6.3)
	GRU	74.8	77.5 (+2.8)	*81.6*	81.2 (-0.4)	79.4	79.2 (-0.2)	81.3	81.1 (-0.2)
	Bi-GRU	76.9	78.3 (+1.3)	**81.8**	80.1 (-1.6)	80.1	79.1 (-1.0)	81.3	80.4 (-0.9)
	CNN-3	66.1	72.5 (+6.4)	73.4	73.3 (-0.1)	72.6	73.0 (+0.4)	*75.2*	74.4 (-0.8)
	CNN-3,5,7,9	73.9	75.6 (+1.7)	75.9	77.9 (+2.0)	77.5	75.9 (-1.6)	*78.4*	77.4 (-0.9)

TABLE B.3: **STS15-16 and Benchmark** scores in Pearson's $r \times 100$. These evaluations are done for before and after tuning the embeddings for a specific algebraic composition operation. We also trained embeddings without pretraining (*None*).

		Word2Vec		GloVe		fastText		Paragram	
		Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
STS12 (All)	GRU	46.8	56.7 (+9.9)	56.0	56.8 (+0.7)	49.6	56.9 (+7.3)	57.6	*58.4* (+0.8)
	Bi-GRU	49.6	55.9 (+6.4)	58.2	58.0 (-0.2)	54.9	56.3 (+1.3)	**59.8**	58.6 (-1.2)
	CNN-3,5,7,9	28.7	44.9 (+16.2)	51.8	53.7 (+2.0)	45.2	51.3 (+6.1)	55.9	*56.8* (+0.9)
MSRpar	GRU	30.0	34.5 (+4.5)	30.9	32.7 (+1.8)	**43.3**	41.5 (-1.8)	37.5	38.9 (+1.5)
	Bi-GRU	35.1	36.2 (+1.0)	35.0	32.1 (-2.9)	*42.8*	36.2 (-6.6)	40.1	37.2 (-2.9)
	CNN-3,5,7,9	35.8	34.6 (-1.1)	29.7	30.5 (+0.8)	41.0	35.0 (-6.0)	*41.8*	35.1 (-6.8)
MSRvid	GRU	61.7	74.2 (+12.5)	*77.7*	*77.7* (+0.1)	58.5	*78.0* (+19.5)	73.2	*77.5* (+4.3)
	Bi-GRU	59.0	73.2 (+14.2)	78.2	**80.9** (+2.6)	69.5	78.8 (+9.3)	*77.5*	*79.7* (+2.1)
	CNN-3,5,7,9	5.9	54.7 (+48.7)	*72.4*	*75.7* (+3.2)	49.3	68.8 (+19.5)	*77.0*	*78.1* (+1.1)
SMTeuroparl	GRU	28.8	**48.2** (+19.3)	42.3	45.7 (+3.4)	34.0	37.0 (+3.0)	40.1	42.2 (+2.1)
	Bi-GRU	33.3	43.5 (+10.2)	42.9	*44.3* (+1.4)	37.4	40.3 (+3.0)	42.9	43.8 (+0.9)
	CNN-3,5,7,9	28.1	30.2 (+2.1)	37.4	37.9 (+0.5)	24.4	31.4 (+7.0)	38.5	*44.4* (+5.9)
OnWN	GRU	60.4	68.2 (+7.8)	69.2	68.1 (-1.2)	61.1	65.1 (+4.0)	*70.9*	69.5 (-1.4)
	Bi-GRU	65.2	67.9 (+2.7)	72.0	69.7 (-2.3)	65.9	68.2 (+2.3)	**72.3**	71.0 (-1.4)
	CNN-3,5,7,9	47.0	60.2 (+13.2)	64.8	66.8 (+2.1)	62.1	65.2 (+3.1)	66.2	*68.6* (+2.4)
SMTnews	GRU	45.5	53.5 (+8.0)	53.3	53.8 (+0.5)	41.1	54.1 (+13.0)	**61.5**	56.7 (-4.7)
	Bi-GRU	48.4	52.5 (+4.1)	55.8	57.4 (+1.5)	50.1	47.7 (-2.4)	*59.7*	53.0 (-6.8)
	CNN-3,5,7,9	24.3	33.6 (+9.3)	46.4	49.5 (+3.1)	37.3	45.5 (+8.2)	43.7	*49.7* (+6.0)
STS13 (All)	GRU	45.2	60.3 (+15.1)	60.4	65.1 (+4.8)	52.9	62.9 (+10.0)	63.7	*65.6* (+1.8)
	Bi-GRU	49.3	61.0 (+11.7)	62.0	66.9 (+4.9)	59.2	63.0 (+3.7)	64.1	**67.6** (+3.6)
	CNN-3,5,7,9	32.9	52.5 (+19.6)	53.9	56.7 (+2.8)	44.5	53.1 (+8.6)	*62.9*	62.1 (-0.7)
FNWN	GRU	36.0	41.7 (+5.7)	42.1	*47.5* (+5.4)	40.8	39.7 (-1.1)	46.9	41.4 (-5.5)
	Bi-GRU	37.7	45.6 (+8.0)	42.9	52.3 (+9.4)	48.1	47.9 (-0.2)	49.3	**53.5** (+4.2)
	CNN-3,5,7,9	23.8	31.1 (+7.4)	39.8	35.6 (-4.2)	28.6	31.8 (+3.2)	*41.5*	41.0 (-0.5)
OnWN	GRU	40.6	71.7 (+31.1)	69.9	79.6 (+9.7)	55.2	79.1 (+23.9)	73.8	**81.5** (+7.7)
	Bi-GRU	49.9	74.3 (+24.5)	74.1	*80.4* (+6.3)	65.7	73.3 (+7.6)	70.4	79.5 (+9.1)
	CNN-3,5,7,9	14.2	54.6 (+40.3)	52.0	63.5 (+11.4)	33.7	52.6 (+18.9)	*71.7*	68.4 (-3.3)
HDL	GRU	51.0	56.4 (+5.5)	57.9	58.8 (+0.9)	54.2	56.7 (+2.5)	*60.4*	59.7 (-0.7)
	Bi-GRU	51.7	54.8 (+3.1)	57.8	60.5 (+2.7)	57.2	59.0 (+1.8)	**63.0**	62.3 (-0.7)
	CNN-3,5,7,9	49.1	56.3 (+7.2)	58.8	56.9 (-1.9)	56.5	58.8 (+2.3)	61.7	*62.8* (+1.1)
STS14 (All)	GRU	48.2	62.5 (+14.2)	62.9	65.5 (+2.6)	57.9	63.9 (+6.0)	65.5	*67.4* (+1.9)
	Bi-GRU	53.5	62.3 (+8.8)	66.1	67.5 (+1.4)	62.1	64.6 (+2.5)	66.8	**67.9** (+1.1)
	CNN-3,5,7,9	35.5	54.4 (+18.9)	59.2	61.6 (+2.4)	54.0	59.1 (+5.1)	63.7	*64.2* (+0.5)
Deft Forum	GRU	12.2	34.0 (+21.8)	29.5	32.0 (+2.5)	26.3	32.8 (+6.6)	36.4	*38.2* (+1.8)
	Bi-GRU	19.4	28.3 (+8.9)	33.7	38.1 (+4.4)	30.8	29.4 (-1.4)	*41.0*	36.5 (-4.5)
	CNN-3,5,7,9	21.6	31.3 (+9.7)	32.1	35.9 (+3.7)	28.4	30.4 (+2.0)	38.8	**44.3** (+5.5)
Deft News	GRU	50.6	63.9 (+13.3)	63.1	64.0 (+0.9)	65.2	65.6 (+0.4)	*66.5*	65.8 (-0.6)
	Bi-GRU	62.3	63.2 (+0.9)	**67.9**	67.0 (-0.9)	65.9	64.1 (-1.8)	62.5	64.5 (+2.0)
	CNN-3,5,7,9	41.9	57.6 (+15.7)	63.9	63.7 (-0.2)	56.6	62.6 (+6.0)	59.8	*65.0* (+5.2)
HDL	GRU	51.6	55.9 (+4.3)	54.0	56.7 (+2.7)	56.3	57.5 (+1.2)	59.2	*59.8* (+0.6)
	Bi-GRU	51.2	54.8 (+3.6)	58.0	58.2 (+0.2)	56.0	60.7 (+4.7)	60.8	**61.8** (+1.0)
	CNN-3,5,7,9	44.1	54.2 (+10.1)	50.8	51.6 (+0.8)	54.3	57.5 (+3.2)	*59.1*	56.5 (-2.6)
Images	GRU	52.3	71.6 (+19.2)	75.4	77.1 (+1.7)	63.1	72.4 (+9.2)	76.0	*79.4* (+3.4)
	Bi-GRU	63.1	72.5 (+9.3)	76.6	78.5 (+1.9)	70.4	74.1 (+3.8)	78.3	**79.5** (+1.2)
	CNN-3,5,7,9	32.6	60.9 (+28.2)	70.0	74.5 (+4.5)	63.5	68.2 (+4.8)	*76.0*	73.3 (-2.7)
OnWN	GRU	59.8	78.8 (+18.9)	74.6	82.3 (+7.8)	69.3	83.1 (+13.8)	78.3	**85.0** (+6.6)
	Bi-GRU	64.0	80.5 (+16.5)	79.8	82.6 (+2.7)	74.1	79.9 (+5.8)	*77.5*	*83.2* (+5.7)
	CNN-3,5,7,9	32.4	65.5 (+33.1)	63.7	71.0 (+7.3)	52.4	65.9 (+13.6)	*75.5*	74.7 (-0.8)
Tweet-News	GRU	49.9	60.1 (+10.2)	*67.5*	66.6 (-0.9)	59.1	60.6 (+1.5)	65.7	63.7 (-2.0)
	Bi-GRU	52.3	61.2 (+8.9)	**68.7**	68.5 (-0.2)	65.5	65.1 (-0.4)	67.8	67.3 (-0.5)
	CNN-3,5,7,9	38.6	49.5 (+10.9)	*66.8*	64.1 (-2.7)	60.1	60.5 (+0.5)	60.7	64.0 (+3.3)

TABLE B.4: **STS15-16 and Benchmark** scores in Pearson's $r \times 100$. These evaluations are done for before and after tuning the embeddings for a specific algebraic composition operation. We also trained embeddings without pretraining (*None*).

		Word2Vec		GloVe		fastText		Paragram	
		Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
STS15 (All)	GRU	54.3	65.0 (+10.7)	64.6	65.8 (+1.2)	57.4	63.9 (+6.5)	*69.4*	68.9 (-0.5)
	Bi-GRU	59.8	63.6 (+3.9)	67.6	69.6 (+2.0)	63.3	65.5 (+2.2)	**70.4**	70.0 (-0.3)
	CNN-3,5,7,9	44.0	59.8 (+15.8)	64.2	65.2 (+0.9)	60.4	61.5 (+1.1)	69.3	*69.5* (+0.2)
Ans. Forums	GRU	32.5	47.8 (+15.4)	42.4	44.1 (+1.8)	36.4	45.0 (+8.6)	**55.1**	51.4 (-3.7)
	Bi-GRU	44.4	41.8 (-2.5)	51.2	51.1 (-0.0)	41.0	43.1 (+2.2)	50.8	*52.7* (+1.9)
	CNN-3,5,7,9	30.2	42.0 (+11.8)	43.7	46.7 (+3.0)	40.1	*37.7* (-2.4)	50.9	*52.7* (+1.8)
Ans. Students	GRU	63.0	68.0 (+5.0)	70.8	68.9 (-1.9)	60.8	69.2 (+8.4)	*72.8*	72.7 (-0.1)
	Bi-GRU	65.5	70.6 (+5.1)	71.3	**74.9** (+3.6)	70.2	67.9 (-2.3)	74.6	73.2 (-1.4)
	CNN-3,5,7,9	51.7	65.4 (+13.7)	67.8	68.2 (+0.3)	65.7	65.4 (-0.3)	*74.6*	72.1 (-2.4)
Belief	GRU	42.3	61.3 (+19.0)	62.0	65.6 (+3.6)	46.0	56.3 (+10.3)	**68.2**	66.9 (-1.3)
	Bi-GRU	52.2	56.1 (+3.9)	62.4	*67.3* (+5.0)	54.9	58.1 (+3.2)	66.7	66.8 (+0.2)
	CNN-3,5,7,9	33.6	56.0 (+22.4)	63.3	67.3 (+4.0)	49.7	55.4 (+5.7)	63.2	*67.3* (+4.1)
HDL	GRU	51.0	61.6 (+10.6)	59.8	60.9 (+1.1)	58.1	62.0 (+3.9)	*64.8*	63.3 (-1.5)
	Bi-GRU	56.8	60.8 (+4.1)	63.1	63.8 (+0.7)	61.7	65.8 (+4.1)	66.5	**66.9** (+0.4)
	CNN-3,5,7,9	46.2	55.7 (+9.5)	59.8	59.0 (-0.8)	61.3	62.1 (+0.8)	*66.3*	66.2 (-0.1)
Images	GRU	65.8	75.8 (+10.0)	75.6	78.6 (+3.0)	69.6	73.8 (+4.2)	78.4	*80.7* (+2.3)
	Bi-GRU	68.5	74.1 (+5.6)	79.3	80.6 (+1.2)	73.5	77.8 (+4.4)	**81.6**	80.2 (-1.4)
	CNN-3,5,7,9	46.1	69.0 (+22.9)	75.8	76.5 (+0.7)	69.6	72.0 (+2.4)	79.5	*79.7* (+0.2)
STS16 (All)	GRU	51.4	60.7 (+9.2)	61.7	60.8 (-0.9)	54.5	60.8 (+6.3)	*62.1*	60.2 (-1.9)
	Bi-GRU	53.2	57.8 (+4.6)	61.7	*65.4* (+3.7)	59.7	58.3 (-1.5)	63.2	63.0 (-0.3)
	CNN-3,5,7,9	49.4	59.6 (+10.2)	60.9	62.2 (+1.3)	50.3	54.1 (+3.9)	**66.7**	63.1 (-3.6)
Ans.-Ans.	GRU	37.8	45.3 (+7.4)	49.9	47.6 (-2.4)	37.9	42.4 (+4.5)	*62.4*	43.4 (-19.0)
	Bi-GRU	39.4	47.6 (+8.2)	42.0	54.8 (+12.8)	43.5	38.9 (-4.6)	*61.0*	53.1 (-7.9)
	CNN-3,5,7,9	38.8	48.0 (+9.3)	48.1	51.4 (+3.3)	36.2	42.5 (+6.3)	**62.7**	57.1 (-5.7)
HDL	GRU	56.8	60.0 (+3.3)	59.5	59.8 (+0.3)	60.5	*61.2* (+0.7)	60.6	58.1 (-2.5)
	Bi-GRU	61.7	61.1 (-0.5)	**63.7**	62.3 (-1.4)	63.4	61.6 (-1.8)	62.4	63.7 (+1.3)
	CNN-3,5,7,9	53.9	62.5 (+8.6)	59.5	62.1 (+2.6)	60.7	62.1 (+1.4)	61.8	*62.9* (+1.1)
Plagiarism	GRU	61.6	75.4 (+13.8)	71.9	77.8 (+5.9)	64.1	73.7 (+9.6)	71.4	*78.8* (+7.3)
	Bi-GRU	61.5	72.0 (+10.5)	76.1	**79.6** (+3.5)	70.0	74.5 (+4.4)	77.0	79.0 (+2.0)
	CNN-3,5,7,9	55.4	68.8 (+13.4)	70.9	73.6 (+2.7)	68.2	70.0 (+1.8)	*78.0*	74.3 (-3.7)
Post Editing	GRU	70.8	78.2 (+7.5)	79.3	77.5 (-1.8)	72.6	77.1 (+4.5)	81.5	*82.1* (+0.7)
	Bi-GRU	70.6	79.4 (+8.8)	77.8	78.9 (+1.1)	79.5	81.1 (+1.6)	**82.2**	81.4 (-0.7)
	CNN-3,5,7,9	66.8	76.3 (+9.6)	77.1	78.5 (+1.4)	76.8	79.0 (+2.2)	80.1	*81.0* (+0.9)
Quest.-Quest.	GRU	27.8	43.3 (+15.6)	46.8	39.6 (-7.1)	35.8	*49.3* (+13.5)	30.9	37.2 (+6.2)
	Bi-GRU	30.3	25.3 (-5.0)	48.3	**50.5** (+2.2)	40.8	33.4 (-7.4)	29.8	35.0 (+5.2)
	CNN-3,5,7,9	30.1	40.5 (+10.4)	47.9	43.8 (-4.1)	4.1	12.2 (+8.0)	*49.4*	37.6 (-11.8)
STS Benchmark	GRU	62.3	66.9 (+4.6)	67.8	68.6 (+0.8)	67.7	68.1 (+0.4)	*71.3*	68.3 (-2.9)
	Bi-GRU	63.3	67.9 (+4.5)	68.4	70.4 (+2.0)	67.1	69.2 (+2.1)	**71.6**	70.4 (-1.2)
	CNN-3,5,7,9	52.6	56.3 (+3.7)	58.3	*60.3* (+2.0)	57.6	56.9 (-0.8)	58.9	60.1 (+1.2)

TABLE B.5: **STS12-16** p-values for all evaluations. Results that are not statistically significant are displayed in bold.

		Word2Vec		GloVe		fastText		Paragram	
		Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
MSRpar	GRU	4.8e-17	2.4e-22	4.8e-18	3.5e-20	1.5e-35	1.6e-32	2.1e-26	1.4e-28
	Bi-GRU	3.2e-23	1.4e-24	4.6e-23	2.1e-19	9.9e-35	1.2e-24	2.8e-30	5.2e-26
	CNN-3,5,7,9	4.9e-24	1.6e-22	8.8e-17	1.2e-17	9.0e-32	4.3e-23	3.8e-33	4.0e-23
MSRvid	GRU	8.0e-80	6.6e-132	2.2e-152	9.1e-153	4.5e-70	2.4e-154	5.9e-127	1.7e-151
	Bi-GRU	1.5e-71	1.0e-126	5.3e-156	1.8e-174	4.1e-109	1.4e-159	1.6e-151	7.7e-166
	CNN-3,5,7,9	**1.0e-01**	1.0e-59	5.5e-123	2.5e-140	3.9e-47	3.2e-106	3.8e-148	2.6e-155
SMTeuroparl	GRU	3.2e-10	5.1e-28	2.2e-21	4.0e-25	6.6e-14	2.3e-16	3.8e-19	3.3e-21
	Bi-GRU	2.4e-13	1.2e-22	5.1e-22	1.7e-23	1.2e-16	2.3e-19	5.6e-22	5.8e-23
	CNN-3,5,7,9	9.0e-10	3.9e-11	1.1e-16	3.7e-17	1.2e-07	5.7e-12	1.1e-17	1.4e-23
OnWN	GRU	9.3e-76	1.1e-103	4.6e-108	3.7e-103	4.5e-78	1.2e-91	1.7e-115	3.1e-109
	Bi-GRU	6.2e-92	2.2e-102	7.0e-121	2.8e-110	1.0e-94	1.1e-103	1.9e-122	7.9e-116
	CNN-3,5,7,9	1.8e-42	3.5e-75	2.3e-90	3.5e-98	3.6e-81	6.5e-92	1.4e-95	2.7e-105
SMTnews	GRU	8.3e-22	6.2e-31	1.0e-30	2.2e-31	1.1e-17	1.1e-31	8.1e-43	2.4e-35
	Bi-GRU	8.9e-25	1.3e-29	4.2e-34	2.8e-36	9.5e-27	4.6e-24	6.2e-40	3.2e-30
	CNN-3,5,7,9	8.9e-07	5.2e-12	1.0e-22	4.6e-26	1.2e-14	9.1e-22	4.5e-20	2.6e-26
FNWN	GRU	3.7e-07	2.4e-09	1.6e-09	4.7e-12	5.4e-09	1.5e-08	9.9e-12	3.1e-09
	Bi-GRU	9.1e-08	4.1e-11	7.3e-10	1.2e-14	2.4e-12	2.9e-12	5.5e-13	2.2e-15
	CNN-3,5,7,9	9.9e-04	1.3e-05	1.5e-08	4.9e-07	6.5e-05	8.2e-06	2.8e-09	4.6e-09
OnWN	GRU	1.2e-23	1.0e-89	1.8e-83	6.9e-124	3.9e-46	2.0e-121	1.2e-97	1.0e-134
	Bi-GRU	1.3e-36	1.0e-99	7.1e-99	3.1e-128	1.7e-70	1.7e-95	3.8e-85	1.5e-123
	CNN-3,5,7,9	7.1e-04	7.7e-45	3.3e-40	1.5e-64	2.2e-16	2.6e-41	1.6e-89	1.3e-78
HDL	GRU	8.0e-51	2.7e-64	2.8e-68	7.0e-71	1.6e-58	5.0e-65	8.5e-76	1.1e-73
	Bi-GRU	1.5e-52	4.6e-60	4.8e-68	6.1e-76	2.1e-66	1.1e-71	2.6e-84	7.7e-82
	CNN-3,5,7,9	9.1e-47	5.2e-64	5.4e-71	1.8e-65	1.8e-64	6.7e-71	8.1e-80	2.0e-83
Deft Forum	GRU	**9.7e-03**	1.3e-13	1.8e-10	3.7e-12	1.6e-08	9.3e-13	1.5e-15	4.9e-17
	Bi-GRU	3.3e-05	9.3e-10	2.1e-13	5.8e-17	2.4e-11	2.1e-10	1.1e-19	1.3e-15
	CNN-3,5,7,9	3.6e-06	1.1e-11	2.9e-12	4.2e-15	8.5e-10	4.6e-11	1.4e-17	5.2e-23
Deft News	GRU	6.7e-21	7.5e-36	9.8e-35	5.8e-36	1.2e-37	2.8e-38	1.2e-39	1.2e-38
	Bi-GRU	1.4e-33	7.6e-35	7.5e-42	2.2e-40	1.1e-38	3.9e-36	6.5e-34	1.2e-36
	CNN-3,5,7,9	3.4e-14	6.1e-28	8.3e-36	1.4e-35	9.1e-27	4.7e-34	2.0e-30	2.4e-37
HDL	GRU	2.9e-52	6.0e-63	4.3e-58	5.5e-65	6.7e-64	3.3e-67	4.8e-72	6.8e-74
	Bi-GRU	2.0e-51	5.7e-60	1.1e-68	2.8e-69	3.9e-63	1.2e-76	3.9e-77	3.5e-80
	CNN-3,5,7,9	4.4e-37	1.7e-58	2.1e-50	3.2e-52	7.4e-59	2.7e-67	7.4e-72	2.0e-64
Images	GRU	5.6e-54	9.2e-119	2.1e-138	1.1e-148	1.2e-84	1.2e-122	4.9e-142	1.7e-163
	Bi-GRU	1.2e-84	4.8e-123	1.0e-145	6.5e-158	4.0e-113	9.3e-132	1.8e-156	1.7e-164
	CNN-3,5,7,9	4.6e-20	3.5e-77	1.6e-111	7.6e-134	8.3e-86	6.8e-104	7.1e-142	2.5e-127
OnWN	GRU	4.8e-74	1.9e-159	4.2e-134	2.7e-186	2.4e-108	9.3e-193	1.1e-156	2.5e-210
	Bi-GRU	8.3e-88	5.8e-172	9.4e-167	4.5e-188	2.5e-131	3.4e-167	2.1e-151	7.1e-194
	CNN-3,5,7,9	8.9e-20	6.3e-93	2.0e-86	5.9e-116	5.3e-54	1.0e-94	3.9e-139	1.4e-134
Tweet-News	GRU	1.7e-48	9.6e-75	9.6e-101	3.6e-97	6.9e-72	1.6e-76	1.1e-93	1.2e-86
	Bi-GRU	5.5e-54	2.6e-78	5.1e-106	3.9e-105	5.6e-93	1.6e-91	4.9e-102	4.6e-100
	CNN-3,5,7,9	4.6e-28	1.4e-47	3.3e-98	5.5e-88	9.2e-75	3.2e-76	9.1e-77	1.2e-87
Ans. Forums	GRU	1.2e-10	7.8e-23	9.0e-18	2.6e-19	3.2e-13	4.0e-20	4.1e-31	1.1e-26
	Bi-GRU	1.6e-19	2.6e-17	2.0e-26	2.2e-26	1.3e-16	2.0e-18	5.3e-26	3.3e-28
	CNN-3,5,7,9	2.4e-09	1.9e-17	6.2e-19	1.1e-21	6.8e-16	4.2e-14	3.9e-26	3.5e-28
Ans. Students	GRU	3.3e-84	5.4e-103	4.7e-115	8.8e-107	5.3e-77	4.0e-108	1.7e-124	4.7e-124
	Bi-GRU	3.8e-93	2.6e-114	1.9e-117	8.8e-136	3.1e-112	1.5e-102	4.1e-134	1.1e-126
	CNN-3,5,7,9	2.1e-52	1.1e-92	2.8e-102	1.2e-103	8.0e-94	1.2e-92	4.6e-134	1.7e-121
Belief	GRU	9.4e-18	4.2e-40	3.2e-41	1.5e-47	4.8e-21	9.1e-33	1.3e-52	6.4e-50
	Bi-GRU	1.2e-27	1.6e-32	8.2e-42	7.0e-51	5.7e-31	2.9e-35	1.5e-49	7.2e-50
	CNN-3,5,7,9	2.3e-11	2.4e-32	2.3e-43	8.9e-51	9.3e-25	1.7e-31	3.3e-43	7.6e-51
HDL	GRU	5.4e-51	1.1e-79	7.9e-74	2.6e-77	7.1e-69	8.7e-81	1.8e-90	4.2e-85
	Bi-GRU	3.5e-65	3.7e-77	1.1e-84	4.6e-87	8.4e-80	2.9e-94	6.2e-97	1.8e-98
	CNN-3,5,7,9	7.9e-41	3.2e-62	7.0e-74	1.3e-71	1.9e-78	4.8e-81	5.4e-96	1.1e-95
Images	GRU	4.1e-94	4.7e-141	8.8e-140	3.4e-158	6.5e-110	5.0e-130	5.6e-157	4.7e-173
	Bi-GRU	4.7e-105	2.2e-131	1.7e-163	2.2e-172	4.3e-128	2.1e-153	2.1e-180	1.1e-169
	CNN-3,5,7,9	9.7e-41	2.6e-107	3.5e-141	3.3e-145	1.3e-109	7.7e-121	2.1e-164	5.1e-166
Ans.-Ans.	GRU	4.7e-10	3.2e-14	2.0e-17	9.3e-16	4.1e-10	1.6e-12	7.6e-29	4.3e-13
	Bi-GRU	7.5e-11	9.0e-16	2.8e-12	2.9e-21	3.6e-13	1.3e-10	2.8e-27	6.5e-20
	CNN-3,5,7,9	1.6e-10	4.7e-16	4.0e-16	1.5e-18	2.9e-09	1.4e-12	3.5e-29	2.4e-23
HDL	GRU	1.2e-22	8.7e-26	3.3e-25	1.5e-25	2.8e-26	5.1e-27	2.5e-26	7.7e-24
	Bi-GRU	1.8e-27	7.0e-27	9.8e-30	3.9e-28	2.3e-29	2.0e-27	2.9e-28	1.1e-29
	CNN-3,5,7,9	3.8e-20	2.1e-28	3.2e-25	6.3e-28	1.8e-26	6.2e-28	1.4e-27	8.0e-29
Plagiarism	GRU	1.9e-25	1.5e-43	7.7e-38	7.8e-48	5.3e-28	1.2e-40	3.5e-37	7.9e-50
	Bi-GRU	2.6e-25	5.0e-38	8.4e-45	1.1e-51	3.4e-35	6.9e-42	2.5e-46	3.1e-50
	CNN-3,5,7,9	6.9e-20	1.5e-33	1.7e-36	1.5e-40	8.8e-33	3.5e-35	2.4e-48	1.3e-41
Post Editing	GRU	2.3e-38	1.2e-51	5.4e-54	3.2e-50	3.3e-41	2.6e-49	3.4e-59	6.8e-61
	Bi-GRU	4.2e-38	3.3e-54	9.4e-51	3.4e-53	1.9e-54	2.5e-58	5.4e-61	4.3e-59
	CNN-3,5,7,9	7.3e-33	8.1e-48	3.1e-49	3.2e-52	8.2e-49	2.3e-53	7.0e-56	5.6e-58
Quest.-Quest.	GRU	4.7e-05	5.6e-11	9.3e-13	2.9e-09	1.0e-07	3.6e-14	5.3e-06	3.0e-08
	Bi-GRU	8.1e-06	2.2e-04	1.3e-13	5.9e-15	8.7e-10	7.5e-07	1.2e-05	2.1e-07
	CNN-3,5,7,9	9.2e-06	1.1e-09	2.1e-13	3.3e-11	**5.5e-01**	**8.0e-02**	3.0e-14	2.1e-08

Appendix C

Miscellaneous results on training with encyclopedic data

In this appendix we will present miscellaneous results from the experiments described in Chapter 6 on learning composition functions using encyclopedic data.

C.1 Size of the different vocabularies

In Table C.1 we show the different vocabulary sizes and how the known words overlap. This influences the results in the "Pre." or "Org." column. For each separate experiment we use randomly initialized embeddings for the ones that do not have an embedding available in the pretrained set.

In addition we show how the vocabularies of the sentence evaluations compare to the dataset and the pretrained embeddings in Table C.2. We used embeddings for words which were in the pretrained vocabulary (Wiki-p, Wiki-e), embeddings for other words we initialized randomly (WP-r, Wiki-r). As you see WP-r + WP-e is the same for all pretrained embeddings. The same goes for Wiki-e + Wiki-r. Under 'Task' we see the entire vocabulary of the specific sentence representation evaluation task, and under 'Emb.' we see how it overlaps with the pretrained embedding.

In Table C.3 we see which word pairs can be used for the word representation evaluation methods. The amount of word pairs is determined by the words in the vocabulary.

TABLE C.1: Vocabulary size of the different datasets and how they overlap with the vocabularies of the pretrained embeddings.

	WordNet		Wikipedia	
Total vocabulary size	48,944		378,950	
word2vec	34,976	71.4 %	83,888	22.1 %
GloVe	42,742	87.3 %	181,967	48.0 %
fastText	41,554	84.9 %	320,708	85.7 %
Paragram	44,138	90.2 %	297,440	78.5 %

TABLE C.2: Vocabulary size for the sentence representation evaluation methods. We show how the vocabulary overlaps with the pretrained embeddings (Emb.) and with the datasets. WordNet with pretrained embeddings is displayed under (WN-e) and WordNet with random initialization is displayed under (WN-r). The same is true for vocabulary overlap with the Wikipedia dataset under (Wiki).

		word2Vec					GloVe					fastText					Paragram				
	Task	Emb.	WN-e	WN-r	Wiki-e	Wiki-r	Emb.	WN-e	WN-r	Wiki-e	Wiki-r	Emb.	WN-e	WN-r	Wiki-e	Wiki-r	Emb.	WN-e	WN-r	Wiki-e	Wiki-r
MR	20303	15965	11178	385	14621	2019	18589	11559	4	16323	317	17369	11256	307	15665	975	17748	11375	188	15811	829
CR	5674	4965	4042	121	4729	316	5472	4161	2	5026	19	4998	4035	128	4723	322	4763	4038	125	4578	467
SUBJ	22616	17515	12327	528	16465	2937	20855	12850	5	18990	412	19872	12499	356	18297	1105	17629	12140	715	16545	2857
MPQA	6238	6083	5114	50	5802	107	6197	5162	2	5898	11	6179	5141	23	5877	32	5676	4974	190	5506	403
SST	17558	14388	10370	357	13241	1641	16525	10724	3	14687	195	15373	10419	308	14059	823	16830	10726	1	14880	2
TREC	8968	7395	6124	462	7170	1186	8599	6575	11	8252	104	8244	6313	273	7890	466	7009	5937	649	6853	1503
MRPC	17908	12203	9343	571	11876	3563	16074	9907	7	15001	438	14233	9477	437	13488	1951	13292	9467	447	12862	2577
SICK-E	2312	2277	2023	14	2152	13	2305	2035	2	2165	0	2291	2030	7	2157	8	2312	2035	2	2165	0
SICK-R	2312	2277	2023	14	2152	13	2305	2035	2	2165	0	2291	2030	7	2157	8	2312	2035	2	2165	0
STS-B	16012	12093	9438	474	11718	2419	14634	9905	7	13802	335	13477	9564	348	12827	1310	16012	9762	150	13390	747
STS12	8124	6716	5907	240	6496	939	7766	6145	2	7352	83	7147	5932	215	6833	602	7867	6142	5	7403	32
STS13	5191	4499	4047	128	4350	352	4910	4173	2	4667	35	4678	4069	106	4489	213	4896	4170	5	4674	28
STS14	9268	7748	6491	218	7488	781	8586	6707	2	8136	133	8245	6546	163	7842	427	8682	6704	5	8192	77
STS15	7431	6410	5425	189	6169	572	7047	5611	3	6665	76	6706	5461	153	6372	369	7097	5607	7	6684	57
STS16	3988	3489	3124	101	3352	291	3804	3223	2	3605	38	3689	3149	76	3502	141	3559	3162	63	3423	220

TABLE C.3: Number of word pairs used to evaluate the different datasets. We can only use a word pair if both words are in the vocabulary. As we see they vary between WordNet and Wikipedia, therefore we cannot compare the embeddings from these different datasets.

	Pairs	WordNet	Wikipedia
MC-30	30	25	30
MTurk-3000	3000	2726	2998
MTurk-287	287	259	285
MTurk-771	771	752	771
RG-65	65	57	65
RW-STANFORD	2034	840	1364
SIMLEX-999	999	974	996
SimVerb-3500	3500	3457	3424
VERB-143	143	135	143
WS-353	353	333	353
WS-353-REL	252	240	252
WS-353-SIM	203	187	203
YP-130	130	121	130

C.2 Results on word representation evaluations

In Table C.4 we see the results for all word representation evaluation methods described in Section 4.3. We only show results for +, GRU and Bi-GRU since these where the best performing models from the previous chapters.

C.3 Results on sentence representation evaluations

In Table C.5 we see the results for all sentence representation evaluation methods described in Section 4.4. We only show results for +, GRU and Bi-GRU since these where the best performing models from the previous chapters.

C.4 Results on semantic textual similarity

In table C.6 and B.4 we show the results of the evaluations on all subsets for the STS tasks from 2012 to 2016, in addition we show the results of the STS Benchmark. Not all results are statistically significant, however most of them are. All corresponding p-vales can be found in C.8.

TABLE C.4: **Word representation evaluation** scores in Spearman's r $\times 100$, for our learned composition functions. It is important to note that under *original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

		Word2Vec		GloVe		fastText		Paragram	
		Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
WS-353	+	68.3	61.8 (-6.5)	71.0	73.6 (+2.6)	**73.9**	67.9 (-5.9)	70.2	64.4 (-5.8)
	GRU	68.1	64.1 (-4.0)	71.0	71.7 (+0.7)	**73.9**	69.7 (-4.1)	70.0	69.9 (-0.1)
	Bi-GRU	68.1	64.0 (-4.1)	71.0	71.1 (+0.1)	**73.9**	72.7 (-1.2)	69.3	68.5 (-0.7)
WS-353 (Rel.)	+	60.3	51.6 (-8.7)	64.4	65.2 (+0.7)	**68.5**	57.9 (-10.5)	62.9	54.2 (-8.7)
	GRU	59.9	50.9 (-9.0)	64.4	64.7 (+0.3)	**68.5**	61.6 (-6.8)	61.6	61.6 (-0.0)
	Bi-GRU	60.0	56.0 (-4.0)	64.4	64.0 (-0.4)	**68.5**	66.4 (-2.1)	60.9	59.9 (-0.9)
WS-353 (Sim.)	+	77.8	73.7 (-4.1)	*80.2*	77.9 (-2.3)	78.1	74.7 (-3.3)	76.9	72.1 (-4.9)
	GRU	77.8	75.5 (-2.4)	80.2	**80.8** (+0.6)	78.1	75.0 (-3.0)	78.1	77.6 (-0.6)
	Bi-GRU	77.8	72.8 (-5.1)	*80.2*	79.4 (-0.8)	78.1	75.0 (-3.0)	77.3	75.9 (-1.5)
SimLex	+	44.2	42.5 (-1.8)	40.7	42.9 (+2.2)	37.3	35.5 (-1.9)	**63.3**	62.7 (-0.6)
	GRU	44.2	41.5 (-2.7)	40.7	41.1 (+0.4)	37.3	35.9 (-1.5)	*63.2*	62.6 (-0.6)
	Bi-GRU	44.2	40.8 (-3.4)	40.7	41.5 (+0.8)	37.3	37.4 (+0.0)	*63.2*	62.8 (-0.4)
SimVerb	+	36.7	33.7 (-3.0)	28.3	30.8 (+2.6)	25.1	23.6 (-1.5)	50.1	**50.2** (+0.1)
	GRU	36.7	33.6 (-3.1)	28.3	28.8 (+0.5)	25.1	24.9 (-0.2)	*50.1*	50.0 (-0.1)
	Bi-GRU	36.7	32.5 (-4.2)	28.3	28.4 (+0.1)	25.1	24.2 (-0.9)	*50.1*	49.8 (-0.3)
RG-65	+	75.0	72.0 (-3.0)	76.2	77.8 (+1.7)	**79.7**	73.4 (-6.4)	75.2	76.8 (+1.5)
	GRU	75.0	69.2 (-5.8)	76.2	75.1 (-1.0)	**79.7**	75.2 (-4.5)	75.2	73.3 (-1.9)
	Bi-GRU	75.0	70.5 (-4.5)	76.2	76.3 (+0.2)	**79.7**	76.6 (-3.1)	75.2	73.7 (-1.5)
MC-30	+	78.8	74.3 (-4.5)	78.6	**82.3** (+3.7)	81.2	78.1 (-3.2)	75.5	77.2 (+1.7)
	GRU	78.8	72.6 (-6.2)	78.6	79.8 (+1.2)	*81.2*	80.2 (-1.0)	75.5	74.5 (-1.0)
	Bi-GRU	78.8	73.8 (-5.0)	78.6	78.4 (-0.2)	*81.2*	77.0 (-4.2)	75.5	73.3 (-2.2)
Rare Word	+	51.5	50.1 (-1.5)	50.4	**54.0** (+3.6)	49.6	47.6 (-2.0)	22.6	25.0 (+2.3)
	GRU	*51.3*	48.6 (-2.7)	50.3	50.5 (+0.2)	49.6	49.1 (-0.5)	22.0	24.7 (+2.8)
	Bi-GRU	*51.4*	48.9 (-2.5)	50.4	50.7 (+0.3)	49.6	49.1 (-0.5)	22.9	23.1 (+0.2)
YP-130	+	55.9	45.2 (-10.7)	*57.1*	44.9 (-12.2)	54.6	47.2 (-7.4)	56.2	56.3 (+0.1)
	GRU	55.9	55.3 (-0.6)	57.1	57.3 (+0.2)	54.6	53.1 (-1.6)	55.9	**57.9** (+2.0)
	Bi-GRU	55.9	53.7 (-2.2)	57.1	56.5 (-0.6)	54.6	48.1 (-6.5)	*57.8*	56.1 (-1.8)
Verb-143	+	44.4	52.8 (+8.4)	34.1	41.8 (+7.6)	38.3	41.0 (+2.7)	*55.7*	53.7 (-1.9)
	GRU	46.4	33.0 (-13.4)	34.1	34.1 (-0.0)	38.3	29.7 (-8.6)	*55.8*	55.2 (-0.6)
	Bi-GRU	46.9	39.1 (-7.7)	34.1	31.0 (-3.1)	38.3	28.3 (-10.0)	**56.3**	50.0 (-6.3)
MTurk-3k	+	73.5	67.0 (-6.4)	*80.2*	77.0 (-3.2)	75.9	67.9 (-8.0)	66.2	62.6 (-3.6)
	GRU	73.2	70.3 (-2.9)	80.2	**80.2** (+0.1)	75.9	73.5 (-2.5)	66.4	66.5 (+0.1)
	Bi-GRU	73.4	70.1 (-3.4)	*80.2*	80.0 (-0.1)	75.9	73.4 (-2.5)	66.1	66.2 (+0.0)
MTurk-287	+	62.8	51.1 (-11.7)	**69.3**	67.8 (-1.5)	66.3	57.8 (-8.5)	63.5	53.5 (-10.1)
	GRU	64.7	62.5 (-2.2)	**69.3**	68.1 (-1.2)	66.3	62.4 (-3.9)	62.7	62.7 (+0.1)
	Bi-GRU	66.2	62.8 (-3.4)	**69.3**	69.1 (-0.3)	66.3	63.1 (-3.2)	62.9	61.4 (-1.5)
MTurk-771	+	67.1	62.4 (-4.7)	**71.5**	69.2 (-2.3)	66.7	60.5 (-6.2)	68.0	66.0 (-2.0)
	GRU	67.1	62.1 (-5.0)	**71.5**	**71.5** (+0.0)	66.7	62.2 (-4.5)	67.9	67.4 (-0.5)
	Bi-GRU	67.1	61.7 (-5.4)	**71.5**	70.9 (-0.6)	66.7	62.1 (-4.6)	67.9	68.0 (+0.1)

TABLE C.5: **Sentence representation evaluation** scores for STS and SICK-R in Pearson's $r \times 100$, all other scores in accuracies $\times 100$. It is important to note that under *Original*, we did train the models themselves but inputted original embeddings. All results for embeddings under CompVec are tuned using our tuning mechanism.

		Word2Vec		GloVe		fastText		Paragram	
		Org.	Tuned	Org.	Tuned	Org.	Tuned	Org.	Tuned
TREC	+	64.4	60.4 (-4.0)	*70.8*	67.2 (-3.6)	67.0	65.6 (-1.4)	68.8	62.0 (-6.8)
	GRU	71.0	67.6 (-3.4)	70.6	**72.6** (+2.0)	69.8	69.6 (-0.2)	65.8	63.4 (-2.4)
	Bi-GRU	68.0	64.0 (-4.0)	69.6	68.8 (-0.8)	68.8	*69.8* (+1.0)	66.6	65.2 (-1.4)
MRPC	+	67.0	66.5 (-0.5)	*69.0*	68.6 (-0.4)	68.4	67.1 (-1.3)	68.6	67.7 (-0.9)
	GRU	70.6	70.4 (-0.2)	70.9	70.3 (-0.6)	70.1	69.6 (-0.5)	*72.1*	71.4 (-0.7)
	Bi-GRU	70.7	69.8 (-0.9)	71.7	70.7 (-1.0)	70.1	70.1 (+0.0)	**72.3**	72.0 (-0.3)
MR	+	75.0	73.8 (-1.2)	**76.2**	76.0 (-0.2)	74.2	73.5 (-0.7)	75.0	74.5 (-0.5)
	GRU	71.1	68.5 (-2.5)	72.5	*72.8* (+0.3)	69.7	68.7 (-1.0)	71.1	71.6 (+0.5)
	Bi-GRU	71.6	69.2 (-2.3)	*73.4*	72.8 (-0.6)	70.6	69.5 (-1.1)	71.3	72.4 (+1.1)
CR	+	76.9	76.4 (-0.5)	**78.3**	77.5 (-0.8)	76.1	76.0 (-0.0)	76.0	75.5 (-0.5)
	GRU	73.7	73.2 (-0.5)	75.3	75.3 (+0.0)	72.4	73.1 (+0.7)	*75.8*	74.4 (-1.4)
	Bi-GRU	75.1	75.3 (+0.2)	*77.8*	76.3 (-1.5)	72.7	72.5 (-0.2)	76.5	75.4 (-1.0)
SUBJ	+	87.8	86.0 (-1.8)	**89.6**	89.0 (-0.6)	89.3	88.6 (-0.7)	87.9	86.8 (-1.1)
	GRU	85.0	82.4 (-2.6)	*87.0*	86.7 (-0.3)	86.8	86.7 (-0.1)	85.5	86.0 (+0.5)
	Bi-GRU	86.3	85.0 (-1.2)	87.3	*87.4* (+0.1)	87.4	86.9 (-0.4)	85.8	86.5 (+0.7)
MPQA	+	**85.7**	84.1 (-1.6)	85.6	84.8 (-0.8)	85.3	84.3 (-1.0)	84.5	84.1 (-0.4)
	GRU	84.5	83.1 (-1.3)	84.1	84.0 (-0.1)	83.7	83.0 (-0.8)	84.3	*85.0* (+0.7)
	Bi-GRU	*85.3*	83.8 (-1.5)	84.5	83.6 (-0.9)	84.4	83.2 (-1.2)	84.7	83.8 (-0.9)
SST	+	77.3	76.0 (-1.3)	79.3	**79.5** (+0.2)	77.9	76.2 (-1.7)	78.6	78.8 (+0.2)
	GRU	74.9	71.6 (-3.3)	*76.3*	75.2 (-1.1)	72.6	72.1 (-0.5)	75.3	75.2 (-0.1)
	Bi-GRU	74.3	74.0 (-0.3)	*77.5*	77.0 (-0.5)	73.0	72.2 (-0.8)	75.8	76.1 (+0.3)
SICK-E	+	68.4	*69.6* (+1.2)	69.2	68.8 (-0.4)	68.7	68.8 (+0.1)	69.4	69.5 (+0.1)
	GRU	72.9	71.8 (-1.0)	73.3	*73.6* (+0.3)	71.5	71.8 (+0.3)	72.5	73.3 (+0.9)
	Bi-GRU	72.2	72.5 (+0.2)	74.3	73.2 (-1.1)	71.8	70.7 (-1.2)	**74.3**	74.2 (-0.1)
SICK-R	+	75.7	75.1 (-0.7)	76.0	75.2 (-0.7)	76.0	75.2 (-0.8)	76.2	*76.3* (+0.1)
	GRU	75.5	72.0 (-3.5)	76.0	76.5 (+0.5)	73.3	72.7 (-0.6)	76.6	*76.8* (+0.2)
	Bi-GRU	76.2	73.0 (-3.2)	**77.3**	76.9 (-0.4)	75.6	73.4 (-2.2)	76.6	76.7 (+0.1)

TABLE C.6: **STS15-16 and Benchmark** scores in Pearson's $r \times 100$. These evaluations are done for before and after tuning the embeddings for a specific algebraic composition operation. We also trained embeddings without pretraining (*None*).

		Word2Vec		GloVe		fastText		Paragram	
		Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
STS12 (All)	+	57.1	59.1 (+2.0)	57.7	56.8 (-0.9)	58.4	60.4 (+1.9)	*63.3*	62.5 (-0.7)
	GRU	52.3	48.7 (-3.6)	56.0	55.4 (-0.6)	53.7	53.9 (+0.2)	59.7	*60.5* (+0.8)
	Bi-GRU	54.8	53.3 (-1.5)	57.8	56.8 (-1.0)	56.3	55.1 (-1.2)	**63.4**	61.9 (-1.5)
MSRpar	+	52.2	52.7 (+0.5)	42.9	36.1 (-6.8)	55.2	**56.7** (+1.5)	47.9	46.1 (-1.8)
	GRU	44.9	40.9 (-4.1)	30.7	30.5 (-0.2)	*48.1*	46.1 (-1.9)	44.4	44.5 (+0.1)
	Bi-GRU	*48.6*	47.6 (-1.0)	36.6	38.9 (+2.2)	48.5	45.3 (-3.3)	45.9	46.8 (+0.9)
MSRvid	+	80.0	79.9 (-0.2)	79.1	80.4 (+1.3)	80.5	79.7 (-0.7)	82.9	**84.0** (+1.1)
	GRU	69.5	56.8 (-12.6)	75.3	77.5 (+2.2)	70.3	71.8 (+1.5)	75.2	*77.9* (+2.7)
	Bi-GRU	78.3	71.7 (-6.6)	79.3	76.7 (-2.5)	77.5	74.7 (-2.8)	*82.2*	78.7 (-3.5)
SMTeuroparl	+	32.4	43.4 (+11.0)	38.8	40.2 (+1.4)	28.5	42.0 (+13.5)	42.6	*43.8* (+1.2)
	GRU	26.5	27.7 (+1.3)	45.1	40.2 (-5.0)	32.4	33.6 (+1.2)	40.5	*45.8* (+5.3)
	Bi-GRU	22.3	27.8 (+5.4)	44.0	43.9 (-0.1)	33.7	36.8 (+3.1)	**49.7**	46.9 (-2.8)
OnWN	+	67.8	68.1 (+0.3)	67.4	68.4 (+1.0)	67.8	67.1 (-0.7)	**72.2**	71.7 (-0.6)
	GRU	60.0	62.8 (+2.8)	67.3	66.8 (-0.4)	60.4	58.6 (-1.8)	*69.0*	68.1 (-0.8)
	Bi-GRU	63.6	64.2 (+0.6)	68.8	66.7 (-2.1)	65.5	62.1 (-3.4)	*70.8*	70.6 (-0.2)
SMTnews	+	31.4	33.2 (+1.8)	48.7	48.3 (-0.4)	39.9	39.4 (-0.5)	*62.0*	57.5 (-4.6)
	GRU	49.4	45.9 (-3.5)	58.7	56.9 (-1.8)	44.8	49.3 (+4.5)	**63.9**	60.8 (-3.1)
	Bi-GRU	42.9	38.0 (-4.9)	52.6	49.6 (-3.0)	39.5	44.4 (+4.9)	*63.0*	59.9 (-3.1)
STS13 (All)	+	61.4	63.3 (+1.9)	62.7	61.6 (-1.1)	59.4	64.1 (+4.7)	**71.5**	70.4 (-1.1)
	GRU	46.6	42.6 (-4.0)	57.3	56.1 (-1.2)	47.9	46.2 (-1.8)	*65.8*	63.5 (-2.3)
	Bi-GRU	50.5	45.9 (-4.6)	58.8	57.3 (-1.5)	51.9	50.4 (-1.6)	*64.3*	61.7 (-2.6)
FNWN	+	45.1	46.7 (+1.7)	40.5	40.4 (-0.1)	42.7	48.1 (+5.3)	47.8	**49.7** (+2.0)
	GRU	14.7	19.6 (+4.9)	25.8	26.2 (+0.4)	15.9	15.6 (-0.3)	*45.3*	39.0 (-6.3)
	Bi-GRU	22.7	14.6 (-8.1)	37.3	35.7 (-1.5)	35.0	27.2 (-7.9)	40.7	*42.0* (+1.3)
OnWN	+	68.3	60.8 (-7.5)	65.9	57.7 (-8.2)	63.0	59.8 (-3.2)	**76.8**	72.1 (-4.7)
	GRU	47.3	29.7 (-17.6)	56.4	54.8 (-1.6)	41.4	34.8 (-6.6)	*66.3*	62.4 (-3.9)
	Bi-GRU	50.5	37.2 (-13.3)	56.0	49.3 (-6.7)	43.5	40.0 (-3.4)	*63.9*	55.4 (-8.5)
HDL	+	60.4	69.4 (+9.0)	65.9	69.8 (+3.9)	60.9	71.4 (+10.5)	73.5	**74.3** (+0.8)
	GRU	54.2	58.1 (+3.9)	65.8	64.5 (-1.3)	60.8	62.3 (+1.5)	*70.6*	70.5 (-0.1)
	Bi-GRU	57.6	60.3 (+2.7)	66.2	68.7 (+2.5)	62.5	63.9 (+1.4)	70.5	*71.5* (+0.9)
STS14 (All)	+	59.9	66.0 (+6.0)	63.7	67.1 (+3.4)	61.8	67.3 (+5.5)	**74.0**	73.6 (-0.3)
	GRU	51.9	52.9 (+1.0)	62.9	63.2 (+0.2)	55.6	54.4 (-1.1)	67.0	*67.3* (+0.3)
	Bi-GRU	58.6	58.2 (-0.3)	64.7	62.9 (-1.8)	57.1	58.5 (+1.4)	*68.5*	68.4 (-0.0)
Deft Forum	+	27.6	38.2 (+10.7)	35.1	41.1 (+6.1)	29.9	38.6 (+8.7)	**49.4**	48.8 (-0.6)
	GRU	22.4	24.9 (+2.5)	32.1	*37.1* (+5.1)	22.4	23.4 (+1.1)	35.7	35.9 (+0.2)
	Bi-GRU	29.3	31.6 (+2.3)	33.0	36.7 (+3.8)	24.4	29.8 (+5.4)	38.0	*41.2* (+3.2)
Deft News	+	69.3	72.9 (+3.5)	71.0	**77.3** (+6.3)	67.9	72.2 (+4.3)	76.1	75.4 (-0.7)
	GRU	65.4	69.3 (+3.8)	70.7	70.4 (-0.3)	63.2	67.0 (+3.8)	*74.3*	72.9 (-1.4)
	Bi-GRU	69.2	68.7 (-0.4)	73.8	70.8 (-3.0)	66.5	69.6 (+3.1)	73.6	*74.8* (+1.2)
HDL	+	58.6	67.6 (+9.0)	62.2	65.2 (+3.0)	59.2	68.3 (+9.0)	71.0	**71.1** (+0.2)
	GRU	56.1	57.0 (+0.9)	62.1	61.3 (-0.8)	59.3	60.3 (+1.0)	*69.1*	67.5 (-1.6)
	Bi-GRU	59.5	62.7 (+3.2)	63.9	63.0 (-0.8)	58.6	58.9 (+0.3)	67.6	*68.3* (+0.7)
Images	+	68.2	74.5 (+6.3)	74.9	77.2 (+2.3)	70.1	76.2 (+6.1)	82.3	**83.6** (+1.3)
	GRU	58.5	63.9 (+5.4)	74.3	74.9 (+0.7)	69.3	64.5 (-4.9)	73.2	*80.0* (+6.8)
	Bi-GRU	69.3	69.9 (+0.6)	77.8	75.5 (-2.3)	69.8	71.3 (+1.4)	79.9	*80.5* (+0.6)
OnWN	+	77.8	74.0 (-3.9)	73.6	72.2 (-1.4)	75.3	73.8 (-1.5)	**84.0**	81.7 (-2.3)
	GRU	58.9	48.5 (-10.4)	67.2	66.3 (-0.9)	59.8	54.0 (-5.7)	*75.2*	72.5 (-2.7)
	Bi-GRU	66.2	55.8 (-10.4)	72.1	63.7 (-8.3)	60.0	60.8 (+0.8)	*74.4*	70.1 (-4.3)
Tweet-News	+	50.8	61.7 (+10.9)	58.2	65.4 (+7.2)	59.2	66.1 (+6.9)	**72.4**	72.3 (-0.1)
	GRU	46.4	52.6 (+6.2)	63.6	62.8 (-0.8)	50.8	52.6 (+1.7)	*66.2*	65.7 (-0.4)
	Bi-GRU	52.6	56.3 (+3.6)	60.5	61.9 (+1.5)	56.0	56.0 (+0.1)	68.2	*68.7* (+0.5)

TABLE C.7: **STS15-16 and Benchmark** scores in Pearson's $r \times 100$. These evaluations are done for before and after tuning the embeddings for a specific algebraic composition operation. We also trained embeddings without pretraining (*None*).

		Word2Vec		GloVe		fastText		Paragram	
		Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
STS15 (All)	+	66.2	73.3 (+7.1)	67.3	71.6 (+4.3)	65.4	72.9 (+7.5)	77.3	**77.8** (+0.5)
	GRU	55.5	60.2 (+4.7)	66.3	67.9 (+1.6)	60.9	60.7 (-0.2)	72.4	*72.5* (+0.1)
	Bi-GRU	63.0	64.2 (+1.3)	69.2	67.6 (-1.6)	62.8	64.3 (+1.5)	*73.3*	73.0 (-0.3)
Ans. Forums	+	51.8	66.2 (+14.4)	52.9	60.9 (+7.9)	49.8	59.9 (+10.1)	69.8	**72.1** (+2.3)
	GRU	35.8	38.0 (+2.2)	52.1	53.1 (+1.0)	43.1	41.6 (-1.5)	*65.7*	64.5 (-1.2)
	Bi-GRU	44.4	45.7 (+1.3)	55.8	54.4 (-1.4)	43.0	44.1 (+1.1)	*67.2*	66.2 (-1.0)
Ans. Students	+	72.7	73.8 (+1.2)	67.6	70.2 (+2.6)	69.7	73.2 (+3.6)	**74.5**	74.2 (-0.3)
	GRU	57.8	64.0 (+6.2)	61.8	66.1 (+4.4)	62.4	61.3 (-1.1)	*69.4*	68.8 (-0.6)
	Bi-GRU	65.5	66.8 (+1.3)	66.5	63.5 (-3.1)	65.6	66.8 (+1.2)	*68.3*	68.1 (-0.2)
Belief	+	55.2	68.0 (+12.8)	59.0	67.9 (+8.9)	52.2	67.4 (+15.2)	75.9	**76.6** (+0.7)
	GRU	42.2	51.8 (+9.6)	61.0	63.3 (+2.4)	45.5	50.3 (+4.8)	*67.6*	67.4 (-0.1)
	Bi-GRU	50.3	54.3 (+4.0)	64.1	62.3 (-1.8)	43.4	51.1 (+7.7)	71.2	*71.6* (+0.4)
HDL	+	61.7	71.5 (+9.8)	68.6	72.6 (+4.0)	64.0	74.8 (+10.8)	76.9	**77.0** (+0.1)
	GRU	60.0	63.1 (+3.1)	67.9	68.2 (+0.3)	68.3	66.6 (-1.7)	*73.5*	72.0 (-1.4)
	Bi-GRU	62.6	67.5 (+4.9)	70.1	70.6 (+0.5)	67.7	67.6 (-0.2)	74.0	*74.2* (+0.2)
Images	+	76.9	80.9 (+4.0)	77.2	79.4 (+2.2)	77.0	80.1 (+3.1)	84.9	**85.5** (+0.6)
	GRU	65.4	68.9 (+3.5)	78.9	78.9 (+0.0)	68.4	68.9 (+0.5)	80.2	*83.2* (+3.0)
	Bi-GRU	76.4	72.6 (-3.8)	80.2	78.0 (-2.1)	74.7	75.3 (+0.6)	*81.9*	81.0 (-1.0)
STS16 (All)	+	54.9	62.0 (+7.1)	58.3	64.2 (+5.9)	55.6	62.0 (+6.4)	69.2	**70.3** (+1.1)
	GRU	45.8	45.9 (+0.1)	59.2	59.4 (+0.1)	51.2	50.0 (-1.2)	*61.7*	61.5 (-0.2)
	Bi-GRU	51.0	51.3 (+0.3)	60.4	60.6 (+0.1)	51.7	52.6 (+0.9)	62.3	*65.2* (+2.9)
Ans.-Ans.	+	27.0	37.9 (+10.8)	25.8	32.1 (+6.3)	15.7	25.4 (+9.7)	**49.3**	49.3 (-0.1)
	GRU	19.7	18.8 (-1.0)	33.0	31.5 (-1.6)	14.6	16.7 (+2.0)	*42.5*	40.2 (-2.3)
	Bi-GRU	25.5	24.5 (-1.1)	31.5	30.7 (-0.8)	19.1	20.2 (+1.0)	*47.0*	43.1 (-3.9)
HDL	+	59.5	68.1 (+8.6)	65.6	70.1 (+4.4)	61.5	71.8 (+10.3)	71.9	**73.7** (+1.7)
	GRU	56.4	56.4 (+0.1)	66.9	66.7 (-0.2)	64.3	65.1 (+0.7)	70.1	*70.9* (+0.8)
	Bi-GRU	60.0	60.8 (+0.8)	68.6	69.6 (+1.0)	63.2	63.4 (+0.1)	68.6	*71.2* (+2.6)
Plagiarism	+	68.8	74.1 (+5.4)	73.2	74.7 (+1.5)	70.3	75.2 (+4.9)	79.1	**81.2** (+2.1)
	GRU	53.3	55.6 (+2.3)	70.7	69.9 (-0.8)	66.8	60.6 (-6.2)	74.0	*75.1* (+1.1)
	Bi-GRU	59.0	67.2 (+8.2)	73.1	73.6 (+0.5)	64.9	64.0 (-1.0)	77.9	*78.1* (+0.2)
Post Editing	+	77.3	**82.3** (+4.9)	69.1	80.1 (+11.0)	73.9	81.7 (+7.9)	80.9	81.8 (+0.9)
	GRU	63.3	71.7 (+8.4)	74.2	77.5 (+3.3)	67.6	71.0 (+3.4)	79.7	*80.2* (+0.5)
	Bi-GRU	65.0	72.0 (+7.0)	77.2	79.0 (+1.8)	69.0	73.3 (+4.3)	81.5	*82.2* (+0.8)
Quest.-Quest.	+	41.7	47.1 (+5.4)	59.9	65.8 (+5.9)	59.5	57.3 (-2.1)	65.7	**66.4** (+0.7)
	GRU	36.4	25.7 (-10.8)	*51.8*	51.8 (-0.1)	43.8	36.2 (-7.5)	40.1	39.3 (-0.9)
	Bi-GRU	46.0	30.7 (-15.3)	*52.4*	50.2 (-2.1)	43.0	42.8 (-0.2)	33.9	50.7 (+16.7)
STS Benchmark	+	57.9	61.4 (+3.5)	61.9	59.8 (-2.1)	56.9	61.6 (+4.7)	64.8	*66.0* (+1.2)
	GRU	64.6	64.5 (-0.2)	69.5	68.5 (-1.0)	65.0	65.4 (+0.3)	*70.6*	70.0 (-0.6)
	Bi-GRU	64.0	64.7 (+0.7)	69.5	69.4 (-0.1)	67.2	65.8 (-1.5)	**71.1**	70.4 (-0.7)

TABLE C.8: **STS12-16** p-values for all evaluations. Results that are not statistically significant are displayed in bold.

		Word2Vec		GloVe		fastText		Paragram	
		Pre.	Tuned	Pre.	Tuned	Pre.	Tuned	Pre.	Tuned
MSRpar	+	1.1e-53	6.7e-55	6.7e-35	1.8e-24	4.8e-61	6.1e-65	2.3e-44	8.3e-41
	GRU	1.5e-38	1.5e-31	7.1e-18	1.2e-17	1.3e-44	8.9e-41	1.4e-37	1.0e-37
	Bi-GRU	8.2e-46	1.1e-43	3.1e-25	1.8e-28	1.4e-45	3.9e-39	2.7e-40	5.4e-42
MSRvid	+	2.3e-168	5.0e-167	6.4e-162	2.5e-171	1.2e-171	2.8e-166	3.3e-191	1.2e-200
	GRU	4.0e-109	2.1e-65	2.5e-138	2.2e-151	8.3e-113	6.7e-120	1.1e-137	1.1e-153
	Bi-GRU	4.3e-156	3.5e-119	8.3e-163	2.1e-146	3.7e-151	1.5e-134	2.5e-185	3.4e-159
SMTeuroparl	+	1.2e-12	1.8e-22	6.8e-18	3.0e-19	4.9e-10	4.6e-21	1.1e-21	5.7e-23
	GRU	8.4e-09	1.5e-09	2.1e-24	3.2e-19	1.1e-12	1.4e-13	1.4e-19	3.1e-25
	Bi-GRU	1.3e-06	1.4e-09	3.5e-23	4.8e-23	1.1e-13	3.4e-16	6.1e-30	2.0e-26
OnWN	+	4.4e-102	3.0e-103	2.0e-100	1.5e-104	5.9e-102	2.6e-99	5.3e-122	3.5e-119
	GRU	1.5e-74	1.5e-83	6.0e-100	2.9e-98	8.6e-76	1.9e-70	5.8e-107	1.9e-103
	Bi-GRU	2.2e-86	1.9e-88	3.0e-106	8.3e-98	3.8e-93	2.6e-81	3.6e-115	2.5e-114
SMTnews	+	1.4e-10	1.0e-11	3.3e-25	9.2e-25	1.2e-16	3.1e-16	8.4e-44	1.8e-36
	GRU	6.4e-26	3.2e-22	2.2e-38	1.2e-35	4.0e-21	7.5e-26	4.3e-47	1.2e-41
	Bi-GRU	2.5e-19	3.6e-15	9.6e-30	4.0e-26	2.4e-16	1.2e-20	1.4e-45	2.9e-40
FNWN	+	7.6e-11	1.2e-11	7.3e-09	8.0e-09	8.6e-10	2.6e-12	3.7e-12	3.3e-13
	GRU	**4.4e-02**	**6.9e-03**	3.4e-04	2.6e-04	**2.9e-02**	**3.2e-02**	6.1e-11	2.9e-08
	Bi-GRU	**1.7e-03**	**4.5e-02**	1.3e-07	4.4e-07	7.7e-07	1.5e-04	6.2e-09	1.7e-09
OnWN	+	2.1e-78	4.3e-58	3.5e-71	3.6e-51	2.9e-63	1.2e-55	3.2e-110	3.3e-91
	GRU	1.3e-32	6.6e-13	1.7e-48	2.5e-45	1.3e-24	2.2e-17	2.6e-72	7.1e-62
	Bi-GRU	1.3e-37	7.8e-20	1.0e-47	1.0e-35	2.8e-27	5.0e-23	9.7e-66	2.2e-46
HDL	+	1.1e-75	1.2e-108	9.8e-95	1.6e-110	3.4e-77	7.8e-118	3.1e-128	1.0e-132
	GRU	1.9e-58	6.5e-69	2.5e-94	1.3e-89	3.9e-77	5.2e-82	3.9e-114	8.0e-114
	Bi-GRU	1.6e-67	1.8e-75	9.7e-96	1.2e-105	1.5e-82	2.6e-87	7.5e-114	3.1e-118
Deft Forum	+	2.8e-09	4.1e-17	1.8e-14	8.2e-20	9.3e-11	2.0e-17	4.0e-29	2.5e-28
	GRU	1.5e-06	8.2e-08	3.3e-12	3.8e-16	1.6e-06	4.9e-07	5.5e-15	4.0e-15
	Bi-GRU	2.4e-10	6.6e-12	7.3e-13	8.1e-16	1.7e-07	1.2e-10	6.7e-17	7.6e-20
Deft News	+	2.7e-44	6.5e-51	2.4e-47	6.4e-61	7.6e-42	1.3e-49	5.2e-58	2.3e-56
	GRU	4.6e-38	3.5e-44	9.8e-47	2.9e-46	6.7e-35	1.8e-40	6.3e-54	4.6e-51
	Bi-GRU	4.6e-44	2.7e-43	7.0e-53	6.0e-47	1.2e-39	8.0e-45	1.6e-52	4.8e-55
HDL	+	2.4e-70	4.6e-101	2.3e-81	8.0e-92	3.0e-72	4.9e-104	5.8e-116	1.1e-116
	GRU	2.8e-63	7.6e-66	4.1e-81	1.2e-78	1.6e-72	1.9e-75	2.3e-107	7.8e-101
	Bi-GRU	4.9e-73	4.4e-83	3.8e-87	2.3e-84	2.7e-70	3.7e-71	2.1e-101	2.4e-104
Images	+	9.0e-104	6.4e-134	4.0e-136	1.6e-149	5.0e-112	1.7e-143	4.5e-186	5.6e-197
	GRU	5.2e-70	3.0e-87	2.3e-132	4.9e-136	1.5e-108	2.3e-89	1.0e-126	6.3e-168
	Bi-GRU	2.1e-108	3.5e-111	7.9e-153	4.4e-139	9.6e-111	3.0e-117	4.0e-167	1.0e-171
OnWN	+	1.9e-153	8.4e-131	5.3e-129	1.2e-121	4.2e-138	5.2e-130	8.6e-201	8.1e-181
	GRU	2.3e-71	1.5e-45	8.8e-100	3.6e-96	8.9e-74	4.2e-58	1.3e-137	2.7e-123
	Bi-GRU	8.3e-96	1.1e-62	4.6e-121	1.2e-86	1.4e-74	3.9e-77	4.4e-133	7.4e-112
Tweet-News	+	1.9e-50	4.9e-80	3.5e-69	8.9e-93	3.8e-72	2.5e-95	1.0e-122	4.1e-122
	GRU	2.6e-41	1.2e-54	2.6e-86	1.3e-83	1.6e-50	1.7e-54	1.1e-95	5.0e-94
	Bi-GRU	1.1e-54	7.1e-64	5.2e-76	1.2e-80	4.9e-63	3.0e-63	9.6e-104	1.1e-105
Ans. Forums	+	4.0e-27	1.0e-48	1.8e-28	2.1e-39	7.0e-25	8.1e-38	4.9e-56	2.9e-61
	GRU	9.2e-13	2.5e-14	1.9e-27	1.2e-28	2.4e-18	4.0e-17	1.1e-47	1.8e-45
	Bi-GRU	1.4e-19	9.2e-21	4.2e-32	3.0e-30	2.8e-18	2.9e-19	1.6e-50	1.4e-48
Ans. Students	+	3.3e-124	3.6e-130	3.3e-101	3.7e-112	6.0e-110	6.6e-127	7.7e-134	5.3e-132
	GRU	4.9e-68	9.9e-88	3.9e-80	1.5e-95	2.9e-82	1.2e-78	5.8e-109	2.1e-106
	Bi-GRU	6.2e-93	5.8e-98	4.5e-97	9.1e-86	2.5e-93	4.3e-98	3.7e-104	1.8e-103
Belief	+	2.8e-31	3.6e-52	1.6e-36	4.6e-52	1.3e-27	4.3e-51	1.7e-71	1.4e-73
	GRU	1.2e-17	3.5e-27	1.6e-39	1.9e-43	1.6e-20	1.9e-25	2.5e-51	5.0e-51
	Bi-GRU	2.1e-25	3.9e-30	1.1e-44	1.1e-41	1.1e-18	2.2e-26	3.3e-59	3.8e-60
HDL	+	8.4e-80	1.7e-118	2.9e-105	1.7e-123	1.6e-87	3.4e-135	1.9e-147	6.2e-148
	GRU	1.7e-74	1.4e-84	2.0e-102	8.5e-104	3.0e-104	3.0e-97	2.9e-128	5.5e-121
	Bi-GRU	5.3e-83	6.9e-101	3.9e-112	2.2e-114	7.3e-102	4.2e-101	1.0e-130	6.3e-132
Images	+	1.0e-147	1.1e-174	3.7e-149	1.1e-163	2.5e-148	5.4e-169	1.3e-209	9.1e-216
	GRU	1.1e-92	8.4e-107	1.4e-160	1.3e-160	1.0e-104	7.5e-107	2.3e-169	2.1e-193
	Bi-GRU	2.9e-144	1.7e-123	2.7e-169	1.3e-154	1.6e-134	8.0e-138	6.8e-183	2.3e-175
Ans.-Ans.	+	1.3e-05	4.5e-10	3.1e-05	1.7e-07	**1.2e-02**	4.1e-05	5.5e-17	6.2e-17
	GRU	**1.6e-03**	**2.7e-03**	7.0e-08	3.0e-07	**2.0e-02**	**7.8e-03**	1.5e-12	2.9e-11
	Bi-GRU	3.8e-05	8.2e-05	3.0e-07	6.3e-07	**2.2e-03**	**1.2e-03**	2.4e-15	6.6e-13
HDL	+	2.9e-25	2.4e-35	4.6e-32	4.4e-38	2.7e-27	9.7e-41	5.6e-41	7.5e-44
	GRU	2.7e-22	2.4e-22	1.1e-33	2.1e-33	1.9e-30	2.4e-31	3.7e-38	2.5e-39
	Bi-GRU	9.6e-26	1.3e-26	4.9e-36	2.2e-37	3.4e-29	2.3e-29	5.1e-36	8.0e-40
Plagiarism	+	1.6e-33	2.4e-41	6.2e-40	2.7e-42	1.4e-35	3.4e-43	1.4e-50	2.9e-55
	GRU	2.9e-18	4.8e-20	3.4e-36	4.2e-35	4.3e-31	1.8e-24	3.4e-41	4.8e-43
	Bi-GRU	6.4e-23	1.6e-31	9.4e-40	1.8e-40	6.3e-29	7.1e-28	3.4e-48	1.3e-48
Post Editing	+	7.9e-50	2.4e-61	6.4e-36	7.6e-56	2.6e-43	7.0e-60	8.1e-58	5.2e-60
	GRU	9.9e-29	9.6e-40	5.4e-44	3.4e-50	6.9e-34	1.1e-38	6.2e-55	3.6e-56
	Bi-GRU	1.0e-30	2.4e-40	1.6e-49	2.4e-53	6.9e-36	2.2e-42	3.1e-59	3.3e-61
Quest.-Quest.	+	3.4e-10	6.5e-13	9.7e-22	2.6e-27	2.3e-21	1.1e-19	3.7e-27	6.4e-28
	GRU	5.9e-08	1.8e-04	8.9e-16	9.9e-16	3.4e-11	7.0e-08	1.7e-09	4.1e-09
	Bi-GRU	2.3e-12	6.1e-06	4.2e-16	9.5e-15	8.4e-11	1.1e-10	4.9e-07	4.9e-15

Bibliography

[1] Eneko Agirre et al. "A study on similarity and relatedness using distributional and wordnet-based approaches". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2009, pp. 19–27.

[2] Eneko Agirre et al. "sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity". In: *In* SEM 2013: The Second Joint Conference on Lexical and Computational Semantics. Association for Computational Linguistics*. Citeseer. 2013.

[3] Eneko Agirre et al. "Semeval-2012 task 6: A pilot on semantic textual similarity". In: *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics. 2012, pp. 385–393.

[4] Eneko Agirre et al. "SemEval-2014 Task 10: Multilingual Semantic Textual Similarity." In: *SemEval@ COLING*. 2014, pp. 81–91.

[5] Eneko Agirre et al. "SemEval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability." In: *SemEval@ NAACL-HLT*. 2015, pp. 252–263.

[6] Eneko Agirre et al. "SemEval-2016 Task 1: Semantic Textual Similarity, Monolingual and Cross-Lingual Evaluation." In: *SemEval@ NAACL-HLT*. 2016, pp. 497–511.

[7] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. "A simple but tough-to-beat baseline for sentence embeddings". In: (2016).

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[9] Collin F Baker, Charles J Fillmore, and John B Lowe. "The berkeley framenet project". In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*. Association for Computational Linguistics. 1998, pp. 86–90.

[10] Simon Baker, Roi Reichart, and Anna Korhonen. "An Unsupervised Model for Instance Level Subcategorization Acquisition." In: *EMNLP*. 2014, pp. 278–289.

[11] Marco Baroni and Roberto Zamparelli. "Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2010, pp. 1183–1193.

[12] Henri Béjoint. *Modern lexicography: An introduction*. Oxford University Press, 2000.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[14] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[15] Steven Bird. "NLTK: the natural language toolkit". In: *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics. 2006, pp. 69–72.

[16] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[17] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606* (2016).

[18] Denny Britz. *Understanding Convolutional Neural Networks for NLP*. URL: `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/` (visited on 11/07/2015).

[19] Sylvain Bromberger. "What are Words? Comments on Kaplan (1990), on Hawthorne and Lepore, and on the Issue". In: *The Journal of Philosophy* 108.9 (2011), pp. 486–503.

[20] Jane Bromley et al. "Signature verification using a" siamese" time delay neural network". In: *Advances in Neural Information Processing Systems*. 1994, pp. 737–744.

[21] Peter F Brown et al. "Class-based n-gram models of natural language". In: *Computational linguistics* 18.4 (1992), pp. 467–479.

[22] Elia Bruni et al. "Distributional semantics in technicolor". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics. 2012, pp. 136–145.

[23] Daniel Cer et al. "SemEval-2017 Task 1: Semantic Textual Similarity-Multilingual and Cross-lingual Focused Evaluation". In: *arXiv preprint arXiv:1708.00055* (2017).

[24] Olivier Chapelle et al. "Expected reciprocal rank for graded relevance". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM. 2009, pp. 621–630.

[25] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[26] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[27] Junyoung Chung et al. "Gated feedback recurrent neural networks". In: *International Conference on Machine Learning*. 2015, pp. 2067–2075.

[28] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. "Mathematical foundations for a compositional distributional model of meaning". In: *arXiv preprint arXiv:1003.4394* (2010).

[29] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.

[30] Alexis Conneau et al. "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *arXiv preprint arXiv:1705.02364* (2017).

[31] Scott Deerwester. "Improving information retrieval with latent semantic indexing". In: (1988).

[32] Scott Deerwester et al. "Indexing by latent semantic analysis". In: *Journal of the American society for information science* 41.6 (1990), p. 391.

[33] *Discontinuity*. URL: `https://en.wikipedia.org/wiki/Discontinuity_(linguistics)` (visited on 11/20/2017).

[34] William B Dolan and Chris Brockett. "Automatically constructing a corpus of sentential paraphrases". In: *Proc. of IWP*. 2005.

[35] Cícero Nogueira Dos Santos and Maira Gatti. "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts." In: *COLING*. 2014, pp. 69–78.

[36] Timothy Dozat. "Incorporating nesterov momentum into adam". In: (2016).

[37] Katrin Erk and Sebastian Padó. "A structured vector space model for word meaning in context". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2008, pp. 897–906.

[38] Manaal Faruqui and Chris Dyer. "Community Evaluation and Exchange of Word Vectors at wordvectors.org". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, USA: Association for Computational Linguistics, 2014.

[39] Lev Finkelstein et al. "Placing search in context: The concept revisited". In: *Proceedings of the 10th international conference on World Wide Web*. ACM. 2001, pp. 406–414.

[40] John Rupert Firth. *Selected papers of JR Firth, 1952-59*. Indiana University Press, 1968.

[41] Gottlob Frege. "On concept and object". In: *The Frege Reader* (1892), pp. 181–193.

[42] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. 2016, pp. 1050–1059.

[43] Debasis Ganguly et al. "Word embedding based generalized language model for information retrieval". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2015, pp. 795–798.

[44] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. "PPDB: The Paraphrase Database." In: *HLT-NAACL*. 2013, pp. 758–764.

[45] Luca Gasparri and Diego Marconi. "Word Meaning". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2016. Metaphysics Research Lab, Stanford University, 2016.

[46] Daniela Gerz et al. "SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity". In: *EMNLP*. 2016.

[47] Alex Gittens, Dimitris Achlioptas, and Michael W Mahoney. "Skip-gram - zipf + uniform = vector additivity". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2017, pp. 69–76.

[48] Cliff Goddard and Anna Wierzbicka. *Meaning and universal grammar: Theory and empirical findings*. Vol. 1. John Benjamins Publishing, 2002.

[49] David Goldberg. "What every computer scientist should know about floating-point arithmetic". In: *ACM Computing Surveys (CSUR)* 23.1 (1991), pp. 5–48.

[50] William Sealy Gosset. "The probable error of a mean". In: *Biometrika* (1908), pp. 1–25.

[51] Gregory Goth. "Deep or Shallow, NLP is Breaking out". In: *Commun. ACM* 59.3 (Feb. 2016), pp. 13–16. ISSN: 0001-0782. DOI: 10.1145/2874915. URL: http://doi.acm.org/10.1145/2874915.

[52] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: *arXiv preprint arXiv:1410.5401* (2014).

[53] Edward Grefenstette and Mehrnoosh Sadrzadeh. "Experimental support for a categorical compositional distributional model of meaning". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 1394–1404.

[54] H Paul Grice, Peter Cole, Jerry Morgan, et al. "Logic and conversation". In: *1975* (1975), pp. 41–58.

[55] Guy Halawi et al. "Large-scale learning of word relatedness with constraints". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 1406–1414.

[56] Patrick Hanks and K Allan. *Lexicography from earliest times to the present*. 2013.

[57] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification". In: *arXiv preprint arXiv:1703.07737* (2017).

[58] Felix Hill, Roi Reichart, and Anna Korhonen. "Simlex-999: Evaluating semantic models with (genuine) similarity estimation". In: *Computational Linguistics* (2016).

[59] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[60] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications*. World Scientific, 1987, pp. 411–415.

[61] Eduard Hovy et al. "OntoNotes: the 90% solution". In: *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*. Association for Computational Linguistics. 2006, pp. 57–60.

[62] Minqing Hu and Bing Liu. "Mining and summarizing customer reviews". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 168–177.

[63] Ray Jackendoff. "Toward an explanatory semantic representation". In: *Linguistic inquiry* 7.1 (1976), pp. 89–150.

[64] Howard Jackson. *Lexicography: an introduction*. Psychology Press, 2002.

[65] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144* (2016).

[66] Theo MV Janssen. "Frege, contextuality and compositionality". In: *Journal of Logic, Language and Information* 10.1 (2001), pp. 115–136.

[67] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188* (2014).

[68] Tom Kenter, Alexey Borisov, and Maarten de Rijke. "Siamese cbow: Optimizing word embeddings for sentence representations". In: *arXiv preprint arXiv:1606.04640* (2016).

[69] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[70] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[71] Ryan Kiros et al. "Skip-thought vectors". In: *Advances in neural information processing systems*. 2015, pp. 3294–3302.

[72] Jayanth Koushik and Hiroaki Hayashi. "Improving Stochastic Gradient Descent with Feedback". In: *arXiv preprint arXiv:1611.01505* (2016).

[73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[74] Neeraj Kumar, Li Zhang, and Shree Nayar. "What is a good nearest neighbors algorithm for finding similar patches in images?" In: *Computer Vision–ECCV 2008* (2008), pp. 364–378.

[75] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2014, pp. 1188–1196.

[76] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[77] Xin Li and Dan Roth. "Learning question classifiers". In: *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics. 2002, pp. 1–7.

[78] Shujie Liu et al. "A recursive recurrent neural network for statistical machine translation". In: (2014).

[79] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[80] Minh-Thang Luong, Richard Socher, and Christopher D. Manning. "Better Word Representations with Recursive Neural Networks for Morphology". In: *CoNLL*. 2013.

[81] Andrew L Maas et al. "Learning word vectors for sentiment analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 142–150.

[82] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[83] Marco Marelli et al. "A SICK cure for the evaluation of compositional distributional semantic models." In: *LREC*. 2014, pp. 216–223.

[84] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[85] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[86] George Miller and Christiane Fellbaum. *Wordnet: An electronic lexical database*. 1998.

[87] George A. Miller and Walter G. Charles. "Contextual correlates of semantic similarity". In: *Language and Cognitive Processes* 6.1 (1991), pp. 1–28. DOI: 10.1080/01690969108406936. eprint: http://dx.doi.org/10.1080/01690969108406936. URL: http://dx.doi.org/10.1080/01690969108406936.

[88] Jeff Mitchell and Mirella Lapata. "Language models based on semantic composition". In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*. Association for Computational Linguistics. 2009, pp. 430–439.

[89] Jeff Mitchell and Mirella Lapata. "Vector-based Models of Semantic Composition." In: *ACL*. 2008, pp. 236–244.

[90] Tom M Mitchell et al. "Predicting human brain activity associated with the meanings of nouns". In: *science* 320.5880 (2008), pp. 1191–1195.

[91] Andriy Mnih and Geoffrey Hinton. "Three new graphical models for statistical language modelling". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 641–648.

[92] Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. "Dependency tree-based sentiment classification using CRFs with hidden variables". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 786–794.

[93] Douglas L Nelson, Cathy L McEvoy, and Thomas A Schreiber. "The University of South Florida free association, rhyme, and word fragment norms". In: *Behavior Research Methods, Instruments, & Computers* 36.3 (2004), pp. 402–407.

[94] Stephen M. Omohundro. "Five Balltree Construction Algorithms". In: TR-89-063 (1989).

[95] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[96] Bo Pang and Lillian Lee. "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts". In: *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2004, p. 271.

[97] Denis Paperno and Marco Baroni. "When the whole is less than the sum of its parts: How composition affects pmi values in distributional semantic vectors". In: *Computational Linguistics* (2016).

[98] Karl Pearson. "Mathematical contributions to the theory of evolution. III. Regression, heredity, and panmixia". In: *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 187 (1896), pp. 253–318.

[99] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[100] Lutz Prechelt. "Automatic early stopping using cross validation: quantifying the criteria". In: *Neural Networks* 11.4 (1998), pp. 761–767.

[101] Hilary Putnam. "Is semantics possible?" In: *Metaphilosophy* 1.3 (1970), pp. 187–201.

[102] Kira Radinsky et al. "A word at a time: computing word relatedness using temporal semantic analysis". In: *Proceedings of the 20th international conference on World wide web*. ACM. 2011, pp. 337–346.

[103] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[104] Herbert Rubenstein and John B. Goodenough. "Contextual Correlates of Synonymy". In: *Commun. ACM* 8.10 (Oct. 1965), pp. 627–633. ISSN: 0001-0782. DOI: 10.1145/365628.365657. URL: http://doi.acm.org/10.1145/365628.365657.

[105] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[106] Sebastian Ruder. *On word embeddings, Part 1*. URL: http://ruder.io/word-embeddings-1/ (visited on 11/05/2017).

[107] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. "Dynamic Routing Between Capsules". In: *arXiv preprint arXiv:1710.09829* (2017).

[108] G Salton and MJ McGill. *Introduction to modern information Philadelphia, PA. American Association for Artificial Intelligence retrieval*. 1983.

[109] Thijs Scheepers, Efstratios Gavves, and Evangelos Kanoulas. "Analyzing the compositional properties of word embeddings". In: (2017).

[110] Thijs Scheepers, Efstratios Gavves, and Evangelos Kanoulas. "Improving Word Embedding Compositionality using Lexicographic Definitions". In: (2017).

[111] Tobias Schnabel et al. "Evaluation methods for unsupervised word embeddings". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. Ed. by Lluís Màrquez et al. The Association for Computational Linguistics, 2015, pp. 298–307. ISBN: 978-1-941643-32-7. URL: http://aclweb.org/anthology/D/D15/D15-1036.pdf.

[112] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[113] Hinrich Schütze. "Word space". In: *Advances in neural information processing systems*. 1993, pp. 895–902.

[114] John Searle. "The background of meaning". In: *Speech act theory and pragmatics* (1980), pp. 221–232.

[115] John R Searle. *Expression and meaning: Studies in the theory of speech acts*. Cambridge University Press, 1985.

[116] Ali Sharif Razavian et al. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.

[117] Richard Socher et al. "Parsing with Compositional Vector Grammars." In: *ACL*. 2013, pp. 455–465.

[118] Richard Socher et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631. Citeseer. 2013, p. 1642.

[119] Richard Socher et al. "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*. Association for Computational Linguistics. 2012, pp. 1201–1211.

[120] Charles Spearman. "The proof and measurement of association between two things". In: *The American journal of psychology* 15.1 (1904), pp. 72–101.

[121] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[122] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. "End-to-end memory networks". In: *Advances in neural information processing systems*. 2015, pp. 2440–2448.

[123] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[124] Zoltán Gendler Szabó. "Compositionality". In: *Stanford encyclopedia of philosophy* (2010).

[125] Zolt>aacute;n Gendler Szabó<. "Compositionality". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2017. Metaphysics Research Lab, Stanford University, 2017.

[126] Kai Sheng Tai, Richard Socher, and Christopher D Manning. "Improved semantic representations from tree-structured long short-term memory networks". In: *arXiv preprint arXiv:1503.00075* (2015).

[127] Charles Travis. "Saying and understanding: A generative theory of illocutions". In: (1977).

[128] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. "Neural vector spaces for unsupervised information retrieval". In: *arXiv preprint arXiv:1708.02702* (2017).

[129] Sida Wang and Christopher D Manning. "Baselines and bigrams: Simple, good sentiment and topic classification". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics. 2012, pp. 90–94.

[130] Hans H Wellisch. "Ebla: The world's oldest library". In: *The Journal of Library History (1974-1987)* 16.3 (1981), pp. 488–500.

[131] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory networks". In: *arXiv preprint arXiv:1410.3916* (2014).

[132] Janyce Wiebe, Theresa Wilson, and Claire Cardie. "Annotating expressions of opinions and emotions in language". In: *Language resources and evaluation* 39.2 (2005), pp. 165–210.

[133] Anna Wierzbicka. "Semantic primitives". In: (1972).

[134] John Wieting et al. "From paraphrase database to compositional paraphrase model and back". In: *arXiv preprint arXiv:1506.03487* (2015).

[135] John Wieting et al. "Towards universal paraphrastic sentence embeddings". In: *arXiv preprint arXiv:1511.08198* (2015).

[136] Dongqiang Yang and David M. W. Powers. "Verb Similarity on the Taxonomy of Wordnet". In: *In the 3rd International WordNet Conference (GWC-06), Jeju Island, Korea*. 2006.

[137] Ainur Yessenalina and Claire Cardie. "Compositional matrix-space models for sentiment analysis". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 172–182.

[138] Peter N Yianilos. "Data structures and algorithms for nearest neighbor search in general metric spaces". In: *SODA*. Vol. 93. 194. 1993, pp. 311–321.

[139] Mahmood Yousefi-Azar and Len Hamey. "Text summarization using unsupervised deep learning". In: *Expert Systems with Applications* 68 (2017), pp. 93–105.

[140] George Kingsley Zipf. "Selected studies of the principle of relative frequency in language". In: (1932).