Improving Word Embedding Compositionality using Lexicographic Definitions

Thijs Scheepers
University of Amsterdam
thijs.scheepers@student.uva.nl

Evangelos Kanoulas
University of Amsterdam
e.kanoulas@uva.nl

Efstratios Gavves
University of Amsterdam
e.gavves@uva.nl

ABSTRACT

We present an in-depth analysis of four popular word embeddings (*Word2Vec*, *GloVe*, *fastText* and *Paragram*) 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*. 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.

In our analysis, we evaluate original embeddings, as well as tuned embeddings, using existing word similarity and sentence embedding evaluation methods. 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, called *CompVecEval*, to be a key contribution.

ACM Reference Format:
Thijs Scheepers, Evangelos Kanoulas, and Efstratios Gavves. 2018. Improving Word Embedding Compositionality using Lexicographic Definitions. In *The Web Conference 2018, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3178876.3186007

1 INTRODUCTION

The principle of *compositionality* [17] states that the meaning of an expression comprises the meaning of its constituents as well as the rules to combine them. It was first introduced to explain the way humans understand language. Today, the same principle is used to model the way computers represent meaning [49]. While the exact interpretation and implications of word specific meaning is debatable, it is clear that lexicography, i.e. the act of writing dictionaries, is important to illustrate the relationship between words and their meaning [18]. 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 domesticated carnivorous mammal with soft fur, a short snout, and retractable claws."* into the lexical semantic representation of *'cat'*.

Lexical representation using real valued vectors, i.e. *word embeddings* [36], have become an important aspect of neural models for information retrieval, natural language processing as well as other text related model classes. Ever since the paper which made these real valued embeddings popular, there has been interest in their compositional properties [37]. These properties are especially important in the context of deep neural models where, in various architectures, multiple representations can be composed into a single deeper representation.

Finding good representations for words in an unsupervised manner, in practice, often relies on a word's context. Training based on context results in a representation which captures both syntax as well as semantics. While syntactic information is important for composing representations, it is not necessarily useful for applying the meaning of single word embedding in a model. When composing words into a joint representation we often want to create a representation of meaning, i.e. semantics, and often do not care for syntax, even though this information is essential to the act of composition itself. Having a compact representation of meaning can be useful for lots of tasks, such as question answering, web search, machine translation and sentiment analysis. In most of these tasks one would need to combine multiple word embeddings to create a single embedding of the semantics from a combination of words. We define such an operation as *word embedding composition*.

In this work we analyze the practical use of four widely used pretrained word embeddings: *Word2Vec* [37], *GloVe* [44], *fastText* [8] and *Paragram* [57]. These embeddings are often used directly as features or used for *transfer learning* to kick start a model's training operation and improve its final performance. Since most applications of word embeddings require them to be composed, we decided to analyze the most popular pretrained word embeddings on their compositional properties, so we can make more informed decisions on which embeddings to use.

In order to do this, we created a new evaluation method called *CompVecEval*, which tests the compositionality of word embeddings using a test set of lexicographic definitions and lemmas. We used definitions and lemmas from *WordNet* [38]. The novel evaluation method checks if the composed words from the definitions are close to the embedding of the lemma. This means we test specifically

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186007

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 make 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.

Our test differs from others because it uses a balltree ranking [42] 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. Other evaluation methods are invariant to this aspect of the embeddings. 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 [3].

In addition, we built a model to tune the existing embeddings towards better compositionality using the same dictionary definitions. We were able to tune the embeddings using various compositional functions, some of which should be learned and some of which are simple algebraic functions. Using simple algebraic functions is sometimes necessary for large scale production applications that need to process large amounts of data. They are also surprisingly effective as our results show. We were able to create embeddings which performed better or comparable across both the *CompVecEval* task, as well as several word similarity and sentence embedding evaluation tasks. We believe therefore that our tuned embeddings could be an even better candidate for direct use in application specific models or for transfer learning. Our method for tuning these embeddings can be applied using other compositional datasets as well. We call this method of tuning word embeddings using compositional data: *CompVec*[1].

2 RELATED WORK

Compositionality in linguistics was first defined back in 1892 by Frege [17] and later neatly placed into the present context by Janssen [26]. In 2010 the mathematical foundations of compositional semantics were described by Coecke et al. [12].

Mitchell and Lapata [39, 40] 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 are popular today. They compare various operations on word embeddings and how it affects their composition. In their results, multiplicative models are superior to the additive models, which is not the case in our analysis.

Distributional composition. Before the popularity of neural 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 [5] composed adjective-noun phrases using an adjective matrix and a noun vector. Grefenstette and Sadrzadeh [22] did something similar but they use a matrix for relational words, and a vector for argument words. Yessenalina and Cardie [59] 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.

Word embeddings through neural networks. Bengio et al. [6] first coined the term *word embedding*, in the context of training a neural language model. Collobert and Weston [13] 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. [36, 37] 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.

A popular method for creating paragraph representations is called *Paragraph2Vec* or *Doc2Vec* [31], 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.

Wieting et al. [57] showed that word embeddings, such as Word2Vec and GloVe, could be further enhanced by training them to compose sentence embeddings for the purpose of paraphrasing. Their method, which uses 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. Arora et al. [3] improved on Wieting et al. [57] 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.

Kiros et al. [30] 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. Kenter et al. [27] combined approaches from Kiros et al. [30] with the approach from Wieting et al. [57] to create an unsupervised method for learning sentence embeddings using a siamese neural network which tries to predict a sentence from context (CBOW). Kenter et al. also average word embeddings to create a semantic representation of sentences.

Recently Tissier et al. [54] introduced *dict2vec*, which expanded the skip-gram algorithm by incorporating word pair relatedness into the training procedure. These word pairs are extracted from lexicographic definitions, similar to the definitions we use. However, their approach is included directly in the training procedure of lexical word embeddings where we focus on composition.

Algebraic composition. Aside from work by Mitchell and Lapata, there are a lot of applications where algebraic composition is applied

[1]We provide an open source implementation as well as an open source dataset on https://thijs.ai/CompVec/.

as part of a model's architecture. Examples are: weighted averaging in attention mechanisms [4, 34], or in memory networks [56].

Paperno and Baroni [43] provided some mathematical explanations for why algebraic composition is performing well. Arora et al. [3] introduced a mathematical framework which attempts a rigorous theoretical understanding for the performance of averaging skip-gram vectors. Gittens et al. [20] built on this and proofed that the skip-gram algorithm actually ensures additive compositionality in terms of analogy tasks. There are caveats, they assume a specific definition of *composition* and a uniform word distribution. But words are distributed according to Zipf's law [60].

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 [28] composed embeddings using a single layer CNN to perform topic categorization and sentiment analysis. Blunsom et al. [7] presented a new pooling layer to apply CNNs to variable length input sentences. Liu et al. [33] later improved this model by reversing its architecture. Our results show that a convolutional model can be used for composing lexicographic definitions, even though we did not find it to be the most effective method.

Recurrent composition. Models utilizing a *recurrent neural network* (RNN) can read input sentences of varying length. They have been used for *neural machine translation* (NMT) [10] and *neural question answering* (NQA) [21, 56] as well as other model classes. Cho et al. [10] 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) [25]. Sutskever et al. [52] improved upon the encoder-decoder model by stacking recurrent layers and reversing the input. The GRU unit is empirically evaluated by Chung et al. [11] and they found that the GRU is comparable in performance with less computational cost. We use the GRU as an order dependent composition function.

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.

Now we turn to attention mechanisms [4, 34], which are important components of NMT and NQA systems. In such 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.

Recursive composition. Socher et al. [48–50] 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.

Evaluation of composed embeddings. Evaluating word representations in general is a difficult task. This usually happens in terms of the similarity between two words and is handcrafted for specific examples. The methods aggregated by Faruqui and Dyer [15][2] are a popular. Their evaluation combines 13 different word pair similarity sets [16, 19, 24], 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. We use these methods in this work to show that lexical semantic qualities of word embeddings do not decrease when we tune the embeddings.

Conneau et al. [14] aggregated different of sentence evaluation methods[3], including some on compositionality. Downstream performance in applications such as sentiment classification [46, 48] or question answering [32] provide an extrinsic evaluation of compositionality, but results may suffer from other confounding effects that affect the performance of the classifier. The STS tasks [1, 2], from SemEval 2014 and 2015, are evaluation tasks which can be used to determine sentence similarity and are also good candidates to test composition indirectly. Marelli et al. [35] 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 as well.

None of these works seem to evaluate compositional semantics of lexical definitions. Our evaluation method, called *CompVecEval*, fills this gap by directly trying to find the semantic similarity between a dictionary definition from WordNet [38] and the associated lemma.

3 DATASET

In order to test semantic composition, we turn to a dictionary for our data. The words or lemmas in dictionaries all have good definitions, which can be composed semantically into the meaning of that word, and thus ideal for our task. We choose to use WordNet [38] 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 sequence of words where $d = \langle w_1^d, w_2^d \dots w_n^d | w^d \in \mathcal{W} \rangle$. We make sure to remove stop words from this definition. 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, since word embeddings do not necessarily have to be applied on the target side, even though that is still possible.

In the original WordNet synonym graph, 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. We end up with $|\mathcal{X}| = 52,430$ unique data points with a vocabulary of $|\mathcal{D}| = 48,944$ unique words and a target vocabulary of $|\mathcal{L}| = 33,186$ unique lemmas.

[2] More commonly known as https://wordvectors.org.
[3] The set of evaluation methods is provided in the SentEval software library https://github.com/facebookresearch/SentEval.

Pretrained word embeddings. Now that we have a vocabulary of definition words $\mathcal{W}$ as well as a vocabulary of target words $\mathcal{L}$, we find vector representations for each definition and target words from the four popular large pretrained word embeddings: Word2Vec, GloVe, fastText and Paragram. Each of these pretrained word embeddings, were trained on English text and have a dimensionality of 300, as to keep our final comparison relatively fair.

Word2Vec was trained using the original skip-gram algorithm using a context window and walking over a large corpus of 100B tokens from Google News articles. GloVe vectors where trained using a local context window, similar to skip-gram, combined with global matrix factorization. With this global matrix vector, each context word is weighted to its global co-occurrence frequency. We used GloVe representations that where trained on the Common Crawl which has 840B tokens and a vocabulary of 2.2M. fastText uses a training algorithm that is based on skip-gram, however it represents each word as a bag of character n-grams. A vector representation is associated with each n-gram and words are represented as a sum of these n-grams. The fastText embeddings were trained on a large English Wikipedia dataset. The Paragram embeddings we use, are the same as the tuned embeddings from the compositional paraphrasic model of Wieting et al. [57] combined with the larger embedding dataset from Wieting et al. [58].

Keep in mind that we understand that these word embeddings are each trained using both a different algorithm and a different dataset. We are not trying to determine which algorithm is better. We are merely trying to give insights into the compositionality of existing pretrained publicly available word embeddings.

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 49,807 data points and a test dataset of 2,623 data points. We made the dataset and the code to create the entire dataset freely available[4].

4 METHODS OF COMPOSITION

The goal of word embedding composition is to combine multiple word embeddings in to a single composed embedding. Figure 1 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 our evaluation, we will test four simple algebraic composition functions, and six learnable composition functions.

Combining multiple intermediate representations into one simple representation is something which happens in lots of deep learning architectures. But in itself, it is not often studied in detail.

[4]The dataset and companion software can be found at: https://thijs.ai/CompVec/.

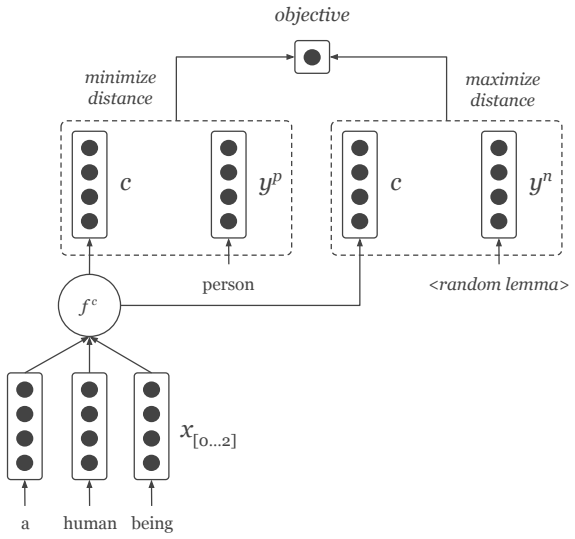

Figure 1: This figure illustrates the *CompVec* model structure. The input embeddings go into the composition function f^c and get composed into a single composed embedding c. We see how this composed vector c gets compared against the positive example y^p and the negative example y^n in the loss function.

In our case we look at the compositionality of word embeddings, but these approaches could very well be extended to other types of representations. Similar evaluations on compositional functions could take place.

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.

4.1 Algebraic composition

First, we will try to create a composition by applying element-wise operations: $+$, $\times$, $\max(d)$ and $\text{average}(d)$. It should be no surprise that composing by simple mathematical operation is not ideal, since the act of composing considers neither the relationship between individual words nor the order of the words. Instead such relationships should already be present in the space of all words under the operation. But 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. For example, the document representation model *Paragraph2Vec* uses an $\text{average}(d)$ operation to compose embeddings of any kind. Our evaluation shows that this is not necessarily the optimal operation.

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.

It should be noted that the magnitude, i.e. norm, of the pretrained embeddings is sometimes assumed to be 1 [45]. 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.

4.2 Tune embeddings while composing

Now that we have defined simple algebraic composition and a dataset for composition we can try tuning the pretrained embeddings for better composition under a specific composition function f^c. In order to do this, we train using stochastic gradient decent and the triplet loss function [23] as can be seen in equation 1. The loss function is related to the more popular contrastive loss function.

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

Aside from this loss function, we also experimented with a triplet loss function based on cosine similarity, similar to the work of [57]. Results from training using the loss function based on cosine-similarity were inferior on all evaluation methods.

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

Descending the gradients of the triplet loss will insure that the composed embedding c moves closer to the reference lemma embedding y^p but at the same time moves away from a random other reference lemma embedding y^n. Using the negative example ensures the embedding values do not converge to 0. The loss function also has a margin parameter α which basically makes sure that if the embeddings c and y^p are already close enough to each other they do not incur a loss. After some experimentation, we choose to use the margin value $\alpha = 5$ for the additive composition and $\alpha = 0.25$ for all other models.

While most word embedding algorithms do not include a form of regularization during the training procedure, we experimented with adding this to our training procedure. While experimenting we found that adding dropout [51] improved the final results considerably. Therefore, our final models are trained with a dropout probability of $P_{dropout} = 0.25$ on the input embeddings.

We trained the model for 205 epochs using batches of 512 data points with the Adam stochastic gradient decent optimizer [29]. We used a learning rate of $lr = 1e^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 1e^{-8}$.

4.3 Learning to compose

In addition to simple algebraic composition we also trained models to compose embeddings. We learned a simple projection layer, recurrent models as well as convolutional models. There are three big advantages of using a learned model over using simple operations. First, we learn the parameters for the compositional function to improve the representation so that we are not bound to the original embedding space and its compositional properties. Second, we can make non-linear transformations when combining representations. Lastly, a RNN and a CNN will be able to take word order into account and thus can do more with syntactical information.

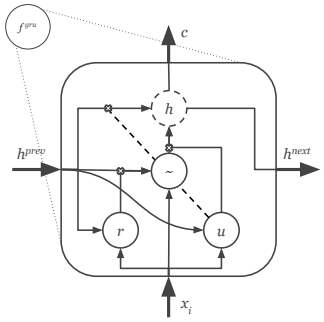

Figure 2: This figure shows the inner workings of a GRU. We see the update and reset gate with their input and output. We see $\tilde{h}$ and how it influences the final output c or h.

We execute the training procedure twice for all learned composition functions. Once with the embeddings fixed, and once where we fix the embeddings for the first part of training (2500 steps or 25 epochs) and then refine the embeddings along with the model's weight matrices. In the results table the models trained with fixed embeddings can be found under the columns titled *Original*, and the models with tuned embeddings under *CompVec*.

Projecting the Composed Vector. Our first and simplest learned composition function is a projection. This is similar to the approach Wieting et al. [57] took.

$$c = \tanh(xW_x + b) \quad (3)$$

We apply this projection to vectors which are composed using the same functions as in the previously discussed algebraic composition functions.

Recurrent Composition. Our first recurrent model uses a plain RNN units for its hidden layer. Our hidden units have a dimensionality of 300, similar to our word embeddings. The RNN is described in the Equation 4.

$$h = \tanh(xW_x + h_{t-1}W_h + b) \quad (4)$$

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 or better than the LSTM [11]. This model has three times the parameters of as our plain RNN model. Figure 2 shows the internals of a GRU schematically. Equation 5 gives us the mathematical definition of a GRU.

$$\begin{aligned}
r &= \sigma(xW_{xr} + h_{t-1}W_{hr} + b_r) && \text{reset gate} \\
u &= \sigma(xW_{xu} + h_{t-1}W_{hu} + b_u) && \text{update gate} \\
\tilde{h} &= \tanh(xW_{x\tilde{h}} + r \cdot h_{t-1}W_{h\tilde{h}} + b_{\tilde{h}}) && \text{candidate update} \\
h &= 1.0 - u \cdot h_{t-1} + u \cdot \tilde{h} && \text{final update}
\end{aligned} \quad (5)$$

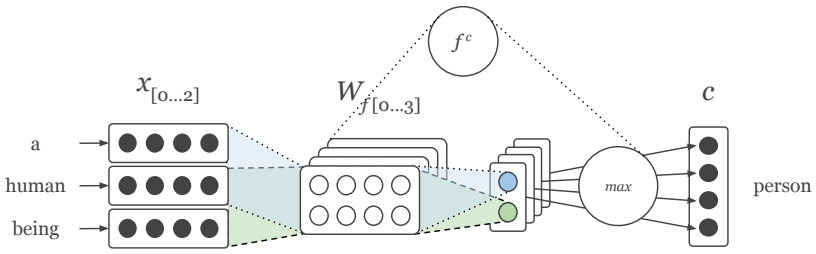

Figure 3: This figure shows the inner workings of a CNN for composing word embeddings. We see the weight matrix for the filters, the max pooling operation as well as c the final output.

Our last recurrent model is a *bi-directional GRU* [47] where we have the input sequence in regular order project to a hidden state of 150 units, and we have a separate model doing so for the input sequence in reverse order.

Convolutional Composition. For our first CNN we will use a filter were 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. We see this clearly in Figure 3.

$$\begin{aligned} \tilde{h}_{ij} &= \sum_{a=0}^{F} \sum_{b=0}^{E} W_{ab}^{f} x_{(i+a)(j+b)} && \text{convolution} \\ h &= \text{relu}(\tilde{h} + b) && \text{nonlinearity} \\ c &= \text{maxpool}(h) && \text{pooling} \end{aligned} \tag{6}$$

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.

5 ANALYSIS AND EVALUATION

In our analysis of the four popular word embeddings *Word2Vec* [37], *GloVe* [44], *fastText* [8] and *Paragram* [57] we test their applicability when using specific types of composition in a deep neural network architecture. We present both results for the original untuned embeddings as well as results for embeddings tuned with our dataset and the architecture described in the previous section. To analyze these word embeddings directly in terms of their compositionality we introduce *CompVecEval*. Additionally, we also analyze the embeddings in terms of their semantic textual similarity, relatedness and entailment using various existing evaluation methods. Lastly, we see if the quality of the word embeddings themselves have been influenced by the tuning optimization by looking at the scores of word similarity tests.

5.1 CompVecEval

We compare our composed representation to our target word representation using nearest neighbor ranking with the ball tree algorithm [42], in contrast to other methods which us magnitude invariant metrics such as cosine similarity. The vector magnitude is important because (intermediate) representations of semantics in neural architectures are affected by it. One does not often normalize each intermediate representation, because it could lead to a loss of information and influence the network's performance.

The balltree algorithm produces a complete ranking of all the 33,186 target words. In the subsequent ranking, we mark all target words from l_d as equally relevant, and all other words as not relevant. This will result in different results for + and average(d), as opposed to them being the same with cosine similarity based ranking approaches.

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.

When we have obtained the ranking and the relevant results, we apply several well-known ranking measures: Mean Reciprocal Rank (MRR) [9], Mean Average Precision (MAP) as well as Mean Precision@10 (MP@10). In addition, we evaluate using Mean Normalized Rank (MNR) which describes the fraction of the total dataset that does not need to be viewed when encountering a relevant result. For MNR and MRR we use the rank of the first relevant target word in l_d.

$$MNR := 1 - \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{\text{rank}_d}{|\mathcal{L}|} \qquad MRR := \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{1}{\text{rank}_d} \tag{7}$$

Here is rank_d the rank of the first relevant target word for the definition d. MRR is the more common method, it is top heavy, i.e. the contribution on lower ranked relevant decreases exponentially. MNR does weight each rank of a relevant result equally. Both of these measures are recall based.

$$MAP := \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{\sum_{r_d=1}^{|\mathcal{L}|} P(r_d) \times \text{rel}(r_d)}{|l_d|} \tag{8}$$

MAP as well as MP@10 captures the possibility of multiple relevant target words into the metric, where MAP is a recall-based metric and MP@10 is a precision-based metric. For all metrics we can state: "higher is better".

For now, we define CompVecEval as computing the MRR measure on the test set of 2,623 data points, since we use the rest for tuning the embeddings. If one's task does not require the training data one could use all the available data points (52,430) for an even more accurate evaluation method. And whenever we want more insight into the compositionality of word embeddings, we can also look at the other ranking methods considered: MRR, MAP and MP@10.

Table 1: This table displays the results of all evaluation measures. For all measures counts: "Higher is better". The results for CompVecEval and SICK-E scores are denoted ×100. All other sentence evaluation measures are denoted in Pearson's $r \times 100$, whereas the word similarity measures are denoted in Spearman's $r \times 100$. It is important to note that in the case of the learned composition functions (RNN, GRU, CNN etc.) under *original*, we *did* train the models themselves but *did not* change the original embeddings whereas under *CompVec* we tuned the embeddings as well.

			Word2Vec		GloVe		fastText		Paragram	
	Measure	Composition	Original	CompVec	Original	CompVec	Original	CompVec	Original	CompVec
CompVecEval	MRR	+	17.0	23.0 (+6.0)	11.9	26.5 (+14.6)	20.7	26.3 (+5.6)	26.6	*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)
		×	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)	6.6	15.6 (+9.0)	13.7	20.1 (+6.4)	14.6	18.6 (+4.0)	20.5	*23.3* (+2.8)
		+ 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)
SentEval	STS14 [2]	+	31.4	61.8 (+30.4)	54.1	65.7 (+11.5)	52.7	65.2 (+12.5)	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)
		×	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)	22.4	62.4 (+39.9)	60.6	*67.1* (+6.5)	42.6	65.1 (+22.5)	61.6	66.3 (+4.7)
		+ 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 [1]	+	36.8	68.7 (+31.9)	58.1	66.6 (+8.5)	58.1	68.8 (+10.7)	75.0	**75.2** (+0.2)
		average(d)	36.3	47.3 (+10.9)	58.1	51.3 (-6.8)	58.2	52.2 (-6.0)	*75.0*	54.6 (-20.3)
		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)
	SICK-E [35]	+	73.3	77.3 (+4.0)	75.8	76.7 (+0.9)	77.0	76.1 (-1.0)	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)
		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)
	SICK-R [35]	+	72.4	78.4 (+6.1)	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)
		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)
WordSim	SimLex [24]	+	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)
		×	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)	44.0	48.2 (+4.2)	40.2	43.0 (+2.7)	37.3	41.1 (+3.8)	*66.2*	66.0 (-0.2)
		+ 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 [19]	+	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)
		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)
	WS-353 [16]	+	70.6	70.1 (-0.5)	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)
		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)

5.2 Sentence Representation Evaluation

To see which composition functions, have an impact on specific aspects of sentence representations we evaluated our composed embeddings using two popular methods: , STS14 [2], STS15 [1] and SICK [35]. Conneau et al. [14] describes how to match different sentence vectors from the SICK dataset. There are 3 matching methods used to extract the relations between 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. The different datasets are evaluated using 10-fold cross validation. We evaluate our compositions using the SICK dataset using both entailment (Entailment) and semantic relatedness (Relatedness). For relatedness we learn to predict the probability distribution of the relatedness scores [53]. Results for relatedness are reported as Pearson's correlation coefficients since other related works also report this metric. Results for entailment are accuracy scores on the classifier with 3 labels.

With STS14 [2] and STS15 we evaluate the embeddings on 11 unsupervised SemEval tasks. The dataset is a combination of post-editing, news, question and plagiarism sentences labeled with a similarity score between 0 and 5. Here we also report the Pearson's correlation coefficient similar to SICK Relatedness.

For all these evaluations, we used the available 48,944 words from our dataset vocabulary to select embeddings for the words we should compose. If a word was not available, we grabbed back to the vocabulary of the original word embeddings (which is much larger in all cases). In this way, our results for the *original* word embeddings are similar to those in other publications.

It is also important to note that we do not directly optimize for the task and we are not trying to improve the state of the art of these evaluation metrics. We are simply showing the impact of different compositional functions on the pretrained embeddings, as well as the impact that tuning for compositionality does not decrease or even slightly increase the sentence similarity, entailment or relatedness performance. Including a task specific learning objective would probably boost results further.

5.3 Word Similarity Evaluation

We performed word similarity tests to see if the quality of the single word embeddings do not suffer under the training for composition. In the results table, we will see that the values before training are exactly the same. This is logical since we are not evaluating the composition itself here. We are only seeing how "training for compositionality" influences the final quality.

WS-353 by Myers et al. [41] is a widely used WordNet based word similarity measure. Naturally, we should improve on this measure since it is based on similar data. This should be noted and might make this result biased. This word similarity set is not very elaborate either with 353 word pairs. SimLex by Hill et al. [24] is a larger word similarity dataset with 999 word pairs. It was specifically created to capture similarity independent of relatedness and association. SimVerb by Gerz et al. [19] focuses on the similarity between 3500 verb pairs from the USF free-association database. For word similarity, we report Spearman's correlation coefficient since other related works also report this metric.

Again, it is important to note that we do not directly optimize for the task and we are not trying to improve the state of the art of these evaluation metrics. We are simply showing that tuning for compositionality does not decrease or even slightly increase the word similarity performance. Including a word similarity learning objective would probably boost results further.

5.4 Results

Table 1 shows the results for all evaluation measures when using all ten different composition functions. We left out under performing composition functions (×, Proj., RNN, CNN etc.) from most measures.

When we compare the four different embeddings used for transfer learning we see clearly that the Paragram embeddings are the best starting point. These embeddings were already compared to the GloVe embeddings with an averaging function [57]. Now we can see how it compares against fastText and Word2Vec with different composition functions as well. The Paragram embeddings benefit from tuning using the lexicographical definitions, albeit only slightly on evaluations other than CompVecEval.

GloVe and Word2Vec seem to benefit the most from tuning. On the additive composition (+) function Word2Vec jumps 30.4 points on the STS14 benchmark, 6.0 points on CompVecEval and 7.4 points on SimLex. The same goes for GloVe with 11.5 points on STS14, 14.6 points on CompVecEval and 9.8 points on SimLex.

We can see that fastText has a lot of benefit from additive composition (+) on CompVecEval without tuning. This is can be explained by looking at the fastText algorithm. It creates embeddings of words by additive composition of n-grams itself, so fastText is intrinsically optimized for additive composition.

Interestingly enough additive composition (+) performs similar or slightly worse when compared to averaging on measures other than CompVecEval. But when we tune on additive composition the tuned model outperforms averaging on our CompVecEval task as well as on STS, SimLex and SimVerb. When tuned on averaging model performance drops significantly for STS tasks, which could be attributed to our choice of loss function. Averaging does yield a bad result on CompVecEval for untuned as well as tuned embeddings.

Multiplicative composition (×) performed, by far, the worst of all composition functions. This directly contradicts results from Mitchell and Lapata [39], however their vectors were not real-valued embeddings as uses in neural networks.

We see that the $\max(d)$ operation or max composition can be a good approach to compose elements. Which is somewhat counter intuitive, because the composition only takes a very small portion of the available information into account. For CompVecEval as well as the other evaluation measures this composition function scores relatively high. In neural networks today, we rarely see an operation such as element-wise maximum used within an architecture outside of CNN's. This could be an interesting direction to explore.

On the projection function, we have results that are not in line with Wieting et al. [57]. 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. Perhaps this could change when we increase the size of the training data. The projection function was outperformed by other learned models

(e.g. GRU) and the additive algebraic composition function on all metrics.

For the learned composition functions, we see that the recurrent models outperform the convolutional models. While the more elaborate convolutional model with larger window sizes (CNN-3,5,7,9) did outperform the simpler CNN as well as the RNN, 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.

CompVecEval. The results for CompVecEval show that the untuned Paragram embeddings are the best untuned embeddings at composing w.r.t. all composition functions. The compositing performance increases if we train for the task of composing lexicographical definitions, as one would expect.

If we look at the best MRR of 30.3 for the bi-directional GRU and the second best MRR of 29.9 for the additive model (+) their difference is minimal. The learned GRU model has noticeable downsides, since the inference operation on a GRU model is a lot more expensive and does require a model to be loaded into memory, while simple addition only requires the word embeddings themselves.

When we look at the best MRR score for Paragram with a tuned algebraic model (+): 29.9, it had a MNR of 92.8 which comes down to seeing an average 3,524 of the total of the total 48,944 target words before encountering a relevant one, which is way better than the random baseline but still not great. To highlight this, we added a randomly generated ranking for all data points. In this random baseline, we see a MNR of 55.2 and not of 50 because there are various data points with more than one relevant target words. All this is good news since it means there is a lot of room for improvement in making embeddings more composable.

The low scores on CompVecEval for averaging could be due to the fact that our balltree absolute distance metric is strongly affected by the embeddings magnitude into account. Whereas sentence embeddings metrics might not be, because they are usually composing two sentences of similar length into a comparable sentence embedding, whereas we have a relatively long source (lexicographic definition) and a very short target (lemma).

Sentence Evaluation. We see that tuning for composition has a very big impact in the STS metrics for Word2Vec, GloVe and fastText. Where the improvements for the Paragram embeddings are also there but less significant. 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.

For the SICK Relatedness and Entailment tests we see comparable results for most composition functions and are not able to improve on them. That could be due to the nature of the SICK data. While our focus is on composing lexical semantics, the focus on the SICK data is on relatedness and entailment. This is not really represented in lexicographical data.

Word Similarity. If we look at the results for the word similarity metrics we see that overall, they increase quite significantly when tuned for composing lexicographic definitions. This even happens when tuning the embeddings to work better with learned composition functions. One would expect that these embeddings would decrease in general performance and increase in the task they are trained for. But the word similarity results suggest that the opposite is the case. Embeddings from the tuned GRU models and our tuned embeddings for additive composition (+) yield the best SimLex and SimVerb performance. It has to be mentioned that the Paragram embeddings [58] were trained on the SimLex [24] dataset. So naturally they will score high on SimLex word similarity.

6 CONCLUSION

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. Our tuned CompVec Paragram embeddings are the best 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 almost every evaluation measure we found that tuning the embeddings on our lexicon-based dataset performed better or comparable. When selecting word embeddings for transfer learning for training a neural network our results can give insights into which embeddings you should choose. We published and open sourced both the dataset as well as the tuned embeddings.

For future research, one could look into multi-layered composition functions. One neural model we would like to explore is a convolutional model containing so called: 'dilated convolutions' [55]. These could provide a solution to the problems with long distance relationships.

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. Our intuition is that using this larger dataset could increase the performance.

Published theoretical works on the understanding of word embeddings is currently limited [3, 20]. This work only provides theory for the original skip-gram algorithm. Further work could look at the theoretical and mathematical properties of the embedding spaces created using other algorithms.

REFERENCES

[1] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, et al. 2015. Semeval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. ACL, 252–263.

[2] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. Semeval-2014 task 10: Multilingual Semantic Textual Similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. ACL, 81–91.

[3] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2016. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. In *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*.

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

[5] Marco Baroni and Roberto Zamparelli. 2010. Nouns are Vectors, Adjectives are Matrices: Representing Adjective-Noun Constructions in Semantic Space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*. ACL, 1183–1193.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research (JMLR)* 3 (2003), 1137–1155.

[7] Phil Blunsom, Edward Grefenstette, and Nal Kalchbrenner. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*. ACM.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics (TACL)* 5 (2017), 135–146.

[9] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected Reciprocal Rank for Graded Relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009)*. ACM, 621–630.

[10] Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. ACL, 1724–1734.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. (2014). Presented in the Deep Learning and Representation Learning Workshop of NIPS 2014.

[12] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. 2010. Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis* 36 (2010), 345–384.

[13] Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*. ACM, 160–167.

[14] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2011. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*. ACL, 670–680.

[15] Manaal Faruqui and Chris Dyer. 2014. Community Evaluation and Exchange of Word Vectors at wordvectors.org. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*. ACL, 19–24.

[16] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing Search in Context: The Concept Revisited. In *Proceedings of the 10th International Conference on World Wide Web (WWW 2001)*. ACM, 406–414.

[17] Gottlob Frege. 1892. On Concept and Object. *The Frege Reader* (1892), 181–193.

[18] Luca Gasparri and Diego Marconi. 2016. Word Meaning. In *The Stanford Encyclopedia of Philosophy* (spring 2016 ed.). Metaphysics Research Lab, Stanford University.

[19] Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*. ACL.

[20] Alex Gittens, Dimitris Achlioptas, and Michael W Mahoney. 2017. Skip-gram - zipf + uniform = vector additivity. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*. ACL, 69–76.

[21] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *arXiv:1410.5401* (2014).

[22] Edward Grefenstette and Mehrnoosh Sadrzadeh. 2011. Experimental Support for a Categorical Compositional Distributional Model of Meaning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*. ACL, 1394–1404.

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

[24] Felix Hill, Roi Reichart, and Anna Korhonen. 2015. Simlex-999: Evaluating Semantic Models with Genuine Similarity Estimation. *Computational Linguistics* 41, 4 (2015), 665–695.

[25] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[26] Theo Janssen. 2001. Frege, Contextuality and Compositionality. *Journal of Logic, Language and Information* 10, 1 (2001), 115–136.

[27] Tom Kenter, Alexey Borisov, and Maarten de Rijke. 2016. Siamese CBOW: Optimizing Word Embeddings for Sentence Representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*. ACL.

[28] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. ACL.

[29] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. (2015).

[30] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-Thought Vectors. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS 2015)*. NIPS, 3294–3302.

[31] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*. IMLS, 1188–1196.

[32] Xin Li and Dan Roth. 2002. Learning Question Classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002*. ACL, 1–7.

[33] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. 2014. A Recursive Recurrent Neural Network for Statistical Machine Translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*. ACL, 1491–1500.

[34] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*. ACL, 1412–1421.

[35] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK Cure for the Evaluation of Compositional Distributional Semantic Models. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC 2014)*. ELRA, 216–223.

[36] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781* (2013).

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS 2013)*. NIPS, 3111–3119.

[38] George Miller and Christiane Fellbaum. 1998. Wordnet: An Electronic Lexical Database. (1998).

[39] Jeff Mitchell and Mirella Lapata. 2008. Vector-based Models of Semantic Composition. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL 2008, HLT)*. ACL, 236–244.

[40] Jeff Mitchell and Mirella Lapata. 2009. Language Models based on Semantic Composition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*. ACL, 430–439.

[41] Jerome L Myers, Arnold Well, and Robert Frederick Lorch. 2010. *Research Design and Statistical Analysis*. Routledge.

[42] Stephen M Omohundro. 1989. *Five Balltree Construction Algorithms*. International Computer Science Institute Berkeley.

[43] Denis Paperno and Marco Baroni. 2016. When the Whole is Less than the Sum of its Parts: How Composition Affects PMI Values in Distributional Semantic Vectors. *Computational Linguistics* 42, 2 (2016), 345–350.

[44] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. ACL, 1532–1543.

[45] Sebastian Ruder. [n. d.]. On word embeddings, Part 1. ([n. d.]). http://ruder.io/word-embeddings-1/

[46] Tobias Schnabel, Igor Labutov, David M. Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*. ACL, 298–307.

[47] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[48] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*. ACL, 455–465.

[49] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic Compositionality through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language*

Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012). ACL, 1201–1211.

[50] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*. ACL, 1631–1642.

[51] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)* 15, 1 (2014), 1929–1958.

[52] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS 2014)*. NIPS, 3104–3112.

[53] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*. ACL, 1556–1566.

[54] Julien Tissier, Christopher Gravier, and Amaury Habrard. 2017. Dict2vec : Learning Word Embeddings using Lexical Dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*. ACL, 254–263.

[55] AÃđron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. In *Proceedings of the 9th ISCA Speech Synthesis Workshop (SSW9)*. ISCA, 125–125.

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

[57] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Towards Universal Paraphrastic Sentence Embeddings. (2016).

[58] John Wieting, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Dan Roth. 2015. From Paraphrase Database to Compositional Paraphrase Model and Back. *Transactions of the Association for Computational Linguistics (TACL)* (2015).

[59] Ainur Yessenalina and Claire Cardie. [n. d.]. Compositional Matrix-space Models for Sentiment Analysis. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*. ACL, 172–182.

[60] George Kingsley Zipf. 1932. Selected Studies of the Principle of Relative Frequency in Language. (1932).