DeepWeight - An Investigation Towards Improving Functional Semantic Composition

Bachelor Thesis Artificial Intelligence, 7.5 ECTS.

L.P.J. van Soest

Student Number 5999995

Abstract

Natural Language Processing inquires about the meaning of lexical items and text segments. In vector semantics, the meaning of words and word combinations is represented by vectors. In natural language, words can be combined. When compared to their components, combined words can represent a different meaning. In an attempt to estimate vectors which represent the meaning of word combinations, using composition models, a set of composition functions is applied to the vectors of the components. Introduced by Dima et al., 2019c, compared to its predecessors, the composition model TransWeight achieves the highest accuracy rates of correctly estimating word composition vectors. For this thesis, it is investigated if further increasing the complexity of the TransWeight model improves on its accuracy. Two experiments are conducted with this investigation. First, a new layer is added to a copy of TransWeight, producing the new investigative model DeepWeight. Second, the length of TransWeights transformations axis is increased to relatively higher values. Subsequently, the evaluation results of both experiments are compared to an evaluation of TransWeight. For both experiments, evaluated estimation ranks were near equal to those of the TransWeight model, possibly suggesting that TransWeight is already complex enough. However, potentially, by using different approaches towards increasing model complexity, future research may unravel significant ways to improve accuracy by further increasing the model complexity.

A thesis presented for the degree of

Bachelor of Science

Supervisor: Dr. D. Paperno

Second Reader: Dr. N.A. Alechina

Utrecht University

May 17 2020

Utrecht University

Contents

1 Introduction

1.1 Natural language processing

Quite often, Natural Language Processing (NLP) requires semantic information about lexical items and text segments. For example, a spam mail detection system might be better at detecting all sorts of spam mails by knowing their contents' semantics. (Schütze, 1993) Furthermore, by understanding the meaning of an instruction given by a human, a digital assistant may be enabled to infer complex constraints, as it can derive the full semantics of a sentence (Albreshne and Pasquier, 2010).

1.2 Semantic Composition

In natural language, words can be combined. Baroni et al., 2014 described this principle by Frege as "(...) the meaning of a complex expression is a function of the meaning of its constituent parts and the mode of their combination." In other words, compared to their components, combined words represent a different meaning. For example, the words *green* and *car* can be combined into *green car*. Semantically, *green car* is different from *green* and *car*, as a green car is more than just green; it is also a car. Given the difference in meaning, a word vector (1.3) representing *green car* is likely to have different characteristics than vectors for *green* or *car*.

1.3 Word Vectors

In vector semantics, in an attempt to capture the meaning of words and word compositions in a given corpus, the meanings of words and word composition are represented by a collection of real numbers. Established by analysing the corpus, each studied word or word composition is assigned a collection of the same size, representing their characteristics relative to the data (Bollegala et al., 2016). Numeric collections for word representation are commonly known as word vectors.

Accurate vectors can only be derived from words or word compositions that occur frequently in a corpus. (Bullinaria and Levy, 2007). Moreover, with the vast amount of possible word compositions, not every combination may frequently occur in a corpus. Therefore, as some do not occur frequently enough, not every possible composition can be studied. Instead, using semantic composition functions based on common mathematical notions (1.4), the vector of the word combination can be estimated (Baroni et al., 2014).

An advantage of using a well-known data representation such as vectors, is that composition functions (1.2) can be defined using standard mathematical notions. Some notions are discussed in (1.4).

1.4 Composition Models

A series of composition functions are defined using a composition model. As listed by Dima et al., 2019c (2 - Previous work in composition models), there exists a plethora of composition models. To provide background on composition models and their mathematical notions, these models are paraphrased below.

input - output Given two 1-dimensional word input tensors $\boldsymbol{u}^{(n)}, \boldsymbol{v}^{(n)}$ of size $n \in \mathbb{N}$, the models derive an output vector $\boldsymbol{p}^{(n)}$.

1.4.1 Composition by Addition

Proposed by Mitchell and Lapata, 2010, the output $\boldsymbol{p}$ is derived by adding tensors $\boldsymbol{u}$ and $\boldsymbol{v}$. Regular Addition, SAddition and VAddition are variations on this notion by using different weighting complexities.

Vector Addition Each element of output tensor $\boldsymbol{p}$ is the result of adding two elements of the two one-dimensional input tensors $\boldsymbol{u}$ and $\boldsymbol{v}$. In other words, $\boldsymbol{p}$ is derived such that

$$\boldsymbol{p} = \boldsymbol{u} + \boldsymbol{v}$$

$$\Leftrightarrow$$

$$\forall_{k \in \mathbb{N}}(0 \leq k < n \rightarrow$$
$$\boldsymbol{p}_k = \boldsymbol{u}_k + \boldsymbol{v}_k)$$

However, this regular form of addition does not take into account word order. Moreover, a different order of the same pair of words may lead to different meanings, e.g. *car factory* versus *factory car*. To compensate, SAddition weighs $\boldsymbol{u}, \boldsymbol{v}$ differently.

Scalar Weighting With SAddition, to apply different weighting, $\boldsymbol{u}$ and $\boldsymbol{v}$ are respectively multiplied by two different scalars α, β. Moreover, before adding $\boldsymbol{u}, \boldsymbol{v}$, SAddition multiplies each element of tensor $\boldsymbol{u}$ with α, and each element of $\boldsymbol{v}$ with β. In other words, $\boldsymbol{p}$ is derived such that

$$\boldsymbol{p} = \alpha\boldsymbol{u} + \beta\boldsymbol{v}$$

$$\Leftrightarrow$$

$$\forall_{k \in \mathbb{N}}(0 \leq k < n \rightarrow$$
$$\boldsymbol{p}_k = \alpha\boldsymbol{u}_k + \beta\boldsymbol{v}_k)$$

Vector Weighting Word vectors may depict complex corpus-wide characteristics. Therefore, with SAddition, multiplying a constant with each element in an input tensor may not be complex enough. To increase complexity, VAddition uses component-wise multiplication. Moreover, two different 1-dimensional tensors $\boldsymbol{a}^{(n)}, \boldsymbol{b}^{(n)}$ are multiplied respectively by $\boldsymbol{a}, \boldsymbol{b}$. In other words, $\boldsymbol{p}$ is derived such that

$$\boldsymbol{p} = \boldsymbol{a}\boldsymbol{u} + \boldsymbol{b}\boldsymbol{u}$$
$$\Leftrightarrow$$
$$\forall_{k \in \mathbb{N}}(0 \leq k < n \rightarrow$$
$$\boldsymbol{p}_k = \boldsymbol{a}_k\boldsymbol{u}_k + \boldsymbol{b}_k\boldsymbol{v}_k)$$

1.4.2 Matrix

The addition models weigh each input word tensor $\boldsymbol{u}, \boldsymbol{v}$, disregarding any possible relationships between characteristics across both input tensors. To compensate, as proposed by Socher et al., 2010, the matrix composition model applies a set of transformation weights across both inputs. To do so, $\boldsymbol{u}, \boldsymbol{v}$ are merged into a single one-dimensional input vector. Then, the input vector is transformed by multiplying the vector with a transformation matrix and by adding a bias vector afterwards. Finally, to derive $\boldsymbol{p}$, a non-linearity function g is used.

Non-linearity According to Glorot et al., 2011, introducing non-linearity by using a rectifier function on the nodes in a neural network improves the performance of the network. The composition models Matrix, FullLex (1.4.3) and TransWeight (1.5) make use of such functions to derive $\boldsymbol{p}$. For example, as introduced by Hahnloser et al., 2000, the paper of Dima et al., 2019c mentions the function $g = ReLu = \lambda x.max(0, x)$. However, when tested, Dima et al., 2019c mention to have found no significant differences when compared to using $g = identity = \lambda x.x$. This is in contrast with Glorot et al., 2011, as using $g = identity$ leaves the nodes untouched, which is equivalent to using no rectifier function at all. Given the contradiction, when explaining the mathematical notions of the models, the non-linearity is defined as g and is specified as such for further interpretation.

Concatenation Input tensors are combined into a single one-dimensional input tensors by concatenating both tensors along the same axes. In other words, all elements of $\boldsymbol{v}$ are appended to $\boldsymbol{u}$, increasing its size from n to $2n$. In other words, for $0 \leq i < 2n$,

$$[\boldsymbol{u}; \boldsymbol{v}]_i = \begin{cases} \boldsymbol{u}_i & i < n \\ \boldsymbol{v}_{i-n} & otherwise \end{cases}$$

Transformation Weighting Taking into account relationships between characteristics across both input tensors $\boldsymbol{u}, \boldsymbol{v}$, to match output tensor $\boldsymbol{p}$'s size n, $[\boldsymbol{u}; \boldsymbol{v}]$ is contracted with a two-dimensional weights vector $\boldsymbol{W}^{(n \times 2n)}$, and is subsequently added by bias tensor $\boldsymbol{B}^{(n)}$.

Contraction By contracting weights vector $\boldsymbol{W}^{(n \times 2n)}$ with $[\boldsymbol{u}; \boldsymbol{v}]$, values on the second axis of $\boldsymbol{W}$ are multiplied with the values on the first and only axis of $[\boldsymbol{u}; \boldsymbol{v}]$. Essentially, to derive the output $\boldsymbol{p}$ with one axis, the contraction reduces the number of axes of $\boldsymbol{W}^{(n \times 2n)}$ from two to one. In other words, with g being a non-linearity function (1.4.2), $\boldsymbol{p}$ is derived such that

$$\boldsymbol{p} = g(\boldsymbol{W}[\boldsymbol{u}; \boldsymbol{v}] + \boldsymbol{B})$$
$$\Leftrightarrow$$
$$\forall_{k \in \mathbb{N}}(0 \leq k < n \rightarrow$$
$$\boldsymbol{p}_k = g(\sum_{i=0}^{2n-1} \boldsymbol{W}_{k,i}[\boldsymbol{u}; \boldsymbol{v}]_i + \boldsymbol{B}_k))$$

1.4.3 FullLex

Proposed by Socher et al., 2012, FullLex builds on the notions of the Matrix model by capturing word-specific interactions. Once found the interactions of one tensor, it is multiplied with the other input tensor. FullLex further increases the complexity of the matrix model.

With FullLex, word-specific interactions are caught using a trainable tensor $\boldsymbol{A}^{(|V| \times n \times n)}$, where $|V|$ is the size of the corpus' vocabulary size; ensuring entries for each word in the corpus. Then, input tensors $\boldsymbol{u}, \boldsymbol{v}$ are transformed by cross-contracting with each other's trained tensor $\boldsymbol{A}^{\boldsymbol{v}}$ and $\boldsymbol{A}^{\boldsymbol{u}}$ and are concatenated to function as input for the matrix model. Subsequently, the input is transformed by using it to contract transformation matrix $\boldsymbol{W}$, which then is added by a bias vector $\boldsymbol{B}$. Like Matrix, $\boldsymbol{b}$ is derived using a non-linearity function. In other words, with g being a non-linearity function, $\boldsymbol{p}$ is derived such that

$$\boldsymbol{p} = g(\boldsymbol{W}[\boldsymbol{A}^{\boldsymbol{v}}\boldsymbol{u}; \boldsymbol{A}^{\boldsymbol{u}}\boldsymbol{v}] + \boldsymbol{B})$$
$$\Leftrightarrow$$
$$\forall_{k \in \mathbb{N}}(0 \leq k < n \rightarrow$$
$$\boldsymbol{p}_k = g(\sum_{i=0}^{2n-1} \boldsymbol{W}_{k,i}(\sum_{j=0}^{n-1}[\boldsymbol{A}^{\boldsymbol{v}}_{i,j}\boldsymbol{u}_j; \boldsymbol{A}^{\boldsymbol{u}}_{i,j}\boldsymbol{v}_j])_i + \boldsymbol{B}_k))$$

1.5 TransWeight

The article of Dima et al., 2019c introduces a model called TransWeight. It states that FullLex is essentially treating each word as in island by disregarding lexical meanings. In an attempt to eliminate this deficit, TransWeight introduces transformation weighting, which will be further discussed in 1.5.1. Opposed to the other models discussed in 1.4, instead of using one layer, TransWeight uses two layers of tensor operations to derive $\boldsymbol{p}$. Following the article, TransWeight turned out to be most accurate.

1.5.1 Transformation weighting

Compared to FullLex (1.4.3), like Matrix (1.4.2), TransWeight refrains from weighting $\boldsymbol{u}, \boldsymbol{v}$ before concatenation. Given the concatenated input, TransWeight uses two layers of operations to derive $\boldsymbol{p}$, which will be discussed below in 1.5.2 and 1.5.3.

1.5.2 Applying Transformations

The concatenation of the input tensors $[\boldsymbol{u}; \boldsymbol{v}]^{(2n)}$ is expanded using a transformation tensor $\boldsymbol{T}^{(t \times 2n \times n)}$. Then, a transformation bias $\boldsymbol{B^t}$ $^{(t \times n)}$ is added to the result. Subsequently, a non-linearity function, such as $g = ReLu = \lambda x.max(0, x)$ (Hahnloser et al., 2000) is applied to the resulting $(t \times n)$-tensor.

Expansion $[\boldsymbol{u}; \boldsymbol{v}]$ is expanded using transformation tensor $\boldsymbol{T}$. Moreover, the values along the axis of $[\boldsymbol{u}; \boldsymbol{v}]$ are multiplied by the values along the second axis T. By doing so, in the next layer, the number of axis of the input is equal the number of axes of T minus the number of axes of $[\boldsymbol{u}; \boldsymbol{v}]$. As a result, the number of axis is equal to two, which is an increment of the number of axes of the previous input $[\boldsymbol{u}; \boldsymbol{v}]$, which was one. In other words, the transformation layer $\boldsymbol{H}$ is defined such that

$$\boldsymbol{H} = g(\boldsymbol{T}[\boldsymbol{u}; \boldsymbol{v}] + \boldsymbol{B})$$

$$\Leftrightarrow$$

$$\forall_{s \in \mathbb{N}} \forall_{j \in \mathbb{N}} (0 \leq s < t \land 0 \leq j < n \rightarrow$$

$$\boldsymbol{H}_{s,j} = g(\sum_{i=0}^{2n-1} \boldsymbol{T}_{s,i,j} [\boldsymbol{u}; \boldsymbol{v}]_i + \boldsymbol{B}^{\boldsymbol{t}}_{s,j})$$

1.5.3 Weighting the Transformations

Using tensor $\boldsymbol{H}^{(t \times n)}$, TransWeight derives $\boldsymbol{p}$ by double contracting $\boldsymbol{H}^{(t \times n)}$ with a three-dimensional weight tensor $\boldsymbol{W}^{(t \times n \times n)}$, added by bias tensor $\boldsymbol{B^w}$ $^{(n)}$. In other words, the model outputs a tensor $\boldsymbol{p}^{(n)}$, such that

$$\boldsymbol{p} = \boldsymbol{W} : \boldsymbol{H} + \boldsymbol{B}$$

$$\Leftrightarrow$$

$$(\forall_{k \in \mathbb{N}} (0 \leq k < n \rightarrow$$

$$\boldsymbol{p_k} = \sum_{s=0}^{t-1} \sum_{j=0}^{n-1} \boldsymbol{H}_{s,j} \boldsymbol{W}_{s,j,k} + \boldsymbol{B}^{\boldsymbol{w}}_k$$

1.5.4 Evaluation Method

To evaluate the model of TransWeight to the other models. The article of Dima et al., 2019c introduces a modified interpretation of the evaluation model of Baroni and Zamparelli, 2010, of which its implementation will be discussed in 2.2. Moreover, according to Baroni and Zamparelli, 2010, based on the assumption that word vectors look like plausible representations of semantic composition, it is to be expected that the closer the estimated vectors are to their target in a corpus, the better they perform in any task that requires access to the composite meaning, e.g. spam detection (1.1). Therefore, as proposed by Baroni and Zamparelli, 2010, to measure the accuracy of such predictions, the distance of the estimation to its target $\tilde{\boldsymbol{p}}$ is compared to the distance of all other estimations of other targets to $\tilde{\boldsymbol{p}}$. Ranked by distance, an estimation is deemed accurate if it is at most the fifth closest vector to the target. This method is elaborated in 2.2.

1.5.5 Successful Model

Using transformations $t = 100$, compared to existing composition models, TransWeight was found to have the highest accuracy on nearly all corpora used for the experiment (Dima et al., 2019c).

1.6 Investigation: Improving TransWeight

As discussed in 1.5, TransWeight consists of two layers. In summary, it uses an interpretation of the Matrix model as its first layer (1.5.2), and adds a second layer of operations to its model by doubly contracting the output of the first layer with a three-dimensional weights vector (1.5.3). From a high-level perspective, TransWeight differs from its predecessors by using two layers of operations instead of one, which essentially increases model complexity. For this thesis, it is investigated if further increasing the complexity of the TransWeight model improves its accuracy towards composition word vector estimation. Moreover, the difference in accuracy is investigated by increasing complexity in two different ways. Namely, in the first experiment, a new layer is added to the TransWeight model (1.6.1). And, in the second experiment, the number of transformations t is increased to more extreme values (1.6.2).

1.6.1 Adding a third layer

To add a new layer to TransWeight, the layer needs to be able to fit between a pair of existing connected layers. The new layer will be added at the end of the network. Specifically, the layer will contract a new weighting tensor $\boldsymbol{V}^{(n \times n \times n)}$ to the output of the second layer. To ensure providing the model with enough complexity, $\boldsymbol{V}$ consists of three dimensions of size n. Subsequently, the output layer $\boldsymbol{p}^{(n)}$ is derived by adding the result of the operation by a new two-dimensional bias tensor $\boldsymbol{B'}^{\,(n \times n)}$.

Modifying the second layer To derive output tensor $\boldsymbol{p}$, any tensor to contract with $\boldsymbol{V}^{(n \times n \times n)}$ must have at least two dimensions of length n to contract with. However, in TransWeight, the second layer is contracted to $\boldsymbol{p}^{(n)}$, which, for further operations, comes short on dimensions to contract with. Therefore, instead of $\boldsymbol{p}^{(n)}$, by removing one contraction, the second layer of TransWeight is modified to have output tensor $\boldsymbol{H'}^{\,(n \times n)}$, such that

$$\boldsymbol{H'} = \boldsymbol{H}\boldsymbol{W}$$

$$\Leftrightarrow$$

$$\forall_{j \in \mathbb{N}} \forall_{k \in \mathbb{N}} (0 \leq j < n \wedge 0 \leq k < n \rightarrow$$

$$\boldsymbol{H'}_{j,k} = \sum_{s=0}^{t-1} \boldsymbol{H}_{s,j} \boldsymbol{W}_{s,j,k} \tag{1}$$

Applying the third layer The new layer will derive one-dimensional output tensor $\boldsymbol{p}^{(n)}$ based on the result of the second layer. In detail, $\boldsymbol{p}$ is derived by contracting the new three-dimensional weights tensor $\boldsymbol{V}^{(n \times n \times n)}$ with $\boldsymbol{H'}^{\,(n \times n)}$ from equation 1.6.1, added by the new two-dimensional bias tensor $\boldsymbol{B'}^{\,(n \times n)}$, such that

$$\boldsymbol{p} = \boldsymbol{H'} : \boldsymbol{V}$$

$$\Leftrightarrow$$

$$\forall_{k \in \mathbb{N}} (0 \leq k < n \rightarrow$$

$$\boldsymbol{p}_k = \sum_{j=0}^{n-1} \sum_{l=0}^{n-1} \boldsymbol{H'}_{j,k} \boldsymbol{V}_{j,k,l} \tag{2}$$

1.6.2 Increasing Transformations

By increasing the complexity of the network, Dima et al., 2019c mention to have found $t = 100$ to be empirically most successful towards achieving high accuracy compared to values for t between 20 and 500. Increasing t would increase the number of calculations in the model. Therefore, next to adding a third layer, by increasing TransWeight's complexity by increasing t, TransWeight's results may improve. In an attempt to gain more information on the effects of increasing the number of transformations t in the second experiment, different values for t: $100, 200, 500, 1000, 10000, 50000$ are tested and compared.

1.6.3 Evaluating results

Conforming to the method of evaluation used by Dima et al., 2019c, results for experiments described by 1.6.1 and 1.6.2 are based on the evaluation method further described in 2.2.

2 Method

2.1 Data

For training and testing, the same data is used as referred to by Dima et al., 2019c. The data contain a total of eight corpora.

Data characteristics For both German and English, the data contain unique corpora for pairs of adjectives with nouns, adverbs with adjectives and nouns with nouns. For Dutch, the data contain two unique corpora for pairs of adverbs with adjectives and nouns with nouns. The lengths of the corpora and their origins are illustrated using table 1 below. For reference, the data sets are available from the data resource hosted by the Tübingen Archive of Language Resources (Dima et al., 2019b).

Using the data When a model is trained, it is trained once for every corpus using the training subset of the corpus. During the development of a model, the implementation can be tested by training a model and then verifying it by evaluation using the development subset of the corpus. Finally, to compare its accuracy within a corpus compared to other trained models, the testing subset is used for evaluation across all models.

Corpus	Training	Testing	Dev	**Total**	*Extracted from (Dima et al., 2019b)*
			German		
Nominal Compounds	22591	6442	3213	**54759**	*GermaNet, version 9.0 (TübingenUniversity, 2018a)*
Adjective-Noun Phrases	83603	23887	11944	**119434**	*TüBa-D/DP (TübingenUniversity, 2018b)*
Adverb-Adjective Phrases	16441	4701	2346	**23488**	*TüBa-D/DP (TübingenUniversity, 2018b)*
			English		
Nominal Compounds	11824	3481	1673	**16978**	*Existing Compound Data Set (Tratz, 2011)*
Adjective-Noun Phrases	167292	47803	23880	**238975**	*ENCOW16AX (Schäfer and Bildhauer, 2012)*
Adverb-Adjective Phrases	16222	4618	2308	**23148**	*ENCOW16AX (Schäfer and Bildhauer, 2012)*
			Dutch		
Adjective-Noun Phrases	58347	16669	8376	**83392**	*Lassy Large (Van Noord et al., 2013)*
Adverb-Adjective Phrases	3183	907	450	**4540**	*Lassy Large (Van Noord et al., 2013)*

Table 1: Corpora used for training. For each corpus, the total number of data points, split into the number of points for training, testing and development is displayed next to the original resource of the corpus.

2.2 Evaluation Methods

As introduced in 1.5.4, similar to the article of Dima et al., 2019c, the evaluation method is based on their interpretation of a ranking system inspired by Baroni and Zamparelli, 2010.

Ranking the Stars Given a model, for each tested pair of words in a corpus, using cosine similarity to calculate distance between tensors, the distance between output tensor $\boldsymbol{p}^{(n)}$ of the model and each existing word tensor in the vocabulary is calculated. Sorted by closest distance, $\boldsymbol{p}^{(n)}$ is considered to be correctly estimated by the model if and only if target word tensor $\tilde{\boldsymbol{p}}$ is at most at rank five of closest word vectors to $\boldsymbol{p}$. As an example, consider a vector domain as illustrated by figure 1 below. Let $\tilde{\boldsymbol{p}}$ be the target word vector, with $\boldsymbol{p}$ as its estimated vector. Furthermore, let the two other vectors $\boldsymbol{r}, \boldsymbol{q}$ be the results of different estimation targets. In this case, $\boldsymbol{p}$ would be of rank 3, as $cos(\boldsymbol{r}, \tilde{\boldsymbol{p}}) < cos(\boldsymbol{q}, \tilde{\boldsymbol{p}}) < cos(\boldsymbol{p}, \tilde{\boldsymbol{p}})$.

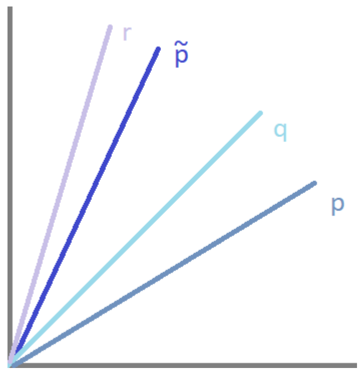

Figure 1: Example of a set of vectors, containing word vectors $\tilde{\boldsymbol{p}}, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{q}$ s.t. $cos(\boldsymbol{r}, \tilde{\boldsymbol{p}}) < cos(\boldsymbol{q}, \tilde{\boldsymbol{p}}) < cos(\boldsymbol{p}, \tilde{\boldsymbol{p}})$.

Reporting Results Given the model, all ranked tests of a corpus are listed and sorted by rank. Given this list, three quartiles and an average accuracy are reported. Moreover, the second quartile is the median of the entirety of the list, the first quartile is the median of the first half of the list and the third is the median of the second half of the list. The accuracy rate of a model represents the percentage of correctly evaluated (rank ≤ 5) word estimations compared to its total number of estimations on the corpus.

2.3 Software Implementation

To reproduce and compare testing results of TransWeight, the other existing models (1.4) and the new model, the same code is used as is published by Dima et al., 2019c in their open-source GitHub repository (Dima et al., 2019a). Moreover, the TransWeight code contains all models described in 1.4 and an implementation of the TransWeight model. Furthermore, by modifying the import variables, any new model can be implemented. To perform the experiments, the code is forked to a new GitHub repository, with the new model inserted (van Soest, 2020). According to the references in the article of Dima et al., 2019c, the repository contains the same code used in the experiments of the article. Continuing on its foundations, for the first experiment, a new layer is inserted to a copy of TransWeight.

2.3.1 Python and TensorFlow

The code provided by Dima et al., 2019c consists of an implementation of the Python TensorFlow libraries (GoogleBrainTeam, 2020a). Using Python, with required packages TensorFlow, Gensim and Keras, given required parameters, the code enables any model to be developed, trained and evaluated.

2.3.2 Computing device specifications

The code is executed on a personal computer with the specifications as described below.

Operating System Windows 10 Pro 64-bit
CPU Intel Core i5 6600K @ 3.50GHz
RAM 16,0GB Dual-Channel @ 1466MHz
Motherboard ASUSTeK Z170 PRO GAMING
Graphics 4095MB NVIDIA GeForce GTX 1070
Storage 465GB Samsung SSD 850 EVO 500GB

2.3.3 TensorFlow GPU

As depicted in 2.3.2, the experiment is run on a system with an NVIDIA GTX 1070 Graphics Card. To accelerate the training process, the code is run on this GPU. Therefore, instead of using the regular *tensorflow=1.13.1* package for python 2.7, the package *tensorflow-gpu=1.13.1* by GoogleBrainTeam, 2020b is used in combination with required graphic card libraries CUDA 10.0 by NVidiaCorporation, 2020a and NVidia cuDNN by NVidiaCorporation, 2020b.

2.3.4 Training utilities

The code contains the script *training.py*, which provides windows cmd instructions to train a given model. To improve ease of use, in the new forked version (van Soest, 2020), path parameters start from the root of the source folder of the training.

2.3.5 Evaluation utilities

The code contains the script *evaluation.py*, which provides windows cmd instructions for outputting the required evaluation results.

2.4 Experiment 1: Adding a Layer

By inserting a copy of the existing TransWeight model in the code, using TensorFlow operations to insert a new layer, the new model for the first experiment is added.

TensorFlow Operations To add a new layer to the model, an understanding of how to implement the mathematical notions of 1.4 - 1.5 is required. TensorFlow provides syntactic sugar to ease up this process, such as tf.einsum (GoogleBrainTeam, 2020c) and tf.tensordot (GoogleBrainTeam, 2020d).

2.5 Experiment 2: Increasing t

With the code, training TransWeight with a higher amount of transformations can be done by changing the training parameters, which will be discussed in 2.6.1.

2.6 Training the models

Using *training.py*, combined with the provided data sets by Dima et al., 2019b (2.1), all existing models in the code and the new model are trained on each corpus. For the second experiment, on each corpus, the model of TransWeight is trained five times, using $t = 100, 200, 500, 1000, 10000$ transformations respectively.

2.6.1 Parameters

For the first experiment, for every model, all parameters are set to their defaults specified by the code, which are displayed in table 2 below.

For the second experiment, TransWeight is trained multiple times, respectively using values $100, 200, 500, 1000, 10000$ for parameter *transforms*.

Default parameters	
batch_size	100
dropout	0.5
dropout2	0.5
patience	5
learning_rate	0.01
seed	1
transforms	100
use_weighting	False
nonlinearity	identity
selection_func	softmax
regularization	0.0
regularizer	l1_regularizer
plot	False
eval_on_test	False
max_rank	1000
eval_batch_size	500
use_nn	False

Table 2: list of all parameter values used for *training.py*

3 Results

3.1 Experiment 1

As shown in table 3, the results of training and evaluation on all corpora for the new model (DeepWeight) are very close to the results of TransWeight. As discussed below in (3.1.1) with a t-test, there is no reason to assume both results differ significantly.

Comp. model / #transforms	TransWeight	DeepWeight
German		
Noun Compounds	$[3.0, 8.0, 44.0], 41.56\%$	$[3.0, 8.0, 44.0], 41.42\%$
Adjective-Noun Phrases	$[1.0, 2.0, 8.0], 68.77\%$	$[1.0, 2.0, 8.0], 68.77\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 5.0], 76.86\%$	$[1.0, 1.0, 5.0], 76.88\%$
English		
Noun Compounds	$[1.0, 2.0, 9.0], 67.65\%$	$[1.0, 2.0, 9.0], 67.60\%$
Adjective-Noun Phrases	$[1.0, 2.0, 6.0], 74.37\%$	$[1.0, 2.0, 6.0], 74.37\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 2.0], 91.45\%$	$[1.0, 1.0, 2.0], 91.60\%$
Dutch		
Adjective-Noun Phrases	$[1.0, 2.0, 6.0], 75.00\%$	$[1.0, 2.0, 5.0], 75.06\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 3.0], 81.37\%$	$[1.0, 1.0, 3.0], 81.70\%$

Table 3: Evaluation results of the TransWeight model and DeepWeight model for every corpus in the data set.

3.1.1 Two-Sample T-Test

Given the evaluation results of the eight corpora as illustrated in table 3, let A be the collection of all estimated ranks over all corpora for TransWeight and let B be the collection of all estimated ranks over all corpora for DeepWeight. As null-hypothesis, H_0 states that both means of A and B are equal. Let $\alpha = 0.05$. We discard H_0 if the calculated p-value is lower than α. Conducting a two-sided t-test over A and B results in $(t \approx -0.245, p \approx 0.807)$. As $p > \alpha$, the t-test concludes that it fails to reject H_0. Therefore, there is no reason to assume both results differ significantly.

3.2 Experiment 2

As shown in tables 4 and 5, varying the number of transformations t for TransWeight with values $100, 200, 500$ and 1000 resulted in near equal results in the performance. As shown in 5, values $t = 10000$ and $t = 50000$ made the TensorFlow library yield an out of memory error, as the resulting amount of nodes by increasing to 10000 or more could not be allocated by the system. As discussed below in (3.2.1), based on an analysis of variance, there is no reason to assume the tested results differ significantly.

	$t = 100$	$t = 200$	$t = 500$
German			
Noun Compounds	$[3.0, 8.0, 44.0], 41.56\%$	$[3.0, 8.0, 44.0], 41.42\%$	$[3.0, 8.0, 44.0], 41.51\%$
Adjective-Noun Phrases	$[1.0, 2.0, 8.0], 68.77\%$	$[1.0, 2.0, 8.0], 68.72\%$	$[1.0, 2.0, 8.0], 68.72\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 5.0], 76.86\%$	$[1.0, 1.0, 5.0], 76.75\%$	$[1.0, 1.0, 5.0], 76.77\%$
English			
Noun Compounds	$[1.0, 2.0, 9.0], 67.65\%$	$[1.0, 2.0, 9.0], 67.51\%$	$[1.0, 2.0, 9.0], 67.31\%$
Adjective-Noun Phrases	$[1.0, 2.0, 6.0], 74.37\%$	$[1.0, 2.0, 6.0], 74.34\%$	$[1.0, 2.0, 6.0], 74.36\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 2.0], 91.45\%$	$[1.0, 1.0, 2.0], 91.21\%$	$[1.0, 1.0, 2.0], 91.29\%$
Dutch			
Adjective-Noun Phrases	$[1.0, 2.0, 6.0], 75.00\%$	$[1.0, 2.0, 6.0], 74.94\%$	$[1.0, 2.0, 6.0], 74.89\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 3.0], 81.37\%$	$[1.0, 1.0, 3.0], 81.15\%$	$[1.0, 1.0, 3.0], 81.15\%$

Table 4: Evaluation results of TransWeight over every corpus in the data set, with varying values of $t = 100, 200, 500$ in each test.

Comp. model / #transforms	$t = 1000$	$t = 10000$	$t = 50000$
German			
Noun Compounds	$[3.0, 8.0, 44.0], 41.51\%$	Out of Memory (OOM)	OOM
Adjective-Noun Phrases	$[1.0, 2.0, 8.0], 68.72\%$	OOM	OOM
Adverb-Adjective Phrases	$[1.0, 1.0, 5.0], 76.73\%$	OOM	OOM
English			
Noun Compounds	$[1.0, 2.0, 9.0], 67.42\%$	OOM	OOM
Adjective-Noun Phrases	$[1.0, 2.0, 6.0], 74.34\%$	OOM	OOM
Adverb-Adjective Phrases	$[1.0, 1.0, 2.0], 91.21\%$	OOM	OOM
Dutch			
Adjective-Noun Phrases	$[1.0, 2.0, 6.0], 74.88\%$	$[1.0, 2.0, 6.0], 74.31\%$	$[1.0, 2.0, 6.0], 74.28\%$
Adverb-Adjective Phrases	$[1.0, 1.0, 3.0], 81.26\%$	$[1.0, 1.0, 3.0], 82.00\%$	$[1.0, 1.0, 3.0], 81.11\%$

Table 5: Evaluation results of TransWeight over every corpus in the data set, with values for $t = 1000, 10000, 50000$ in each test.

3.2.1 Analysis of Variance

To test whether there is a significant difference between the tested groups of corpora, an analysis of variance (ANOVA) is conducted. Moreover, let A be the collection of all estimated ranks over all corpora for TransWeight with number of transformations $t = 100$, let B consist out of the corresponding evaluations for $t = 200$, C for $t = 500$, D for $t = 1000$, E for $t = 10000$ and F for $t = 50000$. As training and evaluating TransWeight with $t = 10000$ and $t = 50000$ resulted in an Out of Memory Error in most cases, sets E and F are excluded from the analysis. The null-hypothesis H_0 states that the means of A, B, C and D are the same. Let $\alpha = 0.05$, H_0 is rejected if the p-value is lower than α. The analysis of variance between sets A, B, C, D resulted in $ANOVA(A, B, C, D) = (t \approx 0.0744, p \approx 0.974)$. Since $p > \alpha$, there is no evidence that H_0 can be rejected.

4 Conclusion

DeepWeight is established in an attempt to further iterate on TransWeight's performance at composition word vector estimation. In summary, TransWeight includes a second layer, which increases overall model complexity. With this research investigating possible improvements on TransWeight's success, further increasing overall model complexity is speculated to attain that goal. However, based on the results of the conducted experiments, just increasing model complexity may not be the solution, as no significant improvements are observed.

4.1 Alternate approaches

TransWeight may already be complex enough, suggesting that increasing complexity may be trivial. However, such a conclusion may be drawn too fast, as this research does not disprove that with a different approach to increasing complexity, better results may be achieved. Moreover, potentially, using different composition functions such as contracting over different axes of tensors, adding a third or fourth layer, better results may be achieved.

4.2 Further research: non-linearity

As discussed in 1.4.2, according to Glorot et al., 2011, introducing non-linearity by using a rectifier function on the nodes in a neural network improves the performance of the network. Dima et al., 2019c reported to find no significant improvements for any model using non-linearity $g = ReLu = \lambda x.max(0, x)$ (Hahnloser et al., 2000) or $g = tanh$. This is in contrast with the findings of Glorot et al., 2011. However, Dima et al., 2019c only reported on comparing non-linearity functions mentioned above. Therefore, in further research, experiments with different rectifier functions can be conducted to observe potentially more or less accurate results.

References

Albreshne, A., & Pasquier, J. (2010). Semantic-based semi-automatic web service composition. *Computer Department, Switzerland*.

Baroni, M., Bernardi, R., Zamparelli, R., Et al. (2014). Frege in space: A program for compositional distributional semantics. *Linguistic Issues in language technology*, *9*(6), 5–110.

Baroni, M., & Zamparelli, R. (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*. Association for Computational Linguistics.

Bollegala, D., Alsuhaibani, M., Maehara, T., & Kawarabayashi, K.-i. (2016). Joint word representation learning using a corpus and a semantic lexicon, In *Thirtieth aaai conference on artificial intelligence*.

Bullinaria, J. A., & Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, *39*(3), 510–526.

Dima, C., de Kok, D., Witte, N., & Hinrichs, E. (2019a). Composition models for words and phrases (in python/tensorflow). Retrieved May 4, 2020, from https://github.com/sfb833-a3/commix

Dima, C., de Kok, D., Witte, N., & Hinrichs, E. (2019b). Data associated with corina dima, daniël de kok, neele witte, erhard hinrichs. 2019. no word is an island — a transformation weighting model for semantic composition. transactions of the association for computational linguistics. Retrieved May 4, 2020, from http://hdl.handle.net/11022/0000-0007-D3BF-4

Dima, C., de Kok, D., Witte, N., & Hinrichs, E. (2019c). No word is an island—a transformation weighting model for semantic composition. *Transactions of the Association for Computational Linguistics*, *7*, 437–451.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks, In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*.

GoogleBrainTeam. (2020a). Tensorflow documentation. Retrieved May 4, 2020, from https://www.tensorflow.org/

GoogleBrainTeam. (2020b). Tensorflow documentation - gpu install guide. Retrieved May 4, 2020, from https://www.tensorflow.org/install/gpu

GoogleBrainTeam. (2020c). Tensorflow documentation - tf.einsum(). Retrieved May 4, 2020, from https://www.tensorflow.org/api_docs/python/tf/einsum

GoogleBrainTeam. (2020d). Tensorflow documentation - tf.tensordot(). Retrieved May 4, 2020, from https://www.tensorflow.org/api_docs/python/tf/tensordot

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*(6789), 947–951.

Mitchell, J., & Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive science*, *34*(8), 1388–1429.

NVidiaCorporation. (2020a). Cuda toolkit 10.0 archive. Retrieved May 4, 2020, from https://developer.nvidia.com/cuda-10.0-download-archive

NVidiaCorporation. (2020b). The nvidia cuda® deep neural network library (cudnn). Retrieved May 4, 2020, from https://developer.nvidia.com/cudnn

Schäfer, R., & Bildhauer, F. (2012). Building large corpora from the web using a new efficient tool chain., In *Lrec*.

Schütze, H. (1993). Word space, In *Advances in neural information processing systems*.

Socher, R., Huval, B., Manning, C. D., & Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces, In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics.

Socher, R., Manning, C. D., & Ng, A. Y. (2010). Learning continuous phrase representations and syntactic parsing with recursive neural networks, In *Proceedings of the nips-2010 deep learning and unsupervised feature learning workshop*.

Tratz, S. (2011). *Semantically-enriched parsing for natural language understanding*. University of Southern California.

TübingenUniversity. (2018a). Germanet - an introduction. Retrieved May 10, 2020, from http://www.sfs.uni-tuebingen.de/GermaNet/

TübingenUniversity. (2018b). Tüba-d/z release 11.0 (06/2018) [final release]. Retrieved May 10, 2020, from https://uni-tuebingen.de/en/faculties/faculty-of-humanities/departments/modernlanguages/department-of-linguistics/chairs/general-and-computational-linguistics/resources/corpora/tueba-dz/

Van Noord, G., Bouma, G., Van Eynde, F., De Kok, D., Van der Linde, J., Schuurman, I., Sang, E. T. K., & Vandeghinste, V. (2013). Large scale syntactic annotation of written dutch: Lassy, In *Essential speech and language technology for dutch*. Springer.

van Soest, L. (2020). Modified github repository forked from composition models for words and phrases (in python/tensorflow). Retrieved May 4, 2020, from https://github.com/larsvansoest/commix