Comparison of two semantically aware composition models for word embeddings

Student Research Paper submitted

to

Prof. Ulf Leser

Humboldt-Universität zu Berlin
Department of Computer Science
Knowledge Management in Bioinformatics

by

Arne Binder
(520999)

Berlin, December 27, 2017

Contents

List of Abbreviations

AFS Argument Facet Similarity

ANN Artificial Neural Network

AVG averaging

FNN Feedforward Neural Network

FC fully connected layer

LSA Latent Semantic Indexing

LSTM Long Short-Term Memory

IR Information Retrieval

MSE Mean Squared Error

NLP Natural Language Processing

PMI Pointwise Mutual Information

PPMI Positive Pointwise Mutual Information

RecNN Recursive Neural Networks

RNN Recurrent Neural Network

SGD Stochastic Gradient Descent

SGNS Skip-Gram model with Negative Sampling

STS Semantic Textual Similarity

SVD Singular Value Decomposition

TF-IDF term frequency-inverse document frequency

std standard deviation

IQR interquartile range

1 Introduction

Large online debatescontain redundant arguments and therefore hinder efficient debating. To tackle this problem, **boltuzic_identifying_2015** proposed to cluster similar arguments and filter only the most representative ones.The Common Round[1] project (**uszkoreit_common_2**(introduces a platform to facilitate large scale online debating with Natural Language Processing (NLP) technologies. The authors define the aggregation of the semantic content of debates as one of their major objectives which they intend to achieve by clustering similar arguments. Moreover, **misra_measuring_2016** define the Argument Facet Similarity (AFS) task as the recognition of the semantic similarity of two sentential arguments.

UL:add examples, numbers

UL:explain further (intention?)

To achieve thesegoals, a semantically consistent similarity measure for arguments is necessary. Frameworks like word2vec (**mikolov_distributed_2013**) or glove (**pennington_glove_2014**) lead to success in many NLP tasks by producing dense embedding vectors for single words. The cosine distance of these word embeddings enables a semantically consistent similarity measure for word tokens. Building on this, **habernal_exploiting_2015**; **boltuzic_identifying_2015**; **misra_measuring_2016** use summed or averaged word embeddings for argument similarity measures. However, **misra_measuring_2016** state that averaging all word embeddings may lose too much information in long sentences. **wang_comparison_2017** present an overview of different embedding composition models for phrase representations. They conclude that the recurrent Long Short-Term Memory (LSTM) model (**hochreiter_long_1997**) just slightly outperforms the additive baseline model in this task. **tai_improved_2015** introduced the recursive TreeLSTM model which further reduced the vanishing gradient problem as compared to recurrent models by shortening the average distance of entities in the computation graph. The authors use dependency parse trees to construct the neural model and present promising results for the SICK phrase relatedness task[2].

AB:move away from arguments as motivation

UL:more precise!

However, these modelsare applied to sentences, but not yet to multi-sentence arguments or paragraphs. Additionally, theirapproach does not use dependency edge type information such as 'nsubj' (nominal subject) or 'prep' (preposition).

UL:specify!

UL:specify!

[1] see http://commonround.dfki.de/

[2] The SICK corpus (**marelli_sick_2014**) contains ~10.000 similarity scored sentence pairs. The system by **tai_improved_2015** achieved a Pearson's r of 0.8676 when predicting the similarities.

1.1 Objective

In this work we examine what degree of semantical awareness is achievable with different existing embedding composition models of varying complexity when applied to sentences. We regard compositions of token embeddings as semantically aware if they yield a distance measure that matches human intuition. Furthermore, our goal is to provide access to the rapid implementation of different neural network topologies including Recursive Neural Networks (RecNN) for semantically aware composition.

UL:clarify sentence level vs. super-sentence level mis-match

1.2 Approach

To achieve these goals we implement the following: (a) an averaging model, and (b) a neural sequence model. We evaluate the semantical awareness with the SICK Semantic Textual Similarity (STS) challenge (**marelli_sick_2014**) and compare the model performances against a TF-IDF baseline.

To prepare for rapid comparison with various neural models, we use the Tensorflow Fold[3] framework (**looks_deep_2017**) for implementation. It allows fast training of neural networks with dynamic computation graphs, i.e. networks whose structure depends on the data that is fed during training. Moreover, it is based on Tensorflow[4] which is widely used as a deep learning framework.

As neural sequence model, we use an approach based on **mueller_siamese_2016** due to its simplicity. It consists of one LSTM as composition model which is applied to two input sentences and uses the Manhattan metric as distance measure. Nevertheless, the authors report a Pearson's r of 0.8822 for the SICK challenge. Furthermore, we analyze the effect of dependency type information by optionally appending edge type embeddingsto the token embedding vectors.

UL:explain

One major drawback of the SICK corpus is its small size. We will use pre-training with paraphrase data from the PPDB[5] corpus (**ganitkevitch_ppdb:_2013**) and evaluate its effect to the model performances. In its smallest, most accurate version the phrasal subset of the corpus contains 1,530,812 pairs of short phrase snippets like ('`maintain international peace and`', '`maintaining world peace and`'). We treat these pairs as similar and extend the dataset with artificially generated negative samples where the two phrases are randomly sampled from the corpus.

[3] see https://github.com/tensorflow/fold

[4] see www.tensorflow.org

[5] see http://paraphrase.org

2 Theoretical Background

This chapter gives an overview of base concepts regarding semantic aware vector space representations and neural machine learning models to produce them.

2.1 Evaluation of Semantic Awareness

Following the Distributional Hypothesis (**sahlgren_distributional_2008**; **harris_distribut** AB:check references **54**), that linguistic items with similar distributions have similar meanings, a conceptional space should be arranged in a way, that distances of concepts follow the human intuition. By doing so, the meaning of these concepts should be as expected as by humans.

We define semantic awareness as a property describing how much meaning a representational space is able to convoy with respect to the space of origin, textual natural language. Furthermore, the state of being semantically aware references the maximal achievable semantic awareness, i.e. convoying the same meaning as the origin of the representation.

With regard to the Distributional Hypothesis semantic awareness can be measured by the correlation of the distances between different concepts in the representational space and the distances between its source concept representations in the original space. Despite the concept of *distance* seems to be more intuitive in the context of representational spaces, its inverse, the concept of *similarity*, commonly serves as base of comparison.

Taking similarity measures in both spaces for granted, that map into the same space $X \subseteq \mathbb{R}$, we can borrow the task of *similarity prediction* for evaluation of semantic awareness AB:add ref when using *semantic relatedness* as underlying similarity metric of X. With the concept of semantic relatedness we refer to the interpretation by **budanitsky_evaluating_2006** *linked by any kind of lexical or functional association.* **resnik_semantic_1999** visualizes this concept by contrasting it with *semantic similarity.* The authors state that "car" and "gasoline" are related, but not semantic similar, whereas "car" and "bicycle" are semantic related *and* similar. Thus, semantic similarity covers only a subset of semantic relatedness and captures relations like synonymy or hypernymy while semantic relatedness further captures antonomy, meronymy and others.

The capability of predicting this kind of similarity in a conceptional space by using an internal representation requires by definition a degree of semantic awareness of this internal space, that we are going to evaluate.

2.1.1 Similarity Prediction

The task of Similarity Prediction can be framed as to predict a score given two entities of a conceptional space, where higher scores represent higher degrees of similarity between them. In the context of NLP these entities are textual documents. The score can model different granularities of similarity, ranging from binary classification, e.g. in the case of Paraphrase Detection[6], weighted aggregation of multi-label classification results, e.g. for instances of discrete questionnaire data, to regression with continuous scores.

We will focus on Similarity Prediction as regression task as it models semantic relatedness based similarity more accurate then binary classification. Semantic Relatedness of arbitrary[7] composed textual entities is no binary phenomena. Otherwise the entity space would disintegrate into distinct clusters each containing similar entities consequently interpreted as identical. As semantic relatedness is defined about *any* kind of lexical or functional association, it is expected that any entity is semantical related with the majority of the entities in the space in a transitively manner allowing only the trivial solution, precisely, that all entities are identical with respect to the similarity. The compensation of this fact increases the amount of necessary data to approximate the real behavior. Furthermore, modeling Similarity Prediction as regression is straightforward compared to aggregation of multi-classification output.

2.1.2 Evaluation Measures

Common measures for evaluation of regression estimators are *Mean Squared Error*, *Pearson correlation coefficient* (**pearson_note_1895**) and *Spearman's rank correlation coefficient* (**spearman_proof_1904**). UL:ref

The Mean Squared Error (MSE) measures the deviation of the predicted values to the correct ones. It incorporates the variance and the bias of the estimator. A value of zero denotes a perfect estimator and larger values indicate lower estimator performance. Given n predictions Y and corresponding correct values X, the MSE is defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - X_i)^2 \tag{1}$$

The Pearson correlation coefficient (Pearson's r) expresses the degree of linear correlation of two variables. It is the covariance of the variables normalized by their standard deviations.

[6] Paraphrase Detection is the task of identifying if two phrases paraphrase each other.

[7] The composition is surely constrained by grammar, linguistic performance and other criteria, but this term is in favor of the tremendous variability of natural language.

It ranges from -1, denoting perfect negative correlation, to 1, denoting perfect positive correlation. A value of zero expresses that the two variables do not correlate at all. Therefore, higher values above zero indicate better estimator performance. Given X and Y, Pearson's r is defined as:

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2 \cdot \sum_{i=1}^{n}(Y_i - \overline{Y})^2}} = \frac{Cov(X,Y)}{\sigma(X)\sigma(Y)}, \tag{2}$$

where $\overline{X}$ is the empirical mean of X, $Cov(X,Y)$ is the empirical covariance of X and Y, and $\sigma(X)$ is the empirical standard deviation.

Spearman's rank correlation coefficient (Spearman's ρ) serves as another correlation based measure, which does not assume linear dependence between X and Y. Hence, it is more robust regarding outliers. Spearman's ρ can be interpreted as special case of Pearson's r where the data is converted into ranks before calculation of Pearson's r. Given X and Y, Spearman's ρ is defined as:

$$\rho = \frac{\sum_{i=1}^{n}(rg(X)_i - \overline{rg(X)})(rg(Y)_i - \overline{rg(Y)})}{\sqrt{\sum_{i=1}^{n}(rg(X)_i - \overline{rg(X)})^2 \cdot \sum_{i=1}^{n}(rg(Y)_i - \overline{rg(Y)})^2}} = \frac{Cov(rg(X), rg(Y))}{\sigma(rg(X))\sigma(rg(Y))}, \tag{3}$$

where $rg(X)$ are the ranks of X.

2.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a machine learning model where input vectors are mapped to predictions by successive application of computational *layers* (**rumelhart_parallel_1986**). Each layer is a differentiable function $l^{(i)} : \mathbb{R}^{n^{(i-1)}} \mapsto \mathbb{R}^{n^{(i)}}$, with $n^{(i)}$ denoting the size of the input vector for the i-th layer, its *layer size*. The arrangement of the layers form the network graph, which is a directed graph defining the specific network architecture. Provided adequate training data, ANNs can be trained efficiently with gradient descent based optimization algorithms. In the next sections we will describe some popular architectures and important components of ANNs.

2.2.1 Feedforward Networks

The simplest class of ANNs are Feedforward Neural Networks (FNNs) that emerged from multilayer Perceptrons (**rosenblatt_perceptron:_1958**). Their network architecture forms an acyclic graph and is defined by $f^{FNN}(v) = (l^{(N)} \circ ... \circ l^{(i)} \circ ... \circ l^{(1)})(v)$ where N is the total amount of layers. The N-th layer is called *output layer*, all other layers are *hidden layers*. In addition, the virtual *input layer* models the input. Figure 1 (**epelbaum_deep_2017**) shows such a structure.

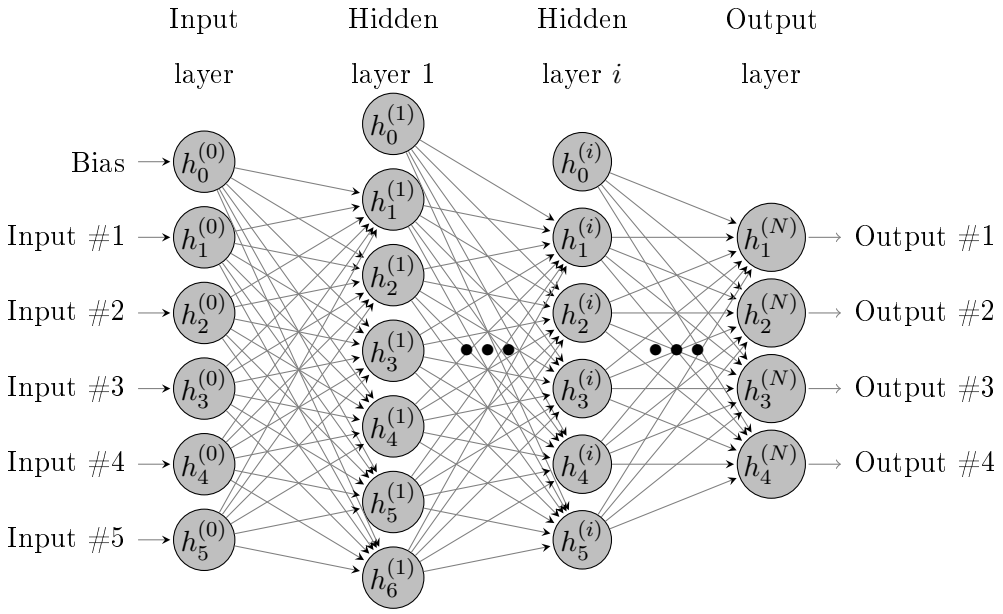


Figure 1: Neural Network with $N+1$ layers ($N-1$ hidden layers).

There exists a diverse landscape of different types of layers. A common instance is the fully connected layer (FC). It connects every node from the previous layer with every node from the current one in the term of a weighted sum. Hence, it calculates an affine transformation of its input. Usually a layer $l^{(i)}$ is parametrized by $\theta^{(i)}$, e.g. a set of *weights* and *biases*. In this manner, the FC can be defined as:

$$l^{FC}(v;\theta^{(i)}) = w^{(i)}v + b^{(i)} \tag{4}$$

where $w^{(i)} \in \mathbb{R}^{n^{(i)} \times n^{(i-1)}}$, $b^{(i)} \in \mathbb{R}^{n^{(i)}}$ and $\theta^{(i)} = \{w^{(i)}, b^{(i)}\}$.

2.2.2 Activation Functions

The Feedforward Neural Network (FNN) described so far is capable of modeling linear mappings only. To enable non-linearity, different non-linear activation functions are applied element-wise to the individual layer outputs. A prominent instance is the *sigmoid* function that takes a real-valued input and "squashes" it to range between 0 and 1. Therefore, it is used to model Bernoulli-Distributions. It is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{5}$$

The *hyperbolic tangent* is another strictly increasing activation function, but maps into the range between -1 and 1:

$$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} = 2\sigma(2x) - 1 \tag{6}$$

Finally, the *softmax* function can be used to model discrete probability distributions, e.g. for classification tasks. In contrast to the previous activation functions it is not element wise independent, since it applies normalization across all input elements to guarantee that its result follow the probability distribution conditions. For an input vector x with entries x_i it is defined as:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{x_j \in x} e^{x_j}} \tag{7}$$

2.2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) introduced by **hopfield_neural_1982** are ANNs that are capable of processing arbitrary long sequences when feeding them stepwise into the network. Especially, they can access information from any previous step and, hence, process the new data conditionally. This is possible via a feedback loop in a *recurrent layer*. Given a sequence of τ-sized vectors $v^{(1)}, ..., v^{(\eta)}$ the t-th output $h^{(t)}$ of a recurrent layer l^{RNN} is defined as:

$$h^{(t)} = l^{RNN}(v^{(t)}; \theta) = g(v^{(t)}; h^{(t-1)}; \theta) \tag{8}$$

where $k, \vartheta \in \mathbb{N}$, k is the inner state size and g is the Recurrent Neural Network (RNN) cell function with $g : (\mathbb{R}^{\tau}; \mathbb{R}^{k}; \mathbb{R}^{\vartheta}) \mapsto \mathbb{R}^{k}$.

The Long Short-Term Memory (LSTM) (**hochreiter_long_1997**) is a prominent RNN. It is a *gated* RNN, meaning its cell function g^{LSTM} consists of multiple interconnected gating functions and, further, models a dedicated cell state $\tilde{c}$. LSTMs tackle one major draw back of the RNN architecture, namely the vanishing gradient problem that occurs when training with long sequences. Without gating, repeated application of one RNN cell to the elements of a sequence leads to exponential decrease of the calculated gradients because common activation functions produce gradients lesser 1 that are multiplied according the chain rule. That effect practically hinders the network to use long-distance information. Section 2.3.2 explains the exact role of gradients in neural network training.

The different gates of the LSTM are responsible for deciding what information should be forgotten (g_f), which new information will be added (g_i) and, finally, which information from $\tilde{c}$ will be presented as output (g_o):

$$
\begin{aligned}
g_f^{(t)} &= \sigma(l^{FC}([\tilde{h}^{(t-1)}, v^{(t)}]; \theta^{(f)})) \\
g_i^{(t)} &= \sigma(l^{FC}([\tilde{h}^{(t-1)}, v^{(t)}]; \theta^{(i)})) \\
g_o^{(t)} &= \sigma(l^{FC}([\tilde{h}^{(t-1)}, v^{(t)}]; \theta^{(o)})) \\
c^{(t)} &= tanh(l^{FC}([h^{(t-1)}, v^{(t)}]; \theta^{(c)})) \\
\tilde{c}^{(t)} &= g_f^{(t)} \odot \tilde{c}^{(t-1)} + g_i^{(t)} \odot \tilde{c} \\
\tilde{h}^{(t)} &= g_o^{(t)} \odot tanh(\tilde{c}^{(t)})
\end{aligned}
\tag{9}
$$

and finally

$$
g^{LSTM}(v^{(t)}; [\tilde{c}^{(t-1)}, \tilde{h}^{(t-1)}]) = [\tilde{c}^{(t)}, \tilde{h}^{(t)}] \tag{10}
$$

where $[a, b]$ is the concatenation of the vectors a and b, and $a \odot b$ denotes their element-wise product (Hadamard product). We omit the parameter set $\theta^{LSTM} = \{\theta^{(f)}, \theta^{(i)}, \theta^{(o)}, \theta^{(c)}\}$ for better readability.

2.3 Supervised Training of ANNs

Supervised training of a model f parametrized by θ takes a set Ω of input-output training tuples (x, y) for granted and tries to find instantiations of θ in the manner that $f(x; \theta) = \hat{y} \approx y, \forall (x, y) \in \Omega$ (**mohri_foundations_2012**). In the context of ANNs, one usually uses a *cost function* J that quantifies the deviation of $\hat{y}$ from y. J is used to guide the parameter adjustment via gradient-based optimization.

2.3.1 Cost Function

A cost function, or *loss function*, commonly used for regression tasks is Mean Squared Error (MSE) as described in section 2.1.2 because it complies to the maximum likelihood of y and $\hat{y}$ if the error is normally distributed and is efficient to calculate in particular. We reformulate AB:add ref? equation (1) to fit into our needs:

$$
J_{\Omega}^{MSE}(\theta) = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{\iota-1} (y_{ij} - \hat{y}_{ij})^2 \tag{11}
$$

where n is the amount of tuples (x_i, y_i) in Ω and $\hat{y}_i = f(x_i; \theta)$ with $y_i, \hat{y}_i \in \mathbb{R}^{\iota}$. Note that, despite the cost function depends on the training samples, these are considered to be fixed.

2.3.2 Gradient-based Optimization

Assuming the loss function J_Ω is differentiable, in theory it would be possible to calculate its global minimum θ^* analytically. However, in fact that is infeasible for common amounts of parameters in θ that can be in the order of millions or billions.

UL:evidence, ref

Gradient-based optimization tackles this problem by shifting the parameters θ stepwise in the negative direction of its gradients ∇J_Ω. By definition of gradients, an infinitesimal displacement of θ in this direction decreases $J_\Omega(\theta)$. Hence, we select the parameters for the next optimization step θ':

$$\theta' = \theta - \alpha \nabla J_\Omega(\theta) \tag{12}$$

where α is the chosen step size, or *learning rate*, that has to be sufficiently small. On the other hand, it is unfavorable to choose a step size too small because that increases convergence time, meaning it slows down training. A lot of different approaches exist that apply dynamical sized stepping like *Momentum* (**polyak_methods_1964**), which takes previous parameter updates into account, or *AdaDelta* (**zeiler_adadelta:_2012**) that tracks individual learning rates for the parameters in θ. ADAM (**kingma_adam:_2014**) is another robust method to guide the parameter optimization, it keeps track of the gradients and its square roots.

As equation (11) states, one update step requires summation over all training examples in Ω, so it is an $O(n)$ operation. Hence, with large training sets calculation of $\nabla J_\Omega(\theta)$ for every step becomes infeasible. Stochastic Gradient Descent (SGD) addresses this problem by randomly partitioning Ω into subsets Ω_i^P of size n^B that are used successively to calculate new parameter sets θ'. Exhausting all Ω_i^P, also referred to as *(mini-)batches*, of a certain partition Ω^P is called finishing an *epoch*. This approach stochastically approximates the calculation of $\nabla J_\Omega(\theta)$.

really? not higher?

Finally, we describe the core algorithm that enables gradient-based training of ANNs, namely *Backpropagation* (**rumelhart_learning_1988**). It uses dynamic programming for efficient calculation of the gradients by exploiting their interconnectivity as present in common ANNs. As stated before, an ANN f can be represented as a composition of layer functions $l^{(i)}$ with $f(v) = (l^{(N)} \circ ... \circ l^{(i)} \circ ... \circ l^{(1)})(v)$ and $l^{(i)}(v^{(i)}) = v^{(i+1)}$. Hence, the *chain rule* of gradient computation defines the derivatives for two successive layers as:

$$\frac{dv^{(i+2)}}{dv^{(i)}} = \frac{dv^{(i+2)}}{dv^{(i+1)}} \frac{dv^{(i+1)}}{dv^{(i)}} \tag{13}$$

$$= l^{(i+1)'}(v^{(i+1)}) l^{(i)'}(v^{(i)}) \tag{14}$$

$$= l^{(i+1)'}(l^{(i)}(v^{(i)})) l^{(i)'}(v^{(i)}) \tag{15}$$

$v^{(i)}$ occurs several times in equation (15). Furthermore, as shown in (13) $\frac{dv^{(i+2)}}{dv^{(i)}}$ depends on $\frac{dv^{(i+2)}}{dv^{(i+1)}}$ immediately. Backpropagation exploits these redundancies by successively calculating all $v^{(i)}$ in a *forward pass*. Then, it calculates all gradients in reverse order while reusing intermediate results in a *backwards pass*. By doing so, the algorithm reduces the complexity from exponential, considering a naive approach, to linear with regard to the edge count of the network graph.

2.4 A Model for Similarity Prediction

We divide similarity prediction with regard to textual inputs into transformation of the inputs into vector space representations and similarity calculation in this embeddings space.

2.4.1 Vector Space Embeddings

We define *embeddings* of documents, words or other linguistically motivated units as vector space representations that are to a degree semantically aware. Furthermore, *embedding models* are functional transformations that map these units into this *embedding space*.

UL:ref, expand!

Because of the relevance of embedding models for many NLP tasks, a diverse landscape of different approaches has evolved. In the following, we distinguish traditional document embedding approaches from composition based models. By traditional approaches we refer to models which process sequences of opaque tokens and directly produce embeddings for these sequences, whereby composition based models initially embed each token independently and compose these token embeddings into a final vector space representation.

UL: REFS!

2.4.2 Traditional Document Embeddings

XXX READ: (**turney_frequency_2010**) As we focus on distributional semantics, we can outline some common features for traditional document embedding models. These models build upon occurrences of terms T in certain contexts C. For instance, a word can occur as a term in a document, i.e. its context, or not. We define $T = \{t_1, ..., t_m\}$ as finite set of opaque symbols with $m = |T|$ and the set of contexts $C \subseteq T^*$ with $n = |C|$, e.g. every context is a sequence of terms. The set of unique terms T is called a vocabulary. We define a *context embedding* as a function $f_C : C \mapsto \mathbb{R}^{\hat{n}}$, where $\hat{n} \in \mathbb{N}$ is the dimensionality of the embedding vectors.

AB:name Vector Space Models!

call it alphabet?

One popular embedding model is based on the term frequency-inverse document frequency (TF-IDF) measure. TF-IDF was introduced as *term specificity* in **sparck_jones_statistical_1972**

Given T and C, the term frequency tf of a certain term t with respect to a context c is defined as the count of occurrences of t in c normalized by the maximum of the term counts in this context:

$$tf(t,c) = \frac{\#(t,c)}{max_{t' \in c} \#(t',c)} \tag{16}$$

where $\#(t,c)$ is the frequency of term t in context c. The inverse document frequency idf of a term t regarding contexts C is defined as:

$$idf(t) = \frac{|C|}{|\{c \in C : t \in c\}|} \tag{17}$$

Finally, the term frequency-inverse document frequency $tf.idf$ is calculated:

$$tf.idf(t,c) = tf(t,c) \cdot idf(t) \tag{18}$$

Hence, given a corpus containing documents as contexts C of terms T, it is possible to calculate a $|T| \times |C|$ TF-IDF matrix $M^{tf.idf}$ with $M^{tf.idf}_{ij} = tf.idf(t_i, c_j)$. The columns of $M^{tf.idf}$ represent embeddings for the contexts, respectively documents. The TF-IDF context embedding $f_C^{tf.idf}$, i.e. the TF-IDF document representation, is defined as $f_C^{tf.idf}(c_j) = M^{tf.idf} \cdot e_j$ where e_j is the j-th unit vector. In the following we call a matrix M^x whose entries M^x_{ij} quantify in any way the occurrences of term t_i in context c_i an *occurrence matrix*.

check order of |T| and |C|

Another common approach utilizes Pointwise Mutual Information (**church_word_1990**). Pointwise Mutual Information (PMI) quantifies the association of two outcomes of discrete random variables X and Y. Given outcomes x and y belonging to these variables, the PMI is defined as:

$$pmi(x;y) \equiv log \frac{p(x,y)}{p(x)p(y)} \tag{19}$$

By interpreting T and C as random variables X and Y an occurrence matrix M^{pmi} can be constructed as follows:

$$\begin{aligned} M^{pmi}_{ij} &= log \frac{\#(t_i, c_j)}{\sum_{c \in C} \#(t_i, c) \cdot \sum_{t \in T} \#(t, c_j)} \\ &= log\, \#(t_i, c_j) - b_i^{(T)} - b_j^{(C)} \end{aligned} \tag{20}$$

where $b_i^{(T)}$ and $b_j^{(C)}$ are term and context biases with

$$\begin{aligned} b_i^{(T)} &= log \sum_{c \in C} \#(t_i, c) \quad \text{and} \\ b_j^{(C)} &= log \sum_{t \in T} \#(t, c_j) \end{aligned} \tag{21}$$

Because $\#(T_i, C_j)$ is zero and, consequently, M_{ij}^{pmi} equals $-\infty$ for a lot of entries, often the Positive Pointwise Mutual Information (PPMI) (**niwa_co-occurrence_1994**) is used:

$$M_{ij}^{ppmi} = max(M_{ij}^{ppmi}, 0) \tag{22}$$

By construction these document representations have as many dimensions as unique terms exist in T. Bur according to Zipf's Law ... This draw backcan be tackled by filtering very frequently occurring words, called *stop words*, assuming they do not convey particular meaning, and, further, very rare words to handleZipf's word frequency law in natural language. But even after ..., the resulting embeddings are very sparse because the majority of words occur only in small subsets of a natural language corpus if the term distributions are not artificially controlled.

UL:explain, why bad -> main argument(?)
UL:explain!
UL:ref
add content
UL:ref

To combat this problem, there are several methods to reduce the embedding dimensionality while maintaining semantic awareness. One common approach is to apply Singular Value Decomposition (SVD) to the occurrence matrix. Latent Semantic Indexing (LSA) (**deerwester_indexing_1990**) implements this in the context of Information Retrieval (IR) by decomposing the occurrence matrix M described above[8]in the following way:

UL:main argument
UL:ref

$$M = U \cdot S \cdot V^T \tag{23}$$

where $U = MM^T$, $V = M^TM$ and S is a diagonal matrix that contains the eigenvalues of M. Furthermore, U and V are orthogonal bases, whereby U spans a term based vector space and V a document based vector space. Using this, a dimensionality reduction can be done by removing the smallest eigenvalues from S resulting in the submatrix $S_{(\hat{n})}$ that contains only the $\hat{n}$ highest eigenvalues. This is a reasonable step because $S_{(\hat{n})}$ and the corresponding matrices $U_{(\hat{n})}$ and $V_{(\hat{n})}$ produce the rank $\hat{n}$ approximation to M with the smallest error[9]. Mapping a document representation d, where d consists of entries as in M, into the reduced dimensional space spanned by $V_{(\hat{n})}$ can be computed by:

UL:ref

$$\hat{d} = S_{(\hat{n})}^{-1} U_{(\hat{n})}^T d \tag{24}$$

where $\hat{d}$ is the $\hat{n}$-dimensional document representation.

Despite its clear theoretical motivation SVD basedVSMs have the major draw back that the decomposition algorithm has to be applied every time a document or a term is added.

UL: more accurate
AB:introduce!
UL:Wozu?

[8]The original LSA makes use of the *term document matrix* M^{td}, a simplification of M^{pmi} without normalization, where $M_{ij}^{td} = \#(T_i, C_j)$.

[9]according to Frobenius norm

SVD is computational expensive, it has a complexity of $O((n+m)^2\hat{n}^3)$. Assuming that T does not change as much as C, approaches arise that build upon (pre-) calculated *word* or *term embeddings*. Then, embedded terms are dynamically composed to document representations leading to *composition models*. We will outline these concepts in the following sections.

UL:ref

UL:ref

2.4.3 Term Embeddings

Similar to context embeddings, we define a term embedding as a function $f_T : T \mapsto \mathbb{R}^{\hat{m}}$, where $\hat{m} \in \mathbb{N}$ is the dimensionality of the resulting embedding vectors.

The methods described in the previous section are directly applicable to produce term embeddings by defining $f_T(t_i) = M^T \cdot e_i$ where $M \in \mathbb{R}^{m \times \hat{m}}$is an occurrence matrix. A common approach to produce term embeddings is to calculate M^{ppmi} and apply SVD. In analogy to $\hat{d} = S_k^{-1} U_k^T d$ (equation (24)), a term embedding $\hat{t}$ of the context based term vector t of term T_i can be calculated as $\hat{t} = S_k^{-1} V_k^T t$.

AB:define or replace m

UL:explain benefit!

Further ideas like the Hyperspace Analogue to Language model (**lund_producing_1996**) build upon co-occurrence counts. They can be cast into the conceptional framework outlined by defining C via a co-occurrences relation R^{CO} as:

$$C_j = \{t_i | (t_i, t_j) \in R^{CO}\} \tag{25}$$

where $R^{CO} = \{(t_i, t_j) | t_i, t_j \in T;\ t_i \text{ and } t_j \text{ are successive tokens}\}$. This approach can be enhanced by relaxing the co-occurrence relation, e.g. defining R^{CO} via a fixed or soft sized co-occurrence window larger than two and optionally adding distance weighting.

AB:refs?

In line with **levy_improving_2015** we refer to the term embedding models presented so far as *count based* models and contrast them with *prediction based* models. Prediction based embedding models commonly are ANNs whose intermediate low[10] dimensional layer results are exploited as term embeddings. They raised attention due to recently published methods that allow efficient embedding calculation with regard to billions of tokens. The Skip-Gram model with Negative Sampling (SGNS)[11] (**mikolov_distributed_2013**) is a popular representative that does not immediatelyrely on an occurrence matrix, but trains a simple, one hidden layer ANN with the objective to predict the context of a term, e.g. the

UL:add figure

UL:introduce idea before "how-to"/technical explanation

UL:repr. of what?

UL:UNCLEAR

[10] with respect to the size of T

[11] A SGNS implementation was released within the word2vec framework (**mikolov_efficient_2013**).

neighbors of the tokens in a textual corpus. The core Skip-Gram idea can be formalized as:

$$\begin{aligned} f_T^{SG}(t_i)_j &= p(c_j|t_i) \\ &= (softmax \circ l_{out}^{FC} \circ l_{hidden}^{FC})(e_i) \cdot e_j \\ &= softmax(l_{out}^{FC}(l_{hidden}^{FC}(e_i; \{w_H; b_H\}); \{w_O; b_O\}) \cdot e_j \\ &= softmax(w_O(w_H \cdot e_i + b_H) + b_O) \cdot e_j \\ &= softmax([w_O, b_O^T] \cdot [w_H, b_H^T] \cdot e_i) \cdot e_j \\ &= softmax(M^{out} \cdot M^{hidden} \cdot e_i) \cdot e_j \end{aligned} \tag{26}$$

Then, the Skip-Gram term embedding can be defined as $f_T^{SG}(t_i) = M^{hidden} \cdot e_i$. Although M^{hidden} does not obviously rely on occurrence counts immediately, **levy_neural_2014** showed that, in fact, Skip-Gram factorizes a shifted PMI matrix in the way that:

$$M^{hidden} \cdot M^{out} + log(k) = M^{pmi} \tag{27}$$

where k is a constant.

Glove (**pennington_glove:_2014**) is a prediction based model that explicitly uses M^{pmi}. Its cost function J^{GLOVE} can be defined as:

UL:introduce! (??)

$$J^{GLOVE}(\{w; b\}) = \sum_{t_i \in T} \sum_{c_j \in C} M_{ij}^B (w_i^T w_j + b_i + b_j - log\,\#(t_i, c_j))^2 \tag{28}$$

where M^B is a fixed matrix of biases and $|T| = |C|$, i.e. the number of possible terms equals the number of possible contexts. The later is feasible because Glove uses co-occurrences to define C as described in equation (25), therefore contexts can be identified by terms.[12]

If we recap the definition of M^{pmi}, $M_{ij}^{pmi} = log\,\#(t_i, c_j) - b_i^{(T)} - b_j^{(C)}$ (see equation (21)), we notice that Glove's objective factorizes M^{pmi}:

$$(w^T w)_{ij} = log\,\#(t_i, c_j) - b_i - b_j \tag{29}$$

$$(w^T w)_{ij} + b_i - b_i^{(T)} + b_j - b_j^{(C)} = M_{ij}^{pmi} \tag{30}$$

Note, that, in contrast to Skip-Gram, this approach implements full symmetry with regard to T and C. By doing so, all information is accumulated in one mapping, context specific information is not outsourced, like it does the default Skip-Gram model with M^{out}[13]:

re-write eq:28 into matrix operations (?)

UL:unclear!

UL: too short! benefit?

$$f_T^{GLOVE}(t_i) = f_C^{GLOVE}(c_i) = w \cdot e_i + b_i \tag{31}$$

[12] The same holds for Skip-Gram.

[13] This can be circumvented for Skip-Gram by using $(f_T^{SG}(t_i) + f_C^{SG}(c_i))$ or $[f_T^{SG}(t_i), f_C^{SG}(c_i)]$ as term embedding for further application, where $f_C^{SG}(c_i) = (M^{out})^T \cdot e_i$.

2.4.4 Composition Models

Composition Models intend to produce document embeddings $\hat{d}$ from a sequence of term embeddings $\hat{s} = [\hat{t}_1, ..., \hat{t}_\kappa]$ with $\forall \hat{t} \in \hat{s} : \hat{t} = f_T(t), t \in T, \hat{t} \subseteq \mathbb{R}^{\hat{m}}$. They can be expressed recursively by:

$$f_{CM}(\hat{s}) = f_N(f_{Rec}(\hat{s}; h_I); |\hat{s}|) \quad \text{and}$$
$$f_{Rec}(\hat{s}; h) = \begin{cases} h & \text{if } |\hat{s}| = 0 \\ f_{Rec}([\hat{t}_2, ..., \hat{t}_\kappa]; f_R(f_M(\hat{t}_1); h)) & \text{else} \end{cases} \tag{32}$$

where $f_M : \mathbb{R}^{\hat{m}} \mapsto \mathbb{R}^{\mathring{m}}$ is the term embedding *mapping function*, $f_R : (\mathbb{R}^{\mathring{m}}, \mathbb{R}^k) \mapsto \mathbb{R}^k$ is the internal *reduction function*, $f_N : (\mathbb{R}^k, \mathbb{N}) \mapsto \mathbb{R}^{\hat{n}}$ is a *normalization function* and $h_I \in \mathbb{R}^k$ is the initial internal state.

In the following we distinguish *order unaware* composition models from *order aware* models. Both types commonly use the identity function or $(htan \circ l^{FC})$ as mapping function f_M. There are also hybrid approaches based on n-grams and Convolutional Neural Networks[14], but these are out of scope for this work.

Order unaware composition

These models are known as Bag-of-Words models. They can be defined by constraining the reduction function f_R to be symmetric. Common instances are *summation* with:

$$f_{R_{sum}}(\mathring{t}; h) = h + \mathring{t} \quad \text{and} \quad h_{I_{sum}} = [0, ..., 0]^T \in \mathbb{R}^k, k = \mathring{m} \tag{33}$$

or *averaging* with:

$$f_{R_{avg}} = f_{R_{sum}}, \quad h_{I_{avg}} = h_{I_{sum}} \quad \text{and} \quad f_{N_{avg}}(\dot{d}; \kappa) = \frac{1}{\kappa} \cdot \dot{d} \tag{34}$$

Order aware composition

These models take the ordering of the input tokens into account. Commonly they build upon Recurrent Neural Networks. So, we can define the reduction function as:

$$f_{R_{rnn}} = g^{RNN} \tag{35}$$

where g^{RNN} is an RNN cell function (see equation (8), assuming θ is fixed) like g^{LSTM} (see equation (10)), for instance.

[14] They fit into the definition of f_{CM} by relaxing f_M, which is, in fact, a convolution kernel of size one, to $f_{M_{cnn}} : (\mathbb{R}^{\hat{m}}, ..., \mathbb{R}^{\hat{m}}) \mapsto \mathbb{R}^{\mathring{m}}$ and using the *max* function for $f_{R_{cnn}}$ to implement max pooling, for instance.

2.4.5 Similarity measures

A similarity measure for embeddings is a symmetric function $f_S : (\mathbb{R}^{\hat{n}}, \mathbb{R}^{\hat{n}}) \mapsto [0, 1] \subset \mathbb{R}$ with:

UL: why not sim(a,a)=1 ?

$$f_S(\hat{d}_i, \hat{d}_i) \geq f_S(\hat{d}_i, \hat{d}_j) \tag{36}$$

The *cosine similarity* is a prominent similarity measure, that is defined by:

$$f_{S_{cos}}(\hat{d}_i, \hat{d}_j) = \frac{\sum_{k=1}^{\hat{n}} \hat{d}_{i_k} \hat{d}_{j_k}}{\|\hat{d}_i\|_2 \|\hat{d}_j\|_2} = \left(\frac{\hat{d}_i}{\|\hat{d}_i\|_2}\right)^T \cdot \frac{\hat{d}_j}{\|\hat{d}_j\|_2} \tag{37}$$

where $\|v\|_2 = \sqrt{\sum_{k=1}^{|v|}(v_k)^2}$ is the L_2 norm of v. As equation (37) shows this measure explicitly normalizes its input. This can be an advantage because it is independent of the *significance* of the embeddings which is, as **schakel_measuring_2015** state, encoded by their norm.

UL explain intuitively! not just on the side...

An *euclidean*distance based similarity measure can be constructed as:

AB:throw away?

$$f_{S_{eucl}}(\hat{d}_i, \hat{d}_j) = exp(-\|\hat{d}_i - \hat{d}_j\|_2) \tag{38}$$

UL:euclidean, in what way?

Similarly, the *manhattan/taxicab* distance can be used as similarity measure as proposed by **mueller_siamese_2016**

$$f_{S_{manh}}(\hat{d}_i, \hat{d}_j) = exp(-\|\hat{d}_i - \hat{d}_j\|_1) \tag{39}$$

where $\|v\|_1 = \sum_{l=1}^{|v|} |v_l|$ is the L_1 norm of v. Compared with the other measures, $f_{S_{manh}}$ has the lowest calculation costs.

2.5 Linguistic Dependency Types

headed, no external (abstract) nodes

AB: add content!!!

2.5.1 Dependency Types (Relations)

universal dependencies project

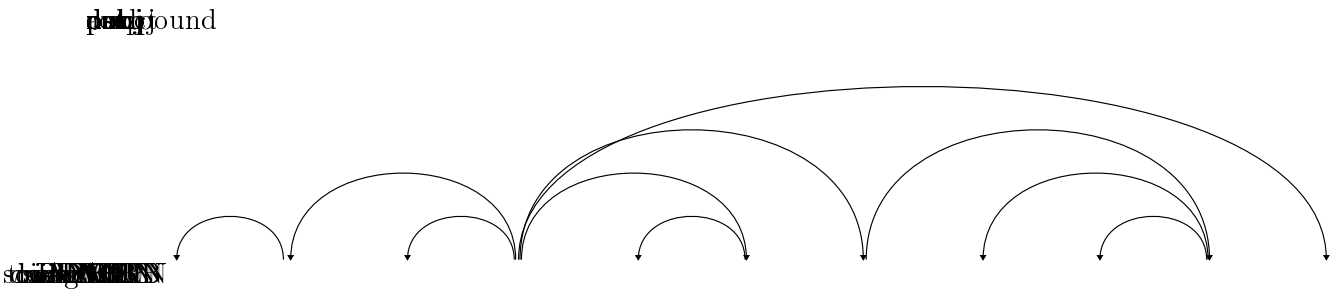


Figure 2: Dependency parse tree

3 Related work

survey (**wang_comparison_2017**);

survey on bi-gram composition (psych. ling. mot.) (**mitchell_composition_2010**) (e.g. semantic priming);

survey on bi-gram comp (sum, mul, RecNN) (**blacoe_comparison_2012**) (avg better)
German compounds (bi-gram) (**dima_reverse-engineering_2015**);
Skip-thought (**kiros_skip-thought_2015**);
RecRNN w/ parse structure (**socher_dynamic_2011**; **socher_semantic_2012**; **socher_recursive_2013**
tai_improved_2015; **wieting_paraphrase_2015**);
RecRNN (deep) (**irsoy_deep_2014**);
RecRNN w/o predefined structure (**zhao_self-adaptive_2015**; **chen_sentence_2015**);
doc2vec (**le_distributed_2014**; **lau_empirical_2016**);
CNN on bag of words (**kalchbrenner_convolutional_2014**);
CNN (**kim_convolutional_2014**; **hu_convolutional_2014**; **yin_convolutional_2015**;
he_multi-perspective_2015);
bi-LSTM + char-embeddings (**ling_finding_2015**);
multi-LSTM (**liu_multi-timescale_2015**);
deep avg (DAN) (**iyyer_deep_2015**);
feature-weighted avg (FCT) (**yu_learning_2015**);
functional composition (ling motivated) (**baroni_frege_2014**; **paperno_practical_2014**);
intra- vs extra-sentential context (ling motivated) (**polajnar_exploration_2015**);
para + doc embeddings (fixed hierarchical LSTM, auto-encoder) (**li_hierarchical_2015**);
multilingual (**hermann_multilingual_2014**);

AVG vs LSTM on paraphrase (**wieting_towards_2015**) impact of word order (**pham_sentence_2013**)
traditional approaches;
SemEval-2016 Task 1: STS (**agirre_semeval-2016_2016**);
SemEval-2017 Task 1: STS (**cer_semeval-2017_2017**);

siamese RNN + manhatten (simple) (**mueller_siamese_2016**);

(**habernal_exploiting_2015**); (**boltuzic_identifying_2015**); (**misra_measuring_2016**);

4 Model architecture and training

This chapter describes the architecture of the neural model used to calculate a similarity score for a given sentence tuple and how this model is trained.

4.1 Model architecture

The model consists of an embedding unit and a similarity function. The embedding unit translates a sequence of tokens into a single embedding vector and is applied to the two input sentences. It calculates the sentence embedding by gathering the individual embeddings for the contained tokens, eventually enhancing them with additional information, and applying a composition function. The two resulting embeddings are fed into the similarity function that produces the similarity score.

Figure shows this architecture. In the following, we describe the respective modules. AB: add figure

4.1.1 Token Embeddings

We use 300 dimensional GloVe embeddings (**pennington_glove:_2014**) to embed individual word tokens. AB:explain why To evaluate the influence of dependency parse edge types, we concatenate the embedding with the one-hot encoded edge type. There are 40 different edge types, resulting in 340 dimensional vectors as input for the composition function. When disabling this feature, we set these 40 entries to zero. The embedding weights are not optimized during training.

4.1.2 Composition function

In this work, we compare two different settings: Using (1) averaging (AVG) or (2) Long Short-Term Memory (LSTM) as composition function. In the AVG case, we apply one fully connected layer (FC) with *tanh* as activation function to every token embedding. Then, averaging the resulting vectors gives the sentence embedding. In the LSTM setting, the token embeddings are fed directly into a LSTM layer whose output is used as sentence embedding. The two settings are visualized in Figure . AB: add figure

To achieve comparability between the two composition functions, we control the amount of effectively trainable parameters in each of them. Depending on whether dependency parse information is used, we use different sizes for the *tanh* FC in the AVG case and the inner state of the LSTM to fix the amount of effective parameters to approximately 111000. Table 1 shows the exact amounts and sizes of the elements.

	AVG (FC size)	LSTM (state size)
w/ dependency edge	110840 (326)	111248 (68)
w/o dependency edge	111000 (370)	111000 (74)

Table 1: Amounts of trainable parameters and sizes of composition function elements

4.1.3 Similarity calculation

We use the cosine similarity to calculate the similarity score between the two sentence embeddings as described in section 2.4.5:

$$f_{SIM_{cos}}(a, b) = \frac{a \cdot b}{\|a\|_2 \|b\|_2} \tag{40}$$

4.1.4 Baseline model

As baseline of comparison we calculated TF-IDF based similarity scores in the following manner. We parse and lemmatize each sentence and filter for verbs, nouns and adjectives according to POS tags. Then, we embed each sentence with TF-IDF as described in section 2.4.2 and apply the similarity measure as usual.

4.2 Training and Implementation

We train the model via batched back-propagation with the MSE loss function described in section 2.3.1 and apply ADAM (**kingma_adam:_2014**) as optimizer. We use a 4:1 train/dev split for early stopping. The training is terminated when the score on the development data set does not increase anymore regarding a smoothing window covering the last 25 epochs.

The model is implemented with the TensorFlow framework (**abadi_tensorflow:_2016**). TensorFlow allows to define and to execute arbitrary dataflow graphs efficiently on different devices as CPUs or GPUs. For tokenization and dependency parsing we use spaCy[15] which is a fast and still accurate[16] NLP framework.

[15] see https://spacy.io/

[16] see **choi_it_2015** for a comparative study

5 Experiments and Evaluation

We conduct several experiments for all four settings to evaluate the influence of order awareness and availability of edge type information to the similarity prediction performance. In the following, we describe the dataset and the hyperparameter setting. Finally, we present the results and an analysis of errors.

5.1 Dataset

We use the SICK corpus[17] (**marelli_sick_2014**) for model training and evaluation. The corpus consists of about approximately 10.000 pairs of English sentences based on the 8K ImageFlickr data set[18] (**hodosh_framing_2013**) and the SemEval 2012 STS MSR-Video Description data set[19] (**agirre_semeval-2012_2012**) that contain sentences describing the same picture or video. These sentences were linguistically normalized, so Named Entities and complex verb constructions are replaced or subordinates are turned into coordinates, for instance. Furthermore, they are manually expanded to generate candidate pairs that were labeled by 5 annotators with one to five point relatedness ratings that are averaged to produce a relatedness score. We rescaled the scores into the interval $[0.0, 1.0]$ to let them fit into our definition of a similarity measure. Table 2 shows some examples of the dataset. The mean of the score for both train and test set is approximately 0.63. The sentences have an average length of 9.6 words and contain 2408 different token types.

AB:explain why SICK

5.2 Training and Hyperparameters

We train the model in batches of 100 examples. The ADAM optimizer is initialized with a learning rate of 0.003. We apply gradient clipping by global norm as specified in **pascanu_difficulty_2012** with a threshold of 5.0 and use dropout (**srivastava_dropout:_2014**) with a keep probability of 0.8 at the LSTM and the FC in the AVG case to prevent from overfitting. We used a grid search at the train/dev set to find this parameter configuration. We applied 5-fold cross validation with regard to the train/dev split for every setting and repeated each experiment 10 times to achieve robust results.

[17] http://alt.qcri.org/semeval2014/task1/index.php?id=data-and-tools

[18] http://nlp.cs.illinois.edu/HockenmaierGroup/data.html

[19] https://www.cs.york.ac.uk/semeval-2012/task6/index.php%3Fid=data.html

sentence A	sentence B	score
A man in a shirt dyed purple is looking at a man in a black shirt who is doing a funny face	A man in a shirt dyed purple is looking at a man in a black shirt who is doing a face which looks funny	1.0
A group of kids is playing in a yard and an old man is standing in the background	A group of boys in a yard is playing and a man is standing in the background	0.875
A brown dog is attacking another animal in front of the man in pants	Two dogs are wrestling and hugging	0.55
A woman is chopping an onion	A woman is washing her feet	0.0

Table 2: Example sentence pairs from SICK corpus with rescaled relatedness score

5.3 Results

We measured performance using the Pearson correlation coefficient and MSE between the predictions and gold scores. The mean Pearson correlation of all runs is about 0.839 with a standard deviation (std) of 0.002 and the mean MSE is 0.024 with std of 0.004. The TF-IDF baseline produces a Pearson score of 0.619 and a MSE of 0.082. All scores averaged over same parameter settings are shown in Table 3a. Figure 3b displays the MSE and Pearson's r results for all parameter combinations as box plots[20]. Note, that the y-axis is inverted for all box plots visualizing MSE scores to show better performing scores above worse ones and provide comparability with plots of Pearson scores.

Enabling the feature **order aware** results in a significant overall performance gain of 0.75% in means of MSE and a performance decrease of −0.21% with regard to Pearson's r ($p < 0.0001$ for two-tailed t-test in both cases). Adding **dependency edge types** improves AB:ref? the overall MSE by 0.05% and Pearson's r increases by 0.24%, whereby the former is not significant (p=0.336), but the later is ($p < 0.0001$). Table 3b shows the specific scores.

5.3.1 Relation of Order awareness and Dependency edge types

5.3.2 Qualitative error analysis

[20] The boxes of all box plots show the interquartile range (IQR), whiskers mark the $[1.5 \times \text{IQR}, 3 \times \text{IQR}]$ range and notches the 10000× bootstrapped confidence intervals.

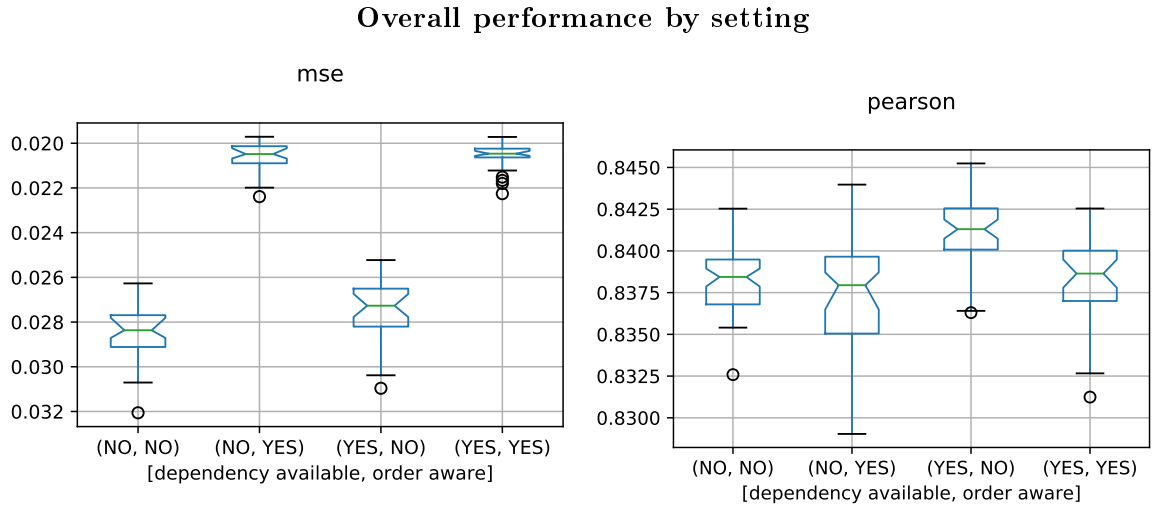
dependency edge	order aware	mse	pearson
NO	NO	0.0284	0.8382
NO	YES	0.0206	0.8375
YES	NO	0.0274	0.8413
YES	YES	0.0205	0.8383
TFIDF		0.0823	0.6189

(a) scores aggregated by setting

parameter	enabled	mse	pearson
dependency edge	NO	0.0279	0.8397
dependency edge	YES	0.0206	0.8379
order aware	NO	0.0245	0.8378
order aware	YES	0.0240	0.8398

(b) scores aggregated by individual parameter assignment

Table 3: Average MSE and Pearson's r scores



(a) MSE for the different settings (inverted y-axis).

(b) Pearson's r for the different settings.

6 Discussion and future work

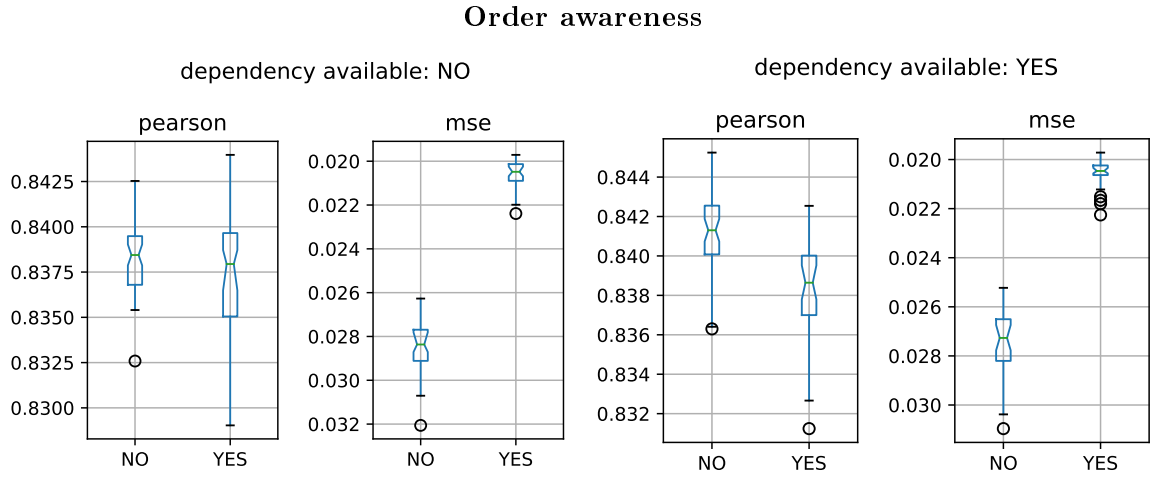


Figure 4: Impact of parameter *order aware* conditioned by parameter *dependency available*.

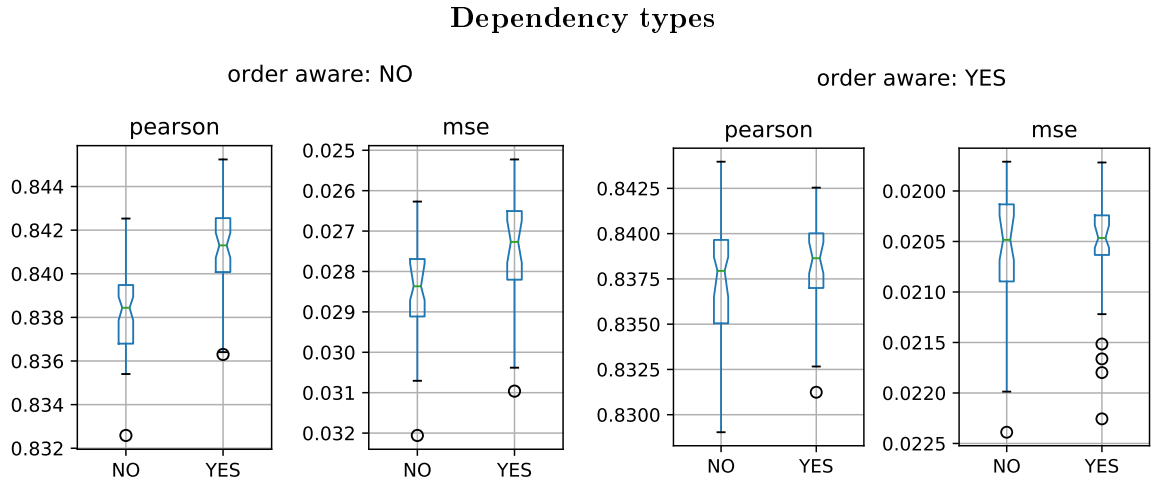


Figure 5: Impact of parameter *dependency available* conditioned by parameter *order aware*.

7 Conclusion

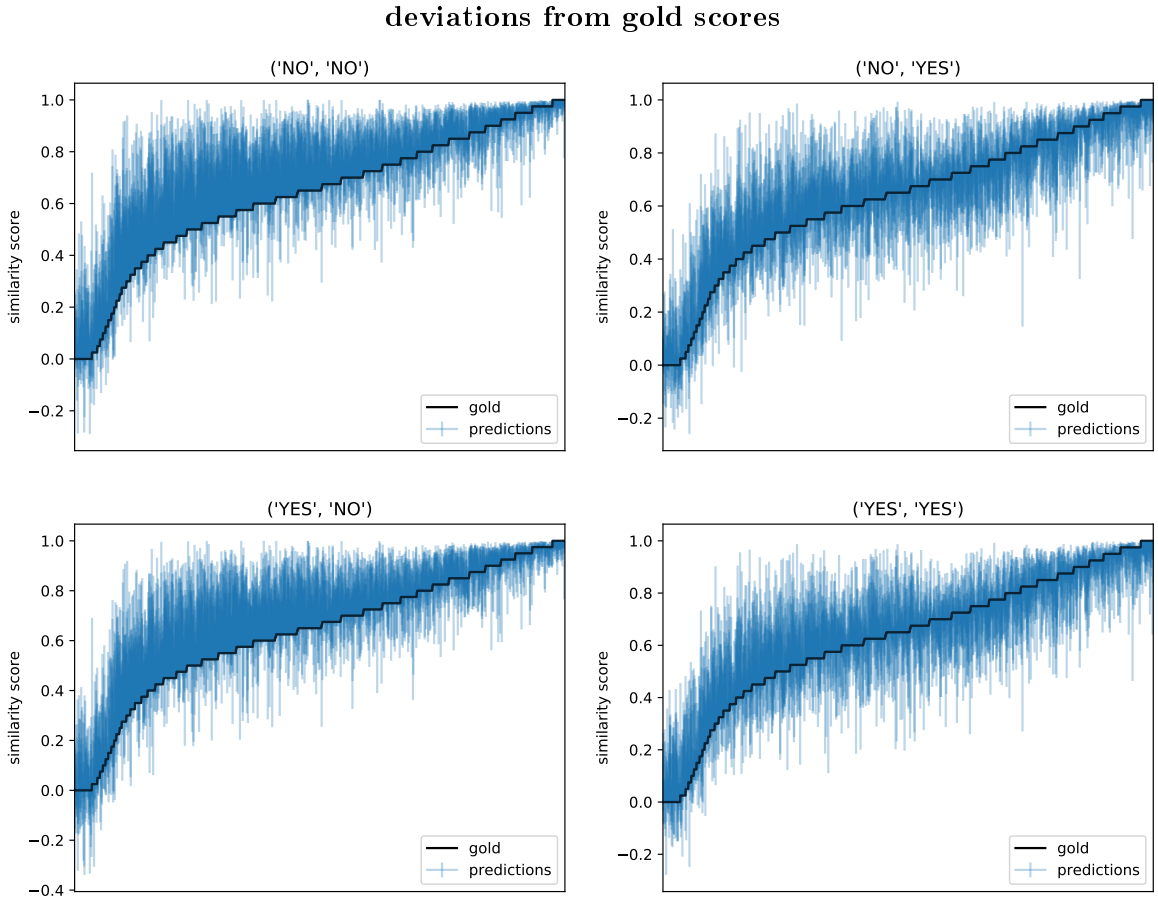


Figure 6: Deviations of the predictions from gold similarity scores by setting (<dependency available>, <order aware>)

Declaration of Authorship

I hereby confirm that I have authored this Student Research Paper independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

Berlin, December 27, 2017

Arne Binder