Supervised and Unsupervised Methods for Learning Representations of Linguistic Units

Sascha Rothe

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

München, den 12. Dezember 2016

Supervised and Unsupervised Methods for Learning Representations of Linguistic Units

Sascha Rothe

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

Erstgutachter:	Prof. Dr. Hinrich Schütze
Zweitgutachter:	Prof. Dr. Roberto Navigli
Drittgutachterin:	Prof. Dr. Vera Demberg
Tag der Einreichung:	12. Dezember 2016
Tag der Disputation:	14. Juni 2017

Eidesstattliche Versicherung
(Siehe Promotionsordnung vom 12.07.11, *x* 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig ohne unerlaubte Beihilfe angefertigt ist.

München, den 12.12.2016

Sascha Rothe

Abstract

Word representations, also called word embeddings, are generic representations, often high-dimensional vectors. They map the discrete space of words into a continuous vector space, which allows us to handle rare or even unseen events, e.g. by considering the nearest neighbors. Many Natural Language Processing tasks can be improved by word representations if we extend the task specific training data by the general knowledge incorporated in the word representations.

The first publication investigates a supervised, graph-based method to create word representations. This method leads to a graph-theoretic similarity measure, CoSimRank, with equivalent formalizations that show CoSimRank's close relationship to Personalized Page-Rank and SimRank. The new formalization is efficient because it can use the graph-based word representation to compute a single node similarity without having to compute the similarities of the entire graph. We also show how we can take advantage of fast matrix multiplication algorithms.

In the second publication, we use existing unsupervised methods for word representation learning and combine these with semantic resources by learning representations for non-word objects like synsets and entities. We also investigate improved word representations which incorporate the semantic information from the resource. The method is flexible in that it can take any word representations as input and does not need an additional training corpus. A sparse tensor formalization guarantees efficiency and parallelizability.

In the third publication, we introduce a method that learns an orthogonal transformation of the word representation space that focuses the information relevant for a task in an ultradense subspace of a dimensionality that is smaller by a factor of 100 than the original space. We use ultradense representations for a Lexicon Creation task in which words are annotated with three types of lexical information – sentiment, concreteness and frequency.

The final publication introduces a new calculus for the interpretable ultradense subspaces, including polarity, concreteness, frequency and part-of-speech (POS). The calculus supports operations like "$1 \cdot \mathit{hate} = \mathit{love}$" and "give me a neutral word for *greasy*" (i.e., *oleaginous*) and extends existing analogy computations like "$\mathit{king} - \mathit{man} + \mathit{woman} = \mathit{queen}$".

Zusammenfassung

Wortrepräsentationen, sogenannte Word Embeddings, sind generische Repräsentationen, meist hochdimensionale Vektoren. Sie bilden den diskreten Raum der Wörter in einen stetigen Vektorraum ab und erlauben uns, seltene oder ungesehene Ereignisse zu behandeln – zum Beispiel durch die Betrachtung der nächsten Nachbarn. Viele Probleme der Computerlinguistik können durch Wortrepräsentationen gelöst werden, indem wir spezifische Trainingsdaten um die allgemeinen Informationen erweitern, welche in den Wortrepräsentationen enthalten sind.

In der ersten Publikation untersuchen wir überwachte, graphenbasierte Methodenn um Wortrepräsentationen zu erzeugen. Diese Methoden führen zu einem graphenbasierten Ähnlichkeitsmaß, CoSimRank, für welches zwei äquivalente Formulierungen existieren, die sowohl die enge Beziehung zum personalisierten PageRank als auch zum SimRank zeigen. Die neue Formulierung kann einzelne Knotenähnlichkeiten effektiv berechnen, da graphenbasierte Wortrepräsentationen benutzt werden können.

In der zweiten Publikation verwenden wir existierende Wortrepräsentationen und kombinieren diese mit semantischen Ressourcen, indem wir Repräsentationen für Objekte lernen, welche keine Wörter sind, wie zum Beispiel Synsets und Entitäten. Die Flexibilität unserer Methode zeichnet sich dadurch aus, dass wir beliebige Wortrepräsentationen als Eingabe verwenden können und keinen zusätzlichen Trainingskorpus benötigen.

In der dritten Publikation stellen wir eine Methode vor, die eine Orthogonaltransformation des Vektorraums der Wortrepräsentationen lernt. Diese Transformation fokussiert relevante Informationen in einen ultra-kompakten Untervektorraum. Wir benutzen die ultra-kompakten Repräsentationen zur Erstellung von Wörterbüchern mit drei verschiedene Angaben – Stimmung, Konkretheit und Häufigkeit.

Die letzte Publikation präsentiert eine neue Rechenmethode für die interpretierbaren ultra-kompakten Untervektorräume – Stimmung, Konkretheit, Häufigkeit und Wortart. Diese Rechenmethode beinhaltet Operationen wie ” 1 *Hass* = *Liebe*” und ”neutrales Wort für *Winkeladvokat*” (d.h., *Anwalt*) und erweitert existierende Rechenmethoden, wie ”*Onkel* *Mann* + *Frau* = *Tante*”.

Contents

Publications and Declaration of Co-Authorship

Chapter 2

Chapter 2 corresponds to the following publication:

> Sascha Rothe and Hinrich Schütze; **CoSimRank: A Flexible & Efficient Graph-Theoretic Similarity Measure**; Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Baltimore, Maryland, USA, June, 2014) Volume 1: Long Papers, pages 1392–1402

I regularly discussed this work with my advisor, but I conceived of the original research contributions and performed implementation and evaluation. I wrote the initial draft of the article and did most of the subsequent corrections. My advisor assisted me in improving the draft.

Chapter 3

Chapter 3 corresponds to the following publication:

> Sascha Rothe and Hinrich Schütze; **AutoExtend: Extending Word Embeddings to Embeddings for Synsets and Lexemes**; Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Beijing, China, July, 2015) Volume 1: Long Papers, pages 1793–1803. IBM Best Student Paper Award

I regularly discussed this work with my advisor, but I conceived of the original research contributions and performed implementation and evaluation. I wrote the initial draft of the article and did most of the subsequent corrections. My advisor assisted me in improving the draft.

Chapter 4

Chapter 4 corresponds to the following publication:

> Sascha Rothe and Sebastian Ebert and Hinrich Schütze; **Ultradense Word Embeddings by Orthogonal Transformation**; Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics (San Diego, California, USA, June, 2016), pages 767–777

Section 4.4 "Text Polarity Classification" of Chapter 4 is Sebastian Ebert's work. I also regularly discussed this work with my coauthors. Apart from these explicitly declared exceptions, I conceived of the original research contributions and performed implementation and evaluation. I wrote the initial draft of the article and did most of the subsequent corrections. My coauthors assisted me in improving the draft.

Chapter 5

Chapter 5 corresponds to the following publication:

> Sascha Rothe and Hinrich Schütze; **Word Embedding Calculus in Meaningful Ultradense Subspaces**; Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Berlin, Germany, August, 2016) Volume 2: Short Papers, pages 512–517

I regularly discussed this work with my advisor, but I conceived of the original research contributions and performed implementation and evaluation. I wrote the initial draft of the article and did most of the subsequent corrections. My advisor assisted me in improving the draft.

Chapter 6

Chapter 6 corresponds to the following publication:

> Sascha Rothe and Hinrich Schütze; **AutoExtend: Combining Word Embeddings with Semantic Resources**; Computational Linguistics (forthcoming 2017)

I regularly discussed this work with my advisor, but I conceived of the original research contributions and performed implementation and evaluation. I wrote the initial draft of the article and did most of the subsequent corrections. My advisor assisted me in improving the draft.

This chapter is the draft of an article that I submitted in 2016 to the journal Computational Linguistics. A revised version of this chapter was accepted for publication on 2017-03-13. This chapter is an extended version of the conference paper reproduced as Chapter 3.

München, 12.12.2016

Sascha Rothe

Chapter 1

Introduction

Word representations, also called word embeddings, are representations of words as mathematical objects. They allow us to use words in Natural Language Processing, e.g., as input to neural networks. An additional benefit is that we can handle rare or even unseen events, e.g., by considering nearest neighbors. The formal definition of an embedding is often given as an injective and structure-preserving map $f : V \; ! \; U$ where V is the discrete space of words – the vocabulary.

A simple word embedding is a one-hot vector. In this case, we create a vocabulary of the n most frequent words. Typical values for n are between 10,000 and 100,000. The word embedding of word v is then given by the canonical unit vector $e_v \; 2 \; \mathsf{R}^{jVj}$. We can now use these embeddings in neural networks or other applications.

However, one-hot vectors do not have the advantage of generalization. For example, given the phrase *the fast car*, we can learn that a *car* is an object that moves, but this phrase would tell us nothing about the word *cars*. As all embeddings have the same distance to each other it is impossible to incorporate information of the nearest neighbors. In this thesis, we will use two methods to overcome this issue.

1. The first approach maps high dimensional sparse one-hot vectors to low dimensional dense vectors. This forces the system to generalize. For example, the system will learn that *car* and *cars* are very similar as they occur in similar contexts. Because the generalization is automatically learned from a training corpus, we call this method unsupervised.

2. The second approach generates a graph, where words are nodes and edges are constructed by linguistic relationships like direct object (e.g., *drive-car*) or hypernymy (e.g., *vehicle-car*). These relationships have to be defined and we therefore call this method supervised. After defining the types of relations we can manually create the graph using linguists or crowdsourcing. We can also learn the edges and nodes by scanning through a large

annotated corpus.

In Natural Language Processing, the mathematical definition we just gave is not always applied strictly. For example, there is often no formal constraint that two different words cannot be mapped to the same vector. This would violate the injectivity. However, injectivity is still a useful property, as it guarantees that we are still able to distinguish words. This also includes synonyms, where we want to have the resulting objects to be similar but not identical. The property of structure preservation is not clearly defined. For example, sometimes we want antonyms to be similar and sometimes we want antonyms to be inverses of each other.

In this introduction, we will briefly cover unsupervised embedding learning methods, some basics of graph theory to handle supervised embeddings and the evaluation of word embeddings. Finally, we will introduce task specific embeddings which can be seen as building blocks of deep neural networks, a subject currently undergoing intense study.

1.1 Unsupervised Training Methods

Among the earliest work on distributed word representations, usually called word embeddings today, was (Rumelhart et al., 1988). Non-neural-network techniques that create low-dimensional word representations also have been used widely, including singular value decomposition (SVD) (Deerwester et al., 1990; Schütze, 1992) and random indexing (Kanerva, 1998, 2009). There has been a resurgence of work on embeddings recently (e.g., Bengio et al. (2003a), Mnih and Hinton (2007a), Collobert et al. (2011)), including methods that are SVD-based (Levy and Goldberg, 2014; Stratos et al., 2015). We will cover the three most frequently used word embedding methods in this section, namely Continuous Bag of Word (CBOW), Skip-gram and Global Vectors (GloVe).

1.1.1 Continuous Bag of Words

The Continuous Bag of Word method, often just called CBOW, was introduced together with Skip-gram (see next subsection) by Mikolov et al. (2013a). Both methods are part of a highly optimized toolkit called word2vec (Mikolov et al., 2013c) and are similar to an autoencoder. First, we have to define a vocabulary V together with an index function v that maps from a position in the training corpus to the index of the corresponding word in the vocabulary. We are able to set the size of the vocabulary quite high. It typically includes 100,000 to 1,000,000 words. Out of vocabulary words (OOVs) can either be replaced by a special token and mapped to the corresponding index or simply be removed from the training

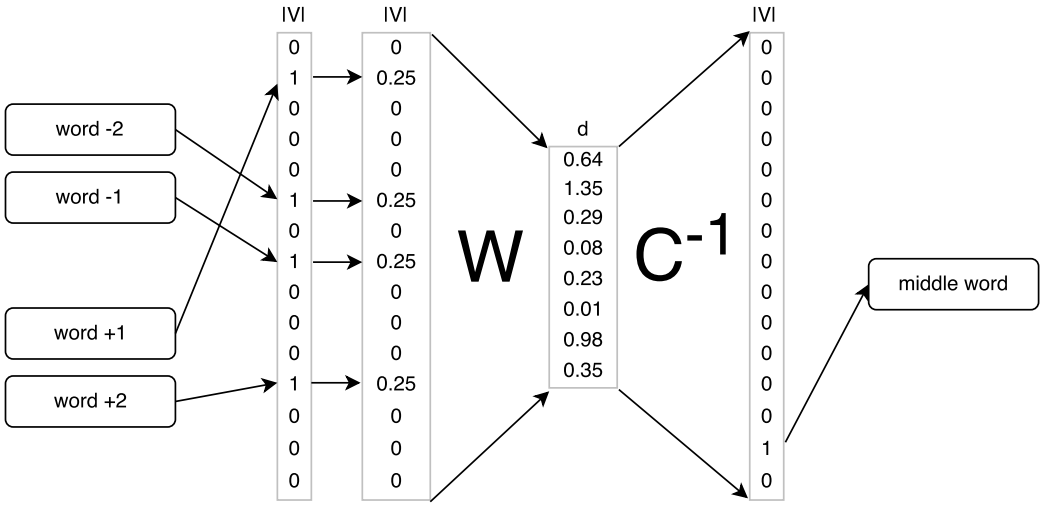

Figure 1.1 – *The CBOW algorithm takes the entire context to predict the word in the middle word.*

corpus. The method also defines a window size k around a word at position t. CBOW then uses all words at position $t + k^{\emptyset}$, with $0 < jk^{\emptyset}j \quad k$, i.e., all words inside the window excluding the middle word, to predict the word in the middle.

Let $e_{v(t)} \; 2 \; \mathsf{R}^{jVj}$ be a canonical unit vector of length jVj where jVj is the number of words in our vocabulary. This vector is also called the one-hot representation of the word at position t. CBOW learns two matrices $W, C \; 2 \; \mathsf{R}^{d \;\; jVj}$ where d is the dimensionality of the embedding space we want to learn. This parameter has to be defined prior to the training. Typical values for the dimensionality of the embeddings space are 50, 100, 300 or 500. We can now state the learning objective of CBOW as follows:

$$\underset{W;C}{\text{argmin}} \left\| \left(\sum_{0<jk^{\emptyset}j \;\; k} \frac{1}{2k} (W e_{v(t+k^{\emptyset})}) \right) \quad C e_{v(t)} \right\| \qquad ,8t \tag{1.1}$$

This is illustrated in Figure 1.1. As all neural networks, the training is performed using the gradient descent algorithm. In contrast to most neural network implementations which use batch gradient descent, a stochastic gradient descent is used. However, the training can be parallelized to multiple GPUs. A corpus of several billion words can therefore be processed in under one day. A word embedding of word v is given by We_v, i.e., the columns of W define the word embeddings. The values in C are discarded.

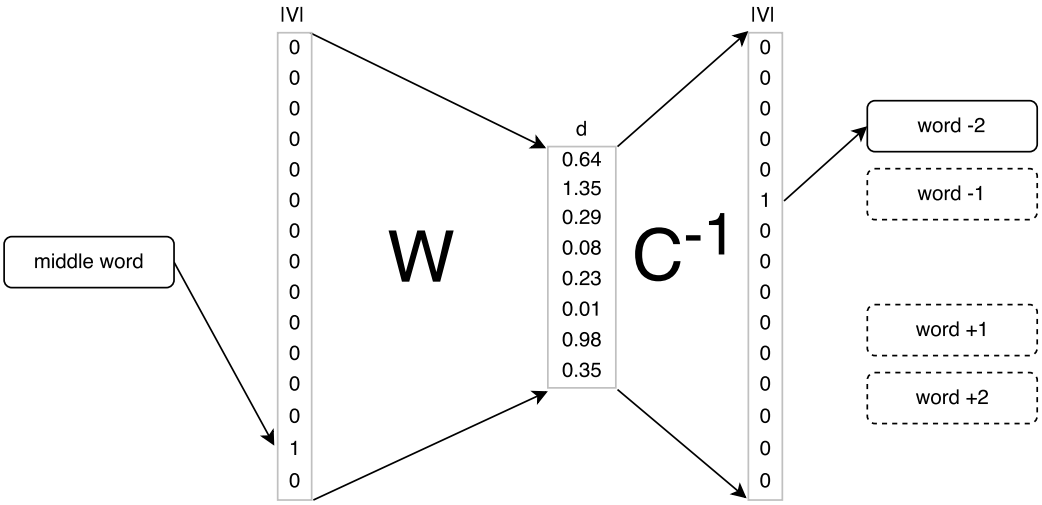

Figure 1.2 – *The Skip-gram algorithm iterates through all context words and pairs one at a time with the middle word.*

1.1.2 Skip-Gram

CBOW uses the context words to predict the word in the middle. Skip-gram vice versa uses the word in the middle to predict surrounding words at position $t + k^{0}$, with $0 < jk^{0}j \quad k$. The learning objective of Skip-gram (Figure 1.2) is given as follows:

$$\underset{W;C}{\operatorname{argmin}} \sum_{0<jk^{0}j \quad k} kWe_{v(t)} \quad Ce_{v(t+k^{0})}k \quad ,8t \tag{1.2}$$

Theoretically Eq. 1.1 and Eq. 1.2 are equivalent when W and C are switched. However, there are practical issues which further distinguish both methods:

1. The window size k is randomly sampled between 0 and k for each gradient descent step.

2. Whereas W is initialized with a uniform distribution, C is set to a zero matrix. This results in slightly shorter word embeddings for vectors in W. However, this effect is negligible for frequent words or if the training lasts for a lot of iterations.

3. CBOW performs one back propagation per middle word whereas Skip-gram performs one for every word in the window. Because of this CBOW is also notably faster than Skip-gram especially when the window size k is large.

4. To optimize the learning process, methods like hierarchical softmax or negative sampling are used. For details on these methods see (Mikolov et al., 2013a) and (Mikolov et al., 2013c)

1.1.3 Global Vectors

Global Vectors, often just called GloVe, are embeddings that were introduced by Pennington et al. (2014b) and aim to combine the advantages of matrix factorization and local context windows. For this, a co-occurrence matrix X is defined, with X_{ij} being the number of times word j occurs in the context of word i. It follows that the probability of word j occurring in the context of word i is given by:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}} \tag{1.3}$$

The idea of the model is that word embeddings should be based on the ratio of co-occurrence probabilities and not on the co-occurrence itself. That means if we want to learn something about the relationship of word i and j we don't look at P_{ij} but on P_{ik}/P_{jk} for all k. Further constraints on the model are linearity and homomorphy. This results in the final learning objective:

$$\underset{W;C}{\text{argmin}} k(We_i)^T Ce_j + b_i + b_j \quad \log(X_{ij})k \qquad ,8i,j \tag{1.4}$$

with b_i and b_j being bias terms. In this model, the context vectors are kept for a small boost in performance. The final word embedding is a combination of both vectors, i.e.:

$$(\lambda W + (1 \quad \lambda)C)e_v$$

1.2 Graph-Based Word Embeddings

The first publication in Chapter 2 covers graph-based embeddings. In this section, we will introduce some basics of graph theory and how graphs fit in our framework of word embeddings. So be $U = f1, 2, 3, \ldots, ng$ the set of nodes. The set of edges E is a subset of $U \quad U$. The pair $G = (U, E)$ is called graph. The edges can be defined by grammatical relationships, e.g., verb-object pairs or hypernymy. Graphs based on these definitions were created by Dorow et al. (2009) and called WordGraph. They used the following relationships for edges: adjectival modifier (adjective-noun), direct object (verb-object) and noun coordination (noun-noun). They were parsed from Wikipedia for two languages, English and German. When a relationship is above a certain threshold an edge was created.

The graphs also included node types, defined by POS; edge types, defined by the different grammatical relationships; and edge weights, defined by the logarithm of the log-likelihood score.

Another graph of words is WordNet by Fellbaum (1998). This graph was manually created and the edges are defined by conceptual-semantic and lexical relations. These include hypernymy (superset), hyponymy (subset), meronymy (part-of), holonymy (opposite of meronymy). Additionally, words are grouped into synsets, so we can easily construct a synonym relationships between words. We will use WordNet in this thesis together with GermaNet by Hamp and Feldweg (1997), which is a similar resource for German, and Freebase (Bollacker et al., 2008), which is a graph-like resource with an emphasis on entities.

To fit in the framework of word embeddings we can simply define the embedding map $f : V \; ! \; U$. However, a mapping to nodes is only slightly useful as most evaluations and applications, e.g., neural networks, rely on real-valued vectors. To overcome this limitation we will propose a similarity measure based on the Personalized PageRank. The Personalized PageRank assigns a vector to each node. So we can also use this method in other evaluations or applications. The new method is presented in Chapter 2 but we will introduce the basics – PageRank and Personalized PageRank – in following two subsections. We will put an emphasis on the mathematical foundations, i.e., existence and uniqueness of solutions.

1.2.1 PageRank

PageRank was developed 1997 by Lawrence Page and Sergei Brin and registered for patent. The name stands for his inventor as well as its primary usage, i.e., ranking web pages. The computation of the PageRank can be done analytically by solving the eigenvalue problem of a matrix similar to an adjacency matrix. To overcome time and space complexity an iterative computation is used in practice. The PageRank of node i at iteration $k + 1$ is defined as:

$$P_i^{(k+1)} = \sum_{(j,i) \in E} \frac{P_j^{(k)}}{g_{out}(j)} \tag{1.5}$$

where $g_{out}(u) = |\{x \in U | (u, x) \in E\}|$ is the outdegree of node u. The computation is started with $P_i^0 = \frac{1}{n}$ for all $i \in E$. The intuitive idea is that pages (i.e., nodes) pass weight through links (i.e., edges) to other web pages. This means that a web page is important if it has important web pages linking to it. Another interpretation is the random surfer model, where an internet surfer randomly clicks links on web pages. The PageRank then corresponds to the probability of that surfer being on a certain page. We can rewrite Eq. 1.5 into a matrix formalization:

$$p^{(k+1)} = Ap^{(k)} \tag{1.6}$$

$$p^{(0)} = \frac{1}{n} j_n = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)^T \tag{1.7}$$

The matrix A is the column normalized adjacency matrix and j_n is a vector of ones and length n. If a node has no outgoing edges we set the corresponding column in A to $\frac{1}{n} j_n$. To ensure fast convergence A must also be strictly positive, which is why an additional damping factor d is introduced.

$$G = dA + (1 \quad d) \left(\frac{1}{n} J_n\right) \tag{1.8}$$

with J_n being a square matrix of ones and size n. The matrix G is also called Google matrix. The iterative computation is then given by:

$$p^{(k+1)} = Gp^{(k)} \tag{1.9}$$

$$p^{(0)} = \frac{1}{n} j_n = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)^T \tag{1.10}$$

with G being column normalized and strictly positive the following statements hold true (Perron, 1907):

1. The spectral radius $\rho(A) > 0$
2. $\rho(A)$ is a single eigenvalue
3. There exists a vector p so that $Ap = \rho(A)p$, $p > 0$ and $kpk_1 = 1$
4. A does not have other eigenvectors $v > 0$ (except for multiples of p)

It is also known that the iteration in Eq. 1.6 converges to the eigenvector corresponding to the dominant eigenvalue, i.e., the spectral radius (Mises and Pollaczek-Geiringer, 1929). We therefore have a unique and well-defined solution p for the iterative computation of PageRank. The vector p is also called Perron vector.

1.2.2 Personalized PageRank

The PageRank assigns each node in the graph a positive value. We want to have a more discrete variable and define the Personalized PageRank p_i. We change Eq. 1.6 as follows:

$$p_i^{(k+1)} = dAp_i^{(k)} + (1 \quad d)e_i \tag{1.11}$$

$$p_i^{(0)} = e_i \tag{1.12}$$

with e_i being the canonical unit vector. Note that this would result in a non-negative but not strictly positive matrix for the vector iteration. However, we can postulate the same conclusions as in the previous section if A is irreducible (Frobenius, 1912). The adjacency matrix of a graph G is irreducible if and only if G is connected, i.e., there is a path between every pair of nodes. WordGraph is undirected and in WordNet, every edge type also has an inverse type, e.g., hypernymy and hyponymy. This condition can be considered to be fulfilled for the graphs we are using.

1.2.3 SimRank

In the previous subsection, we described how to map nodes in a graph to a real-valued vector. These vectors can already be used to compute the similarity between nodes, by using an arbitrary metric of the vector space. We will cover this in more detail in the next section. However, we can also apply a similarity measure directly to the graph. The most used methods for this is SimRank developed by Jeh and Widom (2002) and the idea is similar to PageRank. The intuition that important nodes are connected to important nodes translates to *nodes are similar if their neighbors are similar*. We can write the iterative matrix formalization as follows:

$$S^{(k+1)} = dAS^{(k)}A^T \tag{1.13}$$

$$S^{(0)} = E \tag{1.14}$$

with E being the identity matrix and d being a decay factor similar to the decay factor used in Eq. 1.8. Obviously, this computation converges to zero if $d < 1$. To prevent this the diagonal of S^k is set to one after each iteration. It is now easy to see that SimRank converges to a nontrivial matrix. For a proof please see (Jeh and Widom, 2002) or chapter 2 of this thesis where we prove the convergence of CoSimRank from which the convergence of SimRank immediately follows. The downside of resetting the diagonal after each iteration is that we do not have a local formulation of SimRank, i.e., we always have to compute all similarities in the graph. We will overcome this by introducing CoSimRank.

1.3 Intrinsic Evaluation

As we already mentioned word embeddings shall preserve the structure of the word space. There is no clear definition of the structure of a word space but we might want to preserve similarity, i.e., similar words are mapped to similar word embeddings. This is the basis for the first popular intrinsic task to evaluate word embeddings (Miller and Charles, 1991). The second evaluation, namely Word Analogy, tests the preservation of relations (Mikolov et al., 2013a). We will cover these two evaluation methods in the following subsections. Other evaluations include Word Categorization, i.e., clustering the words into different categories, and selectional preference. Here the task is to determine if a verb-noun pair is a verb-object or a verb-subject pair. For example, is it *drive a car* or *a car drives*. Another task is to find an outlier among a group of words, e.g., *apple*, *banana*, *moon* and *peach*.

1.3.1 Word Similarity

In this task, a set of word pairs is given together with human-assigned similarity scores. Word Similarity test sets for the English language include MC (Miller and Charles, 1991), MEN (Bruni et al., 2014), RG (Rubenstein and Goodenough, 1965), SIMLEX (Hill et al., 2014), RW (Luong et al., 2013) and WordSim-353 (Finkelstein et al., 2001). Table 1.1 gives some examples. The human assigned score is usually based on several annotators. To evaluate the word embeddings we have to define a similarity measure in the embedding space. A frequent choice is the cosine similarity defined by:

$$\mathrm{sim}_{\cos}(v, w) = \frac{v^T w}{\|v\|\|w\|} \tag{1.15}$$

or euclidean similarity defined by:

$$\mathrm{sim}_{L^2}(v, w) = \frac{1}{1 + \|v - w\|} \tag{1.16}$$

After this, we compute a correlation between the human scores and the word embedding based scores. The most basic correlation is Pearson's r (Galton, 1888).

$$\rho_p(X, Y) = \frac{\mathrm{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{1.17}$$

The downside of the Pearson's r is, that it measures the linear dependency between two variables. This is an unnecessary constraint to our models and we therefore use rank correlations. A lot of publications report the Spearman's *rho*

word 1	word 2	similarity
coast	shore	9.10
money	dollar	8.42
tiger	cat	7.35
psychology	clinic	6.58
energy	crisis	5.94
word	similarity	4.75
situation	isolation	3.88
opera	industry	2.63
stock	egg	1.81
professor	cucumber	0.31

Table 1.1 – *Examples of word similarity test pairs, taken out of WordSim-353.*

(Spearman, 1904) which is defined by computing Pearson's *r* of the ranked variables:

$$\rho_s(X,Y) = \rho_p(\mathrm{rg}_X, \mathrm{rg}_Y) = \frac{\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)}{\sigma_{\mathrm{rg}_X}\sigma_{\mathrm{rg}_Y}} \tag{1.18}$$

where *rg* converts scores to ranks, *cov* is the covariance and σ is the standard deviation. Sometimes also Kendall's tau is reported (Kendall, 1948). For this, we consider all pairs of observations (x_i, y_i) and (x_j, y_j). With no loss of generality, we set $x_i < x_j$. We call a pair concordant if $y_i < y_j$ and discordant if $y_i > y_j$. If $x_i = x_j$ or $y_i = y_j$ the pair is called neither concordant nor discordant. With n being the number of observations, Kendall's tau is defined as follows:

$$\tau(X,Y) = \frac{(\text{\# of concordant pairs}) \quad (\text{\# of discordant pairs})}{n(n \quad 1)/2} \tag{1.19}$$

A high correlation indicates high-quality word embeddings. While we already discard absolute values by using a rank correlation this still yields problems. For example, it is obvious that the word pair *car-bus* shall get a higher similarity than *car-table*. However, depending on the context we want to investigate, it is unclear if *bus-train* (both a transportation vehicle) or *bus-truck* (both a big vehicle driving on a road) shall get a higher similarity. Further problems were highlighted by Faruqui et al. (2016).

1.3.2 Word Analogy

For this task, two pairs of words were grouped together. Each pair is related through the same relationship, e.g., country-capital or adjective-comparative. One

word 1A	word 1B	word 2A	word 2B
Paris	France	Berlin	Germany
Europe	euro	USA	dollar
brother	sister	policeman	policewoman
amazing	amazingly	obvious	obviously
slow	slower	cold	colder
fast	fastest	tasty	tastiest
discover	discovering	swim	swimming
China	Chinese	Mexico	Mexican
predicting	predicted	going	went
man	men	eye	eyes
work	works	go	goes

Table 1.2 – *Examples of word analogy test cases. The upper part is of semantic nature whereas the lower part is of syntactic nature.*

out of the four words is hidden and the task is to predict the hidden word. This can be solved by computing an offset vector between one pair and adding it to the single word. A well-known example is the set king-men and *queen-women*. The word *queen* can be found by solving:

$$f(\text{king}) \quad f(\text{men}) + f(\text{women}) = f(\text{queen}) \tag{1.20}$$

whereas f is the word embedding function that maps a word to a word embedding. Note that we either compute the offset vector between *men* and *women* or between *men* and *king*. Both variants lead to the same equation. More examples can be seen in Table 1.2.

After computing the left side of Eq. 1.20 we have to find the nearest neighbors to the result. Cosine distance or euclidean distance is used. If using the euclidean distance this would result in the following optimization problem:

$$\underset{v \in V}{\text{argmin}} \quad \|f(a) \quad f(b) + f(c) \quad f(v)\|$$

To simplify the notation we will use v for a word as well as for the corresponding word embedding $f(v)$. Some implementations normalize all word embeddings first and compute the cosine similarity. This is equivalent to the following optimization problem:

$$\underset{v \in V}{\text{argmax}} \quad \text{sim}_{\cos}(v, a) \quad \text{sim}_{\cos}(v, b) + \text{sim}_{\cos}(v, c)$$

In our example, this means that we seek a word which is similar to *king* and

woman but dissimilar to *men*. An alternative to the computation of the offset vector was proposed by Levy et al. (2014).

$$\underset{v2V}{\text{argmax}} \quad \frac{\text{sim}_{\cos}(v, a)\,\text{sim}_{\cos}(v, c)}{\text{sim}_{\cos}(v, b)}$$

The method is designed to increase the differences between small quantities and reducing the differences between larger ones. Extra care has to be taken to avoid divisions by zero, e.g., by mapping all cosine similarities to $[0, 1]$ and adding ϵ to the division.

Table 1.3 gives examples of the nearest neighbors using the cosine distance. The final evaluation score is simply the accuracy where the correct word was found in position one. A higher accuracy indicates higher quality word embeddings. Current state-of-the-art models achieve an accuracy above 70%. Sometimes also acc@10 is reported, i.e., how often the correct word was found in the list of top 10 candidates. If the list is of arbitrary size or there is more than one correct answer, precision, recall and F1 are also common measures. As we have to find the nearest neighbor the computation of Word Analogy is slightly more expensive than Word Similarity, where we just have to compute a cosine similarity. Word Analogy tests are therefore often reduced to most frequent words, e.g., top 30,000.

Both evaluations, Word Similarity and Word Analogy, are appealing because they are easy to implement and inexpensive to compute. They are also usually a good first indicator. However, they may not always be consistent with performance on downstream tasks (Schnabel et al., 2015). Because of this extrinsic evaluations measure the performance on a certain task, e.g., POS Tagging or Sentiment Analysis (Nayak et al., 2016). On the other side, if word embeddings are optimized for a certain task the embedding training can also be integrated into the learning algorithm. This produces task specific embeddings, which we will cover in the next section.

1.4 Task Specific Embeddings

We already showed how to learn word embeddings on unlabeled data. These embeddings were thought to be general and not specific to any task. The idea of these embeddings is that unlabeled corpora are widely available and the information contained in them can be transferred to tasks where less training data is available. However, there are also tasks where we do have a lot of training data, e.g., machine translation for popular languages or question answering. For these applications, we can also learn task specific embeddings on the fly while training an end-to-end system. In this section, we will briefly discuss word-, word-level-,

query	nearest neighbors
Paris	Heidi, London, France, Dubai, Samuel, Hilton, Rome, Toronto, Las Vegas, Lindsay Lohan
Europe	European, Germany, Spain, England, America, USA, France, Greece, Italy, Sweden, Africa
policeman	policewomen, policemen, constable, cop, taxi driver, soldier, sergeant, police, patrolman
amazing	incredible, awesome, unbelievable, fantastic, phenomenal, astounding, wonderful, remarkable, marvelous, fabulous
fast	quick, rapidly, quickly, slow, faster, rapid, speed
lightning + thunder	thunderstorm, rain, heavy rain, roar, blazing
Friday + Sunday	Saturday, Thursday, Monday, Wednesday, Tuesday, afternoon
Berlin - Germany + France	Paris, French, Brussels, Rome, Toulouse
king - man + woman	queen, monarch, princess, prince, kings, monarchy
slower - slow + cold	colder, warmer, cooler, chilly, frigid, chill

Table 1.3 – *Examples for the nearest neighbors of word embedding expressions. Note that nearest neighbors include synonyms and antonyms. Ambiguous words include related words of both meanings.*

sentence- and document-embeddings for machine comprehension, i.e., where we have a document a question and an answer.

Figure 1.3 gives an example of how such advanced embeddings can be used in a question-answering system. We end this introduction with these advanced embeddings and this architecture to give an outlook on where the area of natural language processing is headed that this thesis makes a contribution on.

1.4.1 Word Embeddings

Word Embeddings can be learned in a downstream task by simply connecting the one-hot vector corresponding to the word via a matrix multiplication to the input of the neural network. Or equivalently we use a lookup table which stores a vector for each word in the vocabulary. To use pre-trained word embedding, like CBOW, we can initialize this lookup table with the pre-trained word embeddings. Often both approaches are combined by initializing the lookup table with pre-trained word embeddings but updating them via the training process.

1.4.2 Word-level Embeddings

In contrast to word embeddings, word-level embeddings are not unique for all occurrences of one word. Instead, they also depend on the context. In terms of precise notation, we have to be careful as this embedding is not a function with the vocabulary V as the domain but only a relation between vocabulary and embedding space or alternative a function $f : V^l \; ! \; \mathsf{R}^d$. The most common approaches to generating these embeddings are Long-Short-Term-Memories (LSTMs) and Convolutional Neural Networks (CNNs). LSTMs can, in theory, have an unlimited context length l but in practice, the context is truncated to the length of an average sentence or document. We will focus on an LSTM implementation by Graves (2013). This architecture is based on the work by Hochreiter and Schmidhuber (1997) which in turn is a modified version of a Recurrent Neural Network (RNN). The implementation is based on several functions using the logistic sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^x} \tag{1.21}$$

The input gate controls the information each cell lets in:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t\ 1} + W_{ci}c_{t\ 1} + b_i) \tag{1.22}$$

The forget gate controls the information each cell maintains over time:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t\ 1} + W_{cf}c_{t\ 1} + b_f) \tag{1.23}$$

After this the following function can be used to compute the cell state:

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c) \tag{1.24}$$

And finally an output gate controls the information that leaves the cell:

$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_t + b_o) \tag{1.25}$$

All vectors are the same size as the hidden state vector h. The next hidden state is computed by:

$$h_t = o_t \tanh(c_t) \tag{1.26}$$

These hidden state vectors h_t are word-level embeddings where x_t are normal word embeddings. A sentence respectively a document is once read from left to right. We call this the Forward LSTM with the corresponding hidden states $\overrightarrow{h}_t$. We can also read the sentence, respectively document from right to left. This is our Backward LSTM with hidden states $\overleftarrow{h}_t$. The Bidirectional LSTM is defined by the concatenation of both:

$$h_t = (\overrightarrow{h}_t, \overleftarrow{h}_t) \tag{1.27}$$

So depending on our needs we can use word-level embeddings which incorporate the left context, the right context or both.

1.4.3 Sentence Embeddings

A sentence embedding encodes an entire sentence. In our framework, we can use them to encode a question. A simple approach is to use the word-level embedding of the first word concatenated with the word-level embedding of the last word. Thus, a sentence embedding is given by:

$$s = (\overrightarrow{h}_T, \overleftarrow{h}_1) \tag{1.28}$$

with T being the sentence length.

Other more sophisticated strategies include encoder-decoder (Bahdanau et al., 2014) or seq2seq models (Sutskever et al., 2014) to auto-encode the sentence. Encoder and decoder are usually LSTMs (see previous subsection). Kiros et al. (2015b) try to mimic the Skip-gram model by predicting surrounding sentences, also using an encoder-decoder model.

1.4.4 Document Embeddings

Document embeddings must be able to either store a lot of information or to concentrate on the relevant information. In theory, we can use the same technique we used for the sentence embedding for an entire document. In practice, this does not work very well, as a document has too much information to encode it into a single vector. We use an attention mechanism which uses the question to concentrate on the relevant information. It can be seen as a forked neural network with the hidden state of the LSTM h_t and the sentence encoding of the question s_q as input:

$$a_t^h = \sigma(W_{ha} h_t + W_{qa} s_q + b_a) \tag{1.29}$$

After the hidden layer a_t^h follows we compute a scalar value a_t^o as output:

$$a_t^o = \sigma(W_{aa} a_t^h + b_{a^\theta}) \tag{1.30}$$

A softmax function over the output of all hidden states gives us the final attention weight a_t:

$$a_t = \frac{\exp(a_t^o)}{\sum_{k=1}^{T} \exp(a_k^o)} \tag{1.31}$$

Instead of using a forked neural network with one hidden layer, we can also use a bilinear term to compute a_t^o:

$$a_t^o = h_t^T W_{BL} s_q \tag{1.32}$$

This method seems to be preferable if we assume s_q and h_t to live in a similar space (Luong et al., 2015). To get the document embedding we compute a weighted sum over all word-level embeddings:

$$d = \sum_{t=1}^{T} a_t h_t \tag{1.33}$$

We can now train a neural network, which minimizes the differences between the document embedding and the correct answer. See (Hermann et al., 2015) for more details. By doing so we are able to learn document embeddings that capture the information relevant to the question. The entire system is visualized in Figure 1.3.

This is a fast-pacing field of research but LSTMs with attention mechanisms together with CNNs are the basis for an increasing number of state-of-the-art systems in Natural Language Processing. While it is feasible to derive the gradient of the models we introduced in the beginning of the introduction, this might be cumbersome for an advanced deep neural network like Figure 1.3. Libraries like

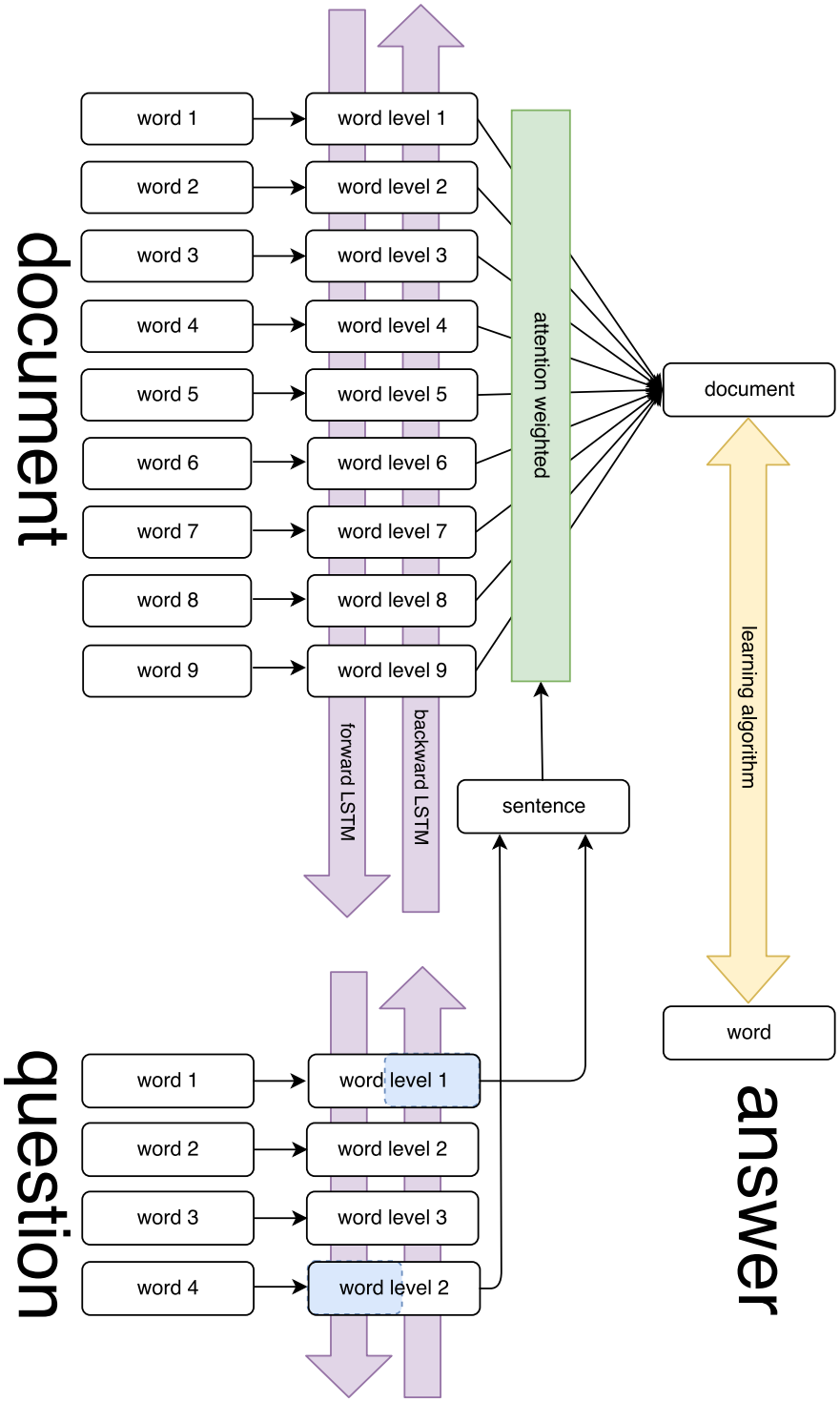

Figure 1.3 – *An end-to-end system for question answering.*

Theano (Theano Development Team, 2016) or TensorFlow (Abadi et al., 2015) take over this task by automatic differentiation, i.e., applying the chain rule to all operation.

Chapter 2

CoSimRank: A Flexible & Efficient Graph-Theoretic Similarity Measure

CoSimRank: A Flexible & Efficient Graph-Theoretic Similarity Measure

Sascha Rothe and Hinrich Schütze
Center for Information & Language Processing
University of Munich
sascha@cis.lmu.de

Abstract

We present *CoSimRank*, a graph-theoretic similarity measure that is efficient because it can compute a single node similarity without having to compute the similarities of the entire graph. We present equivalent formalizations that show CoSimRank's close relationship to Personalized PageRank and SimRank and also show how we can take advantage of fast matrix multiplication algorithms to compute CoSimRank. Another advantage of CoSimRank is that it can be flexibly extended from basic node-node similarity to several other graph-theoretic similarity measures. In an experimental evaluation on the tasks of synonym extraction and bilingual lexicon extraction, CoSimRank is faster or more accurate than previous approaches.

1 Introduction

Graph-theoretic algorithms have been successfully applied to many problems in NLP (Mihalcea and Radev, 2011). These algorithms are often based on PageRank (Brin and Page, 1998) and other centrality measures (e.g., (Erkan and Radev, 2004)). An alternative for tasks involving similarity is SimRank (Jeh and Widom, 2002). SimRank is based on the simple intuition that nodes in a graph should be considered as similar to the extent that their neighbors are similar. Unfortunately, SimRank has time complexity $O(n^3)$ (where n is the number of nodes in the graph) and therefore does not scale to the large graphs that are typical of NLP.

This paper introduces CoSimRank,[1] a new graph-theoretic algorithm for computing node similarity that combines features of SimRank and PageRank. Our key observation is that to compute the similarity of two nodes, we need not consider all other nodes in the graph as SimRank does; instead, CoSimRank starts random walks from the two nodes and computes their similarity at each time step. This offers large savings in computation time if we only need the similarities of a small subset of all n^2 node similarities.

These two cases – computing a few similarities and computing many similarities – correspond to two different representations we can compute CoSimRank on: a vector representation, which is fast for only a few similarities, and a matrix representation, which can take advantage of fast matrix multiplication algorithms.

CoSimRank can be used to compute many variations of basic node similarity – including similarity for graphs with weighted and typed edges and similarity for sets of nodes. Thus, CoSimRank has the added advantage of being a flexible tool for different types of applications.

The extension of CoSimRank to *similarity across graphs* is important for the application of bilingual lexicon extraction: given a set of correspondences between nodes in two graphs A and B (corresponding to two different languages), a pair of nodes $(a \quad A, b \quad B)$ is a good candidate for a translation pair if their node similarity is high. In an experimental evaluation, we show that CoSimRank is more efficient and more accurate than both SimRank and PageRank-based algorithms.

This paper is structured as follows. Section 2 discusses related work. Section 3 introduces CoSimRank. In Section 4, we compare CoSimRank and SimRank. By providing some useful extensions, we demonstrate the great flexibility of CoSimRank (Section 5). We perform an experimental evaluation of CoSimRank in Section 6. Section 7 summarizes the paper.

2 Related Work

Our work is unsupervised. We therefore do not review graph-based methods that make extensive

[1] Code available at code.google.com/p/cistern

use of supervised learning (e.g., de Melo and Weikum (2012)).

Since the original version of *SimRank* (Jeh and Widom, 2002) has complexity $O(n^4)$, many extensions have been proposed to speed up its calculation. A Monte Carlo algorithm, which is scalable to the whole web, was suggested by Fogaras and Rácz (2005). However, in an evaluation of this algorithm we found that it does not give competitive results (see Section 6). A matrix representation of SimRank called *SimFusion* (Xi et al., 2005) improves the computational complexity from $O(n^4)$ to $O(n^3)$. Lizorkin et al. (2010) also reduce complexity to $O(n^3)$ by selecting essential node pairs and using partial sums. They also give a useful overview of SimRank, SimFusion and the Monte Carlo methods of Fogaras and Rácz (2005). A non-iterative computation for SimRank was introduced by Li et al. (2010). This is especially useful for dynamic graphs. However, all of these methods have to run SimRank on the entire graph and are not efficient enough for very large graphs. We are interested in applications that only need a fraction of all $O(n^2)$ pairwise similarities. The algorithm we propose below is an order of magnitude faster in such applications because it is based on a local formulation of the similarity measure.[2]

Apart from SimRank, many other similarity measures have been proposed. Leicht et al. (2006) introduce a similarity measure that is also based on the idea that nodes are similar when their neighbors are, but that is designed for bipartite graphs. However, most graphs in NLP are not bipartite and Jeh and Widom (2002) also proposed a SimRank variant for bipartite graphs.

Another important similarity measure is cosine similarity of *Personalized PageRank* (PPR) vectors. We will refer to this measure as *PPR+cos*. Hughes and Ramage (2007) find that PPR+cos has high correlation with human similarity judgments on WordNet-based graphs. Agirre et al. (2009) use PPR+cos for WordNet and for cross-lingual studies. Like CoSimRank, PPR+cos is efficient when computing single node pair similarities; we therefore use it as one of our baselines below. This method is also used by Chang et al. (2013) for semantic relatedness. They also experimented with Euclidean distance and KL-divergence. Interestingly, a simpler method performed best when comparing with human similarity judgments. In this method only the entries corresponding to the compared nodes are used for a similarity score. Rao et al. (2008) compared PPR+cos to other graph based similarity measures like shortest-path and bounded-length random walks. PPR+cos performed best except for a new similarity measure based on commute time. We do not compare against this new measure as it uses the graph Laplacian and so cannot be computed for a single node pair.

One reason CoSimRank is efficient is that we need only compute a few iterations of the random walk. This is often true of this type of algorithm; cf. (Schütze and Walsh, 2008).

LexRank (Erkan and Radev, 2004) is similar to PPR+cos in that it combines PageRank and cosine; it initializes the sentence similarity matrix of a document using cosine and then applies PageRank to compute lexical centrality. Despite this superficial relatedness, applications like lexicon extraction that look for *similar entities* and applications that look for *central entities* are quite different.

In addition to faster versions of SimRank, there has also been work on extensions of SimRank. Dorow et al. (2009) and Laws et al. (2010) extend SimRank to edge weights, edge labels and multiple graphs. We use their Multi-Edge Extraction (MEE) algorithm as one of our baselines below. A similar graph of dependency structures was built by Minkov and Cohen (2008). They applied different similarity measures, e.g., cosine of dependency vectors or a new algorithm called *path-constrained* graph walk, on synonym extraction (Minkov and Cohen, 2012). We compare CoSimRank with their results in our experiments (see Section 6).

Some other applications of SimRank or other graph based similarity measures in NLP include work on document similarity (Li et al., 2009), the transfer of sentiment information between languages (Scheible et al., 2010) and named entity disambiguation (Han and Zhao, 2010). Hoang and Kan (2010) use SimRank for related work summarization. Muthukrishnan et al. (2010) combine link based similarity and content based similarity for document clustering and classification.

These approaches use at least one of cosine similarity, PageRank and SimRank. CoSimRank can either be interpreted as an efficient version of Sim-

[2] A reviewer suggests that CoSimRank is an efficient version of SimRank in a way analogous to SALSA's (Lempel and Moran, 2000) relationship to HITS (Kleinberg, 1999) in that different aspects of similarity are decoupled.

Rank or as a version of Personalized PageRank for similarity measurement. The novelty is that we compute similarity for vectors that are induced using a new algorithm, so that the similarity measurement is much more efficient when an application only needs a fraction of all $O(n^2)$ pairwise similarities.

3 CoSimRank

We first first give an intuitive introduction of CoSimRank as a Personalized PageRank (PPR) derivative. Later on, we will give a matrix formulation to compare CoSimRank with SimRank.

3.1 Personalized PageRank

Haveliwala (2002) introduced Personalized PageRank – or topic-sensitive PageRank – based on the idea that the uniform damping vector $p^{(0)}$ can be replaced by a personalized vector, which depends on node i. We usually set $p^{(0)}(i) = e_i$, with e_i being a vector of the standard basis, i.e., the i^{th} entry is 1 and all other entries are 0. The PPR vector of node i is given by:

$$p^{(k)}(i) = dAp^{(k-1)}(i) + (1-d)p^{(0)}(i) \quad (1)$$

where A is the stochastic matrix of the Markov chain, i.e., the row normalized adjacency matrix. The damping factor $d \quad (0,1)$ ensures that the computation converges. The PPR vector after k iterations is given by $p^{(k)}$.

To visualize this formula, one can imagine a random surfer starting at node i and following one of the links with probability d or jumping back to the starting node i with probability $(1-d)$. Entry i of the converged PPR vector represents the probability that the random surfer is on node i after an unlimited number of steps.

To simulate the behavior of SimRank we will simplify this equation and set the damping factor $d = 1$. We will re-add a damping factor later in the calculation.

$$p^{(k)} = Ap^{(k-1)} \quad (2)$$

Note that the personalization vector $p^{(0)}$ was eliminated, but is still present as the starting vector of the iteration.

3.2 Similarity of vectors

Let $p(i)$ be the PPR vector of node i. The cosine of two vectors u and v is computed by dividing the inner product u, v by the lengths of the vectors. The cosine of two PPR vectors can be used as a similarity measure for the corresponding nodes (Hughes and Ramage, 2007; Agirre et al., 2009):

$$s(i,j) = \frac{p(i), p(j)}{|p(i)||p(j)|} \quad (3)$$

This measure $s(i,j)$ looks at the probability that a random walker is on a certain edge after an *unlimited number of steps*. This is potentially problematic as the example in Figure 1 shows. The PPR vectors of *suit* and *dress* will have some weight on *tailor*, which is good. However, the PPR vector of *law* will also have a non-zero weight for *tailor*. So *law* and *dress* are similar because of the node *tailor*. This is undesirable.

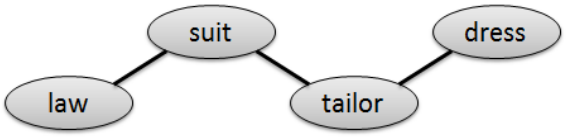

Figure 1: Graph motivating CoSimRank algorithm. Whereas PPR gives relatively high similarity to the pair (law,suit), CoSimRank assigns the pair similarity 0.

We can prevent this type of spurious similarity by *taking into account the path the random surfer took to get to a particular node*. We formalize this by defining CoSimRank $s(i,j)$ as follows:

$$s(i,j) = \sum_{k=0}^{\infty} c^k \; p^{(k)}(i), p^{(k)}(j) \quad (4)$$

where $p^{(k)}(i)$ is the PPR vector of node i from Eq. 2 after k iterations. We compare the PPR vectors at each time step k. The sum of all similarities is the value of CoSimRank, i.e., the final similarity. We add a damping factor c, so that early meetings are more valuable than later meetings.

To compute the similarity of two vectors u and v we use the inner product $\cdot, \cdot$ in Eq. 4 for two reasons:

1. This is similar to cosine similarity except that the 1-norm is used instead of the 2-norm. Since our vectors are probability vectors, we have

$$\frac{p(i), p(j)}{|p(i)||p(j)|} = p(i), p(j)$$

for the 1-norm.[3]

2. Without expensive normalization, we can give a simple matrix formalization of CoSimRank and compute it efficiently using fast matrix multiplication algorithms.

Later on, the following iterative computation of CoSimRank will prove useful:

$$s^{(k)}(i,j) = c^k \; p^{(k)}(i), p^{(k)}(j) \; + s^{(k-1)}(i,j) \tag{5}$$

3.3 Matrix formulation

The matrix formulation of CoSimRank is:

$$\begin{aligned} S^{(0)} &= E \\ S^{(1)} &= cAA^T + S^{(0)} \\ S^{(2)} &= c^2A^2(A^T)^2 + S^{(1)} \\ &\dots \\ S^{(k)} &= c^kA^k(A^T)^k + S^{(k-1)} \end{aligned} \tag{6}$$

We will see in Section 5 that this formulation is the basis for a very efficient version of CoSimRank.

3.4 Convergence properties

As the PPR vectors have only positive values, we can easily see in Eq. 4 that the CoSimRank of one node pair is monotonically non-decreasing. For the dot product of two vectors, the Cauchy-Schwarz inequality gives the upper bound:

$$u, v \;\leq\; u \quad v$$

where x is the norm of x. From Eq. 2 we get $\left\|p^{(k)}\right\|_1 = 1$, where $\cdot_1$ is the 1-norm. We also know from elementary functional analysis that the 1-norm is the biggest of all p-norms and so one has $\left\|p^{(k)}\right\| \leq 1$. It follows that CoSimRank grows more slowly than a geometric series and converges if $|c| < 1$:

$$s(i,j) \leq \sum_{k=0}^{\infty} c^k = \frac{1}{1-c}$$

If an upper bound of 1 is desired for $s(i,j)$ (instead of $1/(1-c)$), then we can use s :

$$s\,(i,j) = (1-c)s(i,j)$$

[3]This type of similarity measure has also been used and investigated by Ó Séaghdha and Copestake (2008), Cha (2007), Jebara et al. (2004) (probability product kernel) and (Jaakkola et al., 1999) (Fisher kernel) among others.

4 Comparison to SimRank

The original SimRank equation can be written as follows (Jeh and Widom, 2002):

$$r(i,j) = \begin{cases} 1, & \text{if } i = j \\ \frac{c}{|N(i)||N(j)|} \sum\limits_{\substack{k \in N(i) \\ l \in N(j)}} r(k,l), & \text{else} \end{cases}$$

where $N(i)$ denotes the nodes connected to i. SimRank is computed iteratively. With A being the normalized adjacency matrix we can write SimRank in matrix formulation:

$$\begin{aligned} R^{(0)} &= E \\ R^{(k)} &= \max\{cAR^{(k-1)}A^T, R^{(0)}\} \end{aligned} \tag{7}$$

where the maximum of two matrices refers to the element-wise maximum. We will now prove by induction that the matrix formulation of CoSimRank (Eq. 6) is equivalent to:

$$S^{\,(k)} = cAS^{\,(k-1)}A^T + S^{(0)} \tag{8}$$

and thus very similar to SimRank (Eq. 7).

The base case $S^{(1)} = S^{\,(1)}$ is trivial. Inductive step:

$$\begin{aligned} S^{\,(k)} &\overset{(8)}{=} cAS^{\,(k-1)}A^T + S^{(0)} \\ &= cA(c^{k-1}A^{k-1}(A^T)^{k-1} + S^{(k-2)})A^T + S^{(0)} \\ &= c^kA^k(A^T)^k + cAS^{(k-2)}A^T + S^{(0)} \\ &= c^kA^k(A^T)^k + S^{(k-1)} \overset{(6)}{=} S^{(k)} \end{aligned}$$

□

Comparing Eqs. 7 and 8, we see that SimRank and CoSimRank are very similar except that they initialize the similarities on the diagonal differently. Whereas SimRank sets each of these entries back to one at each iteration, CoSimRank adds one. Thus, when computing the two similarity measures iteratively, the diagonal element (i,i) will be set to 1 by both methods for those initial iterations for which this entry is 0 for $cAS^{(k-1)}A^T$ (i.e., before applying either max or add). The methods diverge when the entry is $= 0$ for the first time.

Complexity of computing all n^2 similarities. The matrix formulas of both SimRank (Eq. 7) and CoSimRank (Eq. 8) have time complexity $O(n^3)$ or – if we want to take the higher efficiency of computation for sparse graphs into account – $O(dn^2)$ where n is the number of nodes and d the

average degree. Space complexity is $O(n^2)$ for both algorithms.

Complexity of computing k^2 n^2 similarities. In most cases, we only want to compute k^2 similarities for k nodes. For CoSimRank, we compute the k PPR vectors in $O(kdn)$ (Eq. 2) and compute the k^2 similarities in $O(k^2 n)$ (Eq. 5). If $d < k$, then the time complexity of CoSimRank is $O(k^2 n)$. If we only compute a single similarity, then the complexity is $O(dn)$. In contrast, the complexity of SimRank is the same as in the all-similarities case: $O(dn^2)$. It is not obvious how to design a lower-complexity version of SimRank for this case. Thus, we have reduced SimRank's cubic time complexity to a quadratic time complexity for CoSimRank or – assuming that the average degree d does not depend on n – SimRank's quadratic time complexity to linear time complexity for the case of computing few similarities.

Space complexity for computing k^2 similarities is $O(kn)$ since we need only store k vectors, not the complete similarity matrix. This complexity can be exploited even for the all similarities application: If the matrix formulation cannot be used because the $O(n^2)$ similarity matrix is too big for available memory, then we can compute all similarities in batches – and if desired in parallel – whose size is chosen such that the vectors of each batch still fit in memory.

In summary, CoSimRank and SimRank have similar space and time complexities for computing all n^2 similarities. For the more typical case that we only want to compute a fraction of all similarities, we have recast the global SimRank formulation as a local CoSimRank formulation. As a result, time and space complexities are much improved. In Section 6, we will show that this is also true in practice.

5 Extensions

We will show now that the basic CoSimRank algorithm can be extended in a number of ways and is thus a flexible tool for different NLP applications.

5.1 Weighted edges

The use of weighted edges was first proposed in the PageRank patent. It is straightforward and easy to implement by replacing the row normalized adjacency matrix A with an arbitrary stochastic matrix P. We can use this edge weighted PageRank for CoSimRank.

5.2 CoSimRank across graphs

We often want to compute the similarity of nodes in *two different graphs* with a known node-node correspondence; this is the scenario we are faced with in the lexicon extraction task (see Section 6). A variant of SimRank for this task was presented by Dorow et al. (2009). We will now present an equivalent method for CoSimRank. We denote the number of nodes in the two graphs U and V by $|U|$ and $|V|$, respectively. We compute PPR vectors p $\mathbb{R}^{|U|}$ and q $\mathbb{R}^{|V|}$ for each graph. Let $S^{(0)}$ $\mathbb{R}^{|U|\times|V|}$ be the known node-node correspondences. The analog of CoSimRank (Eq. 4) for two graphs is then:

$$s(i,j) = \sum_{k=0}^{\infty} c^k \sum_{(u,v)\ S^{(0)}} p_u^{(k)}(i) q_v^{(k)}(j) \qquad (9)$$

The matrix formulation (cf. Eq. 6) is:

$$S^{(k)} = c^k A^k S^{(0)} (B^T)^k + S^{(k-1)} \qquad (10)$$

where A and B are row-normalized adjacency matrices. We can interpret $S^{(0)}$ as a change of basis. A similar approach for word embeddings was published by Mikolov et al. (2013). They call $S^{(0)}$ the translation matrix.

5.3 Typed edges

To be able to directly compare to prior work in our experiments, we also present a method to integrate a set of typed edges T in the CoSimRank calculation. For this we will compute a similarity matrix for each edge type τ and merge them into one matrix for the next iteration:

$$S^{(k)} = \left(\frac{c}{|T|} \sum_{\tau\ T} A_\tau S^{(k-1)} B_\tau^T \right) + S^{(0)} \qquad (11)$$

This formula is identical to the random surfer model where two surfers only meet iff they are on the same node and used the same edge type to get there. A more strict claim would be to use the same edge type at any time of their journey:

$$S^{(k)} = \frac{c^k}{|T|^k} \sum_{\tau\ T^k} \left(\prod_{i=1}^{k} A_{\tau_i} \right) S^{(0)} \left(\prod_{i=0}^{k-1} B_{\tau_{k-i}}^T \right) + S^{(k-1)} \qquad (12)$$

We will not use Eq. 12 due to its space complexity.

5.4 Similarity of sets of nodes

CoSimRank can also be used to compute the similarity $s(V, W)$ of two sets V and W of nodes, e.g., short text snippets. We are not including this method in our experiments, but we will give the equation here, as traditional document similarity measures (e.g., cosine similarity) perform poorly on this task although there also are known alternatives with good results (Sahami and Heilman, 2006). For a set V, the initial PPR vector is given by:

$$p_i^{(0)}(V) = \begin{cases} \frac{1}{|V|}, & \text{if } i \quad V \\ 0, & \text{else} \end{cases}$$

We then reuse Eq. 4 to compute $s(V, W)$:

$$s(V, W) = \sum_{k=0}^{\infty} c^k \; p^{(k)}(V), p^{(k)}(W)$$

In summary, modifications proposed for SimRank (weighted and typed edges, similarity across graphs) as well as modifications proposed for PageRank (sets of nodes) can also be applied to CoSimRank. This makes CoSimRank a very flexible similarity measure.

We will test the first three extensions experimentally in the next section and leave similarity of node sets for future work.

6 Experiments

We evaluate CoSimRank for the tasks of synonym extraction and bilingual lexicon extraction. We use the basic version of CoSimRank (Eq. 4) for synonym extraction and the two-graph version (Eq. 9) for lexicon extraction, both with weighted edges. Our motivation for this application is that two words that are synonyms of each other should have similar lexical neighbors and that two words that are translations of each other should have neighbors that correspond to each other; thus, in each case the nodes should be similar in the graph-theoretic sense and CoSimRank should be able to identify this similarity.

We use the English and German graphs published by Laws et al. (2010), including edge weighting and normalization. Nodes are nouns, adjectives and verbs occurring in Wikipedia. There are three types of edges, corresponding to three types of syntactic configurations extracted from the parsed Wikipedias: adjective-noun, verb-object and noun-noun coordination. Table 1 gives examples and number of nodes and edges.

Edge types

relation	entities	description	example
amod	a, v	adjective-noun	a fast car
dobj	v, n	verb-object	drive a car
ncrd	n, n	noun-noun	cars and busses

Graph statistics

nodes		nouns	adjectives	verbs
	de	34,544	10,067	2,828
	en	22,258	12,878	4,866
edges		ncrd	amod	dobj
	de	65,299	417,151	143,905
	en	288,878	686,069	510,351

Table 1: Edge types (above) and number of nodes and edges (below)

6.1 Baselines

We propose CoSimRank as an efficient algorithm for computing the similarity of nodes in a graph. Consequently, we compare against the two main methods for this task in NLP: SimRank and extensions of PageRank.

We also compare against the MEE (Multi-Edge Extraction) variant of SimRank (Dorow et al., 2009), which handles labeled edges more efficiently than SimRank:

$$S^{(k)} = \frac{c}{|T|} \sum_{\tau \quad T} A_\tau S^{(k-1)} B_\tau^T$$

$$S^{(k)} = \max\{S^{(k)}, S^{(0)}\}$$

where A_τ is the row-normalized adjacency matrix for edge type τ (see edge types in Table 1).

Apart from SimRank, extensions of PageRank are the main methods for computing the similarity of nodes in graphs in NLP (e.g., Hughes and Ramage (2007), Agirre et al. (2009) and other papers discussed in related work). Generally, these methods compute the Personalized PageRank for each node (see Eq. 1). When the computation has converged, the similarity of two nodes is given by the cosine similarity of the Personalized PageRank vectors. We implemented this method for our experiments and call it PPR+cos.

6.2 Synonym Extraction

We use *TS68*, a test set of 68 synonym pairs published by Minkov and Cohen (2012) for evaluation. This gold standard lists a single word as the

	P@1	P@10	MRR
one-synonym			
PPR+cos	20.6%	52.9%	0.32
SimRank	**25.0**%	61.8%	**0.37**
CoSimRank	**25.0**%	61.8%	**0.37**
Typed CoSimRank	23.5%	**63.2**%	**0.37**
extended			
PPR+cos	32.6%	73.5%	0.48
SimRank	**45.6**%	**83.8**%	**0.59**
CoSimRank	**45.6**%	**83.8**%	**0.59**
Typed CoSimRank	44.1%	**83.8**%	**0.59**

Table 2: Results for synonym extraction on TS68. Best result in each column in bold.

correct synonym even if there are several equally acceptable near-synonyms (see Table 3 for examples). We call this the one-synonym evaluation. Three native English speakers were asked to mark synonyms, that were proposed by a baseline or by CoSimRank, i.e. ranked in the top 10. If all three of them agreed on one word as being a synonym in at least one meaning, we added this as a correct answer to the test set. We call this the "extended" evaluation (see Table 2).

Synonym extraction is run on the English graph. To calculate PPR+cos, we computed 20 iterations with a decay factor of 0.8 and used the cosine similarity with the 2-norm in the denominator to compare two vectors. For the other three methods, we also used a decay factor of 0.8 and computed 5 iterations. Recall that CoSimRank uses the simple inner product $\cdot,\cdot$ to compare vectors.

Our evaluation measures are proportion of words correctly translated by word in the top position (P@1), proportion of words correctly translated by a word in one of the top 10 positions (P@10) and Mean Reciprocal Rank (MRR). CoSimRank's MRR scores of 0.37 (one-synonym) and 0.59 (extended) are the same or better than all baselines (see Table 2). CoSimRank and SimRank have the same P@1 and P@10 accuracy (although they differed on some decisions). CoSimRank is better than PPR+cos on both evaluations, but as this test set is very small, the results are not significant. Table 3 shows a sample of synonyms proposed by CoSimRank.

Minkov and Cohen (2012) tested cosine and random-walk measures on grammatical relationships (similar to our setup) as well as on cooccurrence statistics. The MRR scores for these methods range from 0.29 to 0.59. (MRR is equivalent to MAP as reported by Minkov and Cohen (2012) when there is only one correct answer.) Their best number (0.59) is better than our one-synonym result; however, they performed manual postprocessing of results – e.g., discarding words that are morphologically or semantically related to other words in the list – so our fully automatic results cannot be directly compared.

keyword	expected	extracted
movie	film	**film**
modern	contemporary	**contemporary**
demonstrate	protest	**show**
attractive	appealing	beautiful
economic	profitable	financial
close	shut	open

Table 3: Examples for extracted synonyms. Correct synonyms according to extended evaluation in bold.

6.3 Lexicon Extraction

We evaluate lexicon extraction on TS1000, a test set of 1000 items, (Laws et al., 2010) each consisting of an English word and its German translations. For lexicon extraction, we use the same parameters as in the synonym extraction task for all four similarity measures. We use a seed dictionary of 12,630 word pairs to establish node-node correspondences between the two graphs. We remove a search keyword from the seed dictionary before calculating similarities for it, something that the architecture of CoSimRank makes easy because we can use a different seed dictionary $S^{(0)}$ for every keyword.

Both CoSimRank methods outperform SimRank significantly (see Table 4). The difference between CoSimRank with and without typed edges is not significant. (This observation was also made for SimRank on a smaller graph and test set (Laws et al., 2010).)

PPR+cos's performance at 14.8% correct translations is much lower than SimRank and CoSimRank. The disadvantage of this similarity measure is significant and even more visible on bilingual lexicon extraction than on synonym extraction (see Table 2). The reason might be that we are not comparing the whole PPR vector anymore,

	P@1	P@10
PPR+cos	14.8%†	45.7%†
SimRank MEE	48.0%†	76.0%†
CoSimRank	61.1%	**84.0**%
Typed CoSimRank	**61.4**%	83.9%

Table 4: Results for bilingual lexicon extraction (TS1000 EN → DE). Best result in each column in bold.

	synonym extraction (68 word pairs)	lexicon extraction (1000 word pairs)
PPR+cos	2,228	**2,195**
SimRank	23,423	14,418
CoSimRank	**524**	2,342
Typed CoSimRank	615	6,108

Table 5: Execution times in minutes for CoSimRank and the baselines. Best result in each column in bold.

but only entries which occur in the seed dictionary (see Eq. 9). As the seed dictionary contains 12,630 word pairs, this means that only every fourth entry of the PPR vector (the German graph has 47,439 nodes) is used for similarity calculation. This is also true for CoSimRank, but it seems that CoSimRank is more stable because we compare more than one vector.

We also experimented with the method of Fogaras and Rácz (2005). We tried a number of different ways of modifying it for weighted graphs: (i) running the random walks with the weighted adjacency matrix as Markov matrix, (ii) storing the weight (product of each edge weight) of a random walk and using it as a factor if two walks meet and (iii) a combination of both. We needed about 10,000 random walks in all three conditions. As a result, the computational time was approximately 30 minutes per test word, so this method is even slower than SimRank for our application. The accuracies P@1 and P@10 were worse in all experiments than those of CoSimRank.

6.4 Run time performance

Table 5 compares the run time performance of CoSimRank with the baselines. We ran all experiments on a 64-bit Linux machine with 64 Intel Xenon X7560 2.27Ghz CPUs and 1TB RAM. The calculated time is the sum of the time spent in user mode and the time spent in kernel mode. The actual wall clock time was significantly lower as we used up to 64 CPUs.

Compared to SimRank, CoSimRank is more than 40 times faster on synonym extraction and six times faster on lexicon extraction. SimRank is at a disadvantage because it computes all similarities in the graph regardless of the size of the test set; it is particularly inefficient on synonym extraction because the English graph contains a large number of edges (see Table 1).

Compared to PPR+cos, CoSimRank is roughly four times faster on synonym extraction and has comparable performance on lexicon extraction. We compute 20 iterations of PPR+cos to reach convergence and then calculate a single cosine similarity. For CoSimRank, we need only compute five iterations to reach convergence, but we have to compute a vector similarity in each iteration. The counteracting effects of fewer iterations and more vector similarity computations can give either CoSimRank or PPR+cos an advantage, as is the case for synonym extraction and lexicon extraction, respectively.

CoSimRank should generally be three times faster than typed CoSimRank since the typed version has to repeat the computation for each of the three types. This effect is only visible on the larger test set (lexicon extraction) because the general computation overhead is about the same on a smaller test set.

6.5 Comparison with WINTIAN

Here we address inducing a bilingual lexicon from a seed set based on grammatical relations found by a parser. An alternative approach is to induce a bilingual lexicon from Wikipedia's interwiki links (Rapp et al., 2012). These two approaches have different strengths and weaknesses; e.g., the interwiki-link-based approach does not require a seed set, but it can only be applied to comparable corpora that consist of corresponding – although not necessarily "parallel" – documents.

Despite these differences it is still interesting to compare the two algorithms. Rapp et al. (2012) kindly provided their test set to us. It contains 1000 English words and a single correct German translation for each. We evaluate on a subset we call TS774 that consists of the 774 test word pairs that are in the intersection of words covered by the

†significantly worse than CoSimRank ($\alpha = 0.05$, one-tailed Z-Test)

	P@1	P@10
Wintian	**43.8%**	55.4%†
CoSimRank	43.0%	**73.6%**

Table 6: Results for bilingual lexicon extraction (TS774 DE → EN). Best result in each column in bold.

WINTIAN Wikipedia data (Rapp et al., 2012) and words covered by our data. Most of the 226 missing word pairs are adverbs, prepositions and plural forms that are not covered by our graphs due to the construction algorithm we use: lemmatization, restriction to adjectives, nouns and verbs etc.

Table 6 shows that CoSimRank is slightly, but not significantly worse than WINTIAN on P@1 (43.0 vs 43.8), but significantly better on P@10 (73.6 vs 55.4).[4] The reason could be that CoSimRank is a more effective algorithm than WINTIAN; but the different initializations (seed set vs interwiki links) or the different linguistic representations (grammatical relations vs bag-of-words) could also be responsible.

6.6 Error Analysis

The results on TS774 can be considered conservative since only one translation is accepted as being correct. In reality other translations might also be acceptable (e.g., both *street* and *road* for *Straße*). In contrast, TS1000 accepts more than one correct translation. Additionally, TS774 was created by translating English words into German (using Google translate). We are now testing the reverse direction. So we are doomed to fail if the original English word is a less common translation of an ambiguous German word. For example, the English word *gulf* was translated by Google to *Golf*, but the most common sense of *Golf* is the sport. Hence our algorithm will incorrectly translate it back to *golf*.

As we can see in Table 7, we also face the problems discussed by Laws et al. (2010): the algorithm sometimes picks cohyponyms (which can still be seen as reasonable) and antonyms (which are clear errors).

Contrary to our intuition, the edge-typed variant of CoSimRank did not perform significantly better than the non-edge-typed version. Looking at Table 1, we see that there is only one edge type connecting adjectives. The same is true for verbs. The random surfer only has a real choice between different edge types when she is on a noun node. Combined with the fact that only the last edge type is important this has absolutely no effect for a random surfer meeting at adjectives or verbs.

Two possible solutions would be (i) to use more fine-grained edge types, (ii) to apply Eq. 12, in which the edge type of each step is important. However, this will increase the memory needed for calculation.

[4] We achieved better results for CoSimRank by optimizing the damping factor, but in this paper, we only present results for a fixed damping factor of 0.8.

keyword	gold standard	CoSimRank
arm	poor	**impoverished**
erreichen	reach	**achieve**
gehen	go	**walk**
direkt	directly	**direct**
weit	far	further
breit	wide	narrow
reduzieren	reduce	increase
Stunde	hour	second
Westen	west	southwest
Junge	boy	child

Table 7: Examples for CoSimRank translation errors on TS774. We counted translations as incorrect if they were not listed in the gold standard even if they were correct translations according to *www.dict.cc* (in bold).

7 Summary

We have presented *CoSimRank*, a new similarity measure that can be computed for a single node pair without relying on the similarities in the whole graph. We gave two different formalizations of CoSimRank: (i) a derivation from Personalized PageRank and (ii) a matrix representation that can take advantage of fast matrix multiplication algorithms. We also presented extensions of CoSimRank for a number of applications, thus demonstrating the flexibility of CoSimRank as a similarity measure.

We showed that CoSimRank is superior to SimRank in time and space complexity; and we demonstrated that CoSimRank performs better than PPR+cos on two similarity computation tasks.

Acknowledgments. This work was supported by DFG (SCHU 2246/2-2).

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 19–27.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *WWW*, pages 107–117.

Sung-Hyuk Cha. 2007. Comprehensive survey on distance/similarity measures between probability density functions. *Mathematical Models and Methods in Applied Sciences*, 1(4):300–307.

Ching-Yun Chang, Stephen Clark, and Brian Harrington. 2013. Getting creative with semantic similarity. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 330–333.

Gerard de Melo and Gerhard Weikum. 2012. Uwn: A large multilingual lexical knowledge base. In *ACL (System Demonstrations)*, pages 151–156.

Beate Dorow, Florian Laws, Lukas Michelbacher, Christian Scheible, and Jason Utt. 2009. A graph-theoretic algorithm for automatic extension of translation lexicons. In *Proceedings of the Workshop on Geometrical Models of Natural Language Semantics*, GEMS '09, pages 91–95.

Günes Erkan and Dragomir R. Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Intell. Res. (JAIR)*, 22:457–479.

Dániel Fogaras and Balázs Rácz. 2005. Scaling link-based similarity search. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 641–650.

Xianpei Han and Jun Zhao. 2010. Structural semantic relatedness: a knowledge-based method to named entity disambiguation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 50–59.

Taher H. Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 517–526.

Cong Duy Vu Hoang and Min-Yen Kan. 2010. Towards automated related work summarization. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 427–435.

Thad Hughes and Daniel Ramage. 2007. Lexical semantic relatedness with random graph walks. In *EMNLP-CoNLL*, pages 581–589.

Tommi Jaakkola, David Haussler, et al. 1999. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493.

Tony Jebara, Risi Kondor, and Andrew Howard. 2004. Probability product kernels. *The Journal of Machine Learning Research*, 5:819–844.

Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 538–543.

Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

Florian Laws, Lukas ichelbacher, Beate Dorow, Christian Scheible, Ulrich Heid, and Hinrich Schütze. 2010. A linguistically grounded graph model for bilingual lexicon extraction. In *Coling 2010: Posters*, pages 614–622.

Elizabeth Leicht, Petter Holme, and Mark Newman. 2006. Vertex similarity in networks. *Physical Review E*, 73(2):026120.

Ronny Lempel and Shlomo Moran. 2000. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Computer Networks*, 33(1):387–401.

Pei Li, Zhixu Li, Hongyan Liu, Jun He, and Xiaoyong Du. 2009. Using link-based content analysis to measure document similarity effectively. In *Proceedings of the Joint International Conferences on Advances in Data and Web Management*, APWeb/WAIM '09, pages 455–467.

Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. 2010. Fast computation of simrank for static and dynamic information networks. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 465–476.

Dmitry Lizorkin, Pavel Velikhov, Maxim Grinev, and Denis Turdakov. 2010. Accuracy estimate and optimization techniques for simrank computation. *The VLDB Journal—The International Journal on Very Large Data Bases*, 19(1):45–66.

Rada Mihalcea and Dragomir Radev. 2011. *Graph-based natural language processing and information retrieval*. Cambridge University Press.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Einat Minkov and William W. Cohen. 2008. Learning graph walk based similarity measures for parsed text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 907–916.

Einat Minkov and William W. Cohen. 2012. Graph based similarity measures for synonym extraction from parsed text. In *Workshop Proceedings of TextGraphs-7 on Graph-based Methods for Natural Language Processing*, TextGraphs-7 '12, pages 20–24.

Pradeep Muthukrishnan, Dragomir Radev, and Qiaozhu Mei. 2010. Edge weight regularization over multiple graphs for similarity learning. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 374–383.

Diarmuid Ó Séaghdha and Ann Copestake. 2008. Semantic classification with distributional kernels. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 649–656.

Delip Rao, David Yarowsky, and Chris Callison-Burch. 2008. Affinity measures based on the graph Laplacian. In *Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing*, TextGraphs-3, pages 41–48.

Reinhard Rapp, Serge Sharoff, and Bogdan Babych. 2012. Identifying word translations from comparable documents without a seed lexicon. In *LREC*, pages 460–466.

Mehran Sahami and Timothy D. Heilman. 2006. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 377–386.

Christian Scheible, Florian Laws, Lukas Michelbacher, and Hinrich Schütze. 2010. Sentiment translation through multi-edge graphs. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 1104–1112.

Hinrich Schütze and Michael Walsh. 2008. A graph-theoretic model of lexical syntactic acquisition. In *EMNLP*, pages 917–926.

Wensi Xi, Edward A. Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. 2005. Simfusion: measuring similarity using unified relationship matrix. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 130–137.

Chapter 3

AutoExtend: Extending Word Embeddings to Embeddings for Synsets and Lexemes

AutoExtend: Extending Word Embeddings to Embeddings for Synsets and Lexemes

Sascha Rothe and Hinrich Schütze
Center for Information & Language Processing
University of Munich
sascha@cis.lmu.de

Abstract

We present *AutoExtend*, a system to learn embeddings for synsets and lexemes. It is flexible in that it can take any word embeddings as input and does not need an additional training corpus. The synset/lexeme embeddings obtained live in the same vector space as the word embeddings. A sparse tensor formalization guarantees efficiency and parallelizability. We use WordNet as a lexical resource, but AutoExtend can be easily applied to other resources like Freebase. AutoExtend achieves state-of-the-art performance on word similarity and word sense disambiguation tasks.

1 Introduction

Unsupervised methods for word embeddings (also called "distributed word representations") have become popular in natural language processing (NLP). These methods only need very large corpora as input to create sparse representations (e.g., based on local collocations) and project them into a lower dimensional dense vector space. Examples for word embeddings are SENNA (Collobert and Weston, 2008), the hierarchical log-bilinear model (Mnih and Hinton, 2009), word2vec (Mikolov et al., 2013c) and GloVe (Pennington et al., 2014). However, there are many other resources that are undoubtedly useful in NLP, including lexical resources like WordNet and Wiktionary and knowledge bases like Wikipedia and Freebase. We will simply call these *resources* in the rest of the paper. Our goal is to enrich these valuable resources with embeddings for those data types that are not words; e.g., we want to enrich WordNet with embeddings for synsets and lexemes. A *synset* is a set of synonyms that are interchangeable in some context. A *lexeme* pairs a particular spelling or pronunciation with a particular meaning, i.e., a lexeme is a conjunction of a word and a synset. Our premise is that many NLP applications will benefit if the non-word data types of resources – e.g., synsets in WordNet – are also available as embeddings. For example, in machine translation, enriching and improving translation dictionaries (cf. Mikolov et al. (2013b)) would benefit from these embeddings because they would enable us to create an enriched dictionary for word senses. Generally, our premise is that the arguments for the utility of embeddings for word forms should carry over to the utility of embeddings for other data types like synsets in WordNet.

The insight underlying the method we propose is that *the constraints of a resource can be formalized as constraints on embeddings and then allow us to extend word embeddings to embeddings of other data types like synsets.* For example, the hyponymy relation in WordNet can be formalized as such a constraint.

The advantage of our approach is that it decouples embedding learning from the extension of embeddings to non-word data types in a resource. If somebody comes up with a better way of learning embeddings, these embeddings become immediately usable for resources. And we do not rely on any specific properties of embeddings that make them usable in some resources, but not in others.

An alternative to our approach is to train embeddings on annotated text, e.g., to train synset embeddings on corpora annotated with synsets. However, successful embedding learning generally requires very large corpora and sense labeling is too expensive to produce corpora of such a size.

Another alternative to our approach is to add up all word embedding vectors related to a particular node in a resource; e.g., to create the synset vector of *lawsuit* in WordNet, we can add the word vectors of the three words that are part of the synset (*lawsuit*, *suit*, *case*). We will call this approach

naive and use it as a baseline (S_{naive} in Table 3).

We will focus on WordNet (Fellbaum, 1998) in this paper, but our method – based on a formalization that exploits the constraints of a resource for extending embeddings from words to other data types – is broadly applicable to other resources including Wikipedia and Freebase.

A word in WordNet can be viewed as a composition of several lexemes. Lexemes from different words together can form a synset. When a synset is given, it can be decomposed into its lexemes. And these lexemes then join to form words. These observations are the basis for the formalization of the constraints encoded in WordNet that will be presented in the next section: *we view words as the sum of their lexemes and, analogously, synsets as the sum of their lexemes.*

Another motivation for our formalization stems from the analogy calculus developed by Mikolov et al. (2013a), which can be viewed as a group theory formalization of word relations: we have a set of elements (our vectors) and an operation (addition) satisfying the properties of a mathematical group, in particular, associativity and invertibility. For example, you can take the vector of *king*, subtract the vector of *man* and add the vector of *woman* to get a vector near *queen*. In other words, you remove the properties of *man* and add the properties of *woman*. We can also see the vector of *king* as the sum of the vector of *man* and the vector of a gender-neutral ruler. The next thing to notice is that this does not only work for words that combine several *properties*, but also for words that combine several *senses*. The vector of *suit* can be seen as the sum of a vector representing *lawsuit* and a vector representing *business suit*. AutoExtend is designed to take word vectors as input and unravel the word vectors to the vectors of their lexemes. The lexeme vectors will then give us the synset vectors.

The main contributions of this paper are: (i) We present AutoExtend, a flexible method that extends word embeddings to embeddings of synsets and lexemes. AutoExtend is completely general in that it can be used for any set of embeddings and for any resource that imposes constraints of a certain type on the relationship between words and other data types. (ii) We show that AutoExtend achieves state-of-the-art word similarity and word sense disambiguation (WSD) performance. (iii) We publish the AutoExtend code for extending word embeddings to other data types, the lexeme and synset embeddings and the software to replicate our WSD evaluation.

This paper is structured as follows. Section 2 introduces the model, first as a general tensor formulation then as a matrix formulation making additional assumptions. In Section 3, we describe data, experiments and evaluation. We analyze AutoExtend in Section 4 and give a short summary on how to extend our method to other resources in Section 5. Section 6 discusses related work.

2 Model

We are looking for a model that extends standard embeddings for words to embeddings for the other two data types in WordNet: synsets and lexemes. We want all three data types – words, lexemes, synsets – to live in the same embedding space.

The basic premise of our model is: (i) words are sums of their lexemes and (ii) synsets are sums of their lexemes. We refer to these two premises as *synset constraints*. For example, the embedding of the word *bloom* is a sum of the embeddings of its two lexemes *bloom(organ)* and *bloom(period)*; and the embedding of the synset *flower-bloom-blossom(organ)* is a sum of the embeddings of its three lexemes *flower(organ)*, *bloom(organ)* and *blossom(organ)*.

The synset constraints can be argued to be the simplest possible relationship between the three WordNet data types. They can also be motivated by the way many embeddings are learned from corpora – for example, the counts in vector space models are additive, supporting the view of words as the sum of their senses. The same assumption is frequently made; for example, it underlies the group theory formalization of analogy discussed in Section 1.

We denote word vectors as $w^{(i)}$ $\mathbb{R}^n$, synset vectors as $s^{(j)}$ $\mathbb{R}^n$, and lexeme vectors as $l^{(i,j)}$ $\mathbb{R}^n$. $l^{(i,j)}$ is that lexeme of word $w^{(i)}$ that is a member of synset $s^{(j)}$. We set lexeme vectors $l^{(i,j)}$ that do not exist to zero. For example, the non-existing lexeme *flower(truck)* is set to zero. We can then formalize our premise that the two constraints (i) and (ii) hold as follows:

$$w^{(i)} = \sum_j l^{(i,j)} \quad (1)$$

$$s^{(j)} = \sum_i l^{(i,j)} \quad (2)$$

These two equations are underspecified. We therefore introduce the matrix $E^{(i,j)} \quad \mathsf{R}^{n \times n}$:

$$l^{(i,j)} = E^{(i,j)} w^{(i)} \tag{3}$$

We make the assumption that the dimensions in Eq. 3 are independent of each other, i.e., $E^{(i,j)}$ is a diagonal matrix. Our motivation for this assumption is: (i) This makes the computation technically feasible by significantly reducing the number of parameters and by supporting parallelism. (ii) Treating word embeddings on a per-dimension basis is a frequent design choice (e.g., Kalchbrenner et al. (2014)). Note that we allow $E^{(i,j)} < 0$ and in general the distribution weights for each dimension (diagonal entries of $E^{(i,j)}$) will be different. Our assumption can be interpreted as word $w^{(i)}$ distributing its embedding activations to its lexemes on each dimension separately. Therefore, Eqs. 1-2 can be written as follows:

$$w^{(i)} = \sum_j E^{(i,j)} w^{(i)} \tag{4}$$

$$s^{(j)} = \sum_i E^{(i,j)} w^{(i)} \tag{5}$$

Note that from Eq. 4 it directly follows that:

$$\sum_j E^{(i,j)} = I_n \quad i \tag{6}$$

with I_n being the identity matrix.

Let W be a $|W| \times n$ matrix where n is the dimensionality of the embedding space, $|W|$ is the number of words and each row $w^{(i)}$ is a word embedding; and let S be a $|S| \times n$ matrix where $|S|$ is the number of synsets and each row $s^{(j)}$ is a synset embedding. W and S can be interpreted as linear maps and a mapping between them is given by the rank 4 tensor $\mathbf{E} \quad \mathsf{R}^{|S| \times n \times |W| \times n}$. We can then write Eq. 5 as a tensor product:

$$S = \mathbf{E} \quad W \tag{7}$$

while Eq. 6 states, that

$$\sum_j \mathbf{E}^{i,d_1}_{j,d_2} = 1 \quad i, d_1, d_2 \tag{8}$$

Additionally, there is no interaction between different dimensions, so $\mathbf{E}^{i,d_1}_{j,d_2} = 0$ if $d_1 = d_2$. In other words, we are creating the tensor by stacking the diagonal matrices $E^{(i,j)}$ over i and j. Another sparsity arises from the fact that many lexemes do not exist: $\mathbf{E}^{i,d_1}_{j,d_2} = 0$ if $l^{(i,j)} = 0$; i.e., $l^{(i,j)} = 0$ only if word i has a lexeme that is a member of synset j. To summarize the sparsity:

$$\mathbf{E}^{i,d_1}_{j,d_2} = 0 \quad d_1 = d_2 \quad l^{(i,j)} = 0 \tag{9}$$

2.1 Learning

We adopt an autoencoding framework to learn embeddings for lexemes and synsets. To this end, we view the tensor equation $S = \mathbf{E} \quad W$ as the encoding part of the autoencoder: the synsets are the encoding of the words. We define a corresponding decoding part that decodes the synsets into words as follows:

$$s^{(j)} = \sum_i \bar{l}^{(i,j)}, \quad \overline{w}^{(i)} = \sum_j \bar{l}^{(i,j)} \tag{10}$$

In analogy to $E^{(i,j)}$, we introduce the diagonal matrix $D^{(j,i)}$:

$$\bar{l}^{(i,j)} = D^{(j,i)} s^{(j)} \tag{11}$$

In this case, it is the synset that distributes itself to its lexemes. We can then rewrite Eq. 10 to:

$$s^{(j)} = \sum_i D^{(j,i)} s^{(j)}, \ \overline{w}^{(i)} = \sum_j D^{(j,i)} s^{(j)} \tag{12}$$

and we also get the equivalent of Eq. 6 for $D^{(j,i)}$:

$$\sum_i D^{(j,i)} = I_n \quad j \tag{13}$$

and in tensor notation:

$$\overline{W} = \mathbf{D} \quad S \tag{14}$$

Normalization and sparseness properties for the decoding part are analogous to the encoding part:

$$\sum_i \mathbf{D}^{j,d_2}_{i,d_1} = 1 \quad j, d_1, d_2 \tag{15}$$

$$\mathbf{D}^{j,d_2}_{i,d_1} = 0 \quad d_1 = d_2 \quad l^{(i,j)} = 0 \tag{16}$$

We can state the learning objective of the autoencoder as follows:

$$\underset{\mathbf{E},\mathbf{D}}{\text{argmin}} \ \mathbf{D} \quad \mathbf{E} \quad W - W \tag{17}$$

under the conditions Eq. 8, 9, 15 and 16.

Now we have an autoencoder where input and output layers are the word embeddings and the hidden layer represents the synset vectors. A simplified version is shown in Figure 1. The tensors $\mathbf{E}$

and $\mathbf{D}$ have to be learned. They are rank 4 tensors of size $\approx 10^{15}$. However, we already discussed that they are very sparse, for two reasons: (i) We make the assumption that there is no interaction between dimensions. (ii) There are only few interactions between words and synsets (only when a lexeme exists). In practice, there are only $\approx 10^7$ elements to learn, which is technically feasible.

2.2 Matrix formalization

Based on the assumption that each dimension is fully independent from other dimensions, a separate autoencoder for each dimension can be created and trained in parallel. Let $W \quad \mathbb{R}^{|W| \times n}$ be a matrix where each row is a word embedding and $w^{(d)} = W_{\cdot,d}$ the d-th column of W, i.e., a vector that holds the d-th dimension of each word vector. In the same way, $s^{(d)} = S_{\cdot,d}$ holds the d-th dimension of each synset vector and $E^{(d)} = \mathbf{E}^{\cdot,d}_{\cdot,d} \quad \mathbb{R}^{|S| \times |W|}$. We can write $S = \mathbf{E} \quad W$ as:

$$s^{(d)} = E^{(d)} w^{(d)} \quad d \tag{18}$$

with $E^{(d)}_{i,j} = 0$ if $l^{(i,j)} = 0$. The decoding equation $\overline{W} = \mathbf{D} \quad S$ takes this form:

$$\overline{w}^{(d)} = D^{(d)} s^{(d)} \quad d \tag{19}$$

where $D^{(d)} = \mathbf{D}^{\cdot,d}_{\cdot,d} \quad \mathbb{R}^{|W| \times |S|}$ and $D^{(d)}_{j,i} = 0$ if $l^{(i,j)} = 0$. So E and D are symmetric in terms of non-zero elements. The learning objective becomes:

$$\underset{E^{(d)}, D^{(d)}}{\operatorname{argmin}} \; D^{(d)} E^{(d)} w^{(d)} - w^{(d)} \quad d \tag{20}$$

2.3 Lexeme embeddings

The hidden layer S of the autoencoder gives us synset embeddings. The lexeme embeddings are defined when transitioning from W to S, or more explicitly by:

$$l^{(i,j)} = E^{(i,j)} w^{(i)} \tag{21}$$

However, there is also a second lexeme embedding in AutoExtend when transitioning form S to $\overline{W}$:

$$\bar{l}^{(i,j)} = D^{(j,i)} s^{(j)} \tag{22}$$

Aligning these two representations seems natural, so we impose the following *lexeme constraints*:

$$\underset{E^{(i,j)}, D^{(j,i)}}{\operatorname{argmin}} \left\| E^{(i,j)} w^{(i)} - D^{(j,i)} s^{(j)} \right\| \quad i, j \tag{23}$$

	noun	verb	adj	adv
hypernymy	84,505	13,256	0	0
antonymy	2,154	1,093	4,024	712
similarity	0	0	21,434	0
verb group	0	1,744	0	0

Table 1: # of WN relations by part-of-speech

This can also be expressed dimension-wise. The matrix formulation is given by:

$$\underset{E^{(d)}, D^{(d)}}{\operatorname{argmin}} \left\| E^{(d)} \operatorname{diag}(w^{(d)}) - \left(D^{(d)} \operatorname{diag}(s^{(d)})\right)^T \right\| \quad d \tag{24}$$

with $\operatorname{diag}(x)$ being a square matrix having x on the main diagonal and vector $s^{(d)}$ defined by Eq. 18. While we try to align the embeddings, there are still two different lexeme embeddings. In all experiments reported in Section 4 we will use the average of both embeddings and in Section 4 we will analyze the weighting in more detail.

2.4 WN relations

Some WordNet synsets contain only a single word (lexeme). The autoencoder learns based on the synset constraints, i.e., lexemes being shared by different synsets (and also words); thus, it is difficult to learn good embeddings for single-lexeme synsets. To remedy this problem, we impose the constraint that *synsets related by WordNet (WN) relations should have similar embeddings*. Table 1 shows relations we used. WN relations are entered in a new matrix $R \quad \mathbb{R}^{r \times |S|}$, where r is the number of WN relation tuples. For each relation tuple, i.e., row in R, we set the columns corresponding to the first and second synset to 1 and -1, respectively. The values of R are not updated during training. We use a squared error function and 0 as target value. This forces the system to find similar values for related synsets. Formally, the *WN relation constraints* are:

$$\underset{E^{(d)}}{\operatorname{argmin}} \; R E^{(d)} w^{(d)} \quad d \tag{25}$$

2.5 Implementation

Our training objective is minimization of the sum of synset constraints (Eq. 20), weighted by α, the lexeme constraints (Eq. 24), weighted by β, and the WN relation constraints (Eq. 25), weighted by $1 - \alpha - \beta$.

The training objective cannot be solved analytically because it is subject to constraints Eq. 8,

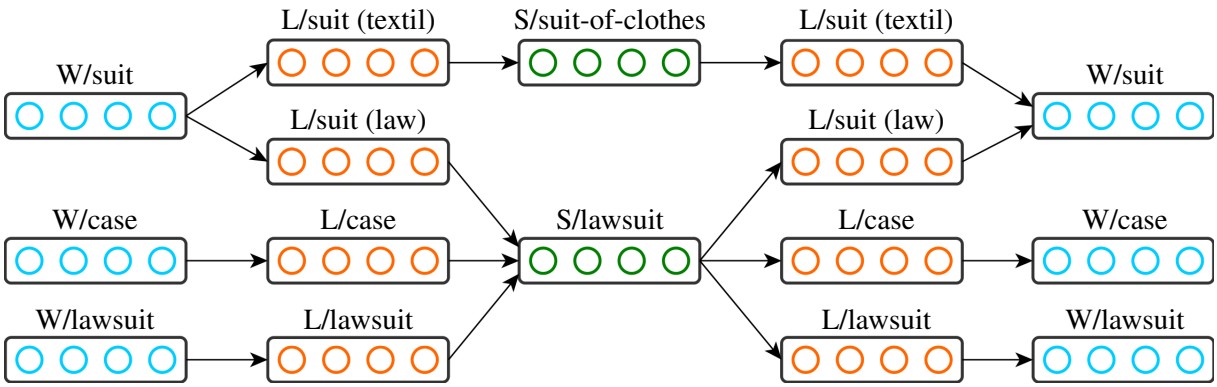

Figure 1: A small subgraph of WordNet. The circles are intended to show four different embedding dimensions. These dimensions are treated as independent. The synset constraints align the input and the output layer. The lexeme constraints align the second and fourth layers.

Eq. 9, Eq. 15 and Eq. 16. We therefore use backpropagation. We do not use regularization since we found that all learned weights are in $[-2, 2]$.

AutoExtend is implemented in MATLAB. We run 1000 iterations of gradient descent. On an Intel Xeon CPU E7-8857 v2 3.00GHz, one iteration on one dimension takes less than a minute because the gradient computation ignores zero entries in the matrix.

2.6 Column normalization

Our model is based on the premise that a word is the sum of its lexemes (Eq. 1). From the definition of $E^{(i,j)}$, we derived that $\mathbf{E} \quad \mathsf{R}^{|S|\times n\times|W|\times n}$ is normalized over the first dimension (Eq. 8). So $E^{(d)} \quad \mathsf{R}^{|S|\times|W|}$ is also normalized over the first dimension. In other words, $E^{(d)}$ is a column normalized matrix. Another premise of the model is that a synset is the sum of its lexemes. Therefore, $D^{(d)}$ is also column normalized. A simple way to implement this is to start the computation with column normalized matrices and normalize them again after each iteration as long as the error function still decreases. When the error function starts increasing, we stop normalizing the matrices and continue with a normal gradient descent. This respects that while $E^{(d)}$ and $D^{(d)}$ should be column normalized in theory, there are a lot of practical issues that prevent this, e.g., OOV words.

3 Data, experiments and evaluation

We downloaded 300-dimensional embeddings for 3,000,000 words and phrases trained on Google News, a corpus of $\approx 10^{11}$ tokens, using word2vec CBOW (Mikolov et al., 2013c). Many words in the word2vec vocabulary are not in WordNet, e.g., inflected forms (*cars*) and proper nouns (*Tony Blair*). Conversely, many WordNet lemmas are not in the word2vec vocabulary, e.g., *42* (digits were converted to 0). This results in a number of empty synsets (see Table 2). Note however that AutoExtend can produce embeddings for empty synsets because we use WN relation constraints in addition to synset and lexeme constraints.

We run AutoExtend on the word2vec vectors. As we do not know anything about a suitable weighting for the three different constraints, we set $\alpha = \beta = 0.33$. Our main goal is to produce compatible embeddings for lexemes and synsets. Thus, we can compute nearest neighbors across all three data types as shown in Figure 2.

We evaluate the embeddings on WSD and on similarity performance. Our results depend directly on the quality of the underlying word embeddings, in our case word2vec embeddings. We would expect even better evaluation results as word representation learning methods improve. Using a new and improved set of underlying embeddings is simple: it is a simple switch of the input file that contains the word embeddings.

3.1 Word Sense Disambiguation

For WSD we use the shared tasks of Senseval-2 (Kilgarriff, 2001) and Senseval-3 (Mihalcea et al., 2004) and a system named IMS (Zhong and

	WordNet	∩ word2vec
words	147,478	54,570
synsets	117,791	73,844
lexemes	207,272	106,167

Table 2: # of items in WordNet and after intersection with word2vec vectors

nearest neighbors of W/suit
S/suit (businessman), L/suit (businessman), L/accomodate, S/suit (be acceptable), L/suit (be acceptable), L/lawsuit, W/lawsuit, S/suit (playing card), L/suit (playing card), S/suit (petition), S/lawsuit, W/countersuit, W/complaint, W/counterclaim
nearest neighbors of W/lawsuit
L/lawsuit, S/lawsuit, S/countersuit, L/countersuit, W/countersuit, W/suit, W/counterclaim, S/counterclaim (n), L/counterclaim (n), S/counterclaim (v), L/counterclaim (v), W/sue, S/sue (n), L/sue (n)
nearest neighbors of S/suit-of-clothes
L/suit-of-clothes, S/zoot-suit, L/zoot-suit, W/zoot-suit, S/garment, L/garment, S/dress, S/trousers, L/pinstripe, L/shirt, W/tuxedo, W/gabardine, W/tux, W/pinstripe

Figure 2: Five nearest word (W/), lexeme (L/) and synset (S/) neighbors for three items, ordered by cosine

Ng, 2010). Senseval-2 contains 139, Senseval-3 57 different words. They provide 8,611, respectively 8,022 training instances and 4,328, respectively 3,944 test instances. For the system, we use the same setting as in the original paper. Preprocessing consists of sentence splitting, tokenization, POS tagging and lemmatization; the classifier is a linear SVM. In our experiments (Table 3), we run IMS with *each feature set by itself* to assess the relative strengths of feature sets (lines 1–7) and on *feature set combinations* to determine which combination is best for WSD (lines 8, 12–15).

IMS implements three standard WSD feature sets: part of speech (POS), surrounding word and local collocation (lines 1–3).

Let w be an ambiguous word with k senses. The three feature sets on lines 5–7 are based on the AutoExtend embeddings $s^{(j)}$, $1 \leq j \leq k$, of the synsets of w and the centroid c of the sentence in which w occurs. The centroid is simply the sum of all word2vec vectors of the words in the sentence, excluding stop words.

The **S-cosine** feature set consists of the k cosines of centroid and synset vectors:

$$< \cos(c, s^{(1)}), \cos(c, s^{(2)}), \ldots, \cos(c, s^{(k)}) >$$

The **S-product** feature set consists of the nk element-wise products of centroid and synset vectors:

$$< c_1 s_1^{(1)}, \ldots, c_n s_n^{(1)}, \ldots, c_1 s_1^{(k)}, \ldots, c_n s_n^{(k)} >$$

where c_i (resp. $s_i^{(j)}$) is element i of c (resp. $s^{(j)}$). The idea is that we let the SVM estimate how important each dimension is for WSD instead of giving all equal weight as in S-cosine.

The **S-raw** feature set simply consists of the $n(k+1)$ elements of centroid and synset vectors:

$$< c_1, \ldots, c_n, s_1^{(1)}, \ldots, s_n^{(1)}, \ldots, s_1^{(k)}, \ldots, s_n^{(k)} >$$

Our main goal is to determine if AutoExtend features improve WSD performance when added to standard WSD features. To make sure that improvements we get are not solely due to the power of word2vec, we also investigate a simple word2vec baseline. For S-product, the AutoExtend feature set that performs best in the experiment (cf. lines 6 and 14), we test the alternative word2vec-based **S_{naive}-product** feature set. It has the same definition as S-product except that we replace the synset vectors $s^{(j)}$ with naive synset vectors $z^{(j)}$, defined as the sum of the word2vec vectors of the words that are members of synset j.

Lines 1–7 in Table 3 show the performance of each feature set by itself. We see that the synset feature sets (lines 5–7) have a comparable performance to standard feature sets. S-product is the strongest of them.

Lines 8–16 show the performance of different feature set combinations. MFS (line 8) is the most frequent sense baseline. Lines 9&10 are the winners of Senseval. The standard configuration of IMS (line 11) uses the three feature sets on lines 1–3 (POS, surrounding word, local collocation) and achieves an accuracy of 65.2% on the English lexical sample task of Senseval-2 and 72.3% on Senseval-3.[1] Lines 12–16 add one additional feature set to the IMS system on line 11; e.g., the system on line 14 uses POS, surrounding word, local collocation and S-product feature sets. The system on line 14 outperforms all previous systems, most of them significantly. While S-raw performs quite reasonably as a feature set alone, it hurts the performance when used as an additional feature set. As this is the feature set that contains the largest number of features ($n(k+1)$), overfitting is the likely reason. Conversely, S-cosine only adds k features and therefore may suffer from underfitting.

We do a grid search (step size .1) for optimal values of α and β, optimizing the average score of Senseval-2 and Senseval-3. The best performing feature set combination is **$S_{optimized}$-product** with

[1] Zhong and Ng (2010) report accuracies of 65.3% / 72.6% for this configuration.

[†] In Table 3 and Table 4, results significantly worse than the best (bold) result in each column are marked † for $\alpha = .05$ and ‡ for $\alpha = .10$ (one-tailed Z-test).

			Senseval-2	Senseval-3
IMS feature sets	1	POS	53.6	58.0
	2	surrounding word	57.6	65.3
	3	local collocation	58.7	64.7
	4	S_{naive}-product	56.5	62.2
	5	S-cosine	55.5	60.5
	6	S-product	58.3	64.3
	7	S-raw	56.8	63.1
system comparison	8	MFS	$47.6^{\dagger}$	$55.2^{\dagger}$
	9	Rank 1 system	$64.2^{\dagger}$	72.9
	10	Rank 2 system	$63.8^{\dagger}$	72.6
	11	IMS	$65.2^{\ddagger}$	$72.3^{\ddagger}$
	12	IMS + S_{naive}-prod.	$62.6^{\dagger}$	$69.4^{\dagger}$
	13	IMS + S-cosine	$65.1^{\ddagger}$	$72.4^{\ddagger}$
	14	IMS + S-product	**66.5**	**73.6**
	15	IMS + S-raw	$62.1^{\dagger}$	$66.8^{\dagger}$
	16	IMS + $S_{optimized}$-prod.	*66.6*	*73.6*

Table 3: WSD accuracy for different feature sets and systems. Best result (excluding line 16) in each column in bold.

$\alpha = 0.2$ and $\beta = 0.5$, with only a small improvement (line 16).

The main result of this experiment is that we achieve an improvement of more than 1% in WSD performance when using AutoExtend.

3.2 Synset and lexeme similarity

We use SCWS (Huang et al., 2012) for the similarity evaluation. SCWS provides not only isolated words and corresponding similarity scores, but also a context for each word. SCWS is based on WordNet, but the information as to which synset a word in context came from is not available. However, the dataset is the closest we could find for sense similarity. Synset and lexeme embeddings are obtained by running AutoExtend. Based on the results of the WSD task, we set $\alpha = 0.2$ and $\beta = 0.5$. Lexeme embeddings are the natural choice for this task as human subjects are provided with two words and a context for each and then have to assign a similarity score. But for completeness, we also run experiments for synsets.

For each word, we compute a context vector c by adding all word vectors of the context, excluding the test word itself. Following Reisinger and Mooney (2010), we compute the lexeme (resp. synset) vector l either as the simple average of the lexeme (resp. synset) vectors $l^{(ij)}$ (method AvgSim, no dependence on c in this case) or as the average of the lexeme (resp. synset) vectors weighted by cosine similarity to c (method AvgSimC).

Table 4 shows that AutoExtend lexeme embeddings (line 7) perform better than previous work,

		AvgSim	AvgSimC
1	Huang et al. (2012)	$62.8^{\dagger}$	$65.7^{\dagger}$
2	Tian et al. (2014)	–	$65.4^{\dagger}$
3	Neelakantan et al. (2014)	67.2	69.3
4	Chen et al. (2014)	$66.2^{\dagger}$	68.9
5	words (word2vec)	$66.6^{\ddagger}$	$66.6^{\dagger}$
6	synsets	$62.6^{\dagger}$	$63.7^{\dagger}$
7	lexemes	**68.9**	**69.8**

Table 4: Spearman correlation ($\rho \times 100$) on SCWS. Best result per column in bold.

including (Huang et al., 2012) and (Tian et al., 2014). Lexeme embeddings perform better than synset embeddings (lines 7 vs. 6), presumably because using a representation that is specific to the actual word being judged is more precise than using a representation that also includes synonyms.

A simple baseline is to use the underlying word2vec embeddings directly (line 5). In this case, there is only one embedding, so there is no difference between AvgSim and AvgSimC. It is interesting that even if we do not take the context into account (method AvgSim) the lexeme embeddings outperform the original word embeddings. As AvgSim simply adds up all lexemes of a word, this is equivalent to the constraint we proposed in the beginning of the paper (Eq. 1). Thus, replacing a word's embedding by the sum of the embeddings of its senses could generally improve the quality of embeddings (cf. Huang et al. (2012) for a similar point). We will leave a deeper evaluation of this topic for future work.

4 Analysis

We first look at the impact of the parameters α, β (Section 2.5) that control the weighting of synset constraints vs lexeme constraints vs WN relation constraints. We investigate the impact for three different tasks. **WSD-alone:** accuracy of IMS (average of Senseval-2 and Senseval-3) if only S-product is used as a feature set (line 6 in Table 3). **WSD-additional:** accuracy of IMS (average of Senseval-2 and Senseval-3) if S-product is used together with the feature sets POS, surrounding word and local collocation (line 14 in Table 3). **SCWS:** Spearman correlation on SCWS (line 7 in Table 4).

For WSD-alone (Figure 3, center), the best performing weightings (red) all have high weights for WN relations and are therefore at the top of triangle. Thus, WN relations are very important for WSD-alone and adding more weight to the

synset and lexeme constraints does not help. However, all three constraints are important in WSD-additional: the red area is in the middle (corresponding to nonzero weights for all three constraints) in the left panel of Figure 3. Apparently, strongly weighted lexeme and synset constraints enable learning of representations that in their interaction with standard WSD feature sets like local collocation increase WSD performance. For SCWS (right panel), we should not put too much weight on WN relations as they artificially bring related, but not similar lexemes together. So the maximum for this task is located in the lower part of the triangle.

The main result of this analysis is that AutoExtend never achieves its maximum performance when using only one set of constraints. All three constraints are important – synset, lexeme and WN relation constraints – with different weights for different applications.

We also analyzed the impact of the four different WN relations (see Table 1) on performance. In Table 3 and Table 4, all four WN relations are used together. We found that any combination of three relation types performs worse than using all four together. A comparison of different relations must be done carefully as they differ in the POS they affect and in quantity (see Table 1). In general, relation types with more relations outperformed relation types with fewer relations.

Finally, the relative weighting of $l^{(i,j)}$ and $\bar{l}^{(i,j)}$ when computing lexeme embeddings is also a parameter that can be tuned. We use simple averaging ($\theta = 0.5$) for all experiments reported in this paper. We found only small changes in performance for $0.2 \leq \theta \leq 0.8$.

5 Resources other than WordNet

AutoExtend is broadly applicable to lexical and knowledge resources that have certain properties. While we only run experiments with WordNet in this paper, we will briefly address other resources. For *Freebase* (Bollacker et al., 2008), we could replace the synsets with Freebase entities. Each entity has several aliases, e.g. Barack Obama, President Obama, Obama. The role of words in WordNet would correspond to these aliases in Freebase. This will give us the synset constraint, as well as the lexeme constraint of the system. Relations are given by Freebase types; e.g., we can add a constraint that entity embeddings of the type "President of the US" should be similar.

To explorer multilingual word embeddings we require the word embeddings of different languages to live in the same vector space, which can easily be achieved by training a transformation matrix L between two languages using known translations (Mikolov et al., 2013b). Let X be a matrix where each row is a word embedding in language 1 and Y a matrix where each row is a word embedding in language 2. For each row the words of X and Y are a translation of each other. We then want to minimize the following objective:

$$\underset{L}{\text{argmin}} \; LX - Y \tag{26}$$

We can use a gradient descent to solve this but a matrix inversion will run faster. The matrix L is given by:

$$L = (X^T \quad X)^{-1}(X^T \quad Y) \tag{27}$$

The matrix L can be used to transform unknown embeddings into the new vector space, which enables us to use a multilingual WordNet like *BabelNet* (Navigli and Ponzetto, 2010) to compute synset embeddings. We can add cross-linguistic relationships to our model, e.g., aligning German and English synset embeddings of the same concept.

6 Related Work

Rumelhart et al. (1988) introduced distributed word representations, usually called word embeddings today. There has been a resurgence of work on them recently (e.g., Bengio et al. (2003) Mnih and Hinton (2007), Collobert et al. (2011), Mikolov et al. (2013a), Pennington et al. (2014)). These models produce only a single embedding for each word. All of them can be used as input for AutoExtend.

There are several approaches to finding embeddings for senses, variously called meaning, sense and multiple word embeddings. Schütze (1998) created sense representations by clustering context representations derived from co-occurrence. The representation of a sense is simply the centroid of its cluster. Huang et al. (2012) improved this by learning single-prototype embeddings before performing word sense discrimination on them. Bordes et al. (2011) created similarity measures for relations in WordNet and Freebase to learn entity embeddings. An energy based model was

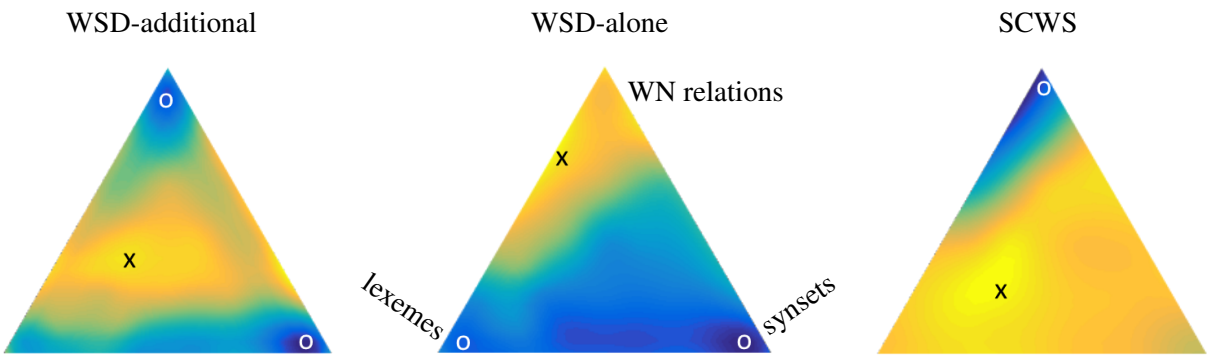

Figure 3: Performance of different weightings of the three constraints (WN relations:top, lexemes:left, synsets:right) on the three tasks WSD-additional, WSD-alone and SCWS. "x" indicates the maximum; "o" indicates a local minimum.

proposed by Bordes et al. (2012) to create disambiguated meaning embeddings and Neelakantan et al. (2014) and Tian et al. (2014) extended the Skip-gram model (Mikolov et al., 2013a) to learn multiple word embeddings. While these embeddings can correspond to different word senses, there is no clear mapping between them and a lexical resource like WordNet. Chen et al. (2014) also modified word2vec to learn sense embeddings, each corresponding to a WordNet synset. They use glosses to initialize sense embedding, which in turn can be used for WSD. The sense disambiguated data can again be used to improve sense embeddings.

This prior work needs a training step to learn embeddings. In contrast, we can "AutoExtend" any set of given word embeddings – without (re)training them.

There is only little work on taking existing word embeddings and producing embeddings in the same space. Labutov and Lipson (2013) tuned existing word embeddings in supervised training, not to create new embeddings for senses or entities, but to get better predictive performance on a task while not changing the space of embeddings.

Lexical resources have also been used to improve word embeddings. In the Relation Constrained Model, Yu and Dredze (2014) use word2vec to learn embeddings that are optimized to predict a related word in the resource, with good evaluation results. Bian et al. (2014) used not only semantic, but also morphological and syntactic knowledge to compute more effective word embeddings.

Another interesting approach to create sense specific word embeddings uses bilingual resources (Guo et al., 2014). The downside of this approach is that parallel data is needed.

We used the SCWS dataset for the word similarity task, as it provides a context. Other frequently used datasets are WordSim-353 (Finkelstein et al., 2001) or MEN (Bruni et al., 2014).

And while we use cosine to compute similarity between synsets, there are also a lot of similarity measures that only rely on a given resource, mostly WordNet. These measures are often functions that depend on the provided information like gloss or the topology like shortest-path. Examples include (Wu and Palmer, 1994) and (Leacock and Chodorow, 1998); Blanchard et al. (2005) give a good overview.

7 Conclusion

We presented AutoExtend, a flexible method to learn synset and lexeme embeddings from word embeddings. It is completely general and can be used for any other set of embeddings and for any other resource that imposes constraints of a certain type on the relationship between words and other data types. Our experimental results show that AutoExtend achieves state-of-the-art performance on word similarity and word sense disambiguation. Along with this paper, we will publish AutoExtend for extending word embeddings to other data types; the lexeme and synset embeddings used in the experiments; and the code needed to replicate our WSD evaluation[2].

Acknowledgments

This work was partially funded by Deutsche Forschungsgemeinschaft (DFG SCHU 2246/2-2). We are grateful to Christiane Fellbaum for discussions leading up to this paper and to the anonymous reviewers for their comments.

[2] http://cistern.cis.lmu.de/

References

Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Knowledge-powered deep learning for word embedding. In *Proceedings of ECML PKDD*.

Emmanuel Blanchard, Mounira Harzallah, Henri Briand, and Pascale Kuntz. 2005. A typology of ontology-based semantic measures. In *Proceedings of EMOI - INTEROP*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of ACM SIGMOD*.

Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of AAAI*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2012. Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of AISTATS*.

Elia Bruni, Nam Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *Journal of Artificial Intelligence Research*, 49(1):1–47.

Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. 2014. A unified model for word sense representation and disambiguation. In *Proceedings of EMNLP*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of WWW*.

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Learning sense-specific word embeddings by exploiting bilingual resources. In *Proceedings of Coling, Technical Papers*.

Iryna Gurevych. 2005. Using the structure of a conceptual network in computing semantic relatedness. In *IJCNLP*.

Birgit Hamp, Helmut Feldweg, et al. 1997. Germanet-a lexical-semantic net for german. In *Proceedings of ACL, Workshops*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2014. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *arXiv preprint arXiv:1408.3456*.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of ACL*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of ACL*.

Adam Kilgarriff. 2001. English lexical sample task description. In *Proceedings of SENSEVAL-2*.

Igor Labutov and Hod Lipson. 2013. Re-embedding words. In *Proceedings of ACL*.

Claudia Leacock and Martin Chodorow. 1998. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283.

Minh-Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of CoNLL*.

Rada Mihalcea, Timothy Chklovski, and Adam Kilgarriff. 2004. The senseval-3 english lexical sample task. In *Proceedings of SENSEVAL-3*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013b. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*.

George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of ICML*.

Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *Proceedings of NIPS*.

Roberto Navigli and Simone Paolo Ponzetto. 2010. Babelnet: Building a very large multilingual semantic network. In *Proceedings of ACL*.

Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. 2014. Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of EMNLP*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*.

Joseph Reisinger and Raymond J Mooney. 2010. Multi-prototype vector-space models of word meaning. In *Proceedings of NAACL*.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive Modeling*, 5:213–220.

Hinrich Schütze. 1998. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123.

Fei Tian, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tie-Yan Liu. 2014. A probabilistic model for learning multi-prototype word embeddings. In *Proceedings of Coling, Technical Papers*.

Zhibiao Wu and Martha Palmer. 1994. Verbs semantics and lexical selection. In *Proceedings of ACL*.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of ACL*.

Torsten Zesch and Iryna Gurevych. 2006. Automatically creating datasets for measures of semantic relatedness. In *Proceedings of the Workshop on Linguistic Distances*.

Zhi Zhong and Hwee Tou Ng. 2010. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of ACL, System Demonstrations*.

Chapter 4

Ultradense Word Embeddings by Orthogonal Transformation

Ultradense Word Embeddings by Orthogonal Transformation

Sascha Rothe and **Sebastian Ebert** and **Hinrich Schütze**
Center for Information and Language Processing
LMU Munich, Germany
{sascha|ebert}@cis.lmu.de

Abstract

Embeddings are *generic* representations that are useful for many NLP tasks. In this paper, we introduce DENSIFIER, a method that learns an *orthogonal transformation* of the embedding space that focuses the information relevant for a task in an *ultradense subspace* of a dimensionality that is smaller by a factor of 100 than the original space. We show that ultradense embeddings generated by DENSIFIER reach state of the art on a lexicon creation task in which words are annotated with three types of lexical information – sentiment, concreteness and frequency. On the SemEval2015 10B sentiment analysis task we show that no information is lost when the ultradense subspace is used, but training is an order of magnitude *more efficient* due to the compactness of the ultradense space.

1 Introduction

Embeddings are useful for many tasks, including word similarity (e.g., Pennington et al. (2014)), named entity recognition (NER) (e.g., Collobert et al. (2011)) and sentiment analysis (e.g., Kim (2014), Kalchbrenner et al. (2014), Severyn and Moschitti (2015)). Embeddings are generic representations, containing different types of information about a word. Statistical models can be trained to make best use of these generic representations for a specific application like NER or sentiment analysis (Ebert et al., 2015).

Our hypothesis in this paper is that the information useful for any given task is contained in an *ultradense subspace* E_u. We propose the new method DENSIFIER to identify E_u. Given a set of word embeddings, DENSIFIER learns an *orthogonal transformation* of the original space E_o on a task-specific training set. The orthogonality of the transformation can be considered a hard regularizer.

The benefit of the proposed method is that embeddings are most useful if learned on unlabeled corpora and performance-enhanced on a broad array of tasks. This means we should try to keep all information offered by them. Orthogonal transformations "reorder" the space without adding or removing information and preserve the bilinear form, i.e., Euclidean distance and cosine. The transformed embeddings concentrate all information relevant for the task in E_u.

The benefits of E_u compared to E_o are (i) *high-quality* and (ii) *efficient* representations. (i) DENSIFIER moves non-task-related information outside of E_u, i.e., into the orthogonal complement of E_u. As a result, E_u provides *higher-quality representations* for the task than E_o; e.g., noise that could result in overfitting is reduced in E_u compared to E_o. (ii) E_u has a *dimensionality smaller by a factor of 100* in our experiments. As a result, training statistical models on these embeddings is much faster. These models also have many fewer parameters, thus again helping to prevent overfitting, especially for complex, deep neural networks.

We show the benefits of ultradense representations in two text polarity classification tasks (SemEval2015 Task 10B, Czech movie reviews).

In the most extreme form, ultradense representations – i.e., E_u – have a single dimension. We exploit this for creating lexicons in which words are

annotated with lexical information, e.g., with sentiment. Specifically, we create high-coverage lexicons with up to 3 million words (i) for three lexical properties: for *sentiment*, *concreteness* and *frequency*; (ii) for five languages: *Czech*, *English*, *French*, *German* and *Spanish*; (iii) for two domains, *Twitter* and *News*, in a domain adaptation setup.

The main advantages of this method of lexicon creation are: (i) We need a training lexicon of only a few hundred words, thus making our method effective for new domains and languages and requiring only a minimal manual annotation effort. (ii) The method is applicable to any set of embeddings, including phrase and sentence embeddings. Assuming the availability of a small hand-labeled lexicon, DENSIFIER automatically creates a domain dependent lexicon based on a set of embeddings learned on a large corpus of the domain. (iii) While the input lexicon is *discrete* – e.g., positive (+1) and negative (-1) sentiment – the output lexicon is *continuous* and this more fine-grained assessment is potentially more informative than a simple binary distinction.

We show that lexicons created by DENSIFIER beat the state of the art on SemEval2015 Task 10E (determining association strength).

One of our goals is to make embeddings more interpretable. The work on sentiment, concreteness and frequency we present in this paper is a first step towards a *general decomposition of embedding spaces into meaningful, dense subspaces*. This would lead to cleaner and more easily interpretable representations – as well as representations that are more effective and efficient.

2 Model

Let Q $\mathsf{R}^{d \times d}$ be an orthogonal matrix that transforms the original word embedding space into a space in which certain types of information are represented by a small number of dimensions. Concretely, we learn Q such that the dimensions D^s $\{1, \ldots, d\}$ of the resulting space correspond to a word's sentiment information and the $\{1, \ldots, d\} \setminus D^s$ remaining dimensions correspond to non-sentiment information. Analogously, the sets of dimensions D^c and D^f correspond to a word's concreteness information and frequency information, respectively. In this paper, we assume that these properties do not correlate and therefore the ultradense subspaces do not overlap, e.g., $D^s \cap D^c =$. However, this might not be true for other settings, e.g., sentiment and semantic information.

If e_w E_o R^d is the original embedding of word w, the transformed representation is Qe_w. We use as a placeholder for s, c and f and call $d = |D|$ the dimensionality of the ultradense subspace of . For each ultradense subspace, we create P $\mathsf{R}^{d \times d}$, an identity matrix for the dimensions in D $\{1, \ldots, d\}$. Thus, the ultradense representation u_w E_u R^d of e_w is defined as:

$$u_w := P\ Qe_w \tag{1}$$

2.1 Separating Words of Different Groups

We assume to have a lexicon resource l in which each word w is annotated for a certain information as either $l(w) = +1$ (positive, concrete, frequent) or $l(w) = -1$ (negative, abstract, infrequent). Let L be a set of word index pairs (v, w) for which $l(v) = l(w)$ holds. We want to maximize:

$$\sum_{(v,w)\ L} u_v - u_w \tag{2}$$

Thus, our objective is given by:

$$\operatorname*{argmax}_Q \sum_{(v,w)\ L} P\ Q(e_w - e_v) \tag{3}$$

or, equivalently, by:

$$\operatorname*{argmin}_Q \sum_{(v,w)\ L} -\ P\ Q(e_w - e_v) \tag{4}$$

subject to Q being an orthogonal matrix.

2.2 Aligning Words of the Same Group

Another goal is to minimize the distance of two words of the same group. Let L be a set of word index pairs (v, w) for which $l(v) = l(w)$ holds. In contrast to Eq. 3, we now want to minimize each distance. Thus, the objective is given by:

$$\operatorname*{argmin}_Q \sum_{(v,w)\ L} P\ Q(e_w - e_v) \tag{5}$$

subject to Q being an orthogonal matrix.

The intuition behind the two objectives is graphically depicted in Figure 1.

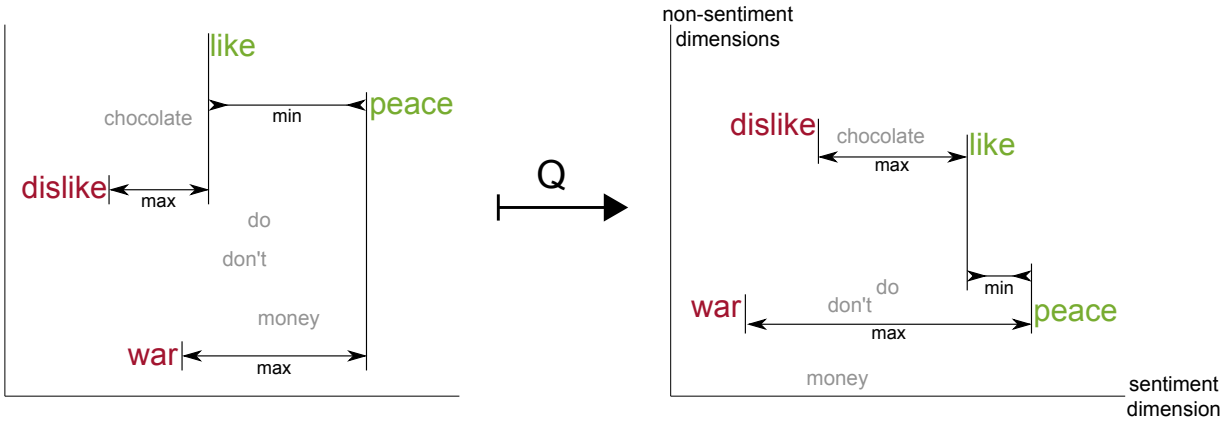

Figure 1: The original word embedding space (left) and the transformed embedding space (right). The training objective for Q is to *minimize* the distances in the sentiment dimension between words of the same group (e.g., positive/green: "like" & "peace") and to *maximize* the distances between words of different groups (e.g., negative/red & positive/green: "war" & "peace"; not necessarily antonyms).

2.3 Training

We combine the two objectives in Eqs. 3/5 for each subspace, i.e., for sentiment, concreteness and frequency, and weight them with α and $1-\alpha$. Hence, there is one hyperparameter α for each subspace. We then perform stochastic gradient descent (SGD). Batch size is 100 and starting learning rate is 5, multiplied by .99 in each iteration.

2.4 Orthogonalization

Each step of SGD updates Q. The updated matrix Q is in general no longer orthogonal. We therefore reorthogonalize Q in each step based on singular value decomposition:

$$Q = USV^T$$

where S is a diagonal matrix, and U and V are orthogonal matrices. The matrix

$$Q := UV^T$$

is the nearest orthogonal matrix to Q in both the 2-norm and the Frobenius norm (Fan and Hoffman, 1955). (Formalizing our regularization directly as projected gradient descent would be desirable. However, gradient descent includes an additive operation and orthogonal matrices are not closed under summation.)

SGD for this problem is sensitive to the learning rate. If the learning rate is too large, a large jump results and the reorthogonalized matrix Q basically is a random new point in the parameter space. If the learning rate is too small, then learning can take long. We found that our training regime of starting at a high learning rate (5) and multiplying by .99 in each iteration is effective. Typically, the cost initially stays about constant (random jumps in parameter space), then cost steeply declines in a small number of about 50 iterations (sweet spot); the curve flattens after that. Training Q took less than 5 minutes per experiment for all experiments in this paper.

3 Lexicon Creation

For lexicon creation, the input is a set of *embeddings* and a lexicon *resource* l, in which words are annotated for a lexical information such as sentiment, concreteness or frequency. DENSIFIER is then trained to produce a *one-dimensional ultra-dense subspace*. The output is an *output lexicon*. It consists of all words covered by the embedding set, each associated with its one-dimensional ultra-dense subspace representation (which is simply a real number), an indicator of the word's strength for that information.

The embeddings and lexicon resources used in this paper cover five languages and three domains (Table 1). The Google News embeddings for English[1] and the FrWac embeddings for French[2] are

[1] https://code.google.com/p/word2vec/
[2] http://fauconnier.github.io/

				train			test			
		#tokens	#types	resource	∩	#words	resource	∩	#words	τ
1	sent CZ web	2.44	3.3	SubLex 1.0	2,492	4,125	SubLex 1.0	319	500	.580
2	sent DE web	1.34	8.0	German PC	10,718	37,901	German PC	573	1,000	.654
3	sent ES web	0.37	3.7	full-strength	824	1,147	full-strength	185	200	.563
4	sent FR web	0.12	1.6	FEEL	7,496	10,979	FEEL	715	1,000	.544
5	sent EN twitter	3.34	5.4	WHM all	12,601	19,329	Trial 10E	198	200	.661
6	sent EN news	3.00	100.0	WHM train	7,633	10,270	WHM val	952	1,000	.622
7	conc EN news	3.00	100.0	BWK	14,361	29,954	BWK	8,694	10,000	.623
8	freq EN news	3.00	100.0	word2vec order	4,000	4,000	word2vec order	1,000	1,000	.361
9	freq FR web	0.12	1.6	word2vec order	4,000	4,000	word2vec order	1,000	1,000	.460

Table 1: Results for lexicon creation. #tokens: size of embedding training corpus (in billion). #types: size of output lexicon (in million). For each resource, we give its size ("#words") and the size of the intersection of resource and embedding set ("∩"). Kendall's τ is computed on "∩".

publicly available. We use word2vec to train 400-dimensional embeddings for English on a 2013 Twitter corpus of size 5.4 $\times 10^9$. For Czech, German and Spanish, we train embeddings on web data of sizes 3.3, 8.0 and 3.8 $\times 10^9$, respectively. We use the following lexicon resources for sentiment: SubLex 1.0 (Veselovská and Bojar, 2013) for Czech; WHM for English [the combination of MPQA (Wilson et al., 2005), Opinion Lexicon (Hu and Liu, 2004) and NRC Emotion lexicons (Mohammad and Turney, 2013)]; FEEL (Abdaoui et al., 2014) for French; German Polarity Clues (Waltinger, 2010) for German; and the sentiment lexicon of Pérez-Rosas et al. (2012) for Spanish. For concreteness, we use BWK, a lexicon of 40,000 English words (Brysbaert et al., 2014). For frequency, we exploit the fact that word2vec stores words in frequency order; thus, the ranking provided by word2vec is our lexicon resource for frequency.

For a resource/embedding-set pair (l, E), we intersect the vocabulary of l with the top 80,000 words of E to filter out noisy, infrequent words that tend to have low quality embeddings and we do not want them to introduce noise when training the transformation matrix.

For the sentiment and concreteness resources, $l(w) \in \{-1, 1\}$ for all words w covered. We create a resource l^f for frequency by setting $l^f(w) = 1$ for the 2000 most frequent words and $l^f(w) = -1$ for words at ranks 20000-22000. 1000 words randomly selected from the 5000 most frequent are the test set.[3] We designate three sets of dimensions D^s, D^c and D^f to represent sentiment, concreteness and frequency, respectively, and arbitrarily set (i) $D^c = \{1\}$ for English and $D^c =$ for the other languages since we do not have concreteness resources for them, (ii) $D^s = \{1\}$ and (iii) $D^f = \{21\}$. Referring to the lines in Table 1, we then learn six orthogonal transformation matrices Q: for CZ-web (1), DE-web (2), ES-web (3), FR-web (4, 9), EN-twitter (5) and EN-news (6, 7, 8).

4 Evaluation

4.1 Top-Ranked Words

Table 2 shows the top 10 positive/negative words (i.e., most extreme values on dimension D^s) when we apply the transformation to the corpora EN-twitter, EN-news and DE-web and the top 10 concrete/abstract words (i.e., most extreme values on dimension D^c) for EN-news. For EN-twitter (leftmost double column), the selected words look promising: they contain highly domain-specific words such as hashtags (e.g., #happy). This is surprising because there is not a single hashtag in the lexicon resource WHM that DENSIFIER was trained on. Results for the other three double columns show likewise extreme examples for the corresponding information and language. This initial evaluation indicates that our method effectively learns high quality lexicons for new domains. Figure 3 depicts values for selected words for the three properties. Illustrative examples are "brother" / "brotherhood" for concreteness and "hate" / "love" for sentiment.

[3] The main result of the frequency experiment below is that τ is low even in a setup that is optimistic due to train/test overlap; presumably it would be even lower without overlap.

EN-twitter		EN-news		EN-news		DE-web	
positive	negative	positive	negative	concrete	abstract	positive	negative
#blessed	rape	expertise	angry	tree	fundamental	herzlichen	gesperrt
inspiration	racist	delighted	delays	truck	obvious	kenntnisse	droht
blessed	horrible	honored	worse	kitchen	legitimate	hervorragende	verurteilt
inspiring	nasty	thank	anger	dog	reasonable	ideale	gefahr
foundation	jealousy	wonderful	foul	bike	optimistic	bestens	falsche
provide	murder	commitment	blamed	bat	satisfied	glückwunsch	streit
wishes	waste	affordable	blame	garden	surprising	optimale	angst
dedicated	mess	passion	complained	homer	honest	anregungen	krankheit
offers	disgusting	exciting	bad	bed	regard	freuen	falschen
#happy	spam	flexibility	deaths	gallon	extraordinary	kompetenzen	verdacht

Table 2: Top 10 words in the output lexicons for the domains Twitter and News (English) and Web (German).

4.2 Quality of Predictions

Table 1 presents experimental results. In each case, we split the resource into train/test, except for Twitter where we used the trial data of SemEval2015 Task 10E for test. We train DENSIFIER on train and compute Kendall's τ on test. The size of the lexicon resource has no big effect; e.g., results for Spanish (small resource; line 3) and French (large resource; line 4) are about the same. See Section 5.2 for a more detailed analysis of the effect of resource size.

The quality of the output lexicon depends strongly on the quality of the underlying word embeddings; e.g., results for French (small embedding training corpus; line 4) are worse than results for English (large embedding training corpus; line 6) even though the lexicon resources have comparable size.

In contrast to sentiment/concreteness, τ values for frequency are low (lines 8-9). For the other three languages we obtain τ [.34, .46] for frequency (not shown). This suggests that word embeddings represent sentiment and concreteness much better than frequency. The reason for this likely is the learning objective of word embeddings: modeling the context. Infrequent words can occur in frequent contexts. Thus, the frequency information in a single word embedding is limited. In contrast negative words are likely to occur in negative contexts.

The nine output lexicons in Table 1 – each a list of words annotated with predicted strength on one of three properties – are available at `www.cis.lmu.de/~sascha/Ultradense/`.

		τ	
	system	all	$\cap$
1	Amir et al. (2015)	.626†	
2	Hamdan et al. (2015)	.621†	
3	Zhang et al. (2015)	.591†	
4	Özdemir and Bergler (2015)	.584†	
5	Plotnikova et al. (2015)	.577†	
6	DENSIFIER	**.654**	**.650**
7	Sentiment140	.508†	.538†
8	DENSIFIER, trial only	.627†	

Table 3: Results for Lexicon Creation. The first column gives the correlation with the entire test lexicon of SemEval2015 10E, the last column only on the intersection of our output lexicon and Sentiment140. Of the 1315 words of task 10E, 985 and 1308 are covered by DENSIFIER and Sentiment140, respectively. $\dagger$: significantly worse than the best (bold) result in the same column ($\alpha = .05$, Fisher z-transformation).

4.3 Determining Association Strength

We also evaluate lexicon creation on SemEval2015 Task 10E. As before, the task is to predict the sentiment score of words/phrases. We use the trial data of the task to tune the hyperparameter, $\alpha^s = .4$. Out-of-vocabulary words were predicted as neutral (7/1315). Table 3 shows that the lexicon computed by DENSIFIER (line 5, Table 1) has a τ of .654 (line 6, column *all*), significantly better than all other systems, including the winner of SemEval 2015 ($\tau = .626$, line 1). DENSIFIER also beats Sentiment140 (Mohammad et al., 2013), a widely used semi-automatic sentiment lexicon. The last column is τ on the intersection of DENSIFIER and Sentiment140. It shows that DENSIFIER again performs significantly better than Sentiment140.

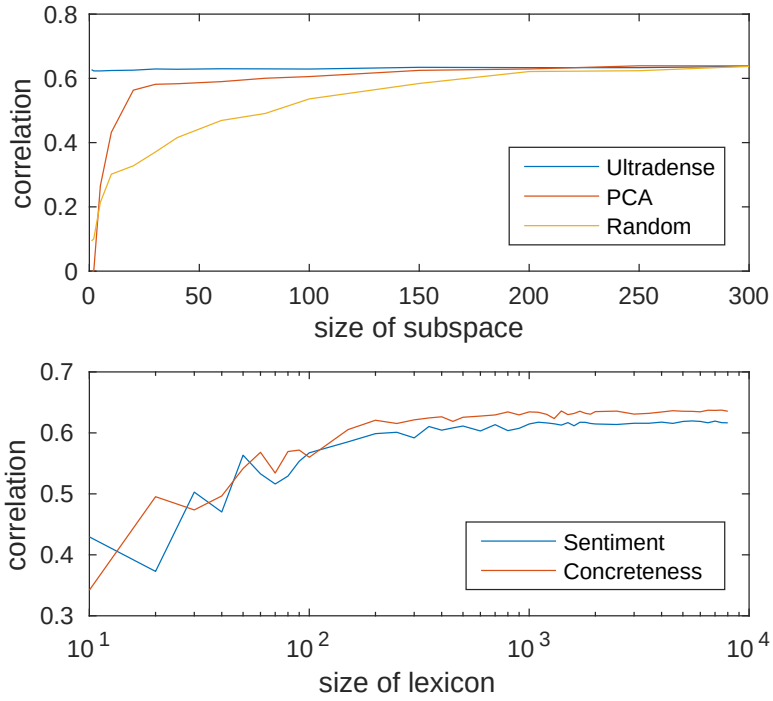

Figure 2: Kendall's τ versus subspace size (top) and training resource size (bottom). See lines 6 & 8, Table 1, for train/test split.

4.4 Text Polarity Classification

We now show that ultradense embeddings decrease model training times without any noticeable decrease in performance compared to the original embeddings. We evaluate on SemEval2015 Task 10B, classification of Twitter tweets as positive, negative or neutral. We reimplement the linguistically-informed convolutional neural network (lingCNN) of Ebert et al. (2015) that has close to state-of-the-art performance on the task. We do not use sentence-based features to focus on the evaluation of the embeddings. We initialize the first layer of lingCNN, the embedding layer, in three different ways: (i) 400-dimensional Twitter embeddings (Section 3); (ii) 40-dimensional ultradense embeddings derived from (i); (iii) 4-dimensional ultradense embeddings derived from (i). The objective weighting is $\alpha^S = .4$, optimized on the development set.

The embedding layer converts a sentence into a matrix of word embeddings. We also add linguistic features for words, such as sentiment lexicon scores. The combination of embeddings and linguistic features is the input for a convolution layer with filters spanning 2-5 words (100 filters each). This is followed by a max pooling layer, a rectifier nonlinearity (Nair and Hinton, 2010) and a fully connected softmax layer predicting the final label. The model is trained with SGD using AdaGrad (Duchi et al., 2011) and ℓ_2 regularization ($\lambda = 5 \times 10^{-5}$). Learning rate is 0.01. Mini-batch size is 100.

We follow the official guidelines and use the SemEval2013 training and development sets as training set, the SemEval2013 test set as development set and the SemEval2015 test set to report final scores (Nakov et al., 2013; Rosenthal et al., 2015). We report macro F_1 of positive and negative classes (the official SemEval evaluation metric) and accuracy over the three classes. Table 4 shows that 40-dimensional ultradense embeddings perform almost as well as the full 400-dimensional embeddings (no significant difference according to sign test). Training time is shorter by a factor of 21 (85/4 examples/second). The 4-dimensional ultradense embeddings lead to only a small loss of 1.5% even though the size of the embeddings is smaller by a factor of 100 (again not a significant drop). Training time is shorter by a factor of 44 (178/4).

We perform the same experiment on CSFD, a

lang.	embeddings	#dim	acc	F_1	ex./sec
en	original	400	.666	.623	4
en	DENSIFIER	40	.662	.620	85
en	DENSIFIER	4	.646	.608	178
cz	original	400	.803	.802	1
cz	DENSIFIER	40	.803	.801	24
cz	DENSIFIER	4	.771	.769	83

Table 4: Performance on Text Polarity Classification

Czech movie review dataset (Habernal et al., 2013), to show the benefits of ultradense embeddings for a low-resource language where only one rather small lexicon is available. As original word embeddings we train new 400 dimensional embeddings on a large Twitter corpus (3.3×10^9 tokens). We use DENSIFIER to create 40 and 4 dimensional embeddings out of these embeddings and SubLex 1.0 (Veselovská and Bojar, 2013). Word polarity features are also taken from SubLex. A simple binary negation indicator is implemented by searching for all tokens beginning with "ne". Since that includes superlative forms having the prefix "nej", we remove them with the exception of common negated words, such as "nejsi" – "you are not". We randomly split the 91,000 dataset instances into 90% train and 10% test and report accuracy and macro F_1 score over all three classes.

Table 4 shows that what we found for English is also true for Czech. There is only a small performance drop when using ultradense embeddings (not significant for 40 dimensional embeddings) while the speed improvement is substantial.

5 Parameter Analysis

In this section, we analyze two parameters: size of ultradense subspace and size of lexicon resource. We leave an evaluation of another parameter, the size of the embedding training corpus, for future work, but empirical results suggest that this corpus should ideally have a size of several billion tokens.

5.1 Size of Subspace

With the exception of the two text polarity classification experiments, all our subspaces have dimensionality $d = 1$. The question arises: does a one-dimensional space perhaps have too low a capacity to encode all relevant information and could we further improve our results by increasing the dimensionality of the subspace to values $d > 1$? The lexicon resources that we train and test on are all binary; thus, if we use values $d > 1$, then we need to map the subspace embeddings to a one-dimensional scale for evaluation. We do this by training, on the train part of the resource, a linear transformation from the ultradense subspace to the one-dimensional scale (e.g., to the sentiment scale).

Figure 2 compares different values of d^s for three different types of subspaces in this setup, i.e., the setup in which the subspace representations are mapped via linear transformation to a one-dimensional sentiment value: (i) *Random*: we take the first d^s dimensions of the original embeddings; (ii) *PCA*: we compute a PCA and take the first d^s principal components; (iii) *Ultradense* subspace of dimensionality d^s. We use the word embeddings and lexicon resources of line 6 in Table 1. For random, the performance starts dropping when the subspace is smaller than 200 dimensions. For PCA, the performance is relatively stable until the subspace becomes smaller than 100. In contrast, ultradense subspaces have almost identical performance for all values of d^s, even for $d^s = 1$. This suggests that a single dimension is sufficient to encode all sentiment information needed for sentiment lexicon creation. However, for other sentiment tasks more dimensions may be needed, e.g., for modeling different emotional dimensions of polarity: fear, sadness, anger etc.

An alternative approach to create a low-dimensional space is to simply train low-dimensional word2vec embeddings. The following experiment suggests that this does not work very well. We used word2vec to train 60-dimensional twitter embeddings with the same settings as on line 5 in Table 1. While the correlation for 400-dimensional embeddings shown in Table 1 is .661, the correlation of 60-dimensional embeddings is only .568. Thus, even though we show that the information in 400-dimensional embeddings that is relevant for sentiment can be condensed into a single dimension, hundreds of dimensions seem to be needed if we use word2vec to collect sentiment information. If we run word2vec with a small dimensionality, only a subset of available sentiment information is "harvested" from the corpus.

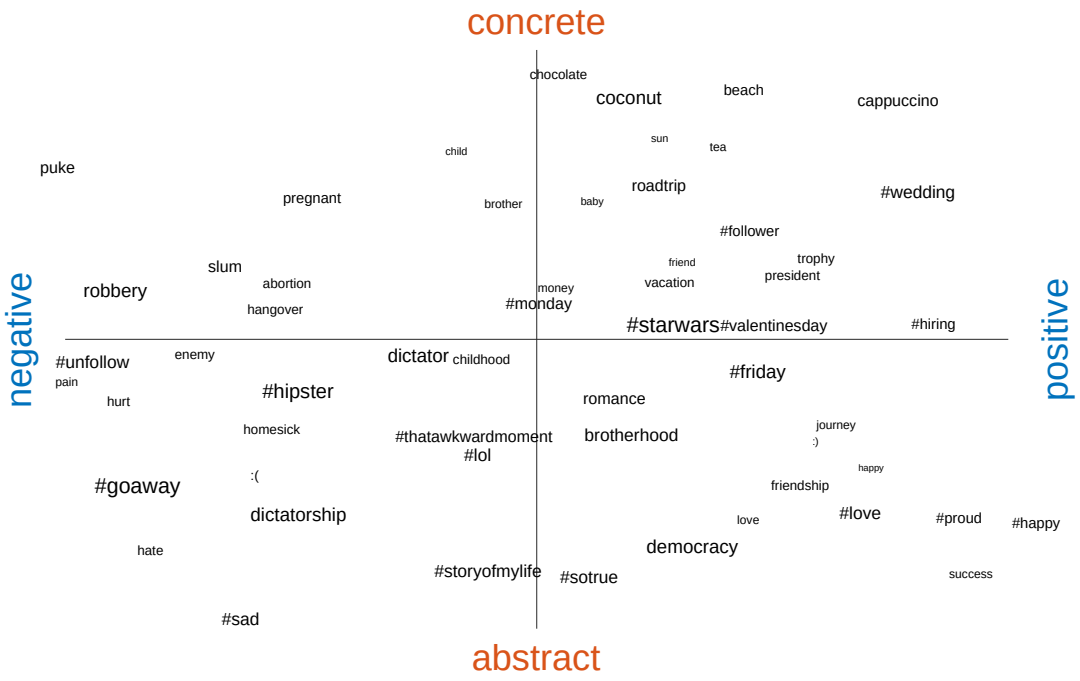

Figure 3: Illustration of EN-twitter output lexicon: DENSIFIER values are x coordinate (sentiment), y coordinate (concreteness) and font size (frequency)

5.2 Size of Training Resource

Next, we analyze what size of training resource is required to learn a good transformation Q. Labeled resources covering many words may not be available or suffer from lack of quality. We use the settings of lines 6 (sentiment) and 7 (concreteness) in Table 1. Figure 2 shows that a small training resource of 300 entries is sufficient for high performance. This suggests that DENSIFIER can create a high quality output lexicon for a new language by hand-labeling only 300 words; and that a small, high-quality resource may be preferable to a large lower-quality resource (semi-automatic or out of domain).

To provide further evidence for this, we train DENSIFIER on only the trial data of SemEval2015 task 10E. To convert the continuous trial data to binary -1 / 1 labels, we discard all words with sentiment values between -0.5 and 0.5 and round the remaining values, giving us 39 positive and 38 negative training words. The resulting lexicon has τ = .627 (Table 3, line 8).[4] This is worse than τ = .654 (line 6) for the setup in which we used several large resources, but still better than all previous work. This indicates that DENSIFIER is especially suited for languages or domains for which little training data is available.

6 Related Work

To the best of our knowledge, this paper is the first to train an orthogonal transformation to reorder word embedding dimensions into ultradense subspaces. However, there is much prior work on postprocessing word embeddings.

Faruqui et al. (2015) perform postprocessing based on a semantic lexicon with the goal of fine-tuning word embeddings. Their transformation is not orthogonal and therefore does not preserve distances. They show that their approach optimizes word embeddings for a given application, i.e., word similarity, but also that it worsens them for other applications like detecting syntactic relations. Faruqui et al. (2015)'s approach also does not have the bene-

[4] Here, we tune α^s on train (equals trial data of SemEval2015 task 10E). This seems to work due to the different objectives for training (maximize/minimize difference) and development (correlation).

fit of ultradense embeddings, in particular the benefit of increased efficiency.

In a tensor framework, Rothe and Schütze (2015) transform the word embeddings to sense (synset) embeddings. In their work, all embeddings live in the same space whereas we explicitly want to change the embedding space to create ultradense embeddings with several desirable properties.

Xing et al. (2015) restricted the work of Mikolov et al. (2013) to an orthogonal transformation to ensure that normalized embeddings stay normalized. This transformation is learned between two embedding spaces of different languages to exploit similarities. They normalized word embeddings in a first step, something that did not improve our results.

As a reviewer pointed out, our method is also related to Oriented PCA (Diamantaras and Kung, 1996). However in contrast to PCA a solution for Oriented PCA is not orthogonal.

Sentiment lexicons are often created semi-automatically, e.g., by extending manually labeled seed sets of sentiment words or adding for each word its syno-/antonyms. Alternatively, words frequently cooccurring with a seed set of manually labeled sentiment words are added (Turney, 2002; Kiritchenko et al., 2014). Heerschop et al. (2011) used WordNet together with a PageRank-based algorithm to propagate the sentiment of the seed set to unknown words. Scheible (2010) presented a semi-automatic approach based on machine translation of sentiment lexicons. The winning system of SemEval2015 10E (Amir et al., 2015) was based on structured skip-gram embeddings with 600 dimensions and support vector regression with RBF kernels. Hamdan et al. (2015), the second ranked team, used the average of six sentiment lexicons as a final sentiment score, a method that cannot be applied to low resource languages. We showed that the lexicons created by DENSIFIER achieve better performance than other semi-automatically created lexicons.

Tang et al. (2014b) train sentiment specific embeddings by extending Collobert & Weston's model and Tang et al. (2014a)'s skip-gram model. The first model automatically labels tweets as positive/negative based on emoticons, a process that cannot be easily transferred to other domains like news. The second uses the Urban Dictionary to expand a small list of 350 sentiment seeds. In our work, we showed that a training resource of about the same size is sufficient without an additional dictionary. DENSIFIER differs from this work in that it does not need a text corpus, but can transform existing, publicly available word embeddings. DENSIFIER is independent of the embedding learning algorithm and therefore extensible to other word embedding models like GloVe (Pennington et al., 2014), to phrase embeddings (Yu and Dredze, 2015) and even to sentence embeddings (Kiros et al., 2015).

7 Conclusion

We have introduced DENSIFIER, a method that is trained to focus embeddings used for an application to an ultradense subspace that contains the information relevant for the application. In experiments on SemEval, we demonstrate two benefits of the ultradense subspace. (i) Information is preserved even if we focus on a subspace that is smaller by a factor of 100 than the original space. This means that unnecessary noisy information is removed from the embeddings and robust learning without overfitting is better supported. (ii) Since the subspace is 100 times smaller, models that use the embeddings as their input representation can be trained more efficiently and have a much smaller number of parameters. The subspace can be learned with just $80 - 300$ training examples, achieving state-of-the-art results on lexicon creation.

We have shown in this paper that up to three orthogonal ultradense subspaces can be created. Many training datasets can be restructured as sets of similar/dissimilar pairs. For instance, in part-of-speech tasks verb/verb pairs would be similar, verb/noun pairs dissimilar. Hence, our objective is widely applicable. In future work, we will explore the possibility of factoring all information present in an embedding into a dozen or so orthogonal subspaces. This factorization would not change the information embeddings contain, but it would make them more compact for any given application, more meaningful and more interpretable.

The nine large DENSIFIER lexicons shown in Table 1 are publicly available.[5]

Acknowledgments. We gratefully acknowledge the support of DFG: grant SCHU 2246/10-1.

[5] www.cis.lmu.de/~sascha/Ultradense/

References

Amine Abdaoui, Jérôme Azé, Sandra Bringay, and Pascal Poncelet. 2014. FEEL: French Extended Emotional Lexicon: ISLRN: 041-639-484-224-2.

Silvio Amir, Ramón Astudillo, Wang Ling, Bruno Martins, Mario J. Silva, and Isabel Trancoso. 2015. Inescid: A regression model for large scale twitter sentiment lexicon induction. In *Proceedings of SemEval*.

Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. 2014. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior research methods*, 46(3):904–911.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537.

K. I. Diamantaras and S. Y. Kung. 1996. *Principal Component Neural Networks: Theory and Applications*. John Wiley & Sons, Inc.

John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159.

Sebastian Ebert, Ngoc Thang Vu, and Hinrich Schütze. 2015. A Linguistically Informed Convolutional Neural Network. In *Proceedings of WASSA*.

Ky Fan and Alan J Hoffman. 1955. Some metric inequalities in the space of matrices. volume 6, pages 111–116.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*.

Ivan Habernal, Tomáš Ptáček, and Josef Steinberger. 2013. Sentiment Analysis in Czech Social Media Using Supervised Machine Learning. In *Proceedings of WASSA*.

Hussam Hamdan, Patrice Bellot, and Frederic Bechet. 2015. Lsislif: Feature extraction and label weighting for sentiment analysis in twitter. In *Proceedings of SemEval*.

Bas Heerschop, Alexander Hogenboom, and Flavius Frasincar. 2011. Sentiment lexicon creation from lexical resources. In *Business Information Systems*.

Minqing Hu and Bing Liu. 2004. Mining and Summarizing Customer Reviews. In *Proceedings of KDD*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of ACL*.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of EMNLP*.

Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. 2014. Sentiment analysis of short informal texts. *JAIR*, pages 723–762.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Proceedings of NIPS*.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Saif M. Mohammad and Peter D. Turney. 2013. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*, 29(3).

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *Proceedings of SemEval*.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of ICML*.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. 2013. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *Proceedings of SemEval*.

Canberk Özdemir and Sabine Bergler. 2015. Clacsentipipe: Semeval2015 subtasks 10 b,e, and task 11. In *Proceedings of SemEval*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*.

Verónica Pérez-Rosas, Carmen Banea, and Rada Mihalcea. 2012. Learning Sentiment Lexicons in Spanish. In *Proceedings of LREC*.

Nataliia Plotnikova, Micha Kohl, Kevin Volkert, Stefan Evert, Andreas Lerner, Natalie Dykes, and Heiko Ermer. 2015. Klueless: Polarity classification and association. In *Proceedings of SemEval*.

Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M. Mohammad, Alan Ritter, and Veselin Stoyanov. 2015. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In *Proceedings of SemEval*.

Sascha Rothe and Hinrich Schütze. 2015. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of ACL*.

Christian Scheible. 2010. Sentiment translation through lexicon induction. In *Proceedings of ACL, Student Research Workshop*.

Aliaksei Severyn and Alessandro Moschitti. 2015. UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. In *Proceedings of SemEval*.

Duyu Tang, Furu Wei, Bing Qin, Ming Zhou, and Ting Liu. 2014a. Building large-scale twitter-specific sentiment lexicon : A representation learning approach. In *Proceedings of COLING*.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014b. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In *Proceedings of ACL*.

Peter D. Turney. 2002. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In *Proceedings of ACL*.

Kateřina Veselovská and Ondřej Bojar. 2013. Czech SubLex 1.0.

Ulli Waltinger. 2010. GermanPolarityClues: A Lexical Resource for German Sentiment Analysis. In *Proceedings of LREC*.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of HLT/EMNLP*.

Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. 2015. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of NAACL*.

Mo Yu and Mark Dredze. 2015. Learning composition models for phrase embeddings. *TACL*, 3:227–242.

Zhihua Zhang, Guoshun Wu, and Man Lan. 2015. Ecnu: Multi-level sentiment analysis on twitter using traditional linguistic features and word embedding features. In *Proceedings of SemEval*.

Chapter 5

Word Embedding Calculus in Meaningful Ultradense Subspaces

Word Embedding Calculus in Meaningful Ultradense Subspaces

Sascha Rothe and **Hinrich Schütze**
Center for Information and Language Processing
LMU Munich, Germany
sascha@cis.lmu.de

Abstract

We decompose a standard embedding space into interpretable orthogonal subspaces and a "remainder" subspace. We consider four interpretable subspaces in this paper: polarity, concreteness, frequency and part-of-speech (POS) subspaces. We introduce a new calculus for subspaces that supports operations like "$-1 \times$ *hate* $=$ *love*" and "give me a neutral word for *greasy*" (i.e., *oleaginous*). This calculus extends analogy computations like "*king* $-$ *man* $+$ *woman* $=$ *queen*". For the tasks of Antonym Classification and POS Tagging our method outperforms the state of the art. We create test sets for Morphological Analogies and for the new task of Polarity Spectrum Creation.

1 Introduction

Word embeddings are usually trained on an objective that ensures that words occurring in similar contexts have similar embeddings. This makes them useful for many tasks, but has drawbacks for others; e.g., antonyms are often interchangeable in context and thus have similar word embeddings even though they denote opposites. If we think of word embeddings as members of a (commutative or Abelian) group, then antonyms should be *inverses* of (as opposed to *similar* to) each other. In this paper, we use DENSIFIER (Rothe et al., 2016) to decompose a standard embedding space into interpretable orthogonal subspaces, including a one-dimensional polarity subspace as well as concreteness, frequency and POS subspaces. We introduce a new calculus for subspaces in which antonyms are inverses, e.g., "$-1 \times$ *hate* $=$ *love*". The formula shows what happens in the polarity subspace; the orthogonal complement (all the remaining subspaces) is kept fixed. We show below that we can predict an entire polarity spectrum based on the subspace, e.g., the four-word spectrum *hate*, *dislike*, *like*, *love*. Similar to polarity, we explore other interpretable subspaces and do operations such as: given a concrete word like *friend* find the abstract word *friendship* (concreteness); given the frequent word *friend* find a less frequent synonym like *comrade* (frequency); and given the noun *friend* find the verb *befriend* (POS).

2 Word Embedding Transformation

We now give an overview of DENSIFIER; see Rothe et al. (2016) for details. Let $Q \quad \mathbb{R}^{d \times d}$ be an orthogonal matrix that transforms the original word embedding space into a space in which certain types of information are represented by a small number of dimensions. The orthogonality can be seen as a hard regularization of the transformation. We choose this because we do not want to add or remove any information from the original embeddings space. This ensures that the transformed word embeddings behave differently only when looking at subspaces, but behave identically when looking at the entire space. By choosing an orthogonal and thus linear transformation we also assume that the information is already encoded linearly in the original word embedding. This is a frequent assumption, as we generally use the vector addition for word embeddings.

Concretely, we learn Q such that the dimensions $D^p \quad \{1, \ldots, d\}$ of the resulting space correspond to a word's polarity information and the $\{1, \ldots, d\} \setminus D^p$ remaining dimensions correspond to non-polarity information. Analogously, the sets of dimensions D^c, D^f and D^m correspond to a word's concreteness, frequency and POS (or morphological) information, respectively. In this paper, we assume that these properties do not corre-

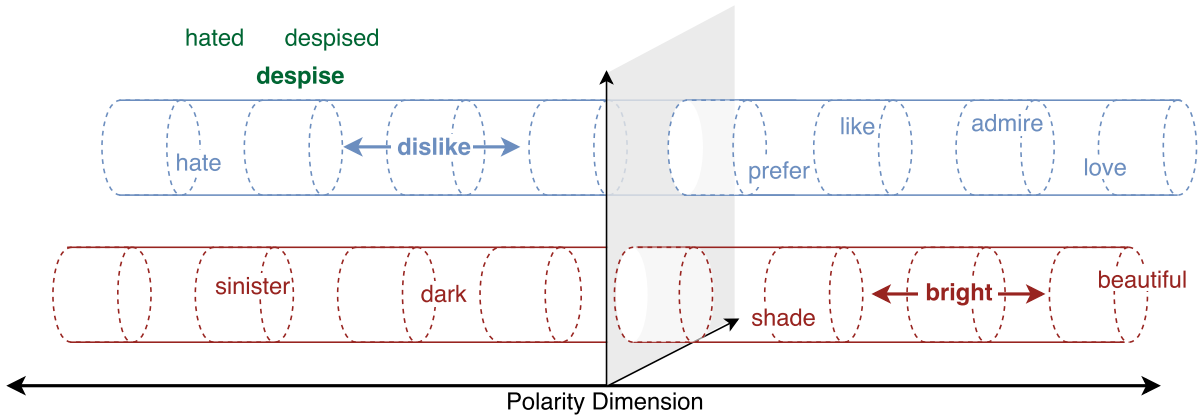

Figure 1: Illustration of the transformed embeddings. The horizontal axis is the polarity subspace. All non-polarity information, including concreteness, frequency and POS, is projected into a two dimensional subspace for visualization (gray plane). A query word (bold) specifies a line parallel to the horizontal axis. We then construct a cylinder around this line. Words in this cylinder are considered to be part of the word spectrum.

late and therefore the ultradense subspaces do not overlap. E.g., $D^p \cap D^c =$. This might not be true for other settings, e.g., sentiment and semantic information. As we are using only four properties there is also a subspace which is in the orthogonal complement of all trained subspaces. This subspace includes the not classified information, e.g., genre information in our case (e.g., "clunker" is a colloquial word for "automobile").

If e_v R^d is the original embedding of word v, the transformed representation is $u_v = Qe_v$. We use as a placeholder for polarity (p), concreteness (c), frequency (f) and POS/morphology (m) and call $d = |D|$ the dimensionality of the ultradense subspace of property . For each ultradense subspace, we create P $\mathsf{R}^{d \times d}$, an identity matrix for the dimensions in D . Thus, the ultradense (UD) representation u_v R^d of word v is defined as:

$$u_v := P\ Qe_v \tag{1}$$

For notational simplicity, u_v will either refer to a vector in R^d or to a vector in R^d where all dimensions $/ D$ are set to zero.

For training, the orthogonal transformation Q we assume we have a lexicon resource. Let L be a set of word index pairs (v, w) with different labels, e.g., positive/negative, concrete/abstract or noun/verb. We want to maximize the distance for pairs in this group. Thus, our objective is:

$$\underset{Q}{\text{argmin}} \sum_{\{p,c,f,m\}} \sum_{(v,w)\ L} - P\ Q(e_v - e_w) \tag{2}$$

subject to Q being an orthogonal matrix. Another goal is to minimize the distance of two words with identical labels. Let L be a set of word index pairs (v, w) with identical labels. In contrast to Eq. 2, we now want to minimize each distance. Thus, the objective is given by:

$$\underset{Q}{\text{argmin}} \sum_{\{p,c,f,m\}} \sum_{(v,w)\ L} P\ Q(e_v - e_w) \tag{3}$$

subject to Q being an orthogonal matrix. For training Eq. 2 is weighted with α and Eq. 3 with $1 - \alpha$. We do a batch gradient descent where each batch contains the same number of positive and negative examples. This means the number of examples in the lexica – which give rise to more negative than positive examples – does not influence the training.

3 Setup and Method

Eqs. 2/3 can be combined to train an orthogonal transformation matrix. We use pretrained 300-dimensional English word embeddings (Mikolov et al., 2013) (W2V). To train the transformation matrix, we use a combination of MPQA (Wilson et al., 2005), Opinion Lexicon (Hu and Liu, 2004) and NRC Emotion lexicons (Mohammad and Turney, 2013) for polarity; BWK, a lexicon of 40,000 English words (Brysbaert et al., 2014), for concreteness; the order in the word embedding file for frequency; and the training set of the FLORS tagger (Schnabel and Schütze, 2014) for POS. The application of the transformation ma-

trix to the word embeddings gives us four subspaces for polarity, concreteness, frequency and POS. These subspaces and their orthogonal complements are the basis for an embedding calculus that supports certain operations. Here, we investigate four such operations. The first operation computes the antonym of word v:

$$\text{antonym}(v) = \text{nn}(u_v - 2u_v^p) \quad (4)$$

where $\text{nn} : \mathbb{R}^d \rightarrow V$ returns the word whose embedding is the nearest neighbor to the input. Thus, our hypothesis is that antonyms are usually very similar in semantics except that they differ on a single "semantic axis," the polarity axis.[1] The second operation is "neutral version of word v":

$$\text{neutral}(v) = \text{nn}(u_v - u_v^p) \quad (5)$$

Thus, our hypothesis is that neutral words are words with a value close to zero in the polarity subspace. The third operation produces the polarity spectrum of v:

$$\text{spectrum}(v) = \{\text{nn}(u_v + xu_v^p) \mid x \in \mathbb{R}\} \quad (6)$$

This means that we keep the semantics of the query word fixed, while walking along the polarity axis, thus retrieving different shades of polarity. Figure 1 shows two example spectra. The fourth operation is "word v with POS of word w":

$$\text{POS}_w(v) = \text{nn}(u_v - u_v^m + u_w^m) \quad (7)$$

This is similar to analogies like *king − man + woman*, except that the analogy is *inferred by the subspace relevant for the analogy*.

We create word spectra for some manually chosen words using the Google News corpus (W2V) and a Twitter corpus. As the transformation was orthogonal and therefore did not change the length of a dimension, we multiply the polarity dimension with 30 to give it a high weight, i.e., paying more attention to it. We then use Eq. 6 with a sufficiently small step size for x, i.e., further reducing the step size does not increase the spectrum. We also discard words that have a cosine distance of more than .6 in the non-polarity space. Table 1 shows examples. The results are highly domain dependent, with Twitter's spectrum indicating more negative views of politicians than news. While *fall* has negative associations, *autumn*'s are positive – probably because *autumn* is of a higher register in American English.

[1] See discussion/experiments below for exceptions

Corpus, Type	Spectrum
News, Polarity	hypocrite, **politician**, legislator, businessman, reformer, statesman, thinker
	fall, winter, summer, **spring**, autumn
	drunks, booze, liquor, lager, **beer**, beers, wine, beverages, wines, tastings
Twitter, Polarity	corrupt, coward, **politician**, journalist, citizen, musician, representative
	stalker, neighbour, gf, bf, cousin, frnd, **friend**, mentor
	#stupid, #problems, #homework, #mylife, #reality, **#life**, #happiness
News, Concreteness	imperialist, conflict, **war**, Iraq, Vietnam War, battlefields, soldiers
	love, friendship, dear friend, friends, **friend**, girlfriend
News, Frequency	redesigned, newer, revamped, **new**
	intellect, insights, familiarity, skills, **knowledge**, experience

Table 1: Example word spectra for polarity, concreteness and frequency on two different corpora. Queries are bold.

	dev set			test set		
	P	R	F_1	P	R	F_1
Adel, 2014	.79	.65	.72	.75	.58	.66
our work	.81	.90	.85	.76	.88	.82

Table 2: Results for Antonym Classification

4 Evaluation

4.1 Antonym Classification.

We evaluate on Adel and Schütze (2014)'s data; the task is to decide for a pair of words whether they are antonyms or synonyms. The set has 2,337 positive and negative pairs each and is split into 80% training, 10% dev and 10% test. Adel and Schütze (2014) collected positive/negative examples from the nearest neighbors of the word embeddings to make it hard to solve the task using word embeddings. We train an SVM (RBF kernel) on three features that are based on the intuition depicted in Figure 1: the three cosine distances in: the polarity subspace; the orthogonal complement; and the entire space. Table 2 shows that improvement of precision is minor (.76 vs. .75), but recall and F_1 improve by a lot (+.30 and +.16).

4.2 Polarity Spectrum Creation

consists of two subtasks. PSC-SET: Given a query word how well can we predict a spectrum? PSC-ORD: How good is the order in the spectrum? Our gold standard is Word Spectrum, included in the Oxford American Writer's Thesaurus (OAWT) and therefore also in MacOS. For each query word

	newsgroups		reviews		weblogs		answers		emails		wsj	
	ALL	OOV	ALL	OOV	ALL	OOV	ALL	OOV	ALL	OOV	ALL	OOV
1 LSJU	89.11†	56.02†	91.43†	58.66†	94.15†	77.13†	88.92†	49.30†	88.68†	58.42†	96.83	90.25
2 SVM	89.14†	53.82†	91.30†	54.20†	94.21†	76.44†	88.96†	47.25†	88.64†	56.37†	96.63	87.96†
3 F	**90.86**	66.42†	**92.95**	75.29†	94.71	83.64†	90.30	62.15†	89.44	62.61†	96.59	90.37
4 F+W2V	90.51	**72.26**	92.46†	78.03	94.70	86.05	90.34	65.16	89.26	63.70†	96.44	91.36
5 F+UD	90.79	72.20	92.84	**78.80**	**94.84**	**86.47**	**90.60**	**65.48**	**89.68**	**66.24**	**96.61**	**92.36**

Table 3: Results for POS tagging. LSJU = Stanford. SVM = SVMTool. F=FLORS. We show three state-of-the-art taggers (lines 1-3), FLORS extended with 300-dimensional embeddings (4) and extended with UD embeddings (5). $\dagger$: significantly better than the best result in the same column (α = .05, one-tailed Z-test).

this dictionary returns a list of up to 80 words of shades of meaning between two polar opposites. We look for words that are also present in Adel and Schütze (2014)'s Antonym Classification data and retrieve 35 spectra. Each word in a spectrum can be used as a query word; after intersecting the spectra with our vocabulary, we end up with 1301 test cases.

To evaluate PSC-SET, we calculate the 10 nearest neighbors of the m words in the spectrum and rank the $10m$ neighbors by the distance to our spectrum, i.e., the cosine distance in the orthogonal complement of the polarity subspace. We report mean average precision (MAP) and weighted MAP where each MAP is weighted by the number of words in the spectrum. As shown in Table 4 there is no big difference between both numbers, meaning that our algorithm does not work better or worse on smaller or larger spectra.

To evaluate PSC-ORD, we calculate Spearman's ρ of the ranks in OAWT and the values on the polarity dimension. Again, there is no significant difference between average and weighted average of ρ. Table 4 also shows that the variance of the measures is low for PSC-SET and high for PSC-ORD; thus, we do well on certain spectra and worse on others. The best one, *beautiful* $\leftrightarrow$ *ugly*, is given as an example. The worst performing spectrum is *fat* $\leftrightarrow$ *skinny* (ρ = .13) – presumably because both extremes are negative, contradicting our modeling assumption that spectra go from positive to negative. We test this hypothesis by separating the spectrum into two subspectra. We then report the weighted average correlation of the optimal separation. For *fat* $\leftrightarrow$ *skinny*, this improves ρ to .67.

	PSC-SET: MAP	PSC-ORD: ρ	avg(ρ_1, ρ_2)
average	.48	.59	.70
weighted avg.	.47	.59	.70
variance	.004	.048	.014
beautiful/ugly	.48	.84	.84
fat/skinny	.56	.13	.67
absent/present	.43	.72	.76

Table 4: Results for Polarity Spectrum Creation: MAP, Spearman's ρ (one spectrum) and average ρ (two subspectra)

4.3 Morphological Analogy.

The previous two subspaces were one-dimensional. Now we consider a POS subspace, because POS is not one-dimensional and cannot be modeled as a single scalar quantity. We create a word analogy benchmark by extracting derivational forms from WordNet (Fellbaum, 1998). We discard words with ≥2 derivational forms of the same POS and words not in the most frequent 30,000. We then randomly select 26 pairs for every POS combination for the dev set and 26 pairs for the test set.[2] An example of the type of equation we solve here is *prediction* − *predict* + *symbolize* = *symbol* (from the dev set). W2V embeddings are our baseline.

We can also rewrite the left side of the equation as POS(*prediction*) + Semantics(*symbolize*); note that this cannot be done using standard word embeddings. In contrast, our method can use meaningful UD embeddings and Eq. 7 with POS(v) being u_v^m and Semantics(v) being $u_v - u_v^m$. The dev set indicates that a 8-dimensional POS subspace is optimal and Table 5 shows that this method out-

[2]This results in an even number of 25 □ 26 = 650 questions per POS combination, 4□2□650 = 5200 in total (4 POS combinations, where each POS can be used as query POS).

	W2V		UD	
	A→B	B→A	A→B	B→A
noun-verb	35.69	6.62	59.69†	50.46†
adj-noun	30.77	27.38	53.85†	43.85†
adj-verb	20.62	3.08	32.15†	24.77†
adj-adverb	45.38	35.54	46.46	43.08†
all	25.63		44.29†	

Table 5: Accuracy @1 on test for Morphological Analogy. †: significantly better than the corresponding result in the same row (α = .05, one-tailed Z-test).

performs the baseline.

4.4 POS Tagging

Our final evaluation is extrinsic. We use FLORS (Schnabel and Schütze, 2014), a state-of-the-art POS tagger which was extended by Yin et al. (2015) with word embeddings as additional features. W2V gives us a consistent improvement on OOVs (Table 3, line 4). However, training this model requires about 500GB of RAM. When we use the 8-dimensional UD embeddings (the same as for Morphological Analogy), we outperform W2V except for a virtual tie on news (Table 3, line 5). So we perform better even though we only use 8 of 300 dimensions! However, the greatest advantage of UD is that we only need 100GB of RAM, 80% less than W2V.

5 Related Work

Yih et al. (2012) also tackled the problem of antonyms having similar embeddings. In their model, the antonym is the inverse of the entire vector whereas in our work the antonym is only the inverse in an ultradense subspace. Our model is more intuitive since antonyms invert only part of the meaning, not the entire meaning. Schwartz et al. (2015) present a method that switches an antonym parameter on or off (depending on whether a high antonym-synonym similarity is useful for an application) and learn *multiple* embedding spaces. We only need a *single* space, but consider different subspaces of this space.

An unsupervised approach using linguistic patterns that ranks adjectives according to their intensity was presented by de Melo and Bansal (2013). Sharma et al. (2015) present a corpus-independent approach for the same problem. Our results (Table 1) suggest that polarity should not be considered to be corpus-independent.

There is also much work on incorporating the additional information into the original word embedding training. Examples include (Botha and Blunsom, 2014) and (Cotterell and Schütze, 2015). However, postprocessing has several advantages. DENSIFIER can be trained on a normal work station without access to the original training corpus. This makes the method more flexible, e.g., when new training data or desired properties are available.

On a general level, our method bears some resemblance with (Weinberger and Saul, 2009) in that we perform supervised learning on a set of desired (dis)similarities and that we can think of our method as learning specialized metrics for particular subtypes of linguistic information or particular tasks. Using the method of Weinberger and Saul (2009), one could learn k metrics for k subtypes of information and then simply represent a word w as the concatenation of (i) the original embedding and (ii) k representations corresponding to the k metrics.[3] In case of a simple one-dimensional type of information, the corresponding representation could simply be a scalar. We would expect this approach to have similar advantages for practical applications, but we view our orthogonal transformation of the original space as more elegant and it gives rise to a more compact representation.

6 Conclusion

We presented a new word embedding calculus based on meaningful ultradense subspaces. We applied the operations of the calculus to Antonym Classification, Polarity Spectrum Creation, Morphological Analogy and POS Tagging. Our evaluation shows that our method outperforms previous work and is applicable to different types of information. We have published test sets and word embeddings at `http://www.cis.lmu.de/~sascha/Ultradense/`.

Acknowledgments

This research was supported by Deutsche Forschungsgemeinschaft (DFG, grant 2246/10-1).

[3] We would like to thank an anonymous reviewer for suggesting this alternative approach.

References

Heike Adel and Hinrich Schütze. 2014. Using mined coreference chains as a resource for a semantic task. In *Proceedings of EMNLP*.

Jan A Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. *arXiv preprint arXiv:1405.4273*.

Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. 2014. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior research methods*, 46(3):904–911.

Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1287–1292, Denver, Colorado, May–June. Association for Computational Linguistics.

Gerard de Melo and Mohit Bansal. 2013. Good, great, excellent: Global inference of semantic intensities. *Transactions of the Association for Computational Linguistics*, 1:279–290.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Minqing Hu and Bing Liu. 2004. Mining and Summarizing Customer Reviews. In *KDD*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Saif M. Mohammad and Peter D. Turney. 2013. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*, 29(3).

Sascha Rothe, Sebastian Ebert, and Hinrich Schütze. 2016. Ultradense word embeddings by orthogonal transformation. *arXiv preprint arXiv:1602.07572*.

Tobias Schnabel and Hinrich Schütze. 2014. Flors: Fast and simple domain adaptation for part-of-speech tagging. *Transactions of the Association for Computational Linguistics*, 2:15–26.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of CoNLL*.

Raksha Sharma, Mohit Gupta, Astha Agarwal, and Pushpak Bhattacharyya. 2015. Adjective intensity and sentiment analysis. In *Proceedings of EMNLP*.

Kilian Q. Weinberger and Lawrence K. Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of HLT/EMNLP*.

Wen-tau Yih, Geoffrey Zweig, and John C Platt. 2012. Polarity inducing latent semantic analysis. In *Proceedings of EMNLP*.

Wenpeng Yin, Tobias Schnabel, and Hinrich Schütze. 2015. Online updating of word representations for part-of-speech tagging. In *Proceedings of EMNLP*.

Chapter 6

AutoExtend: Combining Word Embeddings with Semantic Resources

AutoExtend: Combining Word Embeddings with Semantic Resources

Sascha Rothe
LMU Munich

Hinrich Schütze
LMU Munich

We present AutoExtend*, a system that combines word embeddings with semantic resources by learning embeddings for non-word objects like synsets and entities and learning word embeddings which incorporate the semantic information from the resource. The method is based on encoding and decoding the word embeddings and is flexible in that it can take any word embeddings as input and does not need an additional training corpus. The obtained embeddings live in the same vector space as the input word embeddings. A sparse tensor formalization guarantees efficiency and parallelizability. We use WordNet, GermaNet and Freebase as semantic resources. AutoExtend achieves state-of-the-art performance on Word-in-Context Similarity and Word Sense Disambiguation tasks.*

1 Introduction

Unsupervised methods for learning word embeddings are widely used in natural language processing (NLP). The only data these methods need as input are very large corpora. However, in addition to corpora, there are many other resources that are undoubtedly useful in NLP, including lexical resources like WordNet and Wiktionary and knowledge bases like Wikipedia and Freebase. We will simply refer to these as *resources*. In this article, we present *AutoExtend*, a method for enriching these valuable resources with embeddings for non-word objects they describe; e.g., AutoExtend enriches WordNet with *embeddings for synsets*. The word embeddings and the new non-word embeddings live in the same vector space.

Many NLP applications benefit if non-word objects described by resources – such as synsets in WordNet – are also available as embeddings. For example, in sentiment analysis Balamurali et al. (2011) showed the superiority of sense based features over word based features. Generally, the arguments for the utility of embeddings for words carry over to the utility of embeddings for non-word objects like synsets in WordNet. We demonstrate this by improved performance due to AutoExtend embeddings for non-word objects in experiments on Word-in-Context Similarity, Word Sense Disambiguation and several other tasks.

To extend a resource with AutoExtend, we first formalize it as a graph in which (i) objects of the resource – both word objects and non-word objects – are nodes and (ii) edges describe *relations* between nodes. These relations can be of an additive or a similarity nature. *Additive relations* capture the basic intuition of the *offset calculus* (Mikolov et al. 2013a) as we will discuss in detail in Section 2. *Similarity relations* simply define similar nodes. We then define various constraints based on these relations. For example, one of our constraints states that the embeddings of two synsets related by the similarity relation should be close. Finally, we select the set of embeddings that minimizes the learning objective.

The advantage of our approach is that it *decouples* (i) the learning of word embeddongs on the one hand and (ii) the extension of these word embeddings to non-word objects in a resource on the other hand. If somebody comes up with a better way of learning word embeddings, AutoExtend immediately can extend these embeddings to similarly improved embeddings for non-word objects. We do not rely on any specific properties of word embeddings that make them usable in some resources, but not in others.

The main contributions of this article are as follows. (i) We present AutoExtend, a flexible method that extends word embeddings to embeddings of non-word objects. We demonstrate the generality and flexibility of AutoExtend by running experiments on three different resources: WordNet (Fellbaum 1998), Freebase (Bollacker et al. 2008) and GermaNet (Hamp et al. 1997). (ii) AutoExtend does not require manually labelled corpora. In fact, it does not require any corpora. All we need as input is a set of word embeddings and a resource that can be formally modeled as a graph in the way described above. (iii) We show that AutoExtend achieves state-of-the-art performance on several tasks including Word Sense Disambiguation.

This article is structured as follows. In Section 2, we introduce the AutoExtend model. In Section 3, we describe the three resources we use in our experiments and how we model them. We evaluate the embeddings of word and non-word objects in Section 4 using the tasks of Word Sense Disambiguation, Entity Linking, Word and Word-in-Context Similarity and Synset Alignment. Finally, we give an overview of related work in Section 5 and present our conclusions in Section 6.

2 Model

The graph formalization that underlies AutoExtend is based on the *offset calculus* introduced by Mikolov et al. (2013a). We interpret this calculus as a group theory formalization of word relations: we have a set of elements – the word embeddings – and an operation – vector addition – satisfying the axioms of a commutative group, in particular, commutativity, closure,[1] associativity and invertibility.

The easiest way to see that the original formulation by Mikolov et al. (2013a) corresponds to a commutative group is to conceptualize word embeddings as *sums of property embeddings*. For example, let $\vec{g}_{\text{f}}$ and $\vec{g}_{\text{m}}$ be the embeddings for the properties *feminine gender* and *masculine gender* and let $\vec{r}$ and $\vec{p}$ be the properties of being a ruler and a person. Then we can formalize the semantics of *queen*, *king*, *man* and *woman* and their additive relations as follows:

$$\vec{v}(\text{queen}) = \vec{r} + \vec{g}_{\text{f}} \tag{1}$$

$$\vec{v}(\text{king}) = \vec{r} + \vec{g}_{\text{m}} \tag{2}$$

$$\vec{v}(\text{woman}) = \vec{p} + \vec{g}_{\text{f}} \tag{3}$$

$$\vec{v}(\text{man}) = \vec{p} + \vec{g}_{\text{m}} \tag{4}$$

$$\vec{v}(\text{queen}) - \vec{v}(\text{king}) = \vec{g}_{\text{f}} - \vec{g}_{\text{m}} \tag{5}$$

$$\vec{v}(\text{woman}) - \vec{v}(\text{man}) = \vec{g}_{\text{f}} - \vec{g}_{\text{m}} \tag{6}$$

$$\vec{v}(\text{king}) = \vec{v}(\text{queen}) - \vec{v}(\text{woman}) + \vec{v}(\text{man}) \tag{7}$$

Eqs. 5-6 motivate the name *offset calculus*: The differences of pairs of embeddings that only differ on the same property and have the same relative value settings on those properties (e.g., masculine vs. feminine for the gender property) are modeled as fixed offsets. Eq. 7 is an example of how offsets are used for computing analogies, the application of the offset calculus that has

1 Closure does not hold literally for the set of words of a language represented in a finite corpus: there can be no bijection between the countable set of words and the uncountable set of real-valued vectors. When the sum $x + y$ of the embeddings of two words x and y is not attested as the embedding of a word, then we can see it as the embedding of a longer description. A simple example is that for many animals x, there are special words for the sum of x and *young* (*calf*, *cub*, *chick*), but for others a phrase must be used (*baby koala* in non-Australian varieties of English, *infant baboon*).

received most attention. Eqs. 1-4 are examples of what we take as the underlying assumption of the offset calculus about how embeddings of words are formed: as sums of property vectors.

In addition to semantic properties like gender, the offset calculus has been applied to morphological properties, e.g., *running − walking + walked = ran*; and even to properties of regional varieties of English, e.g., *bonnet − aubergine + eggplant = hood*. We take an expansive view of what a property is and include complex properties that are captured by resources. The most important instance of this expansive view in this article is that we *model a word's embedding as the sum of the embeddings of its senses*. For example, the vector of *suit* is modeled as the sum of a vector representing *lawsuit* and a vector representing *business suit*. Apart from the offset calculus, this can also be motivated by the additivity that underlies many embedding learning algorithms. This is most obvious for the counts in vector space models. They are clearly additive and thus support the view of a word as the sum of its senses. To be precise a word is a weighted sum of its senses, where the weights represent the probability of a sense. Our model incorporates this by simply learning shorter or longer vectors.

The basic idea behind AutoExtend is that it takes the embedding of an object that is a bundle of properties as input and "decodes" or "unravels" this embedding to the embeddings of these properties. For example, AutoExtend unravels the embedding of a word to the embeddings of its senses. These senses are not directly observable, so we can view them as hidden variables.

2.1 General Framework

The basic input to AutoExtend is a semantic resource represented as a graph and an embedding space given as a set of vectors. Each node in the graph is associated with a vector in a high-dimensional vector space. Nodes in the graph can have different types; e.g., in WordNet the types are word, lexeme and synset (see Figure 1). One type is the *input type*. Embeddings of nodes for this type are known. Embeddings of the other types are unknown and will be learned by AutoExtend.

Concretely, to extend a resource with AutoExtend, we (i) formalize it as a graph based on the offset calculus, (ii) assign known objects an input embedding, (iii) define a learning objective on the graph and finally find the set of embeddings that optimizes the learning objective.

(i) Graph formalization of resource

In our formalization of the resource as a graph, objects of the resource – both word objects and non-word objects – are nodes; some edges of the resource describe *additive relations* between nodes. These additive relations are the basic relations of the offset calculus between embeddings of words on the one hand and embeddings of constituents derived from the resource (e.g. semantic properties, morphological properties or senses) on the other hand. More precisely, the embedding of a node x is the sum of the embeddings of all nodes y_i that are connected via an edge (y_i, x). Other edges of the resource describe *similarity relations*. One example of this is that the embeddings of two synsets related by the hyponymy relation should be close. An example for such a graph can be seen in Figure 1.

(ii) Connecting resource and word embeddings

Each node is associated with a vector. The vectors of some nodes are known and the vectors of other nodes (e.g., senses) are not known. Throughout this article a known object is a word; a word can also be a short phrase. An example for the embedding space we want to learn can be seen in Figure 2.

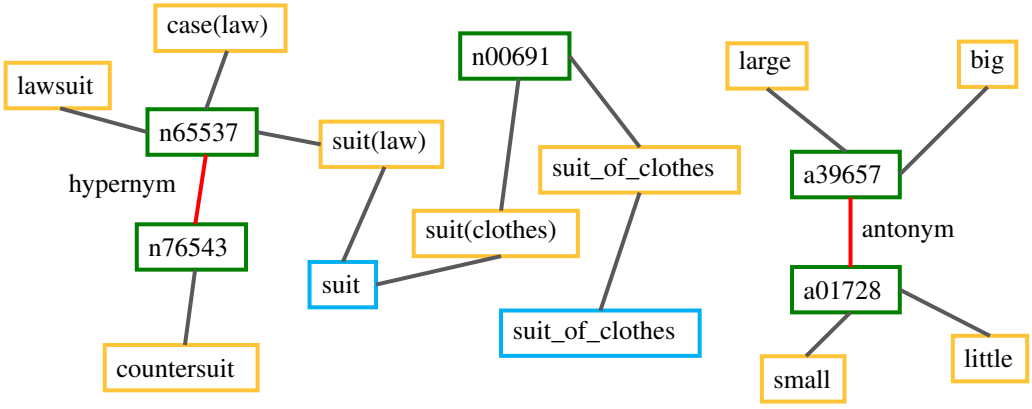

Figure 1
A small subset of the resource WordNet represented as a graph with three different types of nodes. Words (blue) are connected to lexemes (orange) by additive relations (grey). Lexemes are connected to synsets (green) and words. Synsets are connected to lexemes and also to other synsets by (dis)similarity relations (red).

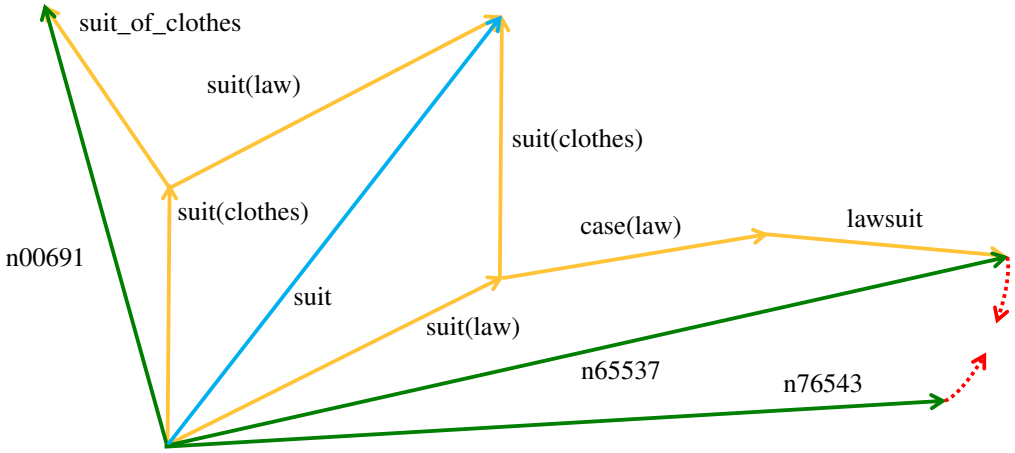

Figure 2
The embedding space we want to learn. The embeddings for words (blue) are given (input type). The embeddings for lexemes (orange) and synsets (green) have to be learned (unknown types). The additive edges define either that lexemes sum up to words or that they sum up to synsets. The similarity edges define which embeddings are similar (e.g., n65537 and n76543).

(iii) Learning objective

We define the learning objective based on various constraints. The *additive relations* define the topology of an autoencoder, which will result in autoencoding constraints that apply if a resource object participates in different additive relations (see next section). We also use *similarity relations* that are specified in the resource. Finally, we select the set of embeddings for non-word objects that minimizes the learning objective. We will assign them to those nodes in the graph (e.g., senses) that do not occur in corpora and do not have corpus-based embeddings.

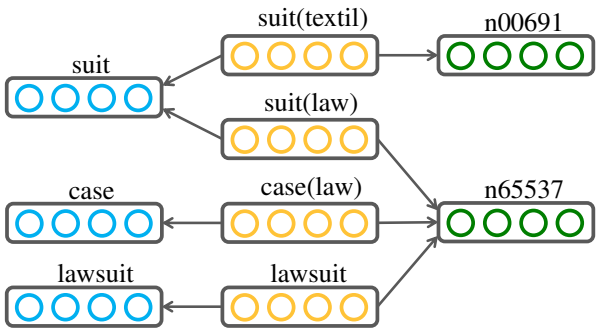

Figure 3
A subgraph of Figure 1 with additive edges only. Words are sums of their lexemes and synsets are sums of their lexemes. The circles are intended to show four different embedding dimensions.

We present AutoExtend in more detail in the following sections. While we could couch the discussion in terms of generic resources, the presentation is easier to follow if a specific resource is used as an example. We will therefore use WordNet as an example resource where appropriate. We now give a brief description of those aspects of WordNet that we make use of in this article.

Words in WordNet are lemmata where a lemma is defined as a particular spelling of the base form of an inflected word form; i.e., a lemma is a sequence of letters with a particular part of speech. A *lexeme* pairs such a spelling with a particular meaning. A *synset* is a set of lexemes with the same meaning in the sense that they are interchangeable for each other in context. Thus, we can also define a lexeme as the conjunction of a word and a synset. Additive relations between lexemes and words and lexemes and synsets correspond to a graph in which each lexeme node is connected to exactly one word node and to exactly one synset node. Additionally, two synset nodes can be connected to indicate a (dis)similarity relation holding between them, e.g., hyponomy or antonymy.

In the context of this paper, word nodes are "known" in the sense that we have learned their vectors from a large corpus. Embeddings for inflected forms are not used in this paper.[2] Lexeme and synset nodes are unknown since they are not directly observable in a corpus and vectors cannot be learned from them using standard embedding learning algorithms.

2.2 Additive Edges

As already mentioned, we will use WordNet as an example resource to simplify the presentation of our model. We will use the additive edges to formulate two basic premises of our model: (i) words are sums of their lexemes and (ii) synsets are sums of their lexemes. For example, the embedding of the word *suit* is a sum of the embeddings of its two lexemes *suit(textile)* and *suit(law)*; and the embedding of the synset *lawsuit-case-suit(law)* is a sum of the embeddings of its three lexemes *lawsuit*, *case(law)* and *suit(law)* (see Figure 3). This is equivalent to saying words split up into their lexemes and lexemes sum up to their synsets. We will formulate this in this subsection.

We denote word vectors as $w^{(i)}$ R^n, synset vectors as $s^{(j)}$ R^n, and lexeme vectors as $l^{(i,j)}$ R^n, where $l^{(i,j)}$ is that lexeme of word $w^{(i)}$ that is a member of synset $s^{(j)}$. We set lexeme

2 We also tried (i) lemmatizing the corpus, (ii) using an averaged embedding of all inflected forms and (iii) using only the embeddings of the most frequent inflected form. These three methods yielded worse performance.

vectors $l^{(i,j)}$ that do not exist to zero. For example, the non-existing lexeme *flower(truck)* is set to zero. We can then formalize our premise that (i) and (ii) hold as follows:

$$w^{(i)} = \sum_j l^{(i,j)} \tag{8}$$

$$s^{(j)} = \sum_i l^{(i,j)} \tag{9}$$

These two equations are underspecified. We therefore introduce the matrix $E^{(i,j)} \quad \mathsf{R}^{n \times n}$:

$$l^{(i,j)} = E^{(i,j)} w^{(i)} \tag{10}$$

We make the assumption that the dimensions in Eq. 10 are independent of each other, i.e., $E^{(i,j)}$ is a diagonal matrix. Our motivation for this assumption is: (i) This makes the computation technically feasible by significantly reducing the number of parameters and by supporting parallelism. (ii) Treating word embeddings on a per-dimension basis is a frequent design choice (e.g., Kalchbrenner et al. (2014b)). (iii) When vectors are treated as elements of a group, the addition is dimension-wise.

Note that we allow $E^{(i,j)} < 0$ and in general the distribution weights for each dimension (diagonal entries of $E^{(i,j)}$) will be different. Our assumption can be interpreted as word $w^{(i)}$ distributing its embedding activations to its lexemes on each dimension separately.

Eqs. 8-9 can then be written as follows:

$$w^{(i)} = \sum_j E^{(i,j)} w^{(i)} \tag{11}$$

$$s^{(j)} = \sum_i E^{(i,j)} w^{(i)} \tag{12}$$

From Eq. 11 it directly follows that:

$$\sum_j E^{(i,j)} = I_n \qquad i \tag{13}$$

with I_n being the identity matrix.

Let W be a $|V| \times n$ matrix where n is the dimensionality of the embedding space, $|V|$ is the number of words and each row $w^{(i)}$ is a word embedding; and let S be a $|S| \times n$ matrix where $|S|$ is the number of synsets and each row $s^{(j)}$ is a synset embedding. W and S can be interpreted as linear maps and a mapping between them is given by the rank 4 tensor $\mathbf{E} \quad \mathsf{R}^{|S| \times n \times |V| \times n}$. We can then write Eq. 12 as a tensor matrix product:

$$S = \mathbf{E} \times W \tag{14}$$

while Eq. 13 states, that

$$\sum_j \mathbf{E}^{i,d_1}_{j,d_2} = 1 \qquad i, d_1, d_2 \tag{15}$$

Additionally, there is no interaction between different dimensions, so $\mathbf{E}^{i,d_1}_{j,d_2} = 0$ if $d_1 = d_2$. In other words, we are creating the tensor by stacking the diagonal matrices $E^{(i,j)}$ over i and j. Another sparsity arises from the fact that many lexemes do not exist: $\mathbf{E}^{i,d_1}_{j,d_2} = 0$ if $l^{(i,j)} = 0$; i.e., $l^{(i,j)} = 0$ only if word i has a lexeme that is a member of synset j. To summarize the sparsity:

$$\mathbf{E}^{i,d_1}_{j,d_2} = 0 \qquad d_1 = d_2 \quad l^{(i,j)} = 0 \tag{16}$$

2.3 Learning Through Autoencoding

We adopt an autoencoding framework to learn embeddings for lexemes and synsets. To this end, we view the tensor equation $S = \mathbf{E} \times W$ as the encoding part of the autoencoder: the synsets are the encoding of the words. For the decoding part, we will take another copy of Figure 3 flip it horizontally and concatenate it to the encoding part (see Figure 4). This time the offset calculus relationships – words are sums of their lexemes and synsets are sums of their lexemes – translate into synsets splitting up into their lexemes and lexemes summing up to their words. We formulate the corresponding decoding part as follows:

$$s^{(j)} = \sum_i {l_2}^{(i,j)} \tag{17}$$

$${w_2}^{(i)} = \sum_j {l_2}^{(i,j)} \tag{18}$$

In analogy to $E^{(i,j)}$, we introduce the diagonal matrix $D^{(j,i)}$:

$${l_2}^{(i,j)} = D^{(j,i)} s^{(j)} \tag{19}$$

In this case, it is the synset that distributes itself to its lexemes. We can then rewrite Eqs. 17-18 to:

$$s^{(j)} = \sum_i D^{(j,i)} s^{(j)} \tag{20}$$

$${w_2}^{(i)} = \sum_j D^{(j,i)} s^{(j)} \tag{21}$$

and we also get the equivalent of Eq. 13 for $D^{(j,i)}$:

$$\sum_i D^{(j,i)} = I_n \qquad j \tag{22}$$

and in tensor notation:

$$W_2 = \mathbf{D} \times S \tag{23}$$

Normalization and sparseness properties for the decoding part are analogous to the encoding part:

$$\sum_i \mathbf{D}^{j,d_2}_{i,d_1} = 1 \qquad j, d_1, d_2 \tag{24}$$

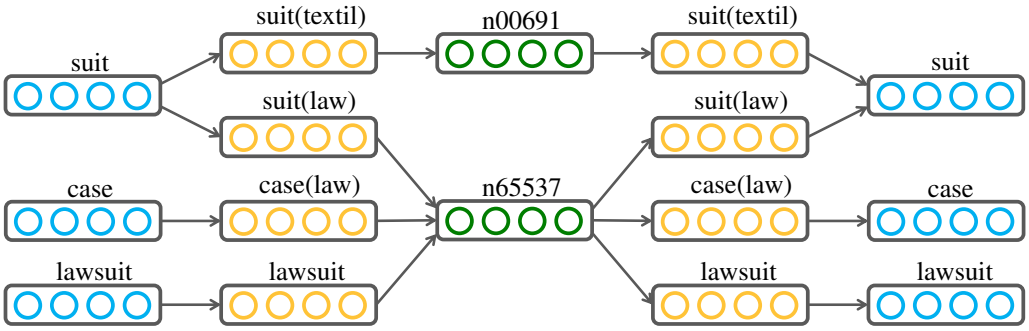

Figure 4
Encode and decode part of AutoExtend. The circles are intended to show four different embedding dimensions. These dimensions are treated as independent. The word constraint aligns the input and the output layer (blue columns), i.e., the difference between encoding input and decoding output is minimized. The lexeme constraint aligns the second and fourth layers (orange columns).

$$\mathbf{D}^{j,d_2}_{i,d_1} = 0 \qquad d_1 = d_2 \qquad l^{(i,j)} = 0 \tag{25}$$

We can state the learning objective of the autoencoder as follows:

$$\underset{\mathbf{E},\mathbf{D}}{\operatorname{argmin}} \; \mathbf{D} \times \mathbf{E} \times W - W \tag{26}$$

under the conditions Eq. 15, 16, 24 and 25.

Now we have an autoencoder where input and output layers are the word embeddings. Aligning these two layers (i.e., minimizing the difference between them) will give us the *word constraint*. The hidden layer represents the synset vectors. The tensors $\mathbf{E}$ and $\mathbf{D}$ have to be learned. They are rank 4 tensors of size $\approx 10^{15}$. However, we already discussed that they are very sparse, for two reasons: (i) We make the assumption that there is no interaction between dimensions. (ii) There are only few interactions between words and synsets (only when a lexeme exists). In practice, there are only $\approx 10^{7}$ elements to learn, which is technically feasible.

2.4 Matrix Formalization

Based on the assumption that each dimension is fully independent from other dimensions, a separate autoencoder for each dimension can be created and trained in parallel. Let $W \quad \mathbb{R}^{|V| \times n}$ be a matrix where each row is a word embedding and $w^{(d)} = W_{:,d}$ the d-th column of W, i.e., a vector that holds the d-th dimension of each word vector. In the same way, $s^{(d)} = S_{:,d}$ holds the d-th dimension of each synset vector and $E^{(d)} = \mathbf{E}^{:,d}_{:,d} \quad \mathbb{R}^{|S| \times |V|}$. We can write $S = \mathbf{E} \times W$ as:

$$s^{(d)} = E^{(d)} w^{(d)} \qquad d \tag{27}$$

with $E^{(d)}_{i,j} = 0$ if $l^{(i,j)} = 0$. The decoding equation $W_2 = \mathbf{D} \times S$ takes this form:

$$w_2{}^{(d)} = D^{(d)} s^{(d)} \qquad d \tag{28}$$

	noun	verb	adj	adv
hypernymy	84,505	13,256	0	0
antonymy	2,154	1,093	4,024	712
similarity	0	0	21,434	0
verb group	0	1,744	0	0

Table 1
Number of similarity relations by part-of-speech

where $D^{(d)} = \mathbf{D}_{:,d}$ $\mathsf{R}^{|V|\times|S|}$ and $D^{(d)}_{j,i} = 0$ if $l^{(i,j)} = 0$. So E and D are symmetric in terms of non-zero elements. The learning objective becomes:

$$\underset{E^{(d)},D^{(d)}}{\text{argmin}} \quad D^{(d)}E^{(d)}w^{(d)} - w^{(d)} \qquad d \tag{29}$$

2.5 Lexeme Embeddings

The hidden layer S of the autoencoder gives us synset embeddings. The lexeme embeddings are defined when transitioning from W to S, or more explicitly by:

$$l^{(i,j)} = E^{(i,j)}w^{(i)} \tag{30}$$

However, there is also a second lexeme embedding in AutoExtend when transitioning from S to W_2:

$$l_2{}^{(i,j)} = D^{(j,i)}s^{(j)} \tag{31}$$

Aligning these two representations (i.e., minimizing the difference between them) seems natural, so we impose the following *lexeme constraint*:

$$\underset{E^{(i,j)},D^{(j,i)}}{\text{argmin}} \quad \left\| E^{(i,j)}w^{(i)} - D^{(j,i)}s^{(j)} \right\| \qquad i,j \tag{32}$$

This can also be expressed dimension-wise. The matrix formulation is given by:

$$\underset{E^{(d)},D^{(d)}}{\text{argmin}} \quad \left\| E^{(d)}\,\mathsf{diag}(w^{(d)}) - \left(D^{(d)}\,\mathsf{diag}(s^{(d)})\right)^T \right\| \quad d \tag{33}$$

with $\mathsf{diag}(x)$ being a square matrix having x on the main diagonal, $w^{(d)} = W_{:,d}$ is again the d-th column of W and vector $s^{(d)}$ defined by Eq. 27. While the lexeme constraint encourages the two embeddings of a lexeme ($l^{(i,j)}$ and $l_2{}^{(i,j)}$) to be similar, they are still two different lexeme embeddings. In all experiments reported in Section 4 we will use the average of both embeddings and in Section 4.6 we will analyze the weighting in more detail.

2.6 Similarity Edges

Some WordNet synsets contain only a single word (lexeme). The autoencoder learns based on the word constraint, i.e., lexemes being shared by different synsets (and also words); thus, it is difficult to learn good embeddings for single-lexeme synsets. To remedy this problem, we use

the similarity edges to impose the constraint that *synsets related by WordNet relations should have similar embeddings*. Table 1 shows relations we used. Note that we also used the antonym relation, as antonyms are often replaceable in context and thus have similar word embeddings in standard word embedding models. Similarity relations are entered in a new matrix R $\mathsf{R}^{r \times |S|}$, where r is the number of relation tuples. For each relation tuple, i.e., row in R, we set the columns corresponding to the first and second synset to 1 and -1, respectively. The values of R are not updated during training. We use a squared error function and 0 as target value. This forces the system to find similar values for related synsets. Formally, the *similarity constraint* is:

$$\underset{E^{(d)}}{\text{argmin}} \; RE^{(d)}w^{(d)} \qquad d \tag{34}$$

2.7 Column Normalization

Our model is based on the premise that a word is the sum of its lexemes (Eq. 8). From the definition of $E^{(i,j)}$, we derived that $\mathbf{E}$ $\mathsf{R}^{|S| \times n \times |V| \times n}$ should be normalized over the first dimension (Eq. 15). So $E^{(d)}$ $\mathsf{R}^{|S| \times |V|}$ should also be normalized over the first dimension. In other words, $E^{(d)}$ should be a column normalized matrix. Another premise of the model is that a synset is the sum of its lexemes. Therefore, $D^{(d)}$ should also be column normalized. We call this the *column normalization constraint* and formalize it as follows:

$$\underset{E^{(d)}}{\text{argmin}} \; ([1, \ldots, 1]\, E^{(d)}) - [1, \ldots, 1] \qquad d \tag{35}$$

$$\underset{D^{(d)}}{\text{argmin}} \; ([1, \ldots, 1]\, D^{(d)}) - [1, \ldots, 1] \qquad d \tag{36}$$

2.8 Implementation

Our training objective is minimization of the sum of all constraints normalized by their output size, i.e., the word constraint (Eq. 29) divided by the number of words, the lexeme constraint (Eq. 33) divided by the number of lexemes and the similarity constraint (Eq. 34) divided by the number of similarities. Our training objective is minimization of the sum of these three normalized constraints, weighted by α (Eq. 29), β (Eq. 33) and $1 - \alpha - \beta$ (Eq. 34). The parameters α and β are tuned on development sets using a grid search with step size 0.1. To save computational cost we explore a "lazy" approach for the column normalization constraint (Eq. 35 & Eq. 36): We start the computation with column normalized matrices and normalize them again after each iteration (doing a gradient descent on the other three constraints) as long as the error function still decreases. When the error function starts increasing, we stop normalizing the matrices and continue with a normal gradient descent. This respects that while $E^{(d)}$ and $D^{(d)}$ should be column normalized in theory, there are a lot of practical issues that prevent this, e.g., out-of-vocabulary words.

The overall training objective cannot be solved analytically because it is subject to Eq. 16 and Eq. 25. We therefore use backpropagation. It turned out to be unnecessary to use regularization: all learned weights in the experiments presented below are in $[-2, 2]$.

3 Data

We test our framework in three different problem settings that cover three resources and two languages.

nearest neighbors of W/suit
S/suit (businessman), L/suit (businessman), L/accomodate, S/suit (be acceptable), L/suit (be acceptable), L/lawsuit, W/lawsuit, S/suit (playing card), L/suit (playing card), S/suit (petition), S/lawsuit, W/countersuit, W/complaint, W/counterclaim
nearest neighbors of W/become
L/become, S/become/suit, L/become/turn, L/become/get, S/become (into exist.), L/become (into exist.), W/becoming, S/become/turn, S/make, L/make, S/turn (into), W/increasingly, W/be, W/emerge
nearest neighbors of W/lawsuit
L/lawsuit, S/lawsuit, S/countersuit, L/countersuit, W/countersuit, W/suit, W/counterclaim, S/counterclaim (n), L/counterclaim (n), S/counterclaim (v), L/counterclaim (v), W/sue, S/sue (n), L/sue (n)
nearest neighbors of S/suit-of-clothes
L/suit-of-clothes, S/zoot-suit, L/zoot-suit, W/zoot-suit, S/garment, L/garment, S/dress, S/trousers, L/pinstripe, L/shirt, W/tuxedo, W/gabardine, W/tux, W/pinstripe

Figure 5
Five nearest word (W/), lexeme (L/) and synset (S/) neighbors for four items, ordered by cosine

3.1 WordNet

We use publicly available 300-dimensional embeddings[3] for 3,000,000 words and phrases trained on Google News, a corpus of $\approx 10^{11}$ tokens, using word2vec CBOW, with a window size of 5 (Mikolov et al. 2013c). Unless stated otherwise we use WordNet 2.1 as the SensEval tasks are based on this version. Many words in the word2vec vocabulary are not in WordNet, e.g., inflected forms (*cars*) and proper nouns (*Tony Blair*). Conversely, many WordNet lemmata are not in the word2vec vocabulary, e.g., *42* (digits were converted to 0). This results in a number of empty synsets (see Table 3). Note however that AutoExtend can produce embeddings for empty synsets because we also use similarity relations, not just additive relations.

We run AutoExtend on the word2vec vectors. Our main goal is to produce compatible embeddings for lexemes and synsets. Thus, we can compute nearest neighbors across all three types as shown in Figure 5.

3.2 GermaNet

For this setup, we train word2vec embeddings for German using settings similar to those that were used to train the English word2vec embeddings. We use the German Wikipedia with 5 10^8 tokens and preprocess them with the word2phrase tool included in word2vec two times, first with a threshold of 200 and then with a threshold of 100. After that, we run word2vec with identical settings as the downloaded word embeddings, i.e. CBOW, window size 5, minimal count 5, negative sampling 3 and hierarchical softmax off. We run 10 iterations to compensate for the smaller corpus. After that we intersect them with words found in GermaNet 9.0. As GermaNet has the same structure as WordNet we can directly apply AutoExtend to it. For similarity relations, we only use hypernymy and antonymy. In GermaNet, antonymy is a relationship between lexemes. To match our model, we extend it to synsets by viewing any pair of synsets as antonyms if they contain lexemes that are antonyms.

3 http://code.google.com/p/word2vec/

	WordNet 2.1 and GermaNet	Freebase
input types	Words W	Words W
unknown types	Lexemes L_1, L_2 Synsets S Improved Words W_1, W_2	Aliases L_1, L_2 Entities S, Types T Improved Words W_1, W_2
input edges	$W \times L$, $L \times S$, $S \times S$	$W \times L$, $L \times S$, $S \times T$
word constraint	(W, W_2)	(W, W_2)
lexeme constraint	(L_1, L_2)	(L_1, L_2)
similarity constraint	(S, S)	(S, T)

Table 2
Overview of input and output data of AutoExtend for different resources. The improved word embedding matrices W_1 and W_2 emerge when summing up all corresponding lexeme embeddings in L_1 and L_2, respectively. The lexeme matrices L_1 and L_2 emerge from the lexemes l in Eq. 30 and l_2 in Eq. 31, respectively.

3.3 Freebase

Freebase contains *word nodes*[4] (whose embeddings are known) and *alias nodes* and *entity nodes* (whose embeddings are unknown). Each entity also has one or more types, e.g. *director*. As we will explain below, we also create *type nodes* and learn embeddings for them. An alias node is connected to exactly one word node and exactly one entity node. An entity node is connected to one or more type nodes.

We use the same English word embeddings as for WordNet and intersect them with words found in Freebase. A Freebase entity has one or more aliases, e.g. the entity *Barack Obama* has the aliases *Barack Obama*, *President Obama* and *Barack Hussein Obama*. Aliases are available in different languages, but we only use English aliases. The role of synsets in WordNet corresponds to the role of entities in Freebase; the role of lexemes in WordNet corresponds to the role of aliases in Freebase, i.e., they connect words and entities. Freebase contains a large number of entities with a single alias; we exclude these since they are usually not completely modeled and contain little information.

Freebase also contains a great diversity of relations, but most of them do not fulfill the requirement of connecting similar entities. For example, the relation *born-in* connects a person and a city, and we do not want to align these embeddings. We therefore only use the relation *same-type*. There are about 26,000 types in Freebase, with different granularity, and well-modeled entities usually have several types. For example, *Barack Obama* has the types *President-of-the-US*, *person* and *author* as well as several other types.[5] For a type with n members, this would give us n^2 relations. This would result in a huge relation matrix which would slow down the AutoExtend computation. To address this, we add *type nodes* to the graph. The similarity relation *same-type* is only constructed between type nodes and entity nodes, but not between entity nodes and entity nodes. An added benefit is that AutoExtend also produces type embeddings; these may be useful for several tasks, e.g., for entity typing (Yaghoobzadeh and Schütze 2015).

4 Recall that our definition of words also includes phrases. Just as we subsume both *suit* and *red tape* under the concept of word in WordNet, we also refer to both *Clinton* and *George Miller* in Freebase as words.

5 Entities also have one notable type, e.g., *President-of-the-US* for *Barack Obama*, but we do not distinguish notable types from other types.

	WordNet 2.1	∩ word2vec	GermaNet 9.0	∩ word2vec		Freebase	∩ word2vec
words	147,478	54,570	109,683	89,160		≈ 23 10^3	17,165
synsets	117,791	73,844	93,246	82,027	e:	≈ 50 10^6	12,362
					t:	≈ 26 10^3	3,516
lexemes	207,272	106,167	124,996	103,926		≈ 47 10^6	27,478

Table 3
Number of items in different resources and after intersection with word2vec vectors. The analogs of synsets in Freebase are entities ("e:") and types ("t:").

4 Experiments and Evaluation

We evaluate AutoExtend embeddings on the following tasks: Word Sense Disambiguation, Entity Linking, Word-in-Context Similarity, Word Similarity and Synset Alignment. Our results depend directly on the quality of the underlying word embeddings. We would expect even better evaluation results as word representation learning methods improve. Using a new and improved set of underlying embeddings in AutoExtend is simple: it is a simple switch of the input file that contains the word embeddings.

4.1 Word Sense Disambiguation

We use IMS (It Makes Sense) for our Word Sense Disambiguation (WSD) evaluation (Zhong and Ng 2010). As in the original paper, preprocessing consists of sentence splitting, tokenization, POS tagging and lemmatization; the classifier is a linear SVM. In our experiments (Table 4), we run IMS with *each feature set by itself* to assess the relative strengths of feature sets (lines 1–7) and on *feature set combinations* to determine which combination is best for WSD (lines 8, 12–15). We use SensEval-2 as development set for SensEval-3 and vice versa. This gives us a weighting of $\alpha = \beta = 0.4$ for both sets.

IMS implements three standard WSD feature sets: part of speech (POS), surrounding word and local collocation (lines 1–3).

Let w be an ambiguous word with k senses. The three feature sets on lines 5–7 are based on the AutoExtend embeddings $s^{(j)}$, $1 \leq j \leq k$, of the k synsets of w and the centroid c of the sentence in which w occurs. The centroid is simply the sum of all word2vec vectors of the words in the sentence, excluding stop words.

The **S-cosine** feature set consists of the k cosines of centroid and synset vectors:

$$< \cos(c, s^{(1)}), \cos(c, s^{(2)}), \ldots, \cos(c, s^{(k)}) >$$

The **S-product** feature set consists of the nk element-wise products of centroid and synset vectors:

$$< c_1 s_1^{(1)}, \ldots, c_n s_n^{(1)}, \ldots, c_1 s_1^{(k)}, \ldots, c_n s_n^{(k)} >$$

where c_i (resp. $s_i^{(j)}$) is element i of c (resp. $s^{(j)}$). The idea is that we let the SVM estimate how important each dimension is for WSD instead of giving all equal weight as in S-cosine.

The **S-raw** feature set simply consists of the $n(k+1)$ elements of centroid and synset vectors:

$$< c_1, \ldots, c_n, s_1^{(1)}, \ldots, s_n^{(1)}, \ldots, s_1^{(k)}, \ldots, s_n^{(k)} >$$

			WSD		Entity Linking	
			SensEval-2	SensEval-3	*FACC dev*	FACC test
		size	4,328	3,944	10,000	10,000
IMS feature sets	1	POS	53.6	58.0	*44.3*	45.2
	2	surrounding word	57.6	**65.3**	***62.3***	**60.3**
	3	local collocation	**58.7**	64.7	*53.7*	53.8
	4	S_{naive}-product	56.5	62.2	*57.7*	58.2
	5	S-cosine	55.6	61.0	*39.3*	39.8
	6	S-product	56.9	62.6	*58.4*	59.2
	7	S-raw	57.2	63.3	*56.1*	56.1
system comparison	8	MFS	47.6†	55.2†	*33.6*†	33.4†
	9	Rank 1 system	64.2†	72.9		
	10	Rank 2 system	63.8†	72.6‡		
	11	IMS	65.2†	72.3†	*62.2*†	61.7†
	12	IMS + S_{naive}-prod.	62.6†	69.4†	*65.0*†	64.9
	13	IMS + S-cosine	65.3‡	72.2†	*62.1*†	61.6†
	14	IMS + S-product	**66.8**	**73.6**	***65.8***	**65.4**
	15	IMS + S-raw	62.4†	66.8†	*63.5*†	63.3†

Table 4
WSD and Entity Linking accuracy for different feature sets and systems. Best result in each column in bold. Results of development sets are italic. Results significantly worse than the best (bold) result in each column are marked $\dagger$ for $\alpha = .05$ and $\ddagger$ for $\alpha = .10$ (one-tailed Z-test).

Based on the experiment, we would like to determine whether AutoExtend features improve WSD performance when added to standard WSD features. To make sure that improvements we get are not solely due to the power of word2vec, we also investigate a simple word2vec baseline. For S-product (the AutoExtend feature set that performs best in the experiment, see line 14), we test the alternative word2vec-based **S_{naive}-product** feature set. It has the same definition as S-product except that we replace the synset vectors $s^{(j)}$ with naive synset vectors $z^{(j)}$, defined as the sum of the word2vec vectors of the words that are members of synset j.

Lines 1–7 in Table 4 show the performance of each feature set by itself. We see that the synset feature sets (lines 5–7) have a comparable performance to standard feature sets. S-product is the strongest of the synset feature sets.

Lines 9–16 show the performance of different feature set combinations. MFS (line 8) is the most frequent sense baseline. Lines 9&10 are the winners of SensEval. The standard configuration of IMS (line 11) uses the three feature sets on lines 1–3 (POS, surrounding word, local collocation) and achieves an accuracy of 65.2% on the English lexical sample task of SensEval-2 (Kilgarriff 2001) and 72.3% on SensEval-3 (Mihalcea et al. 2004).[6] Lines 12–16 add one additional feature set to the IMS system on line 11; e.g., the system on line 14 uses POS, surrounding word, local collocation and S-product feature sets. The system on line 14 outperforms all previous systems, most of them significantly. While S-raw performs quite reasonably as a feature set alone, it hurts the performance when used as an additional feature set. As this is the feature set that contains the largest number of features ($n(k + 1)$), overfitting is the likely reason. Conversely, S-cosine only adds k features and therefore may suffer from underfitting.

The main result of this experiment is that we achieve an improvement of more than 1% in WSD performance when using AutoExtend.

6 Zhong and Ng (2010) report accuracies of 65.3% / 72.6% for this configuration.

word 1	similarity	word 2
... Crew members **advised** passengers to sit quietly in order to increase their chances of survival ...	7.1	... the Rome Statute stipulates that the court may **inform** the Assembly of States Parties or Security Council ...
... and Andy 's getting ready to **pack** his bags and head up to Los Angeles tomorrow to get ready to fly back home on Thursday	2.1	... she encounters Ben (Duane Jones) , who arrives in a pickup truck and defends the house against another **pack** of zombies ...

Table 5
Examples of the SCWS test set. The score indicates the similarity of the words in bold.

4.2 Entity Linking

We use the same IMS system for Entity Linking. The train, development and test sets are created as follows. We start with the annotated FACC corpus and extract all entity annotated words and their surrounding words – ten to the left and ten to the right. Recall that throughout the paper, a word can also be a phrase. We remove aliases that occur less than 0.1 times than the corresponding word and words that have a character length of one or two. We extract at most 400 examples for each entity-word combination. This procedure selects entities that are ambiguous and that are frequent enough to give us a sufficient number of training examples. We randomly select 50 words with 1000 examples each and split each word into 700 train, 100 development and 200 test instances. This results in a test set of 10,000 instances.[7] We optimize the constraint weights on the development set; the optimal values are $\alpha = 0.7$ and $\beta = 0.0$. We incorporate the embeddings in three different ways as described in Section 4.1. The results can be seen in Table 4. Again the element-wise product (line 14) performs better than cosine and raw (lines 13&15). The new feature set achieves an accuracy of 65.4% – significantly better than the baseline IMS system (line 11, 61.7%).

4.3 Word-in-Context Similarity

The third evaluation uses SCWS (Huang et al. 2012). SCWS is a Word Similarity test set but does not only provided isolated words and corresponding similarity scores, but also a context for each word. The similarity score is an average score of 10 human ratings. See Table 5 for examples. In contrast to normal Word Similarity test sets, this data set also contains pairs of two instances of the same word. SCWS is based on WordNet, but the information as to which synset a Word-in-Context came from is not available. However, the data set is the closest we could find for sense similarity. Synset and lexeme embeddings are obtained by running AutoExtend. We set $\alpha = 0.2$ and $\beta = 0.2$ based on Section 4.4. Lexeme embeddings are the natural choice for this task as human subjects are provided with two words and a context for each and then have to assign a similarity score. But for completeness, we also run experiments for synsets.

For each word, we compute a context vector c by adding all word vectors of the context, excluding the test word itself. Following Reisinger and Mooney (2010), we compute the lexeme (resp. synset) vector l either as the *simple average* of the lexeme (resp. synset) vectors $l^{(ij)}$ (resp. $s^{(j)}$) (method AvgSim, no dependence on c in this case) or as the average of the lexeme (resp. synset) vectors *weighted by cosine similarity* to c (method AvgSimC). The latter method is supposed to give higher weights to lexemes that better fit the context.

7 This set is publicly available at `http://cistern.cis.lmu.de/`.

		AvgSim	AvgSimC
1	Huang et al. (2012)	62.8	$65.7^{\dagger}$
2	Tian et al. (2014)	–	$65.4^{\dagger}$
3	Neelakantan et al. (2014)	67.2	69.3
4	Chen et al. (2014)	$66.2^{\ddagger}$	68.9
5	words (word2vec)	$66.7^{\dagger}$	$66.7^{\dagger}$
6	synsets	$63.2^{\dagger}$	$63.5^{\dagger}$
7	lexemes	**68.3**	**70.2**

Table 6
Spearman correlation ($\rho \times 100$) on SCWS. Best result per column in bold. Results significantly worse than the best (bold) result are marked $\dagger$ for $\alpha = .05$ and $\ddagger$ for $\alpha = .10$ (one-tailed Z-test).

Table 6 shows that AutoExtend lexeme embeddings (line 7) perform better than previous work, including (Huang et al. 2012) and (Tian et al. 2014). Lexeme embeddings perform better than synset embeddings (lines 7 vs. 6), presumably because using a representation that is specific to the actual word being judged is more precise than using a representation that also includes synonyms.

A simple baseline is to use the underlying word2vec embeddings directly (line 5). In this case, there is only one embedding, so there is no difference between AvgSim and AvgSimC. It is interesting that even if we do not take the context into account (method AvgSim) the lexeme embeddings outperform the original word embeddings. As AvgSim simply adds up all lexemes of a word, this is equivalent to the motivation we proposed in the beginning of the article (Eq. 8). Thus, replacing a word's embedding by the sum of the embeddings of its senses could generally improve the quality of embeddings – cf. Huang et al. (2012) for a similar argument. We will do a deeper evaluation of this in Section 4.4.

4.4 Word Similarity

The results of the previous experiments motivate us to test the new embeddings also on Word Similarity test sets, namely MC (Miller and Charles 1991), MEN (Bruni et al. 2014), RG (Rubenstein and Goodenough 1965), SIMLEX (Hill et al. 2014), RW (Luong et al. 2013) and WordSim-353 (Finkelstein et al. 2001) for English (using embeddings autoextended based on WordNet) and GUR-65, GUR-350 (Gurevych 2005) and ZG-222 (Zesch and Gurevych 2006) for German (using embeddings autoextended based on GermaNet). As the simple sum of the lexeme vectors (method AvgSim, line 7, Table 6) ignores the context and outperforms the underlying word embeddings (line 5), we expect a similar performance improvement on other Word Similarity test sets. Note that AutoExtend makes available three different word embeddings:

1. the original word embeddings $W_0 = W$, i.e. the input to AutoExtend
2. the word embeddings W_1 that we get when we add lexeme vectors of the *encoding* part (see Eq. 30)
3. the word embeddings W_2 that we get when we add lexeme vectors of the *decoding* part (see Eq. 31 or Eq. 22)

We observe that each pair $(W_i, W_j), i = j$ of word embedding sets corresponds to a constraint of AutoExtend. (i) The column normalization constraint (Eq. 35) will align W_0 and W_1, as we just split the original word embeddings and add them up again. (ii) The word constraint (Eq. 29) will

		MC	MEN	RG	*SIMLEX*	RW	WORDSIM	*GUR*	GUR	ZG
	size	30	3,000	65	*999*	2,034	353	*65*	350	222
	coverage	30	2,922	65	*999*	1,246	332	*47*	213	108
1	W	78.9	77.0	76.1	*44.2*	54.2	**69.9**	*41.0*	39.1	23.0
2	W_1	70.9	67.5†	67.8‡	*37.6*†	49.3†	61.0†	*25.1*†	40.4	28.4‡
3	W_2	**85.2**	**77.5**	**82.5**	***47.4***†	**54.8**	69.0	***63.3***†	**57.1**†	**34.3**†

Table 7
Spearman correlation ($\rho \times 100$). Best result per column in bold. Results of development sets are italic. Results significantly worse or better than the basline (line 1) in each column are marked $\dagger$ for $\alpha = .05$ and $\ddagger$ for $\alpha = .10$ (one-tailed Z-test).

align W_0 and W_2. This was the initial idea of our system. (iii) The lexeme constraint (Eq. 33) will align W_1 and W_2.

As in the previous section we use the cosine similarity of word embeddings to predict a similarity score and report the Spearman correlation. We use W (Table 7, line 1) as our baseline. Lines 2 & 3 are the word embeddings described above. The SIMLEX and GUR-65 test sets are used as development sets to obtain the parameters $\alpha = 0.2$ and $\beta = 0.2$ for both models by optimizing $\max(W_1, W_2)$, i.e. the best result of line 2 & 3. While we observe a significant performance drop from W to W_1, we also observe a small improvement on W_2 on English. The improvement is significant for German, but not for English. This is most likely because of the very strong baseline of the Google News word embeddings, which are used for the English test sets. The German embeddings are trained on the smaller Wikipedia corpus. This suggests that our method is especially suited to improve lower quality embeddings.

4.5 Synset Alignment

In this evaluation, we try to predict whether a German synset corresponds to an English synset or not. This is a useful task when creating multilingual resources. We use the synset embeddings from GermaNet 9.0 and WordNet 3.0. As these embeddings were trained on different corpora we first have to calculate a linear map that transfers the English embedding space to the German one.[8] For this we extract the most frequent 30,000 English words and translate them to German using Google Translate. The resulting pairs are intersected with the most frequent 30,000 German words, leaving 10,684 translation pairs. We hold out 1,000 of them for testing. The remaining 9,684 translation pairs are used to train a linear map. Following (Mikolov et al. 2013b), let W be the matrix containing the German embeddings as rows and V the matrix containing the corresponding English embeddings as rows. The linear map L is given by:

$$L = (W^T W)^{-1} W^T V \tag{37}$$

The linear map L solves the following optimization problem:

$$\operatorname*{argmin}_{L} \; WL - V \tag{38}$$

8 We could also transfer the German embedding space to the English one, but the performance is lower for this setting. The most likely reason is, that the English embeddings are learned on a bigger corpus and thus contain more information. For the linear map, it is easy to drop information, but it is difficult to infer new information.

		Words	Synsets	
			dev	test
	size	2,000	*2,000*	2,000
1	word	.943		
2	synset		***.872***	**.870**
3	$\text{synset}_{\text{naive}}$		$.852^{\ddagger}$	$.826^{\dagger}$

Table 8
Accuracy of development and test set on the Synset Alignment task. Best result per column in bold. Results of development set are italic. Results significantly worse than the best result are marked † for $\alpha = .05$ (one-tailed Z-test) and ‡ for $\alpha = .10$ (one-tailed Z-test).

We create two test sets, one for words and one for synsets. The 1,000 translation pairs we held back are concatenated with 1,000 random German-English word pairs. The task is to predict whether a pair is a translation (positive) or not (negative); the test set contains 1,000 positive and 1,000 negative instances. We construct similar development and test sets for synsets by using the interlingual index provided in GermaNet. The interlingual index allows a mapping of concepts (e.g. synsets) of different languages We randomly collect 1,000 correct German-English synset pairs and 1,000 false synset pairs for development and test each. The development set is used to optimize the parameters α and β of German and English models. The best performance is found for $\alpha = 0.9$ and $\beta = 0.1$ for both German and English. Note that we do not need a development set for words as there are no parameters to tune. Errors in the word test set are probably due to insufficient word embedding models or errors caused by the linear mapping L. As we already mentioned our synset embeddings can only be as good as the underlying word embeddings. So both cases, insufficient word embeddings and insufficient linear map, also affect the performance of the synset embeddings. Because of this, the accuracy of the word test (Line 1 in Table 8) set can be seen as an upper bound. Line 2 shows the performance of our synset embeddings on the development and test sets. Line 3 shows the performance of naive synset embeddings, defined as the sum of the vectors of the words that are members of a synset.

The main result of this experiment is that the synset vectors obtained by AutoExtend perform better in bilingual synsets alignment than a naive sum of words.

4.6 Analysis

The most important parameter of AutoExtend is the weighting α and β given to the objectives. Table 9 shows a summary of all weightings used in this article. We observe that while all constraints are important, the optimal weighting is different for different applications. These differences are due to different corpora and resources. For example, aligning types in Freebase has a different effect than aligning antonyms in WordNet. But more important is the actual task the embeddings are used for. For example, when we compute embedding similarities, we want similar words to have similar embeddings resulting in a big weighting for the similarity constraint (lines 4 & 5). For Synset Alignment we want similar embeddings to have different embeddings in order to better distinguish them, resulting in no weight for the similarity constraint (line 6 & 7).

We found that some applications are not sensitive to the weighting; for example, for Entity Linking (line 2), the differences between weightings that give non-zero weights to all three constraints are negligible (less than 0.3).

We also analyzed the impact of the four different relations in WordNet (see Table 1) on performance. In Table 4 and Table 6, all four relations are used together. We found that any

	task	corpus	resource	α word c.	β lexeme c.	$1-\alpha-\beta$ similarity c.
1	WSD	Google News	WordNet 2.1	0.4	0.4	0.2
2	Entity Linking	Google News	Freebase	0.7	0.0	0.3
3	SCWS	Google News	WordNet 2.1	0.2	0.2	0.6
4	Word Similarity	Google News	WordNet 2.1	0.2	0.2	0.6
5	Word Similarity	Wikipedia	GermaNet 9.0	0.2	0.2	0.6
6	Synset Alignment	Google News	WordNet 3.0	0.9	0.1	0.0
7	Synset Alignment	Wikipedia	GermaNet 9.0	0.9	0.1	0.0

Table 9
Optimal weighting of the three constraints (word, lexeme, similarity) for different tasks.

combination of three relation types performs worse than using all four together. A comparison of different relations must be done carefully as they differ in the POS they affect and in quantity (see Table 1). In general, relation types with more relations outperformed relation types with fewer relations.

5 Related work

Word Embeddings

Among the earliest work on distributed word representations, usually called word embeddings today, was (Rumelhart et al. 1988). Non-neural-network techniques that create low-dimensional word representations also have been used widely, including singular value decomposition (SVD) (Deerwester et al. 1990; Schütze 1992) and random indexing (Kanerva 1998, 2009). There has been a resurgence of work on embeddings recently (e.g., Bengio et al. (2003a), Mnih and Hinton (2007a), Collobert et al. (2011), Mikolov et al. (2013a), Pennington et al. (2014a)), including methods that are SVD-based (Levy and Goldberg 2014; Stratos et al. 2015). All of these models differ from AutoExtend in that they produce only a single embedding for each word, but all of them can be used as input for AutoExtend.

Sense Embeddings not related to Lexical Resources

There are several approaches to finding embeddings for senses, variously called meaning, sense and multiple word embeddings. Schütze (1998) created sense representations by clustering context representations derived from co-occurrence. The centroid of its cluster is used as a representation of a sense. Reisinger and Mooney (2010) and Huang et al. (2012) also presented methods that learn multiple embeddings per word by clustering the contexts. Bordes et al. (2011) created similarity measures for relations in WordNet and Freebase to learn entity embeddings. An energy based model was proposed by Bordes et al. (2012) to create disambiguated meaning embeddings and Neelakantan et al. (2014) and Tian et al. (2014) extended the Skip-gram model (Mikolov et al. 2013a) to learn multiple word embeddings. Another interesting approach to create sense-specific word embeddings uses bilingual resources (Guo et al. 2014). The downside of this approach is that parallel data is needed. While all these embeddings correspond to different word senses, there is no clear mapping between them and a resource like WordNet.

Sense Embeddings related to Lexical Resources

Recently, Bhingardive et al. (2015) used WordNet to create sense embeddings similar to the naive method in this paper. They used these sense embeddings to extract the most frequent synset. Chen et al. (2014) modified word2vec to learn sense embeddings, each corresponding to a WordNet

synset. They use glosses to initialize sense embeddings, which in turn can be used for WSD. The sense disambiguated data can again be used to improve sense embeddings. While WordNet is by far the most used resource, Iacobacci et al. (2015) computed sense embeddings with BabelNet, which is a superset of WordNet. They use a state-of-the-art WSD system to generate a large sense annotated corpus which is used to train sense embeddings. In contrast, our approach can be used to improve WSD without relying on input from an existing WSD system.

Embeddings using Lexical Resources

Other work tries to combine distributed word representations and semantic resources to create better or specialized embeddings. These include the ADMM by Fried and Duh (2014) and the work of Wang et al. (2015). Liu et al. (2015) used WordNet to create ordinal similarity inequalities also to extend the Skip-gram model into a Semantic Word Embedding model. In the Relation Constrained Model, Yu and Dredze (2014) used word2vec to learn embeddings that are optimized to predict a related word in the resource, with good evaluation results. Bian et al. (2014) used not only semantic, but also morphological and syntactic knowledge to compute more effective word embeddings. Cotterell et al. (2016) focus on generating embeddings for inflected forms not observed during training based on morphological resources. Wang et al. (2014) used Freebase to learn embeddings for entities and words. This is done during embedding learning, in contrast to our post-processing method. Zhong et al. (2015) improved this by requiring the embedding vector not only to fit the structured constraints in the knowledge base, but also to be equal to the embedding vector computed from the text description.

Post-processing Embeddings

This prior work needs a training step to learn embeddings. In contrast, we can "AutoExtend" any set of given word embeddings – without (re)training them. There is an increasing amount of work on taking existing word embeddings and combine them with a lexical resource. Labutov and Lipson (2013) re-embedded existing word embeddings in supervised training, not to create new embeddings for senses or entities, but to get better predictive performance on a task while not changing the space of embeddings. A similar approach was chosen by Faruqui et al. (2015a) and called retrofitting. This work is also related to our work in that it uses WordNet. However, it only uses the similarity relations in order to change embeddings for known objects, i.e. words. They do not use additive relations nor do they compute embeddings for non-word objects. Jauhar et al. (2015) used the same retrofitting technique to also model sense embeddings. There work is similar to our approach but instead of distinguishing between additive and similarity relations all edges are treated as similarity relations (see Figure 1). Their results show an improvement for word embeddings but the sense embeddings perform worse then the embeddings they were trained on (0.42 on SCWS, see Table 6). We therefore believe that the additive relation is the superior model for the relationship between words and lexemes as well as for the relationship between synsets and lexemes. Kiela et al. (2015) used retrofitting and joint-learning approaches to specialize their embeddings for either similarity or relatedness tasks.

Other Related Work

In this work we treated WSD and Entity Linking as the same problem and used IMS to solve this task. Moro et al. (2014) exposed the differences of both tasks and also present a unified approach called Babelfy. An overview and analysis of the main approaches to Entity Linking is given by Shen et al. (2015). And while we use cosine to compute similarity between synsets, there are also a lot of similarity measures that only rely on a given resource, mostly WordNet. These measures are often functions that depend on information like glosses or on topological properties like shortest paths. Examples include (Wu and Palmer 1994) and (Leacock and Chodorow 1998);

Blanchard et al. (2005) give a good overview. A purely graph based approach to WSD was presented by Agirre et al. (2014).

6 Conclusion

We presented AutoExtend, a flexible method to learn embeddings for non-word objects in resources. AutoExtend is a general method that can be used for any set of embeddings and for any resource that imposes constraints of a certain type on the relation between words and other objects. Our experimental results show that AutoExtend can be applied to different tasks including Word Sense Disambiguation, Entity Linking, Word-in-Context Similarity, Word Similarity and Synset Alignment. It achieves state-of-the-art performance on Word-in-Context Similarity and Word Sense Disambiguation.

Acknowledgments
This work was funded by Deutsche Forschungsgemeinschaft (DFG SCHU 2246/2-2). We are grateful to Christiane Fellbaum for discussions leading up to this paper and to the anonymous reviewers for their comments.

References

Amine Abdaoui, Jérôme Azé, Sandra Bringay, and Pascal Poncelet. 2014. Feel: French extended emotional lexicon: Islrn: 041-639-484-224-2.

Heike Adel and Hinrich Schütze. 2014. Using mined coreference chains as a resource for a semantic task. In *Proceedings of EMNLP*.

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of NAACL*.

Eneko Agirre, Oier Lopez de Lacalle, and Aitor Soroa. 2014. Random walks for knowledge-based word sense disambiguation. *Computational Linguistics*, 40(1):57–84.

Silvio Amir, Ramón Astudillo, Wang Ling, Bruno Martins, Mario J. Silva, and Isabel Trancoso. 2015. Inesc-id: A regression model for large scale twitter sentiment lexicon induction. In *Proceedings of SemEval*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

A. R. Balamurali, Aditya Joshi, and Pushpak Bhattacharyya. 2011. Harnessing wordnet senses for supervised sentiment classification. In *Proceedings of EMNLP*.

Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. 2003a. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003b. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.

Sudha Bhingardive, Dhirendra Singh, Rudra Murthy V, Hanumant Redkar, and Pushpak Bhattacharyya. 2015. Unsupervised most frequent sense detection using word embeddings. In *Proceedings of ACL*.

Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Knowledge-powered deep learning for word embedding. In *Proceedings of ECML/PKDD*.

Emmanuel Blanchard, Mounira Harzallah, Henri Briand, and Pascale Kuntz. 2005. A typology of ontology-based semantic measures. In *Proceedings of EMOI - INTEROP*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of ACM SIGMOD*.

Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of AAAI*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2012. Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of AISTATS*.

Jan A Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. *arXiv preprint arXiv:1405.4273*.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of WWW*.
Elia Bruni, Nam Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *Journal of Artificial Intelligence Research*, 49(1):1–47.
Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. 2014. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior Research Methods*, 46(3):904–911.
Sung-Hyuk Cha. 2007. Comprehensive survey on distance/similarity measures between probability density functions. *Mathematical Models and Methods in Applied Sciences*, 1(4):300–307.
Ching-Yun Chang, Stephen Clark, and Brian Harrington. 2013. Getting creative with semantic similarity. In *Procedings of ICSC*.
Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. 2014. A unified model for word sense representation and disambiguation. In *Proceedings of EMNLP*.
Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*.
Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.
Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *Proceedings of NAACL*.
Ryan Cotterell, Hinrich Schütze, and Jason Eisner. 2016. Morphological smoothing and extrapolation of word embeddings. In *Proceedings of ACL*.
Gerard de Melo and Mohit Bansal. 2013. Good, great, excellent: Global inference of semantic intensities. *Transactions of the ACL*, 1:279–290.
Gerard de Melo and Gerhard Weikum. 2012. Uwn: A large multilingual lexical knowledge base. In *Proceedings of ACL, System Demonstrations*.
Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.
K. I. Diamantaras and S. Y. Kung. 1996. *Principal Component Neural Networks: Theory and Applications*. John Wiley & Sons, Inc.
Beate Dorow, Florian Laws, Lukas Michelbacher, Christian Scheible, and Jason Utt. 2009. A graph-theoretic algorithm for automatic extension of translation lexicons. In *Proceedings of GEMS*.
John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.
Ted Dunning. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74.
Sebastian Ebert, Ngoc Thang Vu, and Hinrich Schütze. 2015. A Linguistically Informed Convolutional Neural Network. In *Proceedings of WASSA*.
Günes Erkan and Dragomir R. Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479.
Ky Fan and Alan J Hoffman. 1955. Some metric inequalities in the space of matrices. *American Mathematical Society*, 6(1):111–116.
Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015a. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*.
Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015b. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*.
Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.
Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of WWW*.
Dániel Fogaras and Balázs Rácz. 2005. Scaling link-based similarity search. In *Proceedings of WWW*.
Daniel Fried and Kevin Duh. 2014. Incorporating both distributional and relational semantics in word representations. *arXiv preprint arXiv:1412.4369*.
Alex Graves. 2013. Generating sequences with recurrent neural networks. *CoRR*.
Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Learning sense-specific word embeddings by exploiting bilingual resources. In *Proceedings of COLING, Technical Papers*.
Iryna Gurevych. 2005. Using the structure of a conceptual network in computing semantic relatedness. In *Proceedings of IJCNLP*.
Ivan Habernal, Tomáš Ptáček, and Josef Steinberger. 2013. Sentiment Analysis in Czech Social Media Using Supervised Machine Learning. In *Proceedings of WASSA*.

Matthias Hagen, Martin Potthast, Michel Büchner, and Benno Stein. 2015. Webis: An Ensemble for Twitter Sentiment Detection. In *Proceedings of SemEval*.

Hussam Hamdan, Patrice Bellot, and Frederic Bechet. 2015. Lsislif: Feature extraction and label weighting for sentiment analysis in twitter. In *Proceedings of SemEval*.

Birgit Hamp, Helmut Feldweg, et al. 1997. Germanet-a lexical-semantic net for german. In *Proceedings of ACL, Workshops*.

Xianpei Han and Jun Zhao. 2010. Structural semantic relatedness: a knowledge-based method to named entity disambiguation. In *Proceedings of ACL*.

Taher H. Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of WWW*.

Bas Heerschop, Alexander Hogenboom, and Flavius Frasincar. 2011. Sentiment lexicon creation from lexical resources. In *Proceedings if BIS*.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Proceedings of NIPS*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2014. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *arXiv preprint arXiv:1408.3456*.

Cong Duy Vu Hoang and Min-Yen Kan. 2010. Towards automated related work summarization. In *Proceedings of COLING*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Minqing Hu and Bing Liu. 2004. Mining and Summarizing Customer Reviews. In *Proceedings of KDD*.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of ACL*.

Thad Hughes and Daniel Ramage. 2007. Lexical semantic relatedness with random graph walks. In *Proceedings of EMNLP-CoNLL*.

Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2015. Sensembed: learning sense embeddings for word and relational similarity. In *Proceedings of ACL*.

Tommi Jaakkola and David Haussler. 1998. Exploiting generative models in discriminative classifiers. In *Proceedings of NIPS*.

Sujay Kumar Jauhar, Chris Dyer, and Eduard Hovy. 2015. Ontologically grounded multi-sense representation learning for semantic vector space models. In *Proceedings of NAACL*.

Tony Jebara, Risi Kondor, and Andrew Howard. 2004. Probability product kernels. *Journal of Machine Learning Research*, 5:819–844.

Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *Proceedings of KDD*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014a. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of ACL*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014b. A convolutional neural network for modelling sentences. In *Proceedings of ACL*.

Pentti Kanerva. 1998. *Sparse distributed memory*. MIT Press.

Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159.

Douwe Kiela, Felix Hill, and Stephen Clark. 2015. Specializing word embeddings for similarity or relatedness. In *Proceedings of EMNLP*.

Adam Kilgarriff. 2001. English lexical sample task description. In *Proceedings of SENSEVAL-2*.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of EMNLP*.

Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. 2014. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, pages 723–762.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Proceedings of NIPS*.

Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

Igor Labutov and Hod Lipson. 2013. Re-embedding words. In *Proceedings of ACL*.

Florian Laws, Lukas Michelbacher, Beate Dorow, Christian Scheible, Ulrich Heid, and Hinrich Schütze. 2010. A linguistically grounded graph model for bilingual lexicon extraction. In *Proceedings of COLING*.

Claudia Leacock and Martin Chodorow. 1998. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283.

Elizabeth Leicht, Petter Holme, and Mark Newman. 2006. Vertex similarity in networks. *Physical Review E*, 73(2):026120.
Ronny Lempel and Shlomo Moran. 2000. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Computer Networks*, 33(1):387–401.
Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Proceedings of NIPS*.
Pei Li, Zhixu Li, Hongyan Liu, Jun He, and Xiaoyong Du. 2009. Using link-based content analysis to measure document similarity effectively. In *Proceedings of APWeb/WAIM*.
Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. 2010. Fast computation of simrank for static and dynamic information networks. In *Proceedings of EDBT*.
Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. 2015. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of ACL*.
Dmitry Lizorkin, Pavel Velikhov, Maxim Grinev, and Denis Turdakov. 2010. Accuracy estimate and optimization techniques for simrank computation. *The International Journal on Very Large Data Bases*, 19(1):45–66.
Minh-Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of CoNLL*.
Andrew J. McMinn, Yashar Moshfeghi, and Joemon M. Jose. 2013. Building a large-scale corpus for evaluating event detection on twitter. In *Proceedings of CIKM*.
Rada Mihalcea and Dragomir Radev. 2011. *Graph-based natural language processing and information retrieval*. Cambridge University Press.
Rada Mihalcea, Timothy Chklovski, and Adam Kilgarriff. 2004. The senseval-3 english lexical sample task. In *Proceedings of SENSEVAL-3*.
Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013b. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.
Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*.
George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.
Einat Minkov and William W. Cohen. 2008. Learning graph walk based similarity measures for parsed text. In *Proceedings of EMNLP*.
Einat Minkov and William W. Cohen. 2012. Graph based similarity measures for synonym extraction from parsed text. In *Proceedings of TextGraphs-7*.
Andriy Mnih and Geoffrey Hinton. 2007a. Three new graphical models for statistical language modelling. In *Proceedings of ICML*.
Andriy Mnih and Geoffrey E. Hinton. 2007b. Three New Graphical Models for Statistical Language Modelling. In *Proceedings of ICML*.
Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *Proceedings of NIPS*.
Saif M. Mohammad and Peter D. Turney. 2013. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*, 29(3).
Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *Proceedings of SemEval*.
Andrea Moro, Alessandro Raganato, and Roberto Navigli. 2014. Entity linking meets word sense disambiguation: a unified approach. *Transactions of the ACL*.
Pradeep Muthukrishnan, Dragomir Radev, and Qiaozhu Mei. 2010. Edge weight regularization over multiple graphs for similarity learning. In *Proceedings of ICDM*.
Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of ICML*.
Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. 2013. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *Proceedings of SemEval*.
Roberto Navigli and Simone Paolo Ponzetto. 2010. Babelnet: Building a very large multilingual semantic network. In *Proceedings of ACL*.
Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. 2014. Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of EMNLP*.
Kiem-Hieu Nguyen and Cheol-Young Ock. 2012. Semantic relatedness for biomedical word sense disambiguation. In *Proceedings of TextGraphs-7*.

Diarmuid Ó Séaghdha and Ann Copestake. 2008. Semantic classification with distributional kernels. In *Proceedings of COLING*.
Canberk Özdemir and Sabine Bergler. 2015. Clac-sentipipe: Semeval2015 subtasks 10 b,e, and task 11. In *Proceedings of SemEval*.
Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014a. Glove: Global vectors for word representation. In *Proceedings of EMNLP*.
Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014b. GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*.
Verónica Pérez-Rosas, Carmen Banea, and Rada Mihalcea. 2012. Learning Sentiment Lexicons in Spanish. In *Proceedings of LREC*.
Nataliia Plotnikova, Micha Kohl, Kevin Volkert, Stefan Evert, Andreas Lerner, Natalie Dykes, and Heiko Ermer. 2015. Klueless: Polarity classification and association. In *Proceedings of SemEval*.
Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*.
Delip Rao, David Yarowsky, and Chris Callison-Burch. 2008. Affinity measures based on the graph Laplacian. In *Proceedings of TextGraphs-3*.
Reinhard Rapp, Serge Sharoff, and Bogdan Babych. 2012. Identifying word translations from comparable documents without a seed lexicon. In *Proceedings of LREC*.
Joseph Reisinger and Raymond J Mooney. 2010. Multi-prototype vector-space models of word meaning. In *Proceedings of NAACL*.
Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M. Mohammad, Alan Ritter, and Veselin Stoyanov. 2015. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In *Proceedings of SemEval*.
Sascha Rothe and Hinrich Schütze. 2015. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of ACL*.
Sascha Rothe, Sebastian Ebert, and Hinrich Schütze. 2016. Ultradense word embeddings by orthogonal transformation. In *Proceedings of NAACL*.
Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.
David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive Modeling*, 5:213–220.
Josef Ruppenhofer, Jasper Brandes, Petra Steiner, and Michael Wiegand. 2015. Ordering adverbs by their scaling effect on adjective intensity. In *Proceedings of RANLP*.
Mehran Sahami and Timothy D. Heilman. 2006. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of WWW*.
Christian Scheible and Hinrich Schütze. 2012. Bootstrapping sentiment labels for unannotated documents with polarity pagerank. In *Proceedings of LREC*.
Christian Scheible, Florian Laws, Lukas Michelbacher, and Hinrich Schütze. 2010. Sentiment translation through multi-edge graphs. In *Proceedings of COLING*.
Christian Scheible. 2010. Sentiment translation through lexicon induction. In *Proceedings of ACL, Student Research Workshop*.
Helmut Schmid. 1994. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of NeMLaP*.
Helmut Schmid. 2004. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of COLING*.
Tobias Schnabel and Hinrich Schütze. 2014. Flors: Fast and simple domain adaptation for part-of-speech tagging. *Transactions of the ACL*, 2:15–26.
Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of EMNLP*.
Hinrich Schütze and Michael Walsh. 2008. A graph-theoretic model of lexical syntactic acquisition. In *Proceedings of EMNLP*.
Hinrich Schütze. 1992. Dimensions of meaning. In *Proceedings of IEEE - SC*.
Hinrich Schütze. 1998. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123.
Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of CoNLL*.
Aliaksei Severyn and Alessandro Moschitti. 2015. UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. In *Proceedings of SemEval*.
Raksha Sharma, Mohit Gupta, Astha Agarwal, and Pushpak Bhattacharyya. 2015. Adjective intensity and sentiment analysis. In *Proceedings of EMNLP*.

Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.
Karl Stratos, Michael Collins, and Daniel Hsu. 2015. Model-based word embeddings from decompositions of count matrices. In *Proceedings of ACL*.
Duyu Tang, Furu Wei, Bing Qin, Ming Zhou, and Ting Liu. 2014a. Building large-scale twitter-specific sentiment lexicon : A representation learning approach. In *Proceedings of COLING*.
Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014b. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In *Proceedings of ACL*.
Fei Tian, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tie-Yan Liu. 2014. A probabilistic model for learning multi-prototype word embeddings. In *Proceedings of COLING, Technical Papers*.
Peter D. Turney. 2002. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In *Proceedings of ACL*.
Kateřina Veselovská and Ondřej Bojar. 2013. Czech SubLex 1.0.
Ulli Waltinger. 2010. GermanPolarityClues: A Lexical Resource for German Sentiment Analysis. In *Proceedings of LREC*.
Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph and text jointly embedding. In *Proceedings of EMNLP*.
Tong Wang, Abdel-rahman Mohamed, and Graeme Hirst. 2015. Learning lexical embeddings with syntactic and lexicographic knowledge. In *Proceedings of ACL*.
Kilian Q. Weinberger and Lawrence K. Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244.
Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of HLT/EMNLP*.
Zhibiao Wu and Martha Palmer. 1994. Verbs semantics and lexical selection. In *Proceedings of ACL*.
Wensi Xi, Edward A. Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. 2005. Simfusion: measuring similarity using unified relationship matrix. In *Proceedings of SIGIR*.
Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. 2015. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of NAACL*.
Yadollah Yaghoobzadeh and Hinrich Schütze. 2015. Corpus-level fine-grained entity typing using contextual information. In *Proceedings of EMNLP*.
Wen-tau Yih, Geoffrey Zweig, and John C Platt. 2012. Polarity inducing latent semantic analysis. In *Proceedings of EMNLP*.
Wenpeng Yin, Tobias Schnabel, and Hinrich Schütze. 2015. Online updating of word representations for part-of-speech tagging. In *Proceedings of EMNLP*.
Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of ACL*.
Mo Yu and Mark Dredze. 2015. Learning composition models for phrase embeddings. *Transactions of the ACL*, 3:227–242.
Torsten Zesch and Iryna Gurevych. 2006. Automatically creating datasets for measures of semantic relatedness. In *Proceedings of the Workshop on Linguistic Distances*.
Zhihua Zhang, Guoshun Wu, and Man Lan. 2015. Ecnu: Multi-level sentiment analysis on twitter using traditional linguistic features and word embedding features. In *Proceedings of SemEval*.
Zhi Zhong and Hwee Tou Ng. 2010. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of ACL, System Demonstrations*.
Huaping Zhong, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. 2015. Aligning knowledge and text embeddings by entity descriptions. In *Proceedings of EMNLP*.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`.

Amine Abdaoui, Jérôme Azé, Sandra Bringay, and Pascal Poncelet. Feel: French extended emotional lexicon: Islrn: 041-639-484-224-2, 2014.

Heike Adel and Hinrich Schütze. Using mined coreference chains as a resource for a semantic task. In *Proceedings of EMNLP*, 2014.

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of NAACL*, 2009.

Eneko Agirre, Oier Lopez de Lacalle, and Aitor Soroa. Random walks for knowledge-based word sense disambiguation. *Computational Linguistics*, 40 (1):57–84, 2014.

Silvio Amir, Ramón Astudillo, Wang Ling, Bruno Martins, Mario J. Silva, and Isabel Trancoso. Inesc-id: A regression model for large scale twitter sentiment lexicon induction. In *Proceedings of SemEval*, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

A. R. Balamurali, Aditya Joshi, and Pushpak Bhattacharyya. Harnessing wordnet senses for supervised sentiment classification. In *Proceedings of EMNLP*, 2011.

Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003a.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3: 1137–1155, 2003b.

Sudha Bhingardive, Dhirendra Singh, Rudra Murthy V, Hanumant Redkar, and Pushpak Bhattacharyya. Unsupervised most frequent sense detection using word embeddings. In *Proceedings of ACL*, 2015.

Jiang Bian, Bin Gao, and Tie-Yan Liu. Knowledge-powered deep learning for word embedding. In *Proceedings of ECML/PKDD*, 2014.

Emmanuel Blanchard, Mounira Harzallah, Henri Briand, and Pascale Kuntz. A typology of ontology-based semantic measures. In *Proceedings of EMOI - INTEROP*, 2005.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of ACM SIGMOD*, 2008.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of AAAI*, 2011.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of AISTATS*, 2012.

Jan A Botha and Phil Blunsom. Compositional morphology for word representations and language modelling. *arXiv preprint arXiv:1405.4273*, 2014.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of WWW*, 1998.

Elia Bruni, Nam Khanh Tran, and Marco Baroni. Multimodal distributional semantics. *Journal of Artificial Intelligence Research*, 49(1):1–47, 2014.

Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior Research Methods*, 46(3):904–911, 2014.

Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007.

Ching-Yun Chang, Stephen Clark, and Brian Harrington. Getting creative with semantic similarity. In *Procedings of ICSC*, 2013.

Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of ACL*, 2016.

Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. A unified model for word sense representation and disambiguation. In *Proceedings of EMNLP*, 2014.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

Ryan Cotterell and Hinrich Schütze. Morphological word-embeddings. In *Proceedings of NAACL*, 2015.

Gerard de Melo and Mohit Bansal. Good, great, excellent: Global inference of semantic intensities. *Transactions of the ACL*, 1:279–290, 2013.

Gerard de Melo and Gerhard Weikum. Uwn: A large multilingual lexical knowledge base. In *Proceedings of ACL, System Demonstrations*, 2012.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

Konstantinos Diamantaras and Sun-Yuan Kung. *Principal Component Neural Networks: Theory and Applications*. John Wiley & Sons, Inc., 1996.

Beate Dorow, Florian Laws, Lukas Michelbacher, Christian Scheible, and Jason Utt. A graph-theoretic algorithm for automatic extension of translation lexicons. In *Proceedings of GEMS*, 2009.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

Sebastian Ebert, Ngoc Thang Vu, and Hinrich Schütze. A Linguistically Informed Convolutional Neural Network. In *Proceedings of WASSA*, 2015.

Günes Erkan and Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22: 457–479, 2004.

Ky Fan and Alan J Hoffman. Some metric inequalities in the space of matrices. *American Mathematical Society*, 6(1):111–116, 1955.

Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*, 2015a.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*, 2015b.

Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. *arXiv preprint arXiv:1605.02276*, 2016.

Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of WWW*, 2001.

Dániel Fogaras and Balázs Rácz. Scaling link-based similarity search. In *Proceedings of WWW*, 2005.

Daniel Fried and Kevin Duh. Incorporating both distributional and relational semantics in word representations. *arXiv preprint arXiv:1412.4369*, 2014.

Ferdinand Georg Frobenius. Über matrizen aus nicht negativen elementen. *Sitzungsberichte der Akademie der Wissenschaften zu Berlin*, pages 456–477, 1912.

Francis Galton. Co-relations and their measurement, chiefly from anthropometric data. *Proceedings of the Royal Society of London*, 45(273-279):135–145, 1888.

BIBLIOGRAPHY

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, 2013.

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning sense-specific word embeddings by exploiting bilingual resources. In *Proceedings of COLING, Technical Papers*, 2014.

Iryna Gurevych. Using the structure of a conceptual network in computing semantic relatedness. In *Proceedings of IJCNLP*, 2005.

Ivan Habernal, Tomáš Ptáček, and Josef Steinberger. Sentiment Analysis in Czech Social Media Using Supervised Machine Learning. In *Proceedings of WASSA*, 2013.

Matthias Hagen, Martin Potthast, Michel Büchner, and Benno Stein. Webis: An Ensemble for Twitter Sentiment Detection. In *Proceedings of SemEval*, 2015.

Hussam Hamdan, Patrice Bellot, and Frederic Bechet. Lsislif: Feature extraction and label weighting for sentiment analysis in twitter. In *Proceedings of SemEval*, 2015.

Birgit Hamp and Helmut Feldweg. Germanet-a lexical-semantic net for german. In *Proceedings of ACL, Workshops*, 1997.

Xianpei Han and Jun Zhao. Structural semantic relatedness: a knowledge-based method to named entity disambiguation. In *Proceedings of ACL*, 2010.

Taher H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of WWW*, 2002.

Bas Heerschop, Alexander Hogenboom, and Flavius Frasincar. Sentiment lexicon creation from lexical resources. In *Proceedings if BIS*, 2011.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of NIPS*, 2015.

Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *arXiv preprint arXiv:1408.3456*, 2014.

Cong Duy Vu Hoang and Min-Yen Kan. Towards automated related work summarization. In *Proceedings of COLING*, 2010.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Minqing Hu and Bing Liu. Mining and Summarizing Customer Reviews. In *Proceedings of KDD*, 2004.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of ACL*, 2012.

Thad Hughes and Daniel Ramage. Lexical semantic relatedness with random graph walks. In *Proceedings of EMNLP-CoNLL*, 2007.

Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. Sensembed: learning sense embeddings for word and relational similarity. In *Proceedings of ACL*, 2015.

Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Proceedings of NIPS*, 1998.

Sujay Kumar Jauhar, Chris Dyer, and Eduard Hovy. Ontologically grounded multi-sense representation learning for semantic vector space models. In *Proceedings of NAACL*, 2015.

Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *Journal of Machine Learning Research*, 5:819–844, 2004.

Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of KDD*, 2002.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of ACL*, 2014a.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of ACL*, 2014b.

Pentti Kanerva. *Sparse distributed memory*. MIT Press, 1998.

Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009.

Maurice George Kendall. *Rank correlation methods*. Griffin, 1948.

Douwe Kiela, Felix Hill, and Stephen Clark. Specializing word embeddings for similarity or relatedness. In *Proceedings of EMNLP*, 2015.

Adam Kilgarriff. English lexical sample task description. In *Proceedings of SENSEVAL-2*, 2001.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of EMNLP*, 2014.

Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, 50:723–762, 2014.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Proceedings of NIPS*, 2015a.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Proceedings of NIPS*, 2015b.

Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

Igor Labutov and Hod Lipson. Re-embedding words. In *Proceedings of ACL*, 2013.

Florian Laws, Lukas Michelbacher, Beate Dorow, Christian Scheible, Ulrich Heid, and Hinrich Schütze. A linguistically grounded graph model for bilingual lexicon extraction. In *Proceedings of COLING*, 2010.

Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283, 1998.

Elizabeth Leicht, Petter Holme, and Mark Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.

Ronny Lempel and Shlomo Moran. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Computer Networks*, 33(1):387–401, 2000.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of NIPS*, 2014.

Omer Levy, Yoav Goldberg, and Israel Ramat-Gan. Linguistic regularities in sparse and explicit word representations. In *Proceedings of CoNLL*, 2014.

Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. Fast computation of simrank for static and dynamic information networks. In *Proceedings of EDBT*, 2010.

Pei Li, Zhixu Li, Hongyan Liu, Jun He, and Xiaoyong Du. Using link-based content analysis to measure document similarity effectively. In *Proceedings of APWeb/WAIM*, 2009.

Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of ACL*, 2015.

Dmitry Lizorkin, Pavel Velikhov, Maxim Grinev, and Denis Turdakov. Accuracy estimate and optimization techniques for simrank computation. *The International Journal on Very Large Data Bases*, 19(1):45–66, 2010.

Minh-Thang Luong, Richard Socher, and Christopher D Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of CoNLL*, 2013.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, 2015.

Andrew J. McMinn, Yashar Moshfeghi, and Joemon M. Jose. Building a large-scale corpus for evaluating event detection on twitter. In *Proceedings of CIKM*, 2013.

Rada Mihalcea and Dragomir Radev. *Graph-based natural language processing and information retrieval*. Cambridge University Press, 2011.

Rada Mihalcea, Timothy Chklovski, and Adam Kilgarriff. The senseval-3 english lexical sample task. In *Proceedings of SENSEVAL-3*, 2004.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013b.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*, 2013c.

George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.

Einat Minkov and William W. Cohen. Learning graph walk based similarity measures for parsed text. In *Proceedings of EMNLP*, 2008.

Einat Minkov and William W. Cohen. Graph based similarity measures for synonym extraction from parsed text. In *Proceedings of TextGraphs-7*, 2012.

Richard von Mises and Hilda Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *Zeitschrift für Angewandte Mathematik und Mechanik*, 9(2): 152–164, 1929.

Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of ICML*, 2007a.

Andriy Mnih and Geoffrey E. Hinton. Three New Graphical Models for Statistical Language Modelling. In *Proceedings of ICML*, 2007b.

Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Proceedings of NIPS*, 2009.

Saif M. Mohammad and Peter D. Turney. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*, 29(3), 2013.

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *Proceedings of SemEval*, 2013.

Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244, 2014.

Pradeep Muthukrishnan, Dragomir Radev, and Qiaozhu Mei. Edge weight regularization over multiple graphs for similarity learning. In *Proceedings of ICDM*, 2010.

Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of ICML*, 2010.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *Proceedings of SemEval*, 2013.

Roberto Navigli and Simone Paolo Ponzetto. Babelnet: Building a very large multilingual semantic network. In *Proceedings of ACL*, 2010.

Neha Nayak, Gabor Angeli, and Christopher D Manning. Evaluating word embeddings using a representative suite of practical tasks. *Proceedings of ACL*, 2016.

Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of EMNLP*, 2014.

Kiem-Hieu Nguyen and Cheol-Young Ock. Semantic relatedness for biomedical word sense disambiguation. In *Proceedings of TextGraphs-7*, 2012.

Diarmuid Ó Séaghdha and Ann Copestake. Semantic classification with distributional kernels. In *Proceedings of COLING*, 2008.

Canberk Özdemir and Sabine Bergler. Clac-sentipipe: Semeval2015 subtasks 10 b,e, and task 11. In *Proceedings of SemEval*, 2015.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*, 2014a.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, 2014b.

Verónica Pérez-Rosas, Carmen Banea, and Rada Mihalcea. Learning Sentiment Lexicons in Spanish. In *Proceedings of LREC*, 2012.

Oskar Perron. Zur theorie der matrices. *Mathematische Annalen*, 64(2):248–263, 1907.

Nataliia Plotnikova, Micha Kohl, Kevin Volkert, Stefan Evert, Andreas Lerner, Natalie Dykes, and Heiko Ermer. Klueless: Polarity classification and association. In *Proceedings of SemEval*, 2015.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on EMNLP*, 2016.

Delip Rao, David Yarowsky, and Chris Callison-Burch. Affinity measures based on the graph Laplacian. In *Proceedings of TextGraphs-3*, 2008.

Reinhard Rapp, Serge Sharoff, and Bogdan Babych. Identifying word translations from comparable documents without a seed lexicon. In *Proceedings of LREC*, 2012.

Joseph Reisinger and Raymond J Mooney. Multi-prototype vector-space models of word meaning. In *Proceedings of NAACL*, 2010.

Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M. Mohammad, Alan Ritter, and Veselin Stoyanov. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In *Proceedings of SemEval*, 2015.

Sascha Rothe and Hinrich Schütze. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of ACL*, 2015.

Sascha Rothe, Sebastian Ebert, and Hinrich Schütze. Ultradense word embeddings by orthogonal transformation. In *Proceedings of NAACL*, 2016.

Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive Modeling*, 5:213–220, 1988.

Josef Ruppenhofer, Jasper Brandes, Petra Steiner, and Michael Wiegand. Ordering adverbs by their scaling effect on adjective intensity. In *Proceedings of RANLP*, 2015.

Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of WWW*, 2006.

Christian Scheible. Sentiment translation through lexicon induction. In *Proceedings of ACL, Student Research Workshop*, 2010.

Christian Scheible and Hinrich Schütze. Bootstrapping sentiment labels for unannotated documents with polarity pagerank. In *Proceedings of LREC*, 2012.

Christian Scheible, Florian Laws, Lukas Michelbacher, and Hinrich Schütze. Sentiment translation through multi-edge graphs. In *Proceedings of COLING*, 2010.

Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of NeMLaP*, 1994.

Helmut Schmid. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of COLING*, 2004.

Tobias Schnabel and Hinrich Schütze. Flors: Fast and simple domain adaptation for part-of-speech tagging. *Transactions of the ACL*, 2:15–26, 2014.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of EMNLP*, 2015.

Hinrich Schütze. Dimensions of meaning. In *Proceedings of IEEE - SC*, 1992.

Hinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998.

Hinrich Schütze and Michael Walsh. A graph-theoretic model of lexical syntactic acquisition. In *Proceedings of EMNLP*, 2008.

Roy Schwartz, Roi Reichart, and Ari Rappoport. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of CoNLL*, 2015.

Aliaksei Severyn and Alessandro Moschitti. UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. In *Proceedings of SemEval*, 2015.

Raksha Sharma, Mohit Gupta, Astha Agarwal, and Pushpak Bhattacharyya. Adjective intensity and sentiment analysis. In *Proceedings of EMNLP*, 2015.

Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.

Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

Karl Stratos, Michael Collins, and Daniel Hsu. Model-based word embeddings from decompositions of count matrices. In *Proceedings of ACL*, 2015.

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

Duyu Tang, Furu Wei, Bing Qin, Ming Zhou, and Ting Liu. Building large-scale twitter-specific sentiment lexicon : A representation learning approach. In *Proceedings of COLING*, 2014a.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In *Proceedings of ACL*, 2014b.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Fei Tian, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tie-Yan Liu. A probabilistic model for learning multi-prototype word embeddings. In *Proceedings of COLING, Technical Papers*, 2014.

Peter D. Turney. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In *Proceedings of ACL*, 2002.

Kateřina Veselovská and Ondřej Bojar. Czech SubLex 1.0, 2013. URL `http://hdl.handle.net/11858/00-097C-0000-0022-FF60-B`.

Ulli Waltinger. GermanPolarityClues: A Lexical Resource for German Sentiment Analysis. In *Proceedings of LREC*, 2010.

Tong Wang, Abdel-rahman Mohamed, and Graeme Hirst. Learning lexical embeddings with syntactic and lexicographic knowledge. In *Proceedings of ACL*, 2015.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph and text jointly embedding. In *Proceedings of EMNLP*, 2014.

Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of HLT/EMNLP*, 2005.

Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of ACL*, 1994.

Wensi Xi, Edward A. Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. Simfusion: measuring similarity using unified relationship matrix. In *Proceedings of SIGIR*, 2005.

Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of NAACL*, 2015.

Yadollah Yaghoobzadeh and Hinrich Schütze. Corpus-level fine-grained entity typing using contextual information. In *Proceedings of EMNLP*, 2015.

Wen-tau Yih, Geoffrey Zweig, and John C Platt. Polarity inducing latent semantic analysis. In *Proceedings of EMNLP*, 2012.

Wenpeng Yin, Tobias Schnabel, and Hinrich Schütze. Online updating of word representations for part-of-speech tagging. In *Proceedings of EMNLP*, 2015.

Mo Yu and Mark Dredze. Improving lexical embeddings with semantic knowledge. In *Proceedings of ACL*, 2014.

Mo Yu and Mark Dredze. Learning composition models for phrase embeddings. *Transactions of the ACL*, 3:227–242, 2015.

Torsten Zesch and Iryna Gurevych. Automatically creating datasets for measures of semantic relatedness. In *Proceedings of the Workshop on Linguistic Distances*, 2006.

Zhihua Zhang, Guoshun Wu, and Man Lan. Ecnu: Multi-level sentiment analysis on twitter using traditional linguistic features and word embedding features. In *Proceedings of SemEval*, 2015.

Huaping Zhong, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. Aligning knowledge and text embeddings by entity descriptions. In *Proceedings of EMNLP*, 2015.

Zhi Zhong and Hwee Tou Ng. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of ACL, System Demonstrations*, 2010.

Curriculum Vitae

Education

04/2013 – 09/2016	**Ph.D. Student** (CIS, LMU Munich, Germany) Research on Representation Learning
10/2006 – 04/2012	**Student** (LMU Munich, Germany) Diploma in Math, Grade: 1.7 Specializations: Discrete Mathematics Minor: Computer Science

Practical Experience

since 11/2016	**Research Software Engineer** (Google, Zurich) Research on Deep Learning
06/2015 – 09/2015	**Research Intern** (Microsoft, Redmond, USA) Exploring various neural network language models for improved speech recognition systems
10/2008 – 02/2012	**Student Trainee** (Microsoft, Unterschleißheim) Optimizing the SPAM identification process using techniques from information retrieval and machine learning
10/2007 – 08/2008	**Student Trainee** (Fujitsu Siemens Computers, Munich) Developing a GUI (Tcl/Tk) to view, run and automate system security tests
01/2007 – 08/2007	**Student Trainee** (Nokia Siemens Networks , Munich) Concept and realization of an auto login system for an intranet site