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 that 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; for example, 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, Joshi, and Bhattacharyya (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 thanks to AutoExtend embeddings for non-word objects in experiments on Word-in-Context Similarity, Word Sense Disambiguation (WSD), and several other tasks.

[*] Center for Information and Language Processing. E-mail: `rothe@google.com`.

Submission received: 11 January 2016; revised version received: 2 December 2016; accepted for publication: 13 March, 2017.

doi:10.1162/COLI_a_00294

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 embeddings on the one hand and (ii) the extension of these word embeddings to non-word objects in a resource on the other hand. If someone identifies 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. 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, Feldweg et al. 1997). AutoExtend does not require manually labeled 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. We show that AutoExtend achieves state-of-the-art performance on several tasks including WSD.

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 WSD, Entity Linking, Word Similarity, 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

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 $\vec{x} + \vec{y}$ 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*).

and a person. Then we can formalize the semantics of *queen*, *king*, *man*, and *woman* and their additive relations as follows:

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

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

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

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

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

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

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

Equations (5) and (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. Equation (7) is an example of how offsets are used for computing analogies, the application of the offset calculus that has received a lot of attention. Equations (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 the word *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 more 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; for example, 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.

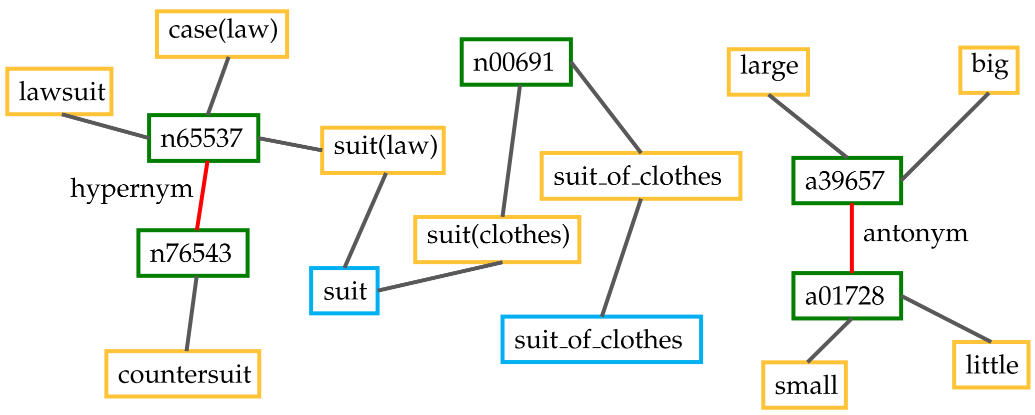

Figure 1
A small subset of the resource WordNet represented as a graph with three different types of nodes. Words (blue) are connected to lexemes (orange) by additive relations (gray). Lexemes are connected to synsets (green) and words. Synsets are connected to lexemes and also to other synsets by (dis)similarity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

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 of 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; note that a word can also be a short phrase. An example for the embedding space we want to learn can be seen in Figure 2.

(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 auto-encoding 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.

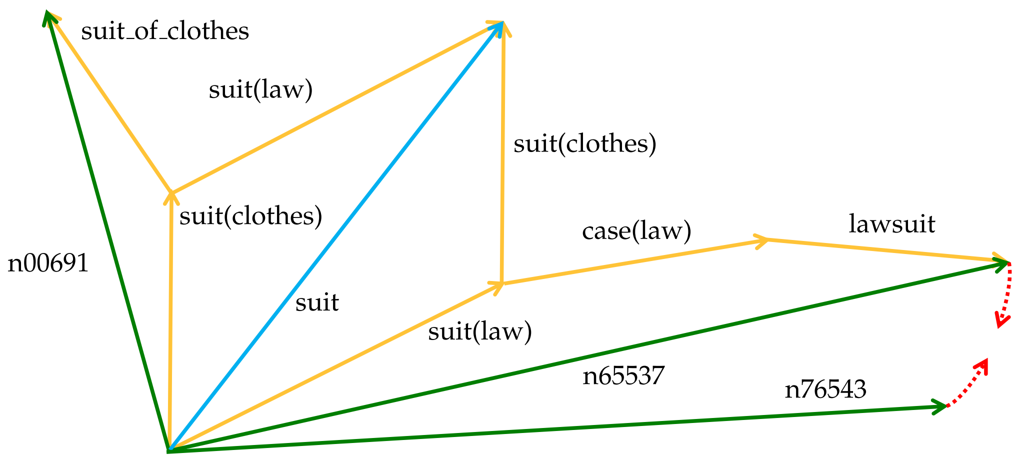

Figure 2
The embedding space we want to learn. The embeddings for words (blue) are given (input type). The embeddings for lexemes (orange) and synsets (green) have to be learned (unknown types). The additive edges define either that lexemes sum up to words or that they sum up to synsets. The similarity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

We present AutoExtend in more detail in the following sections. Although 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 between 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, for example, hyponymy or antonymy.

In the context of this article, 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 because 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

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.

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 $w^{(i)} \in \mathbb{R}^n$, synset vectors as $s^{(j)} \in \mathbb{R}^n$, and lexeme vectors as $l^{(i,j)} \in \mathbb{R}^n$, where $l^{(i,j)}$ 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 (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 $E^{(i,j)} \in \mathbb{R}^{n \times n}$:

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

We make the assumption that the dimensions in Equation (10) are independent of each other, that is, $E^{(i,j)}$ 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, Grefensette, and Blunsom 2014). (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.

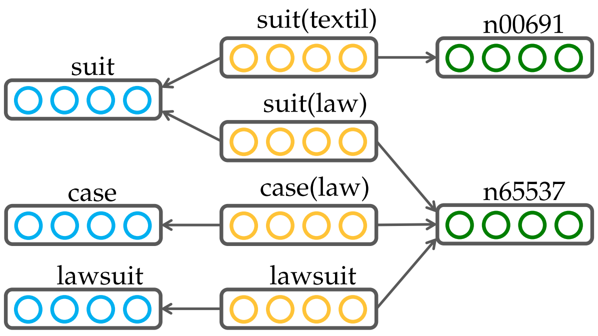

Figure 3
A subgraph of Figure 1 with additive edges only. Words are sums of their lexemes and synsets are sums of their lexemes. The circles are intended to show four different embedding dimensions.

Equations (8) and (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 Equation (11) it directly follows that:

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

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 $\mathbf{E} \in \mathbb{R}^{|S| \times n \times |V| \times n}$. We can then write Equation (12) as a tensor matrix product:

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

and Equation (13) states, that

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

Additionally, there is no interaction between different dimensions, so $\mathbf{E}^{i,d_1}_{j,d_2} = 0$ if $d_1 \neq d_2$. 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., $l^{(i,j)} \neq 0$ only if word i has a lexeme that is a member of synset j. To summarize the sparsity:

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

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

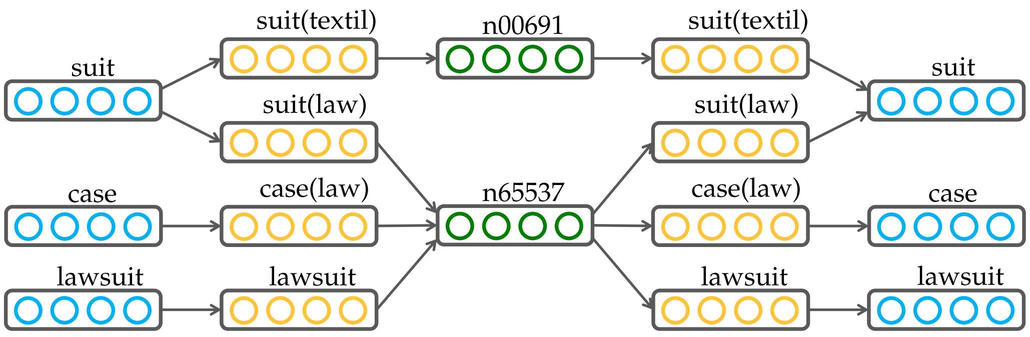

Figure 4
Encode and decode part of AutoExtend. The circles are intended to show four different embedding dimensions. These dimensions are treated as independent. The word constraint aligns the input and the output layer (blue columns), that is, the difference between encoding input and decoding output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

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 Equations (17) and (18) as:

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

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

and we also obtain the equivalent of Equation (13) for $D^{(j,i)}$:

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

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:

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

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

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 of Equations (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 **E** and **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 a 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 of other dimensions, a separate autoencoder for each dimension can be created and trained in parallel. Let $W \in \mathbb{R}^{|V| \times n}$ 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 $E^{(d)} = \mathbf{E}_{\cdot,d}^{\cdot,d} \in \mathbb{R}^{|S| \times |V|}$. We can write $S = \mathbf{E} \times W$ as:

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

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

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

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

$$\underset{E^{(d)},D^{(d)}}{\operatorname{argmin}} \|D^{(d)} E^{(d)} w^{(d)} - w^{(d)}\| \quad \forall d \tag{29}$$

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**:

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

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

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

with diag(x) being a square matrix having x on the main diagonal, $w^{(d)} = W_{\cdot,d}$ is again the d-th column of W, and vector $s^{(d)}$ is defined by Equation (27). Although 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—that is, 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 the

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

	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

relations that 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 $R \in \mathbb{R}^{r \times |S|}$, 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:

$$\underset{E^{(d)}}{\operatorname{argmin}} \| R E^{(d)} w^{(d)} \| \quad \forall d \tag{34}$$

2.7 Column Normalization

Our model is based on the premise that a word is the sum of its lexemes (Equation (8)). From the definition of $E^{(i,j)}$, we derived that $\mathbf{E} \in \mathbb{R}^{|S| \times n \times |V| \times n}$ should be normalized over the first dimension (Equation (15)). So $E^{(d)} \in \mathbb{R}^{|S| \times |V|}$ 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:

$$\underset{E^{(d)}}{\operatorname{argmin}} \| ([1, \ldots, 1]\, E^{(d)}) - [1, \ldots, 1] \, \| \quad \forall d \tag{35}$$

$$\underset{D^{(d)}}{\operatorname{argmin}} \| ([1, \ldots, 1]\, D^{(d)}) - [1, \ldots, 1] \, \| \quad \forall d \tag{36}$$

2.8 Implementation

Our training objective is the minimization of the sum of all constraints normalized by their output size, namely, the word constraint (Equation (29)) divided by the number of words, the lexeme constraint (Equation (33)) divided by the number of lexemes, and the similarity constraint (Equation (34)) divided by the number of similarities. Our training objective is minimization of the sum of these three normalized constraints, weighted by α (Equation (29)), β (Equation (33)), and $1 - \alpha - \beta$ (Equation (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 (Equations (35) and (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 the fact that whereas $E^{(d)}$ and $D^{(d)}$ should be column normalized in theory, there are many practical issues that prevent this (e.g., out-of-vocabulary words).

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

Table 2
Number of items in different resources and after the intersection with word2vec vectors (w2v). The analogs of synsets in Freebase are entities (“e:”) and types (“t:”).

	WordNet 2.1	∩ w2v	GermaNet 9.0	∩ w2v			Freebase	∩ w2v
words	147,478	54,570	109,683	89,160		≈	23,000	17,165
synsets	117,791	73,844	93,246	82,027	e:	≈	50,000,000	12,362
					t:	≈	26,000	3,516
lexemes	207,272	106,167	124,996	103,926		≈	47,000,000	27,478

3. Data

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

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 continuous bag-of-words (CBOW), with a window size of 5 (Mikolov et al. 2013b). 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, for example, inflected forms (*cars*) and proper nouns (*Tony Blair*). Conversely, many WordNet lemmata are not in the word2vec vocabulary, for example, *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 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. In this way, we can compute nearest neighbors across all three types, as shown in Figure 5.

3.2 GermaNet

For this set-up, 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 twice, 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/`.

nearest neighbors of W/suit

S/suit (businessman), L/suit (businessman), L/accommodate, 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.

3.3 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 subsequently, 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). An overview is shown in Table 3. Freebase contains a large number of entities with a single alias; we exclude these because 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

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. 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.

Table 3
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 Equation (30) and l_2 in Equation (31), respectively.

	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)

types.[5] For a type with n members, this would give us n^2 relations. This would result in a huge relation matrix that 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, for example, for entity typing (Yaghoobzadeh and Schütze 2015).

4. Experiments and Evaluation

We evaluate AutoExtend embeddings on the following tasks: WSD, 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 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 individual 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).

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.

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 † for $\alpha = 0.05$ and ‡ for $\alpha = 0.10$ (one-tailed Z-test).

			WSD		Entity Linking	
			SensEval-2	SensEval-3	*FACC dev*	FACC test
		size	4,328	3,944	10,000	10,000
individual 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†

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 (respectively, $s_i^{(j)}$) is element i of c (respectively, $s^{(j)}$). 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:

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

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 obtain 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 $\mathbf{S_{naive}}$**-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 and 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, Chklovski, and Kilgarriff 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. Although S-raw performs quite reasonably as a feature set alone, it hurts the performance when used as an additional feature set. Because 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.

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 (Gabrilovich, Ringgaard, and Subramanya 2013) corpus and extract all entity annotated words and their surrounding words—ten to the left and ten to the right. Recall that throughout this article, a word can also be a phrase. We remove aliases that occur fewer than 0.1 times as 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 1,000 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 and 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 that does not only provide 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.

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

7 This set is publicly available at `http://cistern.cis.lmu.de/`.

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

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 ...

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. 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 (respectively, synset) vector l either as the *simple average* of the lexeme (respectively, synset) vectors $l^{(ij)}$ (respectively, $s^{(j)}$) (method AvgSim, no dependence on c in this case) or as the average of the lexeme (respectively, 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.

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 to note 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

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 † for $\alpha = 0.05$ and ‡ for $\alpha = 0.10$ (one-tailed Z-test).

		AvgSim	AvgSimC
1	Huang et al. (2012)	62.8	65.7†
2	Tian et al. (2014)	–	65.4†
3	Neelakantan et al. (2014)	67.2	69.3
4	Chen, Liu, and Sun (2014)	66.2‡	68.9
5	words (word2vec)	66.7†	66.7†
6	synsets	63.2†	63.5†
7	lexemes	**68.3**	**70.2**

we proposed in the beginning of the article (Equation (8)). Thus, replacing a word's embedding by the sum of the embeddings of its senses could generally improve the quality of embeddings—see Huang et al. (2012) for a similar argument. We will provide 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, Tran, and Baroni 2014), RG (Rubenstein and Goodenough 1965), SIMLEX (Hill, Reichart, and Korhonen 2014), RW (Luong, Socher, and Manning 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). Because 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 obtain when we add lexeme vectors of the *encoding* part (see Equation (30))
3. the word embeddings W_2 that we obtain when we add lexeme vectors of the *decoding* part (see Equations 31 or 22)

We observe that each pair $(W_i, W_j), i \neq j$ of word embedding sets corresponds to a constraint of AutoExtend. (i) The column normalization constraint (Equation (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 (Equation (29)) will align W_0 and W_2. This was the initial idea of our system. (iii) The lexeme constraint (Equation (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 and 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 and 3). Although we observe a significant performance drop from W to W_1, we also observe a small improvement in W_2 for 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

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 baseline (line 1) in each column are marked † for $\alpha = 0.05$ and ‡ for $\alpha = 0.10$ (one-tailed Z-test).

	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**†

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, Le, and Sutskever (2013), 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:

$$\underset{L}{\operatorname{argmin}} \|WL - V\| \tag{38}$$

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. Thus 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 set (line 1 in Table 8) can be seen as an upper bound.

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.

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 $\alpha = 0.05$ (one-tailed Z-test) and ‡ for $\alpha = 0.10$ (one-tailed Z-test).

		Words	Synsets	
			dev	test
	size	2,000	*2,000*	2,000
1	word	0.943		
2	synset		***0.872***	**0.870**
3	$\text{synset}_{\text{naive}}$		*0.852*‡	0.826†

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 although 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. More important, however, is the actual task for which the embeddings are used. 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 and 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 (lines 6 and 7).

We found that some applications are not sensitive to the weighting; for example, for Entity Linking (line 2), the differences between weightings that result in non-zero weights to all three constraints are negligible (less than 0.3).

Table 9
Optimal weighting of the three constraints (word, lexeme, similarity) for different tasks.

				α	β	$1-\alpha-\beta$
	task	corpus	resource	word c.	lexeme c.	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

We also analyzed the impact of the four different relations in WordNet (see Table 1) on performance. In Tables 4 and 6, all four 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.

5. Related Work

5.1 Word Embeddings

Among the earliest work on distributed word representations (usually called "word embeddings" today) was Rumelhart, Hinton, and Williams (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 recently been a resurgence of work on embeddings (e.g., Bengio, Ducharme, and Vincent 2003; Mnih and Hinton 2007; Collobert et al. 2011; Mikolov et al. 2013a; Pennington, Socher, and Manning 2014), including methods that are SVD-based (Levy and Goldberg 2014; Stratos, Collins, and Hsu 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.

5.2 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 are needed. Although all these embeddings correspond to different word senses, there is no clear mapping between them and a resource like WordNet.

5.3 Sense Embeddings Related to Lexical Resources

Recently, Bhingardive et al. (2015) used WordNet to create sense embeddings similar to the naive method in this article. They used these sense embeddings to extract the most frequent synset. Chen, Liu, and Sun (2014) modified word2vec to learn sense embeddings, each corresponding to a WordNet synset. They used 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. Although WordNet is by far the most used resource, Iacobacci, Pilehvar, and Navigli (2015) computed sense embeddings with BabelNet, which is a superset of WordNet. They used a state-of-the-art WSD system to generate a large sense annotated corpus that 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.

5.4 Embeddings Using Lexical Resources

Other work tried 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, Mohamed, and Hirst (2015). Liu et al. (2015) also used WordNet to create ordinal similarity inequalities 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, Gao, and Liu (2014) used not only semantic but also morphological and syntactic knowledge to compute more effective word embeddings. Cotterell, Schütze, and Eisner (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.

5.5 Post-processing Embeddings

This prior work needs a training step to learn embeddings. In contrast, we can "Auto-Extend" any set of given word embeddings—without (re)training them. There is an increasing amount of work on taking existing word embeddings and combining 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 obtain better predictive performance on a task while not changing the space of embeddings. A similar approach was chosen by Faruqui et al. (2015) and called retrofitting. That 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 did not use additive relations nor did they compute embeddings for non-word objects. Jauhar, Dyer, and Hovy (2015) also used the same retrofitting technique to model sense embeddings. Their 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 than the embeddings on which they were trained (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, Hill, and Clark (2015) used retrofitting and joint-learning approaches to specialize their embeddings for either similarity or relatedness tasks.

5.6 Other Related Work

In this work, we treated WSD and Entity Linking as the same problem and used IMS to solve this task. Moro, Raganato, and Navigli (2014) exposed the differences of both tasks and also presented a unified approach, called Babelfy. An overview and analysis of the main approaches to Entity Linking was given by Shen, Wang, and Han (2015).

And whereas we use cosine to compute the similarity between synsets, there are also many 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, de Lacalle, and Soroa (2014).

6. Conclusions

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 article and to the anonymous reviewers for their comments.

References

Agirre, Eneko, Oier Lopez de Lacalle, and Aitor Soroa. 2014. Random walks for knowledge-based word sense disambiguation. *Computational Linguistics*, 40(1):57–84.

Balamurali, A. R., Aditya Joshi, and Pushpak Bhattacharyya. 2011. Harnessing Wordnet senses for supervised sentiment classification. In *Proceedings of EMNLP*, pages 1081–1091, Edinburgh.

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

Bhingardive, Sudha, Dhirendra Singh, Rudra Murthy V, Hanumant Redkar, and Pushpak Bhattacharyya. 2015. Unsupervised most frequent sense detection using word embeddings. In *Proceedings of ACL*, pages 1238–1243, Denver, CO.

Bian, Jiang, Bin Gao, and Tie-Yan Liu. 2014. Knowledge-powered deep learning for word embedding. In *Proceedings of ECML/PKDD*, pages 132–148, Nancy.

Blanchard, Emmanuel, Mounira Harzallah, Henri Briand, and Pascale Kuntz. 2005. A typology of ontology-based semantic measures. In *Proceedings of EMOI - INTEROP*, pages 13–14, Porto.

Bollacker, Kurt, 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*, pages 1247–1250, Vancouver.

Bordes, Antoine, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2012. Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of AISTATS*, pages 127–135, La Palma.

Bordes, Antoine, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of AAAI*, pages 301–306, San Francisco, CA.

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

Chen, Xinxiong, Zhiyuan Liu, and Maosong Sun. 2014. A unified model for word sense representation and disambiguation. In *Proceedings of EMNLP*, pages 1025–1035, Doha.

Collobert, Ronan, 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.

Cotterell, Ryan, Hinrich Schütze, and Jason Eisner. 2016. Morphological smoothing and extrapolation of word embeddings. In *Proceedings of ACL*, pages 1651–1660, Berlin.

Deerwester, Scott, 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.

Faruqui, Manaal, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*, pages 1606–1615, Denver, CO.

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

Finkelstein, Lev, 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*, pages 406–414, Hong Kong.

Fried, Daniel and Kevin Duh. 2014. Incorporating both distributional and relational semantics in word representations. *arXiv preprint arXiv:1412.4369*.

Gabrilovich, E, Ringgaard, M, & Subramanya, A. 2013. FACC1: Freebase annotation of ClueWeb corpora.

Guo, Jiang, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Learning sense-specific word embeddings by exploiting bilingual resources. In *Proceedings of COLING, Technical Papers*, pages 497–507, Dublin.

Gurevych, Iryna. 2005. Using the structure of a conceptual network in computing semantic relatedness. In *Proceedings of IJCNLP*, pages 767–778, Jeju Island.

Hamp, Birgit, Helmut Feldweg, et al. 1997. GermaNet - a lexical-semantic net for German. In *Proceedings of ACL, Workshops*, pages 9–15, Madrid.

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

Huang, Eric H., Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of ACL*, pages 873–882, Jeju Island.

Iacobacci, Ignacio, Mohammad Taher Pilehvar, and Roberto Navigli. 2015. Sensembed: Learning sense embeddings for word and relational similarity. In *Proceedings of ACL*, pages 95–105, Beijing.

Jauhar, Sujay Kumar, Chris Dyer, and Eduard Hovy. 2015. Ontologically grounded multi-sense representation learning for semantic vector space models. In *Proceedings of NAACL*, pages 683–693, Denver, CO.

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of ACL*, pages 655–665, Baltimore, MD.

Kanerva, Pentti. 1998. *Sparse Distributed Memory*. MIT Press.

Kanerva, Pentti. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159.

Kiela, Douwe, Felix Hill, and Stephen Clark. 2015. Specializing word embeddings for similarity or relatedness. In *Proceedings of EMNLP*, pages 2044–2048, Lisbon.

Kilgarriff, Adam. 2001. English lexical sample task description. In *Proceedings of SENSEVAL-2*, pages 17–20, Toulouse.

Labutov, Igor and Hod Lipson. 2013. Re-embedding words. In *Proceedings of ACL*, pages 489–493, Sofia.

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

Levy, Omer and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Proceedings of NIPS*, pages 2177–2185, Montreal.

Liu, Quan, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. 2015. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of ACL*, pages 1501–1511, Beijing.

Luong, Minh-Thang, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of CoNLL*, pages 104–113, Sofia.

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

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

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

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*, pages 3111–3119, Lake Tahoe.

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

Mnih, Andriy and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of ICML*, pages 641–648.

Moro, Andrea, Alessandro Raganato, and Roberto Navigli. 2014. Entity linking meets word sense disambiguation: a unified approach. *Transactions of the ACL*.

Neelakantan, Arvind, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. 2014. Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of EMNLP*, pages 1059–1069, Doha.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, pages 1532–1543, Doha.

Reisinger, Joseph and Raymond J. Mooney. 2010. Multi-prototype vector-space models of word meaning. In *Proceedings of NAACL*, pages 109–117, Los Angeles, CA.

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

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

Schütze, Hinrich. 1992. Dimensions of meaning. In *Proceedings of IEEE - SC*, pages 787–796, Raleigh, NC.

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

Shen, Wei, 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.

Stratos, Karl, Michael Collins, and Daniel Hsu. 2015. Model-based word embeddings from decompositions of count matrices. In *Proceedings of ACL*, pages 1282–1291, Beijing.

Tian, Fei, 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*.

Wang, Tong, Abdel-rahman Mohamed, and Graeme Hirst. 2015. Learning lexical embeddings with syntactic and lexicographic knowledge. In *Proceedings of ACL*, pages 458–463, Beijing.

Wang, Zhen, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph and text jointly embedding. In *Proceedings of EMNLP*, pages 1591–1601, Doha.

Wu, Zhibiao and Martha Palmer. 1994. Verbs semantics and lexical selection. In *Proceedings of ACL*, pages 133–138, Las Cruces.

Yaghoobzadeh, Yadollah and Hinrich Schütze. 2015. Corpus-level fine-grained entity typing using contextual information. In *Proceedings of EMNLP*, pages 715–725, Lisbon.

Yu, Mo and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of ACL*, pages 545–550, Baltimore, MD.

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

Zhong, Huaping, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. 2015. Aligning knowledge and text embeddings by entity descriptions. In *Proceedings of EMNLP*, pages 267–272, Lisbon.

Zhong, Zhi and Hwee Tou Ng. 2010. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of ACL, System Demonstrations*, pages 78–83, Uppsala.