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

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 1793–1803, Beijing, China, July 26-31, 2015.

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

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

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:

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

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 $\mathbf{E} \in \mathbb{R}^{|S| \times n \times |W| \times n}$. We can then write Eq. 5 as a tensor product:

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

while Eq. 6 states, that

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

Additionally, there is no interaction between different dimensions, so $\mathbf{E}^{i,d_1}_{j,d_2} = 0$ if $d_1 \neq d_2$. 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:

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

2.1 Learning

We adopt an autoencoding framework to learn embeddings for lexemes and synsets. To this end, we view the tensor equation $S = \mathbf{E} \otimes W$ 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 $D^{(j,i)}$:

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

and in tensor notation:

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

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

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

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

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

$$\underset{\mathbf{E},\mathbf{D}}{\operatorname{argmin}} \|\mathbf{D} \otimes \mathbf{E} \otimes W - W\| \tag{17}$$

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 $W \in \mathbb{R}^{|W| \times n}$ 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 $E^{(d)} = \mathbf{E}^{\cdot,d}_{\cdot,d} \in \mathbb{R}^{|S| \times |W|}$. We can write $S = \mathbf{E} \otimes W$ as:

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

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

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

where $D^{(d)} = \mathbf{D}^{\cdot,d}_{\cdot,d} \in \mathbb{R}^{|W| \times |S|}$ 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:

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

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 $\overline{W}$:

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

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

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

	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:

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

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

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

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,

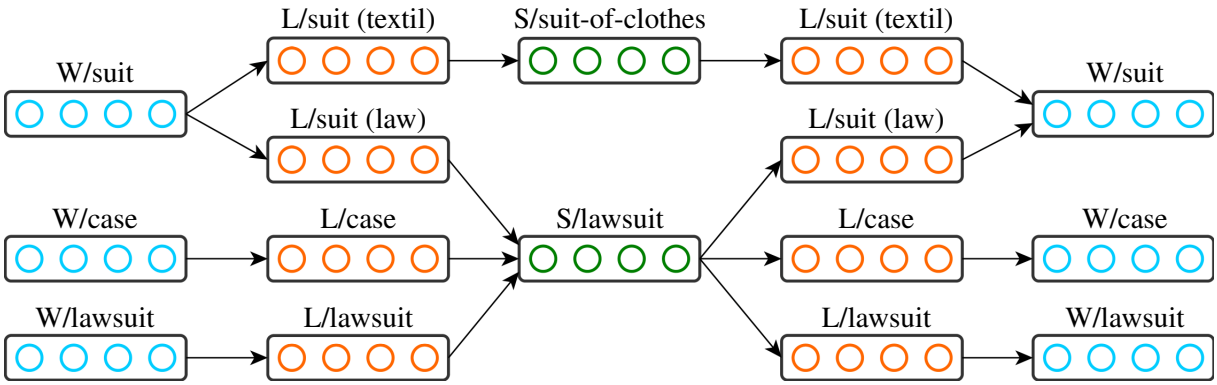

Figure 1: A small subgraph of WordNet. The circles are intended to show four different embedding dimensions. These dimensions are treated as independent. The synset constraints align the input and the output layer. The lexeme constraints align the second and fourth layers.

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 $\mathbf{E} \in \mathbb{R}^{|S| \times n \times |W| \times n}$ is normalized over the first dimension (Eq. 8). So $E^{(d)} \in \mathbb{R}^{|S| \times |W|}$ 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†	55.2†
	9	Rank 1 system	64.2†	72.9
	10	Rank 2 system	63.8†	72.6
	11	IMS	65.2‡	72.3‡
	12	IMS + S_{naive}-prod.	62.6†	69.4†
	13	IMS + S-cosine	65.1‡	72.4‡
	14	IMS + S-product	**66.5**	**73.6**
	15	IMS + S-raw	62.1†	66.8†
	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†	65.7†
2	Tian et al. (2014)	–	65.4†
3	Neelakantan et al. (2014)	67.2	69.3
4	Chen et al. (2014)	66.2†	68.9
5	words (word2vec)	66.6‡	66.6†
6	synsets	62.6†	63.7†
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 $\overline{l}^{(i,j)}$ 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:

$$\underset{L}{\operatorname{argmin}} \|LX - Y\| \tag{26}$$

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

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

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

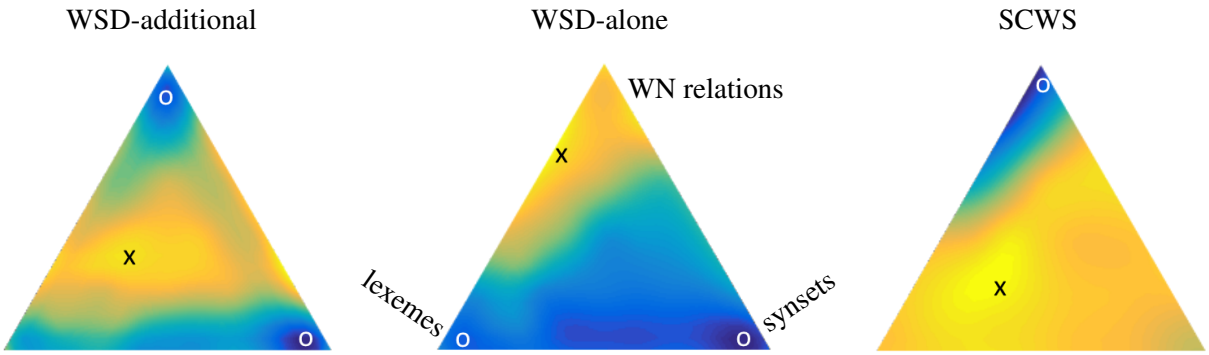

Figure 3: Performance of different weightings of the three constraints (WN relations:top, lexemes:left, synsets:right) on the three tasks WSD-additional, WSD-alone and SCWS. "x" indicates the maximum; "o" indicates a local minimum.

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*.

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.

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*.

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*.