Splitting Compounds by Semantic Analogy

Joachim Daiber* **Lautaro Quiroz†** **Roger Wechsler†** **Stella Frank***

*Institute for Logic, Language and Computation
University of Amsterdam
Science Park 107, 1098 XG Amsterdam
{J.Daiber,S.C.Frank}@uva.nl

†Graduate School of Informatics
University of Amsterdam
Science Park 904, 1098 XH Amsterdam
first.last@student.uva.nl

Abstract

Compounding is a highly productive word-formation process in some languages that is often problematic for natural language processing applications. In this paper, we investigate whether distributional semantics in the form of word embeddings can enable a deeper, i.e., more knowledge-rich, processing of compounds than the standard string-based methods. We present an unsupervised approach that exploits regularities in the semantic vector space (based on analogies such as "bookshop is to shop as bookshelf is to shelf") to produce compound analyses of high quality. A subsequent compound splitting algorithm based on these analyses is highly effective, particularly for ambiguous compounds. German to English machine translation experiments show that this semantic analogy-based compound splitter leads to better translations than a commonly used frequency-based method.

1 Introduction

In languages such as German, compound words are a frequent occurrence leading to difficulties for natural language processing applications, and in particular machine translation. Several methods for dealing with this issue—from shallow count-based methods to deeper but more complex neural network-based processing methods—have been proposed. The recent surge in practical models for distributional semantics has enabled a multitude of practical applications in many areas, most recently in morphological analysis (Soricut and Och, 2015). In this paper, we investigate whether similar methods can be utilized to perform deeper, i.e. more knowledge-rich, processing of compounds. A great asset of word embeddings are the regularities that their multi-dimensional vector space exhibits. Mikolov et al. (2013) showed that regularities such as "*king* is to *man* what *queen* is to *woman*" can be expressed and exploited in the form of basic linear algebra operations on the vectors produced by their method. This often-cited example can be expressed as follows: $v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$, where $v(.)$ maps a word into its word embedding in vector space.

In a very recent approach, Soricut and Och (2015) exploit these regularities for unsupervised morphology induction. Their method induces vector representations for basic morphological transformations in a fully unsupervised manner. String prefix and suffix replacement rules are induced directly from the data based on the idea that morphological processes can be modeled on the basis of *prototype* transformations, i.e. vectors that are good examples of a morphological process are applied to a word vector to retrieve its inflected form. A simple example of this idea is $\uparrow d_{car\underline{s}} = v(\text{cars}) - v(\text{car})$ and $v(\text{dogs}) \approx v(\text{dog}) + \uparrow d_{car\underline{s}}$, which expresses the assumption that the word *car* is to *cars* what *dog* is to *dogs*. The direction vector $\uparrow d_{car\underline{s}}$ represents the process of adding the plural morpheme *-s* to a noun.

While this intuition works well for frequently occurring inflectional morphology, it is not clear whether it extends to more semantically motivated derivational processes such as compounding. We study this question in the present paper. Our experiments are based on the German language, in which compounding is a highly productive phenomenon allowing for a potentially infinite number of combinations of words into compounds. This fact, coupled with the issue that many compounds are observed infrequently in data, leads to a data sparsity problem that hinders the processing of such languages. Our

contributions are as follows: After reviewing related work (Section 2), we study whether the regularities exhibited by the vector space also apply to compounds (Section 3). We examine the relationship between the components within compounds, as illustrated by the analogical relationship "*Hauptziel* is to *Ziel* what *Hauptader* is to *Ader*."[1] By leveraging this analogy we can then analyze the novel compound *Hauptmann* (captain) by searching for known string prefixes (e.g. *Haupt-*) and testing whether the resulting split compound (*Haupt|mann*) has a similar relation between its components (*haupt*, *mann*) as the prototypical example (*Haupt|ziel*). We induce the compound components and their prototypes and apply them in a greedy compound splitting algorithm (Section 4), which we evaluate on a gold standard compound splitting task (Section 4.3) and as a preprocessing step in a machine translation setup (Section 5).

2 Related work

Our methodology follows from recent work on morphology induction (Soricut and Och, 2015), which combines string edits with distributional semantics to split words into morphemes. In this model, morphemes are represented as string edits plus vectors, and are linked into derivation graphs. The authors consider prefix and suffix morphemes up to six characters in length; in contrast, our approach to noun compound splitting only considers components at least four characters long.

2.1 Splitting compounds for SMT

Dealing with word compounding in statistical machine translation (SMT) is essential to mitigate the sparse data problems that productive word generation causes. There are several issues that need to be addressed: *splitting* compound words into their correct components (i.e. disambiguating between split points), *deciding* whether to split a compound word at all, and, if translating into a compounding language, *merging* components into a compound word (something we do not address, but see Fraser et al. (2012) and Cap et al. (2014) for systems that do). Koehn and Knight (2003) address German compound splitting using a straightforward approach based on component frequency. They also present splitting approaches based on word alignments and POS tag information, but find that while the more resource-intensive approaches give better splitting performance (measured by gold-standard segmentations) the frequency-based method results in the best SMT performance (measured by BLEU). This is attributed to the fact that phrase-based MT system do not penalize the frequency-based method for over-splitting, since it can handle components as a phrase.

Nießen and Ney (2000), Popović et al. (2006) and Fritzinger and Fraser (2010) explore using morphological analyzers for German compound splitting, with mixed results. Since these approaches use heavy supervision within the morphological analyzer, they are orthogonal to our unsupervised approach.

It may be advantageous to split only compositional compounds, and leave lexicalized compounds whole. Weller et al. (2014) investigate this question by using distributional similarity to split only words that pass a certain threshold (i.e., where the parts proposed by the morphological analyzer are similar to the compound). Contrary to their hypothesis, they find no advantage in terms of SMT, again indicating that oversplitting is not a problem for phrase-based SMT. The use of distributional similarity as a cue for splitting is similar to the work presented in this paper. However, the approach we follow in this paper is fully unsupervised, requiring only word embeddings estimated from a monolingual corpus. Additionally, it stands out for its simplicity, making it easy to understand and implement.

2.2 Semantic compositionality

Noun compounding has also been treated within the field of distributional semantics. Reddy et al. (2011) examine English noun compounds and find that distributional co-occurrence can capture the relationship between compound parts and whole, as judged by humans in terms of 'literalness'. Schulte im Walde et al. (2013) replicate this result for German, and also show that simple window-based distributional vectors outperform syntax-based vectors.

[1] In vector algebra: $\uparrow d_{\underline{\text{Hauptziel}}} = v(\text{Hauptziel}) - v(\text{Ziel})$ and $v(\text{Hauptader}) \approx v(\text{Ader}) + \uparrow d_{\underline{\text{Hauptziel}}}$. The compounds translate to main goal (*Hauptziel*) and main artery (*Hauptader*). As a separate noun, *Haupt* means head.

3 Towards deeper processing of compound words

3.1 Unsupervised morphology induction from word embeddings

Our approach is based on the work of Soricut and Och (2015), who exploit regularities in the vector space to induce morphological transformations. The authors extract morphological transformations in the form of prefix and suffix replacement rules up to a maximum length of 6 characters. The method requires an initial candidate set which contains all possible prefix and suffix rules that occur in the monolingual corpus. For English, the candidate set contains rules such as `suffix:ed:ing`, which represents the suffix *ed* replaced by *ing* (e.g. *walked*→*walking*). This candidate set also contains overgenerated rules that do not reflect actual morphological transformations; for example `prefix:S:`ϵ[2] in *scream*→*cream*.

The goal is to filter the initial candidate set to remove spurious rules while keeping useful rules. For all word pairs a rule applies to, word embeddings are used to calculate a vector representing the transformation. For example, the direction vector for the rule `suffix:ing:ed` based on the pair (*walking*, *walked*) would be $\uparrow d_{walk\underline{\text{ing}\rightarrow\text{ed}}} = v(walked) - v(walking)$. For each rule there are thus potentially as many direction vectors as word pairs it applies to. A direction vector is considered to be meaning-preserving if it successfully predicts the affix replacements of other, similar word pairs. Specifically, each direction vector is applied to the first word in the other pair and an ordered list of suggested words is produced. For example, the direction vector $\uparrow d_{walk\underline{\text{ing}\rightarrow\text{ed}}}$ can be evaluated against (*playing*, *played*) by applying $\uparrow d_{walk\underline{\text{ing}\rightarrow\text{ed}}}$ to *playing* to produce the predicted word form: $v(played*) = v(playing) + \uparrow d_{walk\underline{\text{ing}\rightarrow\text{ed}}}$. This prediction is then compared against the true word embedding $v(played)$ using a generic evaluation function $E(v(played), v(playing) + \uparrow d_{walk\underline{\text{ing}\rightarrow\text{ed}}})$.[3] If the evaluation function passes a certain threshold, we say that the direction vector *explains* the word pair. Some direction vectors explain many word pairs while others might explain very few. To judge the explanatory power of a direction vector, a *hit rate* metric is calculated, expressing the percentage of applicable word pairs for which the vector makes good predictions.[4] Each direction vector has a hit rate and a set of word pairs that it explains (its evidence set). Apart from their varying explanatory power, morphological transformation rules are also possibly ambiguous. For example, the rule `suffix:`ϵ`:s` can describe both the pluralization of a noun (one *house*→two *houses*) and the 3rd person singular form of a verb (I *find*→she *finds*). Different direction vectors might explain the nouns and verbs separately.

Soricut and Och (2015) retain only the most explanatory vectors by applying a recursive procedure to find the minimal set of direction vectors explaining most word pairs. We call this set of direction vectors *prototypes*, as they represent a prototypical transformation for a rule and other words are formed *in analogy to* this particular word pair. Finally, Soricut and Och (2015) show that their prototypes can be applied successfully in a word similarity task for several languages.

3.2 Compound words and the semantic vector space

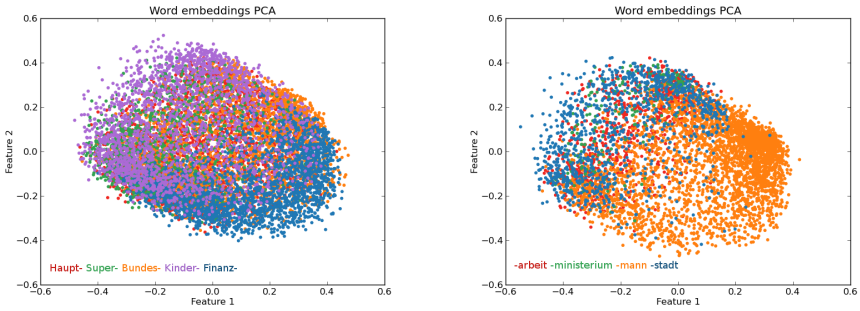
According to Lieber and Štekauer (2009), compounds can be classified into several groups based on whether the semantic head is part of the compound (*endocentric* compounds; a doghouse is a also a house) or whether the semantic head is outside of the compound (*exocentric* compounds; a skinhead is not a head). In this paper, we focus on endocentric compounds, which are also the most frequent type in German. Endocentric compounds consist of a modifier and a semantic head. The semantic head specifies the basic meaning of the word and the modifier restricts this meaning. In German, the modifiers come before the semantic head; hence, the semantic head is always the last component in the compound. When applying the idea of modeling morphological processes by semantic analogy to compounds, we can represent either the semantic head or the modifier of the compound as the transformation (like the morpheme rules above). Since the head carries the compound's basic meaning, we add the modifier's vector representation to the head word in order to restrict its meaning. We expect the resulting compound to be in the neighborhood of the head word in the semantic space (e.g., a *doghouse* is close to *house*).

[2] ϵ denotes the empty string.

[3] We follow Soricut and Och (2015) in defining E as either the *cosine* distance or the *rank* (position in the predictions).

[4] A transformation is considered a *hit* if the evaluated score is above a certain threshold for each evaluation method E.

[5] Gloss for modifiers: (a) main, (b) federal, (c) children, (d) finance. Heads: (e) piece of work, (f) ministry, (g) man, (h) city.



(a) Compounds with the same modifier. (b) Compounds with the same head.

Figure 1: Semantic representations of compounds based on (a) their modifiers and (b) their heads.

We illustrate this intuition by visualizing compound words and their parts in the vector space. All visualizations are produced by performing principal component analysis (PCA) to reduce the vector space from 500 to 2 dimensions. Figure 1 presents the visualization of various compounds with either the same head or the same modifier. For Figure 1a, we plot all German compounds in our dataset that have one of the modifiers *Haupt-*,[5a] *Super-, Bundes-*,[5b] *Kinder-*[5c] or *Finanz-*.[5d] Figure 1b, on the other hand, shows a plot for all German compounds that have one of the heads *-arbeit,*[5e] *-ministerium,*[5f] *-mann*[5g] or *-stadt*.[5h] Hence, the two plots illustrate the difference between learning vector representations for compound modifiers or heads. Words with the same modifier do not necessarily appear in close proximity in the embedding space. This is particularly true for modifiers that can be applied liberally to many head words, such as *Super-* or *Kinder-*.[5c] On the other hand, compounds with the same head are close in the embedding space. This observation is crucial to our method, as we aim to find direction vectors that generalize to as many word pairs as possible.

4 Compound induction from word embeddings

4.1 Compound extraction

Candidate extraction We compile an initial set of modifier candidates by extracting all possible prefixes with a minimum length of 4 characters.[6] We retain a modifier as a candidate if both the modifier and the rest of the word, i.e. the potential head of the compound, occur in the vocabulary. The initial candidate set contains 281K modifiers, which are reduced to 165K candidates by removing the modifiers occurring in only one word. The length of the average support set (i.e., the set of all compounds the modifier applies to) is 13.5 words. Table 1a shows the ten candidate modifiers with the biggest support sets. At this stage, the candidate set contains any modifier-head split that can be observed in the data, including candidates that do not reflect real compound splits.[7] Compound splits are not applied recursively here, as we assume that internal splits can be learned from the occurrences of the heads as individual words.[8]

Prototype extraction To find the prototype vectors that generalize best over the most words in the support set, we apply the same recursive algorithm as Soricut and Och (2015). The algorithm initially computes the direction vector for each *(modifier, compound)* pair in the support set by subtracting the embedding of the head from the embedding of the compound, e.g. $\uparrow d_{\underline{\text{dog}}\textit{house}} = v(\textit{doghouse}) - v(\textit{house})$. Each direction vector is then evaluated by applying it to all the word pairs in the support set, for example $v(\textit{owner}) + \uparrow d_{\underline{\text{dog}}\textit{house}} \stackrel{?}{=} v(\textit{dogowner})$ for the word pair *dog*|*owner*. If the resulting vector is close (according to E) to the vector of the actual target compound, we add it to the evidence set of the vector. The direction vector with the largest evidence set is selected as a prototype. All pairs this prototype explains are then removed and the algorithm is applied recursively until no direction vector explains

[6]For efficient computation, we use a directed acyclic word graph: `https://pypi.python.org/pypi/pyDAWG`.
[7]For example, as *Para* (a river) and *dies* (this) occur in the data, an incorrect candidate split occurs for *Para*|*dies* (paradise).
[8]For example, for *Haupt*|*bahn*|*hof* (main train station), we observe both *Haupt*|*bahnhof* and *Bahn*|*hof*.

	Modifier	Support		Modifier	Support
1.	*Land-*	8387	6.	*Landes-*	5189
2.	*Kinder-*	6249	7.	*Schul-*	5011
3.	*Haupt-*	5855	8.	*Jugend-*	4855
4.	*Lande-*	5637	9.	*Ober-*	4799
5.	*Stadt-*	5327	10.	*Groß-*	4656

(a) Modifiers by size of support set.

Prototype	Evidence words
$v_{\text{-Zeiger}}$	-Bewegung -Klicks -Klick -Tasten -Zeiger
$v_{\text{-Stämme}}$	-Mutanten -Gene -Hirnen -Stämme
$v_{\text{-Kostüm}}$	-Knopf -Hirn -Hirns -Kostüm
$v_{\text{-Steuerung}}$	-Ersatz -Bedienung -Steuerung

(b) Prototypes and evidence words for *Maus-*.[9]

Table 1: Overall most common modifiers and the prototypes extracted for the modifier *Maus-*.

at least t_{evd} compounds. As the evaluation function E we use the rank of the correct word in the list of predictions and experiment with $t_{\text{evd}} = \{10, 6, 4\}$. Lastly, for efficient computation we sample the evidence set down to a maximum number of 500 words.

4.2 Implementation considerations

We now turn to implementation considerations and perform an intrinsic evaluation of the prototypes.

Word embeddings We use the German data of the *News Crawl Corpora* (2007-2014).[10] The text is truecased and tokenized, and all punctuation characters are removed, resulting in approximately 2B tokens and a vocabulary size of 3M. We use *word2vec* to estimate the word embeddings.[11] We train 500-dimensional word embeddings using the skip-gram model, a window size of 5 and a minimum word frequency threshold of 2. The latter ensures that we find word embeddings for all words that occur at least twice in the corpus, which is useful as long compounds may occur only very few times.

Treatment of interfixes (Fugenelemente) For mostly phonetic reasons, German allows the insertion of a limited set of characters between the modifier and the head. As learning this set is not the aim of our work, we simply allow the fixed set of interfixes {*-s-*, *-es-*} to occur. For any combination of interfix and casing of the head word, we add the tuple of the two to the support set of the corresponding modifier.

What do the prototypes encode? An inspection of the prototypes for each modifier shows that the differences between them are not always clear cut. Often, however, each prototype expresses one specific sense of the modifier. Table 1b illustrates this on the example of the German modifier *Maus-* (Engl. *mouse*), which can refer to both the animal and the computer device. Although there are more than two prototype vectors, it is interesting to observe that the two word senses are almost fully separated.

Calculating the hit rate To evaluate the quality of the prototypes, we use the hit rate metric defined by Soricut and Och (2015). A direction vector's hit rate is the percentage of relevant word pairs that can be explained by the vector. A prediction is explainable if the actual target word is among the top t_{rank} predictions and, optionally, if the cosine similarity between the two is at least t_{sim}.

The implementation of this evaluation function E requires the calculation of the cosine distance between a newly created vector and the word vector of every item in the vocabulary. Since this score is calculated N times for every of the N word pairs (i.e., N^2 times), this is a computationally extremely expensive process. For more efficient computation, we use an approximate k-nearest neighbor search method.[12] While this is not a lossless search method, it offers an adjustable trade-off between the model's prediction accuracy and running time.[13] For a standard setting ($t_{\text{evd}} = 6, t_{\text{rank}} = 80$), the hit rates using approximate and exact rank are 85.9% and 60.9% respectively. This shows that the hit rates obtained with the approximate method are more optimistic, which will affect how the prototype vectors are extracted. Additionally, restricting both *rank* and *similarity* ($t_{\text{rank}} = 80, t_{\text{sim}} \geq 0.5$) leads to lower hit rates (25.9% for approximate and 15% for exact rank).

[9] Words are related to mouse pointer (*Zeiger*), biological genus (*Stämme*), mouse costume (*Kostüm*) and control (*Steuerung*).

[10] `http://www.statmt.org/wmt15/translation-task.html`

[11] `https://code.google.com/p/word2vec/`

[12] `https://github.com/spotify/annoy`

[13] With this fast approx. search method the total training time would be just below 7 days if run on a single 16 core machine.

	(a) Mean hit rate		(b) Mean cosine sim.		(c) % with prototypes		(d) Mean # of prototypes	
$t_{\text{rank}} =$	80	100	80	100	80	100	80	100
$t_{\text{evd}} = 4$	26%	22%	0.39	0.39	8.93%	9.52%	4.20	4.16
$t_{\text{evd}} = 6$	31%	26%	0.43	0.43	5.13%	5.47%	3.29	3.30
$t_{\text{evd}} = 10$	36%	31%	0.45	0.45	2.91%	3.14%	2.25	2.29

Table 2: Overview of the influence of the hyperparameters on prototype extraction.

Influence of thresholds Table 2 compares the parameters of our model based on (a) the mean hit rate, (b) cosine similarity, (c) the percentage of candidate modifiers with at least one prototype and (d) the mean number of prototypes per rule. Higher values of t_{evd} (minimum evidence set size) lead to better quality in terms of hit rate and cosine similarity as prototypes have to be able to cover a larger number of word pairs in order to be retained. The rank threshold t_{rank} also behaves as expected. Reducing t_{rank} to 80 means that the predicted vectors are of higher quality as they need to be closer to the true compound embeddings. Tables (c) and (d) illustrate that the more restrictive parameter settings reduce the amount of modifiers for which prototypes can be extracted. From a total of 165399 candidate prefixes, only 3%-10% are retained in the end for our settings. Similarly, the average number of prototypes per modifier also decreases with more restrictive settings. Interestingly, however, for the most restrictive setting ($t_{\text{evd}} = 10$, $t_{\text{rank}} = 80$), this number is still a relatively high 2 prototypes per vector.

4.3 Compound splitting

To obtain a clearer view of the quality of the extracted compound representations, we apply the prototypes to a compound splitting task.

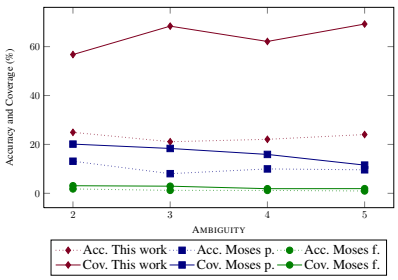
Splitting compounds by semantic analogy The extracted compound modifiers and their prototypes can be employed directly to split a compound into its components. Algorithm 1 presents the greedy algorithm applied to every word in the text. V is the word embedding vocabulary, M is the set of extracted modifiers with their prototypes, and PREFIXES(.) is a function returning all string prefixes.

```
1:  procedure DECOMPOUND(word, V, M)
2:      modifiers ← {m | p ← PREFIXES(word) if p ∈ M}
3:      if modifiers = ∅ OR word ∉ V then
4:          return word
5:      bestModifier ← ∅
6:      for modifier ∈ modifiers do
7:          head ← word without modifier                ▷ e.g. house ← doghouse without dog-
8:          if head ∈ V then
9:              for (head_proto, word_proto) ∈ modifier do
10:                 Evaluate "word is to head what word_proto is to head_proto"
11:                                                     ▷ e.g. doghouse is to house what dogowner is to owner
12:             Update bestModifier if this is the best match so far
13:     return word split based on bestModifier
```

Algorithm 1: Greedy compound splitting algorithm.

Compounds may only be split if (a) the full compound word is in the vocabulary V, i.e. it has been observed at least twice in the training data (Line 3), (b) it has a string prefix in the modifier set and this modifier has at least one prototype (Line 3), (c) the potential head word resulting from splitting the compound based on the modifier is also in our vocabulary (Line 8). The last case, namely that the compound head candidate is not in the vocabulary can occur for two reasons: either this potential head is a valid word that has not been observed frequently enough or, the more common reason, the substring is not a valid word in the language.[14] The algorithm's coverage can be increased by backing off to a frequency-based method if conditions (a) or (c) are violated. The core of the algorithm is the evaluation of meaning

[14]For example, when applying the algorithm to *Herrengarderobe* (male cloak room), two possible prefixes apply: *Herr* and *Herren*. In the first case, the remaining slice is *engarderobe*, which is not a valid word and thus the candidate prefix is discarded.



(a) Evaluation of highly ambiguous compounds.

	This work		Moses (partial)		Moses (full)	
Scenario	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.
Full test set	27.43	58.45	18.04	31.41	6.57	13.75
2 splits	24.94	56.75	13.13	20.13	1.79	3.11
3 splits	21.10	68.37	8.04	18.35	1.21	2.92
4 splits	22.09	62.11	9.98	15.91	1.19	1.90
5 splits	24.04	69.23	9.62	11.54	0.96	1.92

(b) Evaluation of all compounds and highly ambiguous compounds only.

Table 3: Gold standard evaluation of compound splitting.

preservation in Line 10. This evaluation is performed using the *rank*-based and *cosine similarity*-based evaluation functions. Modifiers that do not pass the thresholds defined for these functions are discarded as weak splits. To split compounds with more than two components, the algorithm is applied recursively.

General evaluation We use the test set from Henrich and Hinrichs (2011), which contains a list of 54569 compounds annotated with binary splits. As we only consider prefixes with a minimal length of 4 characters, we filter the test set accordingly, leaving 50651 compounds. Moses (Koehn et al., 2007) offers a compound splitter that splits a word if the geometric average of the frequencies of its components is higher than the frequency of the compound. We trained two instances of this compound splitter to use as references: one using the German monolingual dataset used to train the Word2Vec models and a second using a subset of the previous dataset.[15] Unlike our method, the two baseline systems do not consider the meaning preservation criteria of the compound splitting rules that are applied. Results for the full test set (accuracy and coverage, i.e. $\frac{|\text{correct splits}|}{|\text{compounds}|}$ and $\frac{|\text{compounds split}|}{|\text{compounds}|}$) are presented in the first row of Table 3b.

Evaluation of highly ambiguous compounds The strength of our method resides in the capacity to discriminate good candidate splits from bad ones. By capturing the meaning relation between compounds and their components, we are able to decide for a given word which splitting rule is the most appropriate. With this in mind, our approach should stand out in contexts where multiple split points may apply to a compound. We simulate different ambiguity scenarios based on Henrich and Hinrich's gold standard dataset: We extract compounds for which we find 2, 3, 4, and 5 potential split points.[16] The resulting test sets consists of 18571, 1815, 842 and 104 compounds, respectively. For all compound splitting experiments, we use the prototype vectors extracted with the parameters $t_{\text{evd}} = 6$ and $t_{\text{rank}} = 100$.

Table 3b presents accuracy and coverage for the compounds within the different ambiguity scenarios. To better visualize the trends for highly ambiguous compounds, we plot the accuracy and coverage scores in relation to the ambiguity of the compounds in Table 3a. The analogy-based method outperforms the frequency-based baselines in both coverage and accuracy. While for the Moses splitter, the coverage decreases with increasing ambiguity, the opposite behavior is shown by our approach, as having more possible splits results in a higher number of direction vectors increasing the likelihood of obtaining meaning-preserving splits. This experiment shows that the analogy-based compound splitter is advantageous for words that can potentially be explained by several candidate splits.

5 Compound splitting for machine translation

Translation setup We use the Moses decoder (Koehn et al., 2007) to train a phrase-based MT system on the English–German *Common crawl* parallel corpus and *WMT news test* 2010 (tuning). Word alignment is performed with Giza++ (Och and Ney, 2003). We use a 3rd order language model estimated using IRSTLM (Federico et al., 2008), as well as lexicalized reordering. The test data set is *WMT news*

[15] Subset: *News Crawl 2007-2009* (275M tokens, 2.09M types). Full set: *News Crawl 2007-2014* (2B tokens, 3M types).

[16] Each string prefix which occurs as a separate word produces a potential split point (indicated by }). The potential split points may not be linguistically motivated and can lead to correct (*general|stabs*) or incorrect splits (*gene}rals}tabs*). Examples include Einkauf}s}wagen, Eis}en}bahn}unternehmen, Wissen}s}chaft}s}park and Gene}ra}l}s}tab}s.

	(a) No comp. splitting			(b) OOV only			(c) Rare: c(w) < 20			(d) All words		
	Splits	BLEU	MTR	Splits	BLEU	MTR	Splits	BLEU	MTR	Splits	BLEU	MTR
Moses splitter	0	17.6	25.5	226	17.6	25.7[A]	231	17.6	25.7	244	17.9	25.8[A]
This work				317	17.6	25.8[A]	744	**18.2**[ABC]	**26.1**[ABC]	1616	17.7	26.3[A]

[A] Stat. sign. against (a) at $p < 0.05$ [B] Stat. sign. against Moses splitter at same c(w) at $p < 0.05$ [C] Stat. sign. against best Moses splitter (d) at $p < 0.05$

Table 4: Translation results for various integration methods.

test 2015,[17] which contains approx. 2100 de-en sentence pairs and 10000 tokens (with one reference translation). We compare our method against a baseline translation system with no compound splitter, and the same system implementing Moses' default compound splitting tool. The test set contains 2111 out-of-vocabulary word types (natural OOV words), which yields a total of 2765 unknown tokens, consisting mostly of compounds, brand names, and city names. This implies that 22.16% (word types) resp. 7.15% (tokens) of the test corpus are unknown to the baseline system.

Translation experiments To test the analogy-based compound splitter on a realistic setting, we perform a standard machine translation task. We translate a German text using a translation baseline system with no compound handling (a), a translation system integrating the standard Moses compound splitter tool trained using the best-performing settings, and a translation system using our analogy-based compound splitter. We test the following basic methods of integration: Splitting only words that are OOV to the translation model (b), splitting all words that occur less than 20 times in the training corpus (c),and applying the compound splitters to every word in the datasets (d). Table 4 shows the results of these translation experiments. For each experiment, we report BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), and the number of compound splits performed on the test set. Statistical significance tests are performed using bootstrap resampling (Koehn, 2004).

Discussion The results show that when applied without restrictions, our method splits a large number of words and leads to minor improvements. When applied only to rare words the splitter produces statistically significant improvements in both BLEU and METEOR over the best frequency-based compound splitter. This difference indicates that a better method for deciding which words the splitter should be applied to could lead to further improvements. Overall, the output of the analogy-based compound splitter is more beneficial to the machine translation system than the baseline splitter.

6 Conclusion

In this paper, we have studied whether regularities in the semantic word embedding space can be exploited to model the composition of compound words based on analogy. To approach this question, we made the following contributions: First, we evaluated whether properties of compounds can be found in the semantic vector space. We found that this space lends itself to modeling compounds based on their semantic head. Based on this finding, we discussed how to extract compound transformations and prototypes following the method of Soricut and Och (2015) and proposed an algorithm for applying these structures to compound splitting. Our experiments show that the analogy-based compound splitter outperforms a commonly used compound splitter on a gold standard task. Our novel compound splitter is particularly adept at splitting highly ambiguous compounds. Finally, we applied the analogy-based compound splitter in a machine translation task and found that it compares favorably to the commonly used shallow frequency-based method.

Acknowledgements Joachim Daiber is supported by the EXPERT (EXPloiting Empirical appRoaches to Translation) Initial Training Network (ITN) of the European Union's Seventh Framework Programme. Stella Frank is supported by funding from the European Unions Horizon 2020 research and innovation programme under grant agreement Nr. 645452.

[17] http://www.statmt.org/wmt15/translation-task.html

References

Fabienne Cap, Alexander Fraser, Marion Weller, and Aoife Cahill. 2014. How to produce unseen teddy bears: Improved morphological processing of compounds in SMT. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*.

Marcello Federico, Nicola Bertoldi, and Mauro Cettolo. 2008. IRSTLM: An open source toolkit for handling large scale language models. In *Proceedings of Interspeech 2008 - 9th Annual Conference of the International Speech Communication Association*.

Alexander Fraser, Marion Weller, Aoife Cahill, and Fabienne Cap. 2012. Modeling inflection and word-formation in SMT. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Fabienne Fritzinger and Alexander Fraser. 2010. How to avoid burning ducks: Combining linguistic analysis and corpus statistics for German compound processing. In *Proceedings of the ACL 2010 Joint Fifth Workshop on Statistical Machine Translation and Metrics (MATR)*.

Verena Henrich and Erhard W. Hinrichs. 2011. Determining immediate constituents of compounds in GermaNet. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing 2011*.

Philipp Koehn and Kevin Knight. 2003. Empirical methods for compound splitting. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 9th Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Rochelle Lieber and Pavol Štekauer. 2009. *The Oxford handbook of compounding*. Oxford University Press.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Sonja Nießen and Hermann Ney. 2000. Improving SMT quality with morpho-syntactic analysis. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Maja Popović, Daniel Stein, and Hermann Ney. 2006. Statistical machine translation of German compound words. In *Proceedings of FinTal - 5th International Conference on Natural Language Processing*.

Siva Reddy, Diana McCarthy, and Suresh Manandhar. 2011. An empirical study on compositionality in compound nouns. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*.

Sabine Schulte im Walde, Stefan Müller, and Stephen Roller. 2013. Exploring vector space models to predict the compositionality of German noun-noun compounds. In *Proceedings of the 2nd Joint Conference on Lexical and Computational Semantics (*SEM)*.

Radu Soricut and Franz Och. 2015. Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Marion Weller, Fabienne Cap, Stefan Müller, Sabine Schulte im Walde, and Alexander Fraser. 2014. Distinguishing degrees of compositionality in compound splitting for statistical machine translation. In *Proceedings of the First Workshop on Computational Approaches to Compound Analysis (ComaComa) at COLING*.