Word Sense Disambiguation for 158 Languages using Word Embeddings Only

Varvara Logacheva\textsuperscript{1}, Denis Teslenko\textsuperscript{2}, Artem Shelmanov\textsuperscript{1}, Steffen Remus\textsuperscript{2}, Dmitry Ustalov\textsuperscript{1}, Andrey Kutuzov\textsuperscript{3}, Ekaterina Artemova\textsuperscript{5}, Chris Biemann\textsuperscript{3}, Simone Paolo Ponzetto\textsuperscript{4}, Alexander Panchenko\textsuperscript{1}

\textsuperscript{1}Skolkovo Institute of Science and Technology, Moscow, Russia
\textsuperscript{2}Ural Federal University, Yekaterinburg, Russia
\textsuperscript{3}Universität Hamburg, Hamburg, Germany
\textsuperscript{4}Universität Mannheim, Mannheim, Germany
\textsuperscript{5}University of Oslo, Oslo, Norway
\textsuperscript{6}Higher School of Economics, Moscow, Russia

\texttt{v.logacheva@skoltech.ru}

Abstract

Disambiguation of word senses in context is easy for humans, but is a major challenge for automatic approaches. Sophisticated supervised and knowledge-based models were developed to solve this task. However, (i) the inherent Zipfian distribution of supervised training instances for a given word and/or (ii) the quality of linguistic knowledge representations motivate the development of completely unsupervised and knowledge-free approaches to word sense disambiguation (WSD). They are particularly useful for under-resourced languages which do not have any resources for building either supervised and/or knowledge-based models. In this paper, we present a method that takes as input a standard pre-trained word embedding model and induces a fully-fledged word sense inventory, which can be used for disambiguation in context. We use this method to induce a collection of sense inventories for 158 languages on the basis of the original pre-trained fastText word embeddings by \textsuperscript{[Grave et al., 2018]}, enabling WSD in these languages. Models and system are available online.

Keywords: word sense induction, word sense disambiguation, word embeddings, sense embeddings, graph clustering

1. Introduction

There are many polysemous words in virtually any language. If not treated as such, they can hamper the performance of all semantic NLP tasks \cite{Resnik2006}. Therefore, the task of resolving the polysemy and choosing the most appropriate meaning of a word in context has been an important NLP task for a long time. It is usually referred to as Word Sense Disambiguation (WSD) and aims at assigning meaning to a word in context. The majority of approaches to WSD are based on the use of knowledge bases, taxonomies, and other external manually built resources \cite{Moro2014, Upadhyay2018}. However, different senses of a polysemous word occur in very diverse contexts and can potentially be discriminated with their help. The fact that semantically related words occur in similar contexts, and diverse words do not share common contexts, is known as distributional hypothesis and underlies the technique of constructing word embeddings from unlabelled texts. The same intuition can be used to discriminate between different senses of individual words. There exist methods of training word embeddings that can detect polysemous words and assign them different vectors depending on their contexts \cite{Athiwarat2018, Jain2019}. Unfortunately, many widespread word embedding models, such as GloVe \cite{Pennington2014}, word2vec \cite{Mikolov2013}, fastText \cite{Bojanowski2017}, do not handle polysemous words. Words in these models are represented with single vectors, which were constructed from diverse sets of contexts corresponding to different senses. In such cases, their disambiguation needs knowledge-rich approaches.

We tackle this problem by suggesting a method of post-hoc unsupervised WSD. It does not require any external knowledge and can separate different senses of a polysemous word using only the information encoded in pre-trained word embeddings. We construct a semantic similarity graph for words and partition it into densely connected subgraphs. This partition allows for separating different senses of polysemous words. Thus, the only language resource we need is a large unlabelled text corpus used to train embeddings. This makes our method applicable to under-resourced languages. Moreover, while other methods of unsupervised WSD need to train embeddings from scratch, we perform retrofitting of sense vectors based on existing word embeddings.

We create a massively multilingual application for on-the-fly word sense disambiguation. When receiving a text, the system identifies its language and performs disambiguation of all the polysemous words in it based on pre-extracted word sense inventories. The system works for 158 languages, for which pre-trained fastText embeddings available \cite{Grave2018}. The created inventories are based on these embeddings. To the best of our knowledge, our system is the only WSD system for the majority of the presented languages. Although it does not match the state of the art for resource-rich languages, it is fully unsupervised and can be used for virtually any language.

The contributions of our work are the following:

\footnote{The full list languages is available at \texttt{fasttext.cc} and includes English and 157 other languages for which embeddings were trained on a combination of Wikipedia and CommonCrawl texts.}
• We release word sense inventories associated with fastText embeddings for 158 languages.
• We release a system that allows on-the-fly word sense disambiguation for 158 languages.
• We present egvi (Ego-Graph Vector Induction), a new algorithm of unsupervised word sense induction, which creates sense inventories based on pre-trained word vectors.

2. Related Work

There are two main scenarios for WSD: the supervised approach that leverages training corpora explicitly labelled for word sense, and the knowledge-based approach that derives sense representation from lexical resources, such as WordNet (Miller, 1995). In the supervised case WSD can be treated as a classification problem. Knowledge-based approaches construct sense embeddings, i.e. embeddings that separate various word senses.

SupWSD (Papandrea et al., 2017) is a state-of-the-art system for supervised WSD. It makes use of linear classifiers and a number of features such as POS tags, surrounding words, local collocations, word embeddings, and syntactic relations. GlossBERT model (Huang et al., 2019), which is another implementation of supervised WSD, achieves a significant improvement by leveraging gloss information. This model benefits from sentence-pair classification approach, introduced by Devlin et al. (2019) in their BERT contextualized embedding model. The input to the model consists of a context (a sentence which contains an ambiguous word) and a gloss (sense definition) from WordNet. The context-gloss pair is concatenated through a special token ([SEP]) and classified as positive or negative.

On the other hand, sense embeddings are an alternative to traditional word vector models such as word2vec, fastText or GloVe, which represent monosemous words well but fail for ambiguous words. Sense embeddings represent individual senses of polysemous words as separate vectors. Sense embeddings represent individual senses of polysemous words as separate vectors. They can be linked to an explicit inventory (Iacobacci et al., 2015) or induce a sense inventory from unlabelled data (Iacobacci andNavigli, 2019). LSTMEmbed (Iacobacci andNavigli, 2019) aims at learning sense embeddings linked to BabelNet (Navigli and Ponzetto, 2012), at the same time handling word ordering, and using pre-trained embeddings as an objective. Although it was tested only on English, the approach can be easily adapted to other languages present in BabelNet. However, manually labelled datasets as well as knowledge bases exist only for a small number of well-resourced languages. Thus, to disambiguate polysemous words in other languages one has to resort to fully unsupervised techniques.

The task of Word Sense Induction (WSI) can be seen as an unsupervised version of WSD. WSI aims at clustering word senses and does not require to map each cluster to a predefined sense. Instead of that, word sense inventories are induced automatically from the clusters, treating each cluster as a single sense of a word. WSI approaches fall into three main groups: context clustering, word ego-network clustering and synonyms (or substitute) clustering.

Context clustering approaches consist in creating vectors which characterise words’ contexts and clustering these vectors. Here, the definition of context may vary from window-based context to latent topic-aliike context. Afterwards, the resulting clusters are either used as senses directly (Kutuzov, 2018), or employed further to learn sense embeddings via Chinese Restaurant Process algorithm (Li and Jurafsky, 2015). AdaGram, a Bayesian extension of the Skip-Gram model (Bartunov et al., 2016), AutoSense, an extension of the LDA topic model (Amplayo et al., 2019), and other techniques.

Word ego-network clustering is applied to semantic graphs. The nodes of a semantic graph are words, and edges between them denote semantic relatedness which is usually evaluated with cosine similarity of the corresponding embeddings (Pelevina et al., 2016) or by PMI-like measures (Hope and Keller, 2013b). Word senses are induced via graph clustering algorithms, such as Chinese Whispers (Biemann, 2006) or MaxMax (Hope and Keller, 2013a). The technique suggested in our work belongs to this class of methods and is an extension of the method presented by Pelevina et al. (2016).

Synonyms and substitute clustering approaches create vectors which represent synonyms or substitutes of polysemous words. Such vectors are created using synonymy dictionaries (Ustalov et al., 2019) or context-dependent substitutes obtained from a language model (Amrami and Goldberg, 2018). Analogously to previously described techniques, word senses are induced by clustering these vectors.

3. Algorithm for Word Sense Induction

The majority of word vector models do not discriminate between multiple senses of individual words. However, a polysemous word can be identified via manual analysis of its nearest neighbours—they reflect different senses of the word. Table 1 shows manually sense-labelled most similar words. Here, the definition of context may vary from window-based context to latent topic-aliike context. Afterwards, the resulting clusters are either used as senses directly (Kutuzov, 2018), or employed further to learn sense embeddings via Chinese Restaurant Process algorithm (Li and Jurafsky, 2015). AdaGram, a Bayesian extension of the Skip-Gram model (Bartunov et al., 2016), AutoSense, an extension of the LDA topic model (Amplayo et al., 2019), and other techniques.

Word ego-network clustering is applied to semantic graphs. The nodes of a semantic graph are words, and edges between them denote semantic relatedness which is usually evaluated with cosine similarity of the corresponding embeddings (Pelevina et al., 2016) or by PMI-like measures (Hope and Keller, 2013b). Word senses are induced via graph clustering algorithms, such as Chinese Whispers (Biemann, 2006) or MaxMax (Hope and Keller, 2013a). The technique suggested in our work belongs to this class of methods and is an extension of the method presented by Pelevina et al. (2016).

Synonyms and substitute clustering approaches create vectors which represent synonyms or substitutes of polysemous words. Such vectors are created using synonymy dictionaries (Ustalov et al., 2019) or context-dependent substitutes obtained from a language model (Amrami and Goldberg, 2018). Analogously to previously described techniques, word senses are induced by clustering these vectors.

3. Algorithm for Word Sense Induction

The majority of word vector models do not discriminate between multiple senses of individual words. However, a polysemous word can be identified via manual analysis of its nearest neighbours—they reflect different senses of the word. Table 1 shows manually sense-labelled most similar words. Here, the definition of context may vary from window-based context to latent topic-aliike context. Afterwards, the resulting clusters are either used as senses directly (Kutuzov, 2018), or employed further to learn sense embeddings via Chinese Restaurant Process algorithm (Li and Jurafsky, 2015). AdaGram, a Bayesian extension of the Skip-Gram model (Bartunov et al., 2016), AutoSense, an extension of the LDA topic model (Amplayo et al., 2019), and other techniques.

Word ego-network clustering is applied to semantic graphs. The nodes of a semantic graph are words, and edges between them denote semantic relatedness which is usually evaluated with cosine similarity of the corresponding embeddings (Pelevina et al., 2016) or by PMI-like measures (Hope and Keller, 2013b). Word senses are induced via graph clustering algorithms, such as Chinese Whispers (Biemann, 2006) or MaxMax (Hope and Keller, 2013a). The technique suggested in our work belongs to this class of methods and is an extension of the method presented by Pelevina et al. (2016).

Synonyms and substitute clustering approaches create vectors which represent synonyms or substitutes of polysemous words. Such vectors are created using synonymy dictionaries (Ustalov et al., 2019) or context-dependent substitutes obtained from a language model (Amrami and Goldberg, 2018). Analogously to previously described techniques, word senses are induced by clustering these vectors.

3. Algorithm for Word Sense Induction

The majority of word vector models do not discriminate between multiple senses of individual words. However, a polysemous word can be identified via manual analysis of its nearest neighbours—they reflect different senses of the word. Table 1 shows manually sense-labelled most similar words. Here, the definition of context may vary from window-based context to latent topic-aliike context. Afterwards, the resulting clusters are either used as senses directly (Kutuzov, 2018), or employed further to learn sense embeddings via Chinese Restaurant Process algorithm (Li and Jurafsky, 2015). AdaGram, a Bayesian extension of the Skip-Gram model (Bartunov et al., 2016), AutoSense, an extension of the LDA topic model (Amplayo et al., 2019), and other techniques.

Word ego-network clustering is applied to semantic graphs. The nodes of a semantic graph are words, and edges between them denote semantic relatedness which is usually evaluated with cosine similarity of the corresponding embeddings (Pelevina et al., 2016) or by PMI-like measures (Hope and Keller, 2013b). Word senses are induced via graph clustering algorithms, such as Chinese Whispers (Biemann, 2006) or MaxMax (Hope and Keller, 2013a). The technique suggested in our work belongs to this class of methods and is an extension of the method presented by Pelevina et al. (2016).

Synonyms and substitute clustering approaches create vectors which represent synonyms or substitutes of polysemous words. Such vectors are created using synonymy dictionaries (Ustalov et al., 2019) or context-dependent substitutes obtained from a language model (Amrami and Goldberg, 2018). Analogously to previously described techniques, word senses are induced by clustering these vectors.
3.1. SenseGram: A Baseline Graph-based Word Sense Induction Algorithm

SenseGram is the method proposed by Pelevina et al. (2016) that separates nearest neighbours to induce word senses and constructs sense embeddings for each sense. It starts by constructing an ego-graph (semantic graph centred at a particular word) of the word and its nearest neighbours. The edges between the words denote their semantic relatedness, e.g. the two nodes are joined with an edge if cosine similarity of the corresponding embeddings is higher than a pre-defined threshold. The resulting graph can be clustered into subgraphs which correspond to senses of the word. The sense vectors are then constructed by averaging embeddings of words in each resulting cluster. In order to use these sense vectors for word sense disambiguation in text, the authors compute the probabilities of sense vectors of a word given its context or the similarity of the sense vectors to the context.

3.2. ego-Graph (Ego-Graph Vector Induction): A Novel Word Sense Induction Algorithm

Induction of Sense Inventories: One of the downsides of the described above algorithm is noise in the generated graph, namely, unrelated words and wrong connections. They hamper the separation of the graph. Another weak point is the imbalance in the nearest neighbour list, when a large part of it is attributed to the most frequent sense, not sufficiently representing the other senses. This can lead to construction of incorrect sense vectors.

We suggest a more advanced procedure of graph construction that uses the interpretability of vector addition and subtraction operations in word embedding space (Mikolov et al., 2013) while the previous algorithm only relies on the list of nearest neighbours in word embedding space. The key innovation of our algorithm is the use of vector subtraction to find pairs of most dissimilar graph nodes and construct the graph only from the nodes included in such “anti-edges”. Thus, our algorithm is based on graph-based word sense induction, but it also relies on vector-based operations between word embeddings to perform filtering of graph nodes. Analogously to the work of Pelevina et al. (2016), we construct a semantic relatedness graph from a list of nearest neighbours, but we filter this list using the following procedure:

1. Extract a list $\mathcal{N} = \{w_1, w_2, ..., w_N\}$ of $N$ nearest neighbours for the target (ego) word $w$.

2. Compute a list $\Delta = \{\delta_1, \delta_2, ..., \delta_N\}$ for each $w_i$ in $\mathcal{N}$, where $\delta_i = w - w_i$. The vectors in $\delta$ contain the components of sense of $w$ which are not related to the corresponding nearest neighbours from $\mathcal{N}$.

3. Compute a list $\overline{\mathcal{N}} = \{\overline{w_1}, \overline{w_2}, ..., \overline{w_N}\}$, such that $\overline{w_i}$ is in the top nearest neighbours of $\delta_i$ in the embedding space. In other words, $\overline{w_i}$ is a word which is the most similar to the target (ego) word $w$ and least similar to its neighbour $w_i$. We refer to $\overline{w_i}$ as an anti-pair of $w_i$.

To clarify this, consider the target (ego) word $w = \text{python}$, its top similar term $w_1 = \text{Java}$ and the resulting anti-pair $\overline{w_1} = \text{snake}$ which is the top related term of $\delta_1 = w - w_1$. Together they form an anti-edge $(w_1, \overline{w_1}) = (\text{Java}, \text{snake})$ composed of a pair of semantically dissimilar terms.

4. Construct $V$, the set of vertices of our semantic graph $G = (V, E)$ from the list of anti-edges $\overline{E}$, with the following recurrent procedure: $V = V \cup \{w_i, \overline{w_i} : w_i \in \mathcal{N}, \overline{w_i} \in \mathcal{N}\}$, i.e. we add a word from the list of nearest neighbours and its anti-pair only if both of them are nearest neighbours of the original word $w$. We do not add $w$’s nearest neighbours if their anti-pairs do not belong to $\mathcal{N}$. Thus, we add only words which can help discriminating between different senses of $w$.

5. Construct the set of edges $E$ as follows. For each $w_i \in \mathcal{N}$ we extract a set of its $K$ nearest neighbours $\mathcal{N}'_i = \{u_1, u_2, ..., u_K\}$ and define $E = \{(w_i, u_j) : w_i \in V, u_j \in V, u_j \in \mathcal{N}'_i, u_j \neq \overline{w_i}\}$. In other words, we remove edges between a word $w_i$ and its nearest neighbour $u_j$ if $u_j$ is also its anti-pair. According to our hypothesis, $w_i$ and $\overline{w_i}$ belong to different senses of $w$, so they should not be connected (i.e. we never add anti-edges into $E$). Therefore, we consider any connection between them as noise and remove it.

Note that $N$ (the number of nearest neighbours for the target word $w$) and $K$ (the number of nearest neighbours of $w_{\text{na}}$) do not have to match. The difference between these parameters is the following. $N$ defines how many words will be considered for the construction of ego-graph. On the other hand, $K$ defines the degree of relatedness between words in the ego-graph — if $K = 50$, then we will connect vertices $w$ and $u$ with an edge only if $u$ is in the list of 50 nearest neighbours of $w$. Increasing $K$ increases the graph connectivity and leads to lower granularity of senses. According to our hypothesis, nearest neighbours of $w$ are grouped into clusters in the vector space, and each of the clusters corresponds to a sense of $w$. The described vertices selection procedure allows picking the most representative members of these clusters which are better at discriminating between the clusters. In addition to that, it helps dealing with the cases when one of the clusters is over-represented in the nearest neighbour list. In this case, many elements of such a cluster are not added to $V$ because their anti-pairs fall outside the nearest neighbour list. This also improves the quality of clustering.

After the graph construction, the clustering is performed using the Chinese Whispers algorithm (Biemann, 2006). This is a bottom-up clustering procedure that does not require to pre-define the number of clusters, so it can correctly process polysemous words with varying numbers of senses as well as unambiguous words.

Figure I shows an example of the resulting pruned graph of the word $\text{Ruby}$ for $N = 50$ nearest neighbours in terms of the fastText cosine similarity. In contrast to the baseline method by Pelevina et al. (2016) where all 50 terms are
clustered, in the method presented in this section we sparsify the graph by removing 13 nodes which were not in the set of the “anti-edges” i.e. pairs of most dissimilar terms out of these 50 neighbours. Examples of anti-edges i.e. pairs of most dissimilar terms for this graph include: (Haskell, Sapphire), (Garret, Rails), (Opal, Rubyist), (Hazel, Ruby-On-Rails), and (CoffeeScript, Opal).

**Labelling of Induced Senses** We label each word cluster representing a sense to make them and the WSD results interpretable by humans. Prior systems used hyponyms to label the clusters (Ruppert et al., 2015 [Panchenko et al., 2017], e.g. “animal” in the “python (animal)”). However, neither hyponyms nor rules for their automatic extraction are available for all 158 languages. Therefore, we use a simpler method to select a keyword which would help to interpret each cluster. For each graph node \( v \in V \) we count the number of anti-edges it belongs to: \( \text{count}(v) = |\{(w_i, \pi_j) : (w_i, \pi_j) \in E \land (v = w_i \lor v = \pi_j)\}| \). A graph clustering yields a partition of \( V \) into \( n \) clusters: \( V = \{V_1, V_2, ..., V_n\} \). For each cluster \( V_i \) we define a keyword \( w_i^\text{key} \) as the word with the largest number of anti-edges \( \text{count}(\cdot) \) among words in this cluster.

**Word Sense Disambiguation** We use keywords defined above to obtain vector representations of senses. In particular, we simply use word embedding of the keyword \( w_i^\text{key} \) as a sense representation \( s_i \) of the target word \( w \) to avoid explicit computation of sense embeddings like in (Pelevina et al., 2016). Given a sentence \( \{w_1, w_2, ..., w_i, w, w_{j+1}, ..., w_n\} \) represented as a matrix of word vectors, we define the context of the target word \( w \) as \( \mathbf{c}_w = \sum_{j=1}^n w_j \). Then, we define the most appropriate sense \( \hat{s} \) as the sense with the highest cosine similarity to the embedding of the word’s context:

\[
\hat{s} = \arg \max_{s_i} \frac{\mathbf{c}_w \cdot s_i}{||\mathbf{c}_w|| \cdot ||s_i||}.
\]

### 4. System Design

We release a system for on-the-fly WSD for 158 languages. Given textual input, it identifies polysemous words and retrieves senses that are the most appropriate in the context.

#### 4.1. Construction of Sense Inventories

To build word sense inventories (sense vectors) for 158 languages, we utilised GPU-accelerated routines for search of similar vectors implemented in Faiss library (Johnson et al., 2019). The search of nearest neighbours takes substantial time, therefore, acceleration with GPUs helps to significantly reduce the word sense construction time. To further speed up the process, we keep all intermediate results in memory, which results in substantial RAM consumption of up to 200 Gb.

The construction of word senses for all of the 158 languages takes a lot of computational resources and imposes high requirements to the hardware. For calculations, we use in parallel 10–20 nodes of the Zhores cluster (Zacharov et al., 2019) empowered with Nvidia Tesla V100 graphic cards. For each of the languages, we construct inventories based on 50, 100, and 200 neighbours for 100,000 most frequent words. The vocabulary was limited in order to make the computation time feasible. The construction of inventories for one language takes up to 10 hours, with 6.5 hours on average. Building the inventories for all languages took more than 1,000 hours of GPU-accelerated computations.

We release the constructed sense inventories for all the available languages. They contain all the necessary information for using them in the proposed WSD system or in other downstream tasks.

#### 4.2. Word Sense Disambiguation System

The first text pre-processing step is language identification, for which we use the fastText language identification models by Bojanowski et al. (2017). Then the input is tokenised. For languages which use Latin, Cyrillic, Hebrew, or Greek scripts, we employ the Europarl tokeniser. For Chinese, we use the Stanford Word Segmenter (Tseng et al., 2005). For Japanese, we use Mecab (Kudo, 2006). We tokenise Vietnamese with UETsegmenter (Nguyen and Le, 2016). All other languages are processed with the ICU tokeniser, as implemented in the PyICU project.

After the tokenisation, the system analyses all the input words with pre-extracted sense inventories and defines the most appropriate sense for polysemous words.

![Figure 1: The graph of nearest neighbours of the word Ruby can be separated according several senses: programming languages, female names, gems, as well as a cluster of different spellings of the word Ruby.](https://www.statmt.org/europarl)

---

https://www.statmt.org/europarl

https://pypi.org/project/PyICU
Figure 2: Interface of our WSD module with examples for the English language. Given a sentence, it identifies polysemous words and retrieves the most appropriate sense (labelled by the centroid word of a corresponding cluster).

by Ustalov et al. (2018), extending it with a back-end for multiple languages, language detection, and sense browsing capabilities.

5. Evaluation

We first evaluate our converted embedding models on multi-language lexical similarity and relatedness tasks, as a sanity check, to make sure the word sense induction process did not hurt the general performance of the embeddings. Then, we test the sense embeddings on WSD task.

5.1. Lexical Similarity and Relatedness

Experimental Setup
We use the SemR-11 dataset (Barzegar et al., 2018), which contain word pairs with manually assigned similarity scores from 0 (words are not related) to 10 (words are fully interchangeable) for 12 languages: English (en), Arabic (ar), German (de), Spanish (es), Farsi (fa), French (fr), Italian (it), Dutch (nl), Portuguese (pt), Russian (ru), Swedish (sv), Chinese (zh). The task is to assign relatedness scores to these pairs so that the ranking of the pairs by this score is close to the ranking defined by the oracle score. The performance is measured with Pearson correlation of the rankings. Since one word can have several different senses in our setup, we follow Remus and Biemann (2018) and define the relatedness score for a pair of words as the maximum cosine similarity between any of their sense vectors.

We extract the sense inventories from fastText embedding vectors. We set $N = K$ for all our experiments, i.e. the number of vertices in the graph and the maximum number of vertices’ nearest neighbours match. We conduct experiments with $N = K$ set to 50, 100, and 200. For each cluster $V_i$ we create a sense vector $s_i$ by averaging vectors that belong to this cluster. We rely on the methodology of Remus and Biemann (2018) shifting the generated sense vector to the direction of the original word vector: $s_i = \lambda w + (1 - \lambda) \frac{1}{n} \sum_{u \in V_i} \cos(w, u) \cdot u$, where $\lambda \in [0, 1]$, $w$ is the embedding of the original word, $\cos(w, u)$ is the cosine similarity between $w$ and $u$, and $n = |V_i|$. By introducing the linear combination of $w$ and $u \in V_i$ we enforce the similarity of sense vectors to the original word important for this task. In addition to that, we weight $u$ by their similarity to the original word, so that more similar neighbours contribute more to the sense vector. The shifting parameter $\lambda$ is set to 0.5, following Remus and Biemann (2018).

A fastText model is able to generate a vector for each word even if it is not represented in the vocabulary, due to the use of subword information. However, our system cannot assemble sense vectors for out-of-vocabulary words, for such words it returns their original fastText vector. Still, the coverage of the benchmark datasets by our vocabulary is at least 85% and approaches 100% for some languages, so we do not have to resort to this back-off strategy very often.

We use the original fastText vectors as a baseline. In this case, we compute the relatedness scores of the two words as a cosine similarity of their vectors.

Discussion of Results
We compute the relatedness scores for all benchmark datasets using our sense vectors and compare them to cosine similarity scores of original fastText vectors. The results vary for different languages. Figure 3 shows the change in Pearson correlation score when switching from the baseline fastText embeddings to our sense vectors. The new vectors significantly improve the relatedness detection for German, Farsi, Russian, and Chinese, whereas for Italian, Dutch, and Swedish the score slightly falls behind the baseline. For other languages, the performance of sense vectors is on par with regular fastText.

5.2. Word Sense Disambiguation

The purpose of our sense vectors is disambiguation of polysemous words. Therefore, we test the inventories constructed with egvi on the Task 13 of SemEval-2013 — Word Sense Induction (Jurgens and Klapaftis, 2013). The task is to identify the different senses of a target word in context in a fully unsupervised manner.

Experimental Setup The dataset consists of a set of polysemous words: 20 nouns, 20 verbs, and 10 adjectives and specifies 20 to 100 contexts per word, with the total of 4,664 contexts, drawn from the Open American National Corpus. Given a set of contexts of a polysemous word, the participants of the competition had to divide them into clusters by sense of the word. The contexts are manually labelled with WordNet senses of the target words, the gold clusters by sense of the word. The contexts are manually labelled with WordNet senses of the target words, the gold standard clustering is generated from this labelling. The task allows two setups: graded WSI where participants can submit multiple senses per word and provide the probability of each sense in a particular context, and non-graded WSI where a model determines a single sense for a word in context. In our experiments we performed non-graded WSI. We considered the most suitable sense as the one with the highest cosine similarity with embeddings of the context, as described in Section 3.2.

The performance of WSI models is measured with three metrics that require mapping of sense inventories (Jaccard Index, Kendall’s τ, and WNDCG) and two cluster comparison metrics (Fuzzy NMI and Fuzzy B-Cubed).

Discussion of Results We compare our model with the models that participated in the task, the baseline ego-graph clustering model by Pelevina et al. (2016), and AdaGram (Bartunov et al., 2016), a method that learns sense embeddings based on a Bayesian extension of the Skip-gram model. Besides that, we provide the scores of the simple baselines originally used in the task: assigning one sense to all words, assigning the most frequent sense to all words, and considering each context as expressing a different sense. The evaluation of our model was performed using the open source context-eval tool.

Table 2 shows the performance of these models on the SemEval dataset. Due to space constraints, we only report the scores of the best-performing SemEval participants, please refer to Jurgens and Klapaftis (2013) for the full results. The performance of AdaGram and SenseGram models is reported according to Pelevina et al. (2016).

The table shows that the performance of egvi is similar to state-of-the-art word sense disambiguation and word sense induction models. In particular, we can see that it outperforms SenseGram on the majority of metrics. We should note that this comparison is not fully rigorous, because SenseGram induces sense inventories from word2vec as opposed to fastText vectors used in our work.

5.3. Analysis

In order to see how the separation of word contexts that we perform corresponds to actual senses of polysemous words, we visualise ego-graphs produced by our method. Figure 1 shows the nearest neighbours clustering for the word Ruby, which divides the graph into five senses: Ruby-related programming tools, e.g. RubyOnRails (orange cluster), female names, e.g. Josie (magenta cluster), gems, e.g. Sapphire (yellow cluster), programming languages in general, e.g. Haskell (red cluster). Besides, this is typical for fastText embeddings featuring sub-string similarity, one can observe a cluster of different spelling of the word Ruby in green.

Analogously, the word python (see Figure 3) is divided into the senses of animals, e.g. crocodile (yellow cluster), programming languages, e.g. perl5 (magenta cluster), and conference, e.g. pycon (red cluster).

In addition, we show a qualitative analysis of senses of mouse and apple. Table 3 shows nearest neighbours of the original words separated into clusters (labels for clusters were assigned manually). These inventories demonstrate clear separation of different senses, although it can be too fine-grained. For example, the first and the second cluster for mouse both refer to computer mouse, but the first one addresses the different types of computer mice, and the second one is used in the context of mouse actions. Similarly, we see that iphone and macbook are sep-
Table 2: WSD performance on the SemEval-2013 Task 13 dataset for the English language.

<table>
<thead>
<tr>
<th>Model</th>
<th>Supervised Evaluation</th>
<th>Clustering Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jacc. Ind.</td>
<td>Kendall’s $\tau$</td>
</tr>
<tr>
<td>Baselines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One sense for all</td>
<td>0.192</td>
<td>0.609</td>
</tr>
<tr>
<td>One sense per instance</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Most Frequent Sense</td>
<td>0.455</td>
<td>0.465</td>
</tr>
<tr>
<td>SemEval-2013 participants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AI-KU (base)</td>
<td>0.197</td>
<td>0.620</td>
</tr>
<tr>
<td>AI-KU (remove5-add1000)</td>
<td>0.244</td>
<td>0.642</td>
</tr>
<tr>
<td>Unimelb (50k)</td>
<td>0.213</td>
<td>0.620</td>
</tr>
<tr>
<td>Sense embeddings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaGram, $\alpha = 0.05$, 100 dim. vectors</td>
<td>0.274</td>
<td>0.644</td>
</tr>
<tr>
<td>SenseGram (word2vec)</td>
<td>0.197</td>
<td>0.615</td>
</tr>
<tr>
<td>egvi (fastText, K=200)</td>
<td>0.229</td>
<td>0.625</td>
</tr>
</tbody>
</table>

Figure 5: Distribution of the number of senses per word in the generated inventories for all 158 languages for the number of neighbours set to: $N \in \{50, 100, 200\}$, $K \in \{50, 100, 200\}$ with $N = K$.

We present egvi, a new algorithm for word sense induction based on graph clustering that is fully unsupervised and relies on graph operations between word vectors. We

6. Conclusions and Future Work

We present egvi, a new algorithm for word sense induction based on graph clustering that is fully unsupervised and relies on graph operations between word vectors. We
apply this algorithm to a large collection of pre-trained fastText word embeddings, releasing sense inventories for 158 languages. These inventories contain all the necessary information for constructing sense vectors and using them in downstream tasks. The sense vectors for polysemous words can be directly retrofitted with the pre-trained word embeddings and do not need any external resources. As one application of these multilingual sense inventories, we present a multilingual word sense disambiguation system that performs unsupervised and knowledge-free WSD for 158 languages without the use of any dictionary or sense-labelled corpus.

The evaluation of quality of the produced sense inventories is performed on multilingual word similarity benchmarks, showing that our sense vectors improve the scores compared to non-disambiguated word embeddings. Therefore, our system in its present state can improve WSD and downstream tasks for languages where knowledge bases, taxonomies, and annotated corpora are not available and supervised WSD models cannot be trained.

A promising direction for future work is combining distributional information from the induced sense inventories with lexical knowledge bases to improve WSD performance. Besides, we encourage the use of the produced word sense inventories in other downstream tasks.

### Acknowledgements

We acknowledge the support of the Deutsche Forschungsgemeinschaft (DFG) foundation under the “JOIN-T 2” and “ACQuA” projects. Ekaterina Artemova was supported by the framework of the HSE University Basic Research Program and Russian Academic Excellence Project “5-100”.

---


---

<table>
<thead>
<tr>
<th>Label</th>
<th>Nearest neighbours</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer mouse types</td>
<td>touch-pad, logitech, scrollwheel, <em>mouses</em>, mouse.\textit{It}, mouse.\textit{Th}, joystick, trackpads, nano-receiver, 800DPI, nony, track-pad, M325, keyboard-controlled, Intellipoint, MouseI, intellimouse, Swiftpoint, Evoluent, 800dpi, moused, game-pad, steelseries, ErgoMotion, Intellimouse, \textit{&lt;}\textit{...}&gt;</td>
</tr>
<tr>
<td>computer mouse actions</td>
<td>Ctrl-Alt, right-mouse, cursor, left-clicks, spacebar, runUse, mouseclick, \textbf{click}, mousepointer, keystroke, cusror, mousewheel, MouseMove, mousebutton, leftclick, click-dragging, mouse-button, cursor., arrow-key, double-clicks, mouse-down, ungrah, mouseX, arrow-keys, right-button, \textit{&lt;}\textit{...}&gt;</td>
</tr>
<tr>
<td>rodent</td>
<td>Rodent, rodent, mousehole, rats, \textbf{mice}, mice., hamster, SOD1G93A, meeses, mice.\textit{Th}, PDAPP, hedgehog, Maukie, r\textit{Tg}4510, mousey, meeces, rodents, cat, White-footed, rat, Mice, \textit{&lt;}\textit{...}&gt;</td>
</tr>
<tr>
<td>keyboard</td>
<td>keybord, keyboard, keybord, keyboard\textit{Th}, the, keyboard, keyboar, \textbf{Keyboard}, keboard, keyboard\textit{ll}, keyb, keyboard.\textit{Th}, s, keyboard, keyboard</td>
</tr>
<tr>
<td>medical</td>
<td>\textit{SENCAR}, \textit{mTERT}, \textbf{mouse-specific}</td>
</tr>
<tr>
<td>Latin</td>
<td>\textit{Apodemus, Micormys}</td>
</tr>
<tr>
<td>Latin</td>
<td>\textbf{Akodon}</td>
</tr>
<tr>
<td>apple</td>
<td>apples, apple-producing, Honeycrisp, apple-y, Macouns, apple-growing, pear, apple-pear, Gravensteins, apple-like, Apples, Honeycrisps, apple-related, Borkh, Braeburns, Starkrimson, Apples., SweeTango, Elstar</td>
</tr>
<tr>
<td>macbook</td>
<td>macbook, macbookpro, macbookair, iMac, iBooks, \textbf{tuaw}, ox, macintosh, imacs, apple.com, applestore, Tagsapp, stejobs, applecare</td>
</tr>
<tr>
<td>fruit, typos pinklady</td>
<td>blackerry, blackberry, blueberry, aple, cidar, \textbf{apple.\textit{The}}, apple.I, aple, appple, calvados, pie.\textit{It},</td>
</tr>
<tr>
<td>tokenisation issues, typos</td>
<td>Apple.This, AMApple, it.Apple, too.Apple, AppleApple, \textbf{up.Apple}, AppleA, Aplle, Apple.\textit{Apple}</td>
</tr>
<tr>
<td>Apple criticism</td>
<td>anti-apple, Aple, Crapple, isheep, iDiots, crapple, Appple, iCrap, \textbf{non-apple}</td>
</tr>
<tr>
<td>Bible</td>
<td>\textbf{Adam}</td>
</tr>
<tr>
<td>cooking</td>
<td>\textbf{caramel-dipped}</td>
</tr>
<tr>
<td>iphone</td>
<td>mobileme, idevice, carplay, iphones, icloud, iwatch, ios5, ipod, \textbf{iphone}, android, ifans, iphone.I, iphone4, iphone5, idevices, ipad, ios, ipod., iphone5, iphone., ios7</td>
</tr>
<tr>
<td>fruit</td>
<td>apples, apple-producing, Honeycrisp, apple-y, Macouns, apple-growing, pear, apple-pear, Gravensteins, apple-like, Apples, Honeycrisps, apple-related, Borkh, Braeburns, Starkrimson, Apples., SweeTango, Elstar</td>
</tr>
<tr>
<td>macbook</td>
<td>macbook, macbookpro, macbookair, iMac, iBooks, \textbf{tuaw}, ox, macintosh, imacs, apple.com, applestore, Tagsapp, stejobs, applecare</td>
</tr>
<tr>
<td>apple criticism</td>
<td>\textit{anti-apple, Aple, Crapple, isheep, iDiots, crapple, Appple, iCrap, non-apple}</td>
</tr>
<tr>
<td>Bible</td>
<td>\textbf{Adam}</td>
</tr>
<tr>
<td>cooking</td>
<td>\textbf{caramel-dipped}</td>
</tr>
<tr>
<td>iphone</td>
<td>mobileme, idevice, carplay, iphones, icloud, iwatch, ios5, ipod, \textbf{iphone}, android, ifans, iphone.I, iphone4, iphone5, idevices, ipad, ios, ipod., iphone5, iphone., ios7</td>
</tr>
<tr>
<td>fruit</td>
<td>apples, apple-producing, Honeycrisp, apple-y, Macouns, apple-growing, pear, apple-pear, Gravensteins, apple-like, Apples, Honeycrisps, apple-related, Borkh, Braeburns, Starkrimson, Apples., SweeTango, Elstar</td>
</tr>
<tr>
<td>macbook</td>
<td>macbook, macbookpro, macbookair, iMac, iBooks, \textbf{tuaw}, ox, macintosh, imacs, apple.com, applestore, Tagsapp, stejobs, applecare</td>
</tr>
<tr>
<td>apple criticism</td>
<td>anti-apple, Aple, Crapple, isheep, iDiots, crapple, Appple, iCrap, \textbf{non-apple}</td>
</tr>
<tr>
<td></td>
<td>\textit{Apple.This, AMApple, it.Apple, too.Apple, AppleApple, up.Apple, AppleA, Aplle, Apple.\textit{Apple}}</td>
</tr>
<tr>
<td>Apple</td>
<td>Apple.\textit{The}, apple.I, aple, appple, calvados, pie.\textit{It},</td>
</tr>
<tr>
<td>Bible</td>
<td>\textbf{Adam}</td>
</tr>
<tr>
<td>cooking</td>
<td>\textbf{caramel-dipped}</td>
</tr>
<tr>
<td>iphone</td>
<td>mobileme, idevice, carplay, iphones, icloud, iwatch, ios5, ipod, \textbf{iphone}, android, ifans, iphone.I, iphone4, iphone5, idevices, ipad, ios, ipod., iphone5, iphone., ios7</td>
</tr>
<tr>
<td>fruit</td>
<td>apples, apple-producing, Honeycrisp, apple-y, Macouns, apple-growing, pear, apple-pear, Gravensteins, apple-like, Apples, Honeycrisps, apple-related, Borkh, Braeburns, Starkrimson, Apples., SweeTango, Elstar</td>
</tr>
<tr>
<td>macbook</td>
<td>macbook, macbookpro, macbookair, iMac, iBooks, \textbf{tuaw}, ox, macintosh, imacs, apple.com, applestore, Tagsapp, stejobs, applecare</td>
</tr>
<tr>
<td>apple criticism</td>
<td>anti-apple, Aple, Crapple, isheep, iDiots, crapple, Appple, iCrap, \textbf{non-apple}</td>
</tr>
</tbody>
</table>

Table 4: Clustering of senses for words \textit{mouse} and \textit{apple} produced by our method. Cluster labels in this table were assigned manually for illustrative purposes. For on-the-fly disambiguation we use centroid words in clusters as sense labels (shown here \textbf{in bold}).

---

Links to the produced datasets, online demo, and source codes are available at: [http://uhh-lt.github.io/158](http://uhh-lt.github.io/158).
8. Bibliographical References

Amplayo, R. K., Hwang, S.-w., and Song, M. (2019). Au-

Amrami, A. and Goldberg, Y . (2018). Word Sense Induc-


Barzegar, S., Davis, B., Zarrouk, M., Handschuh, S., and

Bartunov, S., Kondrashkin, D., Osokin, A., and Vetrov,

Hope, D. and Keller, B. (2013a). MaxMax: A Graph-

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T.

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and

Hope, D. and Keller, B. (2013a). MaxMax: A Graph-

Iacobacci, I. and Navigli, R. (2019). LSTMEmbed: Learn-

Jain, S., Bodapati, S. B., Nallapati, R., and Anandkumar, A.


Kutuzov, A. (2018). Russian word sense induction by clus-

Li, J. and Jurafsky, D. (2015). Do Multi-Sense Embed-


