fastSense: An Efficient Word Sense Disambiguation Classifier

Tolga Uslu, Alexander Mehler, Daniel Baumartz, Wahed Hemati
TTLab, Goethe University Frankfurt
Frankfurt, Germany
{uslu, mehler, baumartz, hemati}@em.uni-frankfurt.de

Abstract
The task of Word Sense Disambiguation (WSD) is to determine the meaning of an ambiguous word in a given context. In spite of its importance for most NLP pipelines, WSD can still be seen to be unsolved. The reason is that we currently lack tools for WSD that handle big data – “big” in terms of the number of ambiguous words and in terms of the overall number of senses to be distinguished. This desideratum is exactly the objective of fastSense, an efficient neural network-based tool for word sense disambiguation introduced in this paper. We train and test fastSense by means of the disambiguation pages of the German Wikipedia. In addition, we evaluate fastSense in the context of Senseval and SemEval. By reference to Senseval and SemEval we additionally perform a parameter study. We show that fastSense can process huge amounts of data quickly and also surpasses state-of-the-art tools in terms of F-measure.

Keywords: WSD, Big Data, Wikipedia

1. Introduction
One of the core tasks in natural language processing is Word Sense Disambiguation. Without disambiguation, we just consider a word as a combination of characters and not the meaning behind it. Without properly disambiguating the lexical constituents of a text, it is almost impossible to process its content automatically. Our goal is to solve this problem and to make it applicable to large amounts of data. To this end, we present a neural network-based classifier for WSD called fastSense. We take sequences of words as input and compute a sense label per ambiguous word in that sequence as output. This approach was motivated by the classifier called fastText (Joulin et al., 2016). As the name suggests, fastText is designed to perform text classifications as quickly as possible. However, fastText is not suitable for disambiguating words. In addition, the neural network used by fastText does not support training of multi-labels. Therefore, we implemented our own word embedding-based neural network by analogy to the architecture of fastText. This allows us to apply fastSense to WSD efficiently even on big data. In order to test the time complexity of our approach, we created a disambiguation corpus from the German Wikipedia with over 50,000,000 training and test sets. We use the disambiguation pages and the link structure of Wikipedia to match words with their corresponding Wikipedia senses. In this paper, we deal with the German Wikipedia. In terms of size or space complexity, its sense model is far beyond what is normally studied, for example, in the framework of Senseval or SemEval. However, in order to show that our approach is language independent, we additionally perform multiple tests related to Senseval and SemEval. These tests show that our model keeps up with state-of-the-art tools by reaching 73.47% at Senseval-2, 73.48% at Senseval-3 and up to 87.57% on SemEval 2007 tasks.

The paper is structured as follows: In Section 2, we contrast fastSense with related approaches to WSD. In Section 3, we introduce the architecture of fastSense. In Section 4, we explain the experiments carried out to evaluate fastSense and show the results achieved by it. In Section 5.3, we discuss our findings and in Section 5, we give a summary of the paper.

2. Related Work
Our approach is motivated by fastText (Joulin et al., 2016). This relates to the very efficient and successful way by which fastText allows for classifying data. The main purpose of fastText is text classification. Its architecture is similar to word2vec (Mikolov et al., 2013); both approaches are based on a bag-of-words model. Further, both of them use a single hidden layer. The difference between word2vec and fastText is that the latter requires to define a label for any input text, while word2vec uses context windows of lexical units to predict single words or vice versa. We transpose fastText to word sense disambiguation in order to efficiently determine the meaning of ambiguous words even in cases in which we face big data. By this we mean scenarios in which hundreds of thousands of different words are ambiguous. fastSense is characterized by its simplicity, speed and quality. This distinguishes it from similar tools. For instance, (Mihalcea and Csomai, 2007; Ferragina and Scialli, 2010; Ratinov et al., 2011b; Ratinov et al., 2011a; Agerri et al., 2013; Moro et al., 2014) present approaches to Entity Linking. More specifically, they link tokens in texts to knowledge databases such as DBpedia, Wikipedia or WordNet to identify instances of entities. These approaches are similar to ours, with the difference that we focus on ambiguous words, while the latter approaches also link words that have only one meaning. The disadvantage of these approaches is their speed. For large amounts of data, they may take weeks to produce an output (see Table 2 for an estimation of this time effort). (Mihalcea, 2007) uses a
technique similar to the one presented here to build a sense-tagged Wikipedia corpus using the link structure of Wikipedia to match senses. However, this corpus has not been used to disambiguate ambiguous words according to Wikipedia’s disambiguation pages, but to compare them with the data of Senseval 2. (Mihalcea et al., 2004) use a PageRank algorithm operating on semantic networks to perform WSD. The underlying network is spanned by means of semantic relations of synsets, entailment and other WordNet relations. The PageRank algorithm assigns scores to words and chooses the disambiguating synset of highest score. (Yuan et al., 2016) present two WSD algorithms, achieving the best results by means of a semi-supervised algorithm combining labeled sentences with unlabeled ones and propagating labels based on sentence similarity. (Tripodi and Pelillo, 2016) describe an approach to WSD based on evolutionary game theory, in which words tend to adapt senses of their neighborhood so that WSD is reformulated as a kind of constraint satisfaction. (Zhong and Ng, 2010) present a framework for English all-words WSD. It disambiguates each content word of a given sentence using a linear kernel-based SVM (Joachims, 2002). (Jacobacci et al., 2016) show that the use of word embeddings achieves an improvement in WSD compared to standard features. (Chaplot et al., 2015) propose a graph based unsupervised WSD system which requires WordNet, a dependency parser and a POS-Tagger. They model WSD as a maximum-a-posteriori inference query operating on a Markov random field. (Raganato et al., 2017a) define WSD in terms of a sequence learning problem. This is done by means of a bidirectional LSTM-based neural network (Hochreiter and Schmidhuber, 1997). (Melamud et al., 2016) present context2vec which is also based on bidirectional LSTMs for learning disambiguating word contexts.

Unlike these approaches, we present a method that can handle big data: in terms of the number of senses to be distinguished and in terms of the number of units to be disambiguated. On the one hand, knowledge driven approaches using, for example, WordNet and related resources are limited in terms of the number of senses distinguished by them. GermaNet, for example, distinguishes 33,630 senses of 13,445 ambiguous words – that is much less than considered by us. On the other hand, approaches that rely on algorithms like PageRank or classifiers like SVMs or LSTMs are limited in terms of their time efficiency: it is a computational challenge to maintain, for example, SVMs for each of the 825,179 senses of the 221,965 ambiguous words of the German Wikipedia which, however, are easily covered by our approach. Thus, we are in need of a flexible, easy-to-compute, but efficient method for WSD as presented in the next section.

3. Model architecture

During training, fastSense requires text as input and the corresponding senses as output (see Figure 1). Its single hidden layer is an embedding layer in which word indexes from the input layer are converted into word vectors. More specifically, the number of hidden nodes corresponds to the dimension of the pre-trained word embedding vectors so that the weights of edges between input and hidden nodes correspond to the respective coordinates of the latter vectors. We computed the word embeddings by means of word2vec (Mikolov et al., 2013) using Wikipedia as the underlying corpus. The embeddings are then merged in the hidden layer according to the global averaging pooling principle (Lin et al., 2013).

![Figure 1: Model architecture of fastSense.](image-url)

To support multi-label training we used the sigmoid function as an activation function of the output layer. For the sake of optimizing, Adamax, a special variant of Adam (Kingma and Ba, 2014), is a very efficient choice in practice. It uses the infinity norm, which makes it possible to stabilize the training over longer periods of time and, thus, to achieve faster and better results.

To prevent overfitting, we used Dropout (Srivastava et al., 2014) as regularization method. Dropout removes nodes during each training session, ignores them and does not train with them. After the training process, the nodes are reinserted with their original weights.

To apply this model to WSD, we additionally developed a method for post-processing the output of the neural network. Usually, the sense of highest probability is selected as output. However, since the output layer contains all senses of all ambiguous words, it is unlikely that the target sense of a word to be disambiguated equals the top ranked sense. Thus, we do not necessarily select the label of highest probability, but go through the list of rank-ordered candidates until the first occurrence of a tagged by a corresponding sense number is reached. This sense unit is then produced as output of disambiguating x. As an example, consider processing the ambiguous word bank as depicted in Figure 2 when observing an occurrence of this word
in a sentence about a financial topic, a classifier like fastText will likely suggest topic labels such as finance, money or financial institute because of the fact that the input sentence is about such a topic. However, what we are looking for is the sense of the word and not the most strongly associated topic label. Thus, we descend the sorted output of fastSense given the input sentence until we reach a candidate sense prefixed by bank (i.e., bank_2) that is taken to predict the sense of the word in this sentence. In this way, fastSense can be used as a tool for WSD. Conversely speaking, we reconstructed WSD as a kind of topic labeling that is performed by fastSense by analogy to fastText.

![Figure 2: fastSense in work: bank_2 is selected as the first occurrence prefixed by bank.](image)

4. Experiment

We perform two disambiguation experiments. The first one uses the German Wikipedia to demonstrate the efficiency of fastSense. The second one is based on Senseval and SemEval. It aims at comparing fastSense with state-of-the-art tools. Table 1 lists the parameters used for these evaluations.

<table>
<thead>
<tr>
<th>Short</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS</td>
<td>Part of speech is considered.</td>
</tr>
<tr>
<td>Lemma</td>
<td>Lemma information is considered.</td>
</tr>
<tr>
<td>Token</td>
<td>Token information is considered.</td>
</tr>
<tr>
<td>WP</td>
<td>POS information is added.</td>
</tr>
<tr>
<td>x-Nb</td>
<td>x neighbors (left and right) are considered as context.</td>
</tr>
<tr>
<td>MinContext(k)</td>
<td>Any input text must contain at least k tokens to be used in training or testing.</td>
</tr>
</tbody>
</table>

Table 1: Parameters used for evaluating fastSense.

4.1. Wikipedia-based Disambiguation

In order to show that our approach allows for capturing large amounts of data, we created a corpus using the disambiguation pages of the whole German Wikipedia. For preprocessing this data we used the TextImager pipeline. Every word listed on a disambiguation page in Wikipedia corresponds to a different meaning of the corresponding lemma (page title). In total, we processed 221,965 disambiguation pages related to 825,179 senses. On average, this gives 3.72 senses per word. The disambiguation page Bank title). In total, we processed 221,965 disambiguation pages related to 825,179 senses. On average, this gives 3.72 senses per word. The disambiguation page Bank title). In total, we processed 221,965 disambiguation pages related to 825,179 senses. On average, this gives 3.72 senses per word. The disambiguation page Bank

![Table 2: Runtime-related evaluation regarding similar tools using 1, 500, 1000 and all test instances.](image)

![Table 3: Wikipedia-based evaluation of fastSense in comparison to fastText and the most frequent sense (MFS) baseline.](image)
4.2. Senseval and SemEval related Disambiguation

SemCor [Mihalcea, 2016] provides texts with semantically annotated WordNet senses, which are automatically mapped to WordNet. We trained on SemCor 3.0 for performing Senseval and SemEval related tests. Because of the small amount of data provided by this corpus (234,136 disambiguated words), we were able to perform a parameter study to search for the best performing parameter settings. Candidates for feature selection are POS, token, lemma and combinations thereof (see Table 1). Next, we tested different word context sizes (Context), word-n-grams (NGrams), learning rates (LR), dimensions of the hidden layer (Dim) and the number of epochs (Epoch).

After each optimization step, we used the best performing parameter settings. Candidates for feature selection are POS, token, lemma and combinations thereof (see Table 1). Next, we tested different word context sizes (Context), word-n-grams (NGrams), learning rates (LR), dimensions of the hidden layer (Dim) and the number of epochs (Epoch).

After completion we applied the optimal settings on SemCor 3.0. We also tested fastSense on Senseval 2, 3 and SemEval 7, 13 and 15 tasks using SemCor for training.

<table>
<thead>
<tr>
<th>Model</th>
<th>SE2</th>
<th>SE3</th>
<th>SE7</th>
<th>SE7'</th>
<th>SE13</th>
<th>SE15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iacobacci, 2016</td>
<td>0.634</td>
<td>0.653</td>
<td>0.894</td>
<td>0.578</td>
<td>0.673</td>
<td>0.715</td>
</tr>
<tr>
<td>Tripodi, 2016</td>
<td>0.660</td>
<td>0.647</td>
<td>0.828</td>
<td>0.585</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yuan, 2016</td>
<td>0.736</td>
<td>0.692</td>
<td>0.828</td>
<td>0.642</td>
<td>0.670</td>
<td></td>
</tr>
<tr>
<td>Chaplot, 2015</td>
<td>0.605</td>
<td>0.586</td>
<td></td>
<td>0.506</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhong, 2010</td>
<td>0.623</td>
<td>0.650</td>
<td>0.879</td>
<td>0.565</td>
<td>0.653</td>
<td>0.695</td>
</tr>
<tr>
<td>Raganato, 2017</td>
<td>0.720</td>
<td>0.702</td>
<td></td>
<td>0.648</td>
<td>0.669</td>
<td>0.724</td>
</tr>
<tr>
<td>Melamud, 2016</td>
<td>0.718</td>
<td>0.691</td>
<td></td>
<td></td>
<td>0.613</td>
<td>0.656</td>
</tr>
<tr>
<td>fastSense</td>
<td>0.735</td>
<td>0.735</td>
<td>0.876</td>
<td>0.624</td>
<td>0.662</td>
<td>0.732</td>
</tr>
</tbody>
</table>

Table 6: Comparison of state-of-the-art WSD tools on the Senseval 2, 3 and SemEval 7, 13 and 15 tasks using SemCor for training.

4.3. Discussion

We successfully used Wikipedia as a disambiguation corpus and show that fastSense can handle such a large amount of data (see Table 3). fastSense not only stands out for its speed, but also for the quality of its classification. We outperform the baseline considerably and show that similar approaches cannot keep up with fastSense in terms of runtime. Thus fastSense is a step towards performing WSD in relation to large amounts of data.

Since the SemCor data is many times smaller than the data derived from Wikipedia, we were able to carry out a parameter study in the Senseval- and SemEval-related experiments. Most interesting is our finding concerning the size of the context window. Using one neighbor of a word as context (1-Nb) and word-3-grams perform best. We also found that token in combination with POS-related information (see the parameter list in Table 1 and Table 4) perform best.

Note that fastSense is comparably fast: it takes only 20 minutes for disambiguating 4,729,480 instances on a single thread – that is, ca. 3,941 senses per second. Further, as mentioned in Section 3, fastText is less suited for WSD; accordingly, it performs worse compared to fastSense (see Table 3).

In this sense, though being based on a related architecture, fastSense better fits the needs of WSD.

5. Conclusion

We presented a novel approach to word sense disambiguation called fastSense. We tested this model in the framework of Senseval and SemEval tasks as well as in terms of a big data-experiment based on the German Wikipedia. We achieve an F-score of up to 81.00% using the Wikipedia-based data. Further, we achieved 73.47% and 73.48% on Senseval 2 and 3, 87.57% and 87.57% on Senseval 2 and 3, 87.57% and 87.57% on Senseval 2 and 3. Since the SemCor data is many times smaller than the data derived from Wikipedia, we were able to carry out a parameter study in the Senseval- and SemEval-related experiments. Most interesting is our finding concerning the size of the context window. Using one neighbor of a word as context (1-Nb) and word-3-grams perform best. We also found that token in combination with POS-related information (see the parameter list in Table 1 and Table 4) perform best.

Note that fastSense is comparably fast: it takes only 20 minutes for disambiguating 4,729,480 instances on a single thread – that is, ca. 3,941 senses per second. Further, as mentioned in Section 3, fastText is less suited for WSD; accordingly, it performs worse compared to fastSense (see Table 3). In this sense, though being based on a related architecture, fastSense better fits the needs of WSD.

<table>
<thead>
<tr>
<th>Input</th>
<th>SE2</th>
<th>SE3</th>
<th>SE7</th>
<th>SE7'</th>
<th>SE13</th>
<th>SE15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token WP</td>
<td>0.735</td>
<td>0.735</td>
<td>0.662</td>
<td>0.732</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Token</td>
<td></td>
<td></td>
<td>0.876</td>
<td>0.624</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: F1-scores of the Senseval/SemEval-related tasks.

We successfully used Wikipedia as a disambiguation corpus and show that fastSense can handle such a large amount of data (see Table 3). fastSense not only stands out for its speed, but also for the quality of its classification. We outperform the baseline considerably and show that similar approaches cannot keep up with fastSense in terms of runtime. Thus fastSense is a step towards performing WSD in relation to large amounts of data.

Since the SemCor data is many times smaller than the data derived from Wikipedia, we were able to carry out a parameter study in the Senseval- and SemEval-related experiments. Most interesting is our finding concerning the size of the context window. Using one neighbor of a word as context (1-Nb) and word-3-grams perform best. We also found that token in combination with POS-related information (see the parameter list in Table 1 and Table 4) perform best.

Note that fastSense is comparably fast: it takes only 20 minutes for disambiguating 4,729,480 instances on a single thread – that is, ca. 3,941 senses per second. Further, as mentioned in Section 3, fastText is less suited for WSD; accordingly, it performs worse compared to fastSense (see Table 3). In this sense, though being based on a related architecture, fastSense better fits the needs of WSD.

5. Conclusion

We presented a novel approach to word sense disambiguation called fastSense. We tested this model in the framework of Senseval and SemEval tasks as well as in terms of a big data-experiment based on the German Wikipedia. We achieve an F-score of up to 81.00% using the Wikipedia-based data. Further, we achieved 73.47% and 73.48% on Senseval 2 and 3, 87.57% and 87.57% on Senseval 2 and 3. Since the SemCor data is many times smaller than the data derived from Wikipedia, we were able to carry out a parameter study in the Senseval- and SemEval-related experiments. Most interesting is our finding concerning the size of the context window. Using one neighbor of a word as context (1-Nb) and word-3-grams perform best. We also found that token in combination with POS-related information (see the parameter list in Table 1 and Table 4) perform best.

Note that fastSense is comparably fast: it takes only 20 minutes for disambiguating 4,729,480 instances on a single thread – that is, ca. 3,941 senses per second. Further, as mentioned in Section 3, fastText is less suited for WSD; accordingly, it performs worse compared to fastSense (see Table 3). In this sense, though being based on a related architecture, fastSense better fits the needs of WSD.
6. Bibliographical References


