CS224N Final Project Report

Syntactico Semantic Word Representations in Multiple Languages

Jianfeng Hu, Guan (Bell) Wang, Zhemin Li

jfhu@stanford.edu guanw@stanford.edu zhemin@stanford.edu

1 Introduction

Our project is an extension of the project "*Syntactico Semantic Word Representations in Multiple Languages*"[1]. The previous project aims to improve the semantical representation of English vocabulary via incorporating the local context with global context and supplying homonymy and polysemy for multiple embeddings per word. It also introduces a new neural network architecture that learns the word embeddings from both local and global context and multiple embeddings of each word with homonymy and polysemy. Based on this neural network learning model, our project learns the embeddings for German words and improves the semantical representation of German vocabulary. After the learning procedure, we produce a new dataset of German word embeddings and visualize them in t-SNE figures.

2 Related Work

An n-gram[4] is a contiguous sequence of n items from a given sequence of text or speech. N-grams model can be used to predict the next word though combining the estimator of the probability of the next word given the current word and the word before the current word[5]. However, if a large corpus is used, the data may be too sparse for the n-gram language model[1].

A distributed representation[6] of a symbol is a vector of features which characterize the meaning of the symbol. Distributed representation of different words can model the similarity between those words. A neural language model[7] is used to learn distributed representations based on neural networks. It is powerful to accurately predict the next word given previously seen words. The neural language model can solve the data spareness of n-gram language model[1]. With a neural network language model, configuration of these features depends on the learning algorithms used[7]. In the previous project, it uses relative local context and larger global context for training input, and Ranking-loss is used as the objective function for training word embeddings[1].

Vector-space models are used to represent the syntactical and semantical meaning of a word with vectors. However, representing a word's meaning with a single prototype vector is problematic because of lexical ambiguity. Reisinger and Mooney introduced the concept of multiple word prototypes and presented a method that uses clustering to produce multiple "sense-specific"

vectors for each word[3]. Those vectors are context-dependent vector representations of word meaning that naturally accommodates homonymy and polysemy[3]. In the previous project, multiple-prototype approach is adopted to the neural language model through representing each context window by single-prototype embeddings and then cluster such embeddings to perform word sense discrimination[1].

3 Method, Modification and Implementation

Our project collects training corpus from the wikipedia website and dictionary from a Translation Website[8]. The German word embeddings are initialized based on the related English word embeddings. With the data preprocessing and initialization, neural language model from the previous project is used to learn the word embeddings for German vocabulary. Finally, KNN method is used to find the K-nearest neighbors for each word in the vocabulary for evaluation.

3.1 Data Collection - Wikipedia Dataset

Our training corpus comes from Wikipedia. Wikipedia is an enormous source for texts in multiple languages. It is free and has been growing and kept updated all the time. Our learning package exploits Wikipedia's data and is easy to be reused later on other languages.

When building the package, we used German as the target language. Some special characters in German are UTF8 encoded in the Wikipedia. By correctly processing the German corpus, we could be confident in processing other language which are left-to-right and whose words are space-delimited.

Wikipedia provides its own dump so that people can make use of it and don't have to crawl their own versions. For German Wikipedia, the dump file is a 20G-sized xml file which contains all the wiki pages and their metadata. For our training purpose, we need to remove extra metadata as well as markup tags. Then, the remaining sentences should be tokenized and feed to our next step. We used a heavily modified Python library[2] to parse the xml structure and remove the markup tags. Then, we tokenized the articles and serve them as our training corpus.

3.2 Building Dictionary

Before running the training algorithm, we already have the word embeddings for English words from Huang et al[1]. In order to provide a better result, we decided to use the existing embeddings to initialize our training algorithm. For each German word, we will find its English synonyms, and assign an initial embedding to the German word by averaging all English embeddings.

In order to support the initialization, we need to find synonyms between English and German words. Resources are very limited on the Internet, so we crawled our own dictionary. For each of the 100,000 English tokens whose corresponding embeddings are available, we send requests to query its German translations. Queries are sent in parallel to increase the performance. Because many of the English tokens are not actual words, we were able to find German translations for around 17,000 English words. Some of the words have multiple translations in German. With the result, we created our own English-German dictionary.

3.3 Embedding Initialization

Since we already have the trained English word embeddings, we use them to initialize the German word embeddings. From some English-German dictionary, we can get the corresponding German words for an English word. Hence we assign the English word embedding to the corresponding German words as their initial embeddings. Here we assume that, as the corresponding German words are translated from the same English word, they should have similar meanings. Based on this assumption, initializing these German words with the same embeddings can improve the performance of the training process.

3.4 Learning Method

In order to learn the German word embeddings, we use the neural language model presented in the previous project *Syntactico Semantic Word Representations in Multiple Languages*. The cost function[1] (also called *rank loss* for pair (s,d)) is:

$$C_{s,d} = \sum_{w \in V} \max(0, 1 - g(s,d) + g(s^w, d))$$

where s is a word sequence, d is a document in which the sequence occurs, s^w is s with the last word replaced by word w, and $g(\cdot,\cdot)$ is the scoring function that represents the neural networks used.

A neural network with one hidden layer is used to compute the score of the local context, and a two-layer neural network is used to compute the global context score[1]. The final score is sum of the score of local context and that of the global context[1]. The training objective is to minimize training cost function. In the learning procedure, a word from the vocabulary is randomly picked for each sequence-document pair, and derivatives are taken in the *rank loss* with respect to weights of the neural networks and the German word embedding matrix[1]. Weights are updated through backpropagation, and word embeddings are learned and updated during the training[1].

3.5 Training Step

After initialization, we train the word embeddings by running the package downloaded from the previous project. We run the package on the CORN machines. Moreover, in order to deal with unexpected exceptions on the CORN machines, we changed the package to make it be able to resume from intermediate saves.

3.6 Evaluation Step

As the system outputs the embeddings for each word in the vocabulary, we evaluate the performance by analyzing whether a word and its nearest neighbor words have related meanings. Therefore, we adopt KNN to find the K-nearest neighbors for each word in the vocabulary.

After obtaining the K-nearest neighbors for the words in the vocabulary, we evaluate the result by counting the number of synonyms appearing in the K-nearest neighbors set. The synonyms set can be collected from WordNet or crawling from online synonym dictionaries. We use the latter one. We also visualize the results by performing t-SNE on the output embeddings.

4 Training and Evaluation

4.1 Overview of Training

We tested our system with a corpus which contains 1 million German Wikipedia pages. We extracted the top 5000 frequent words from the corpus to form the vocabulary. We trained the top 1000 words first. Once we got the embeddings of these 1000 words, we combined them with the initial embeddings of the next 1000 words. Then we continued training these 2000 words. We continued this process until all top 5000 frequent words were in our vocabulary. As our space on the CORN machine was limited, we were not able to put the entire WIkipedia corpus on the corn machine. Hence we divided the entire corpus into several small chunks and after each process, we replaced the current training corpus with a new one.

4.2 T-SNE Visualization

We adopted the t-SNE technique to reduce the word embeddings' dimension from 50 to 2 and visualize them in the following figures. Figure 1 shows all the 5000 word embeddings. We find that the graph have one large cluster with several small clusters.

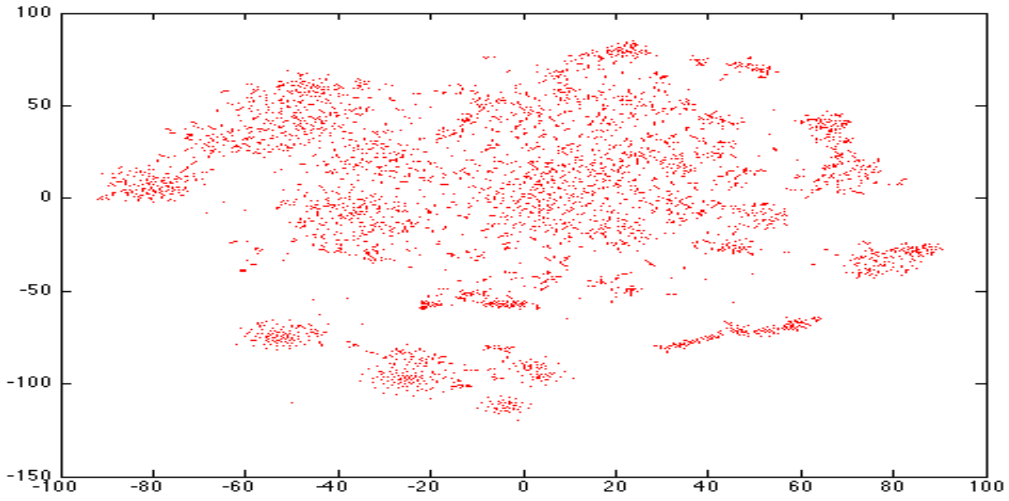

Figure 1

Figure 2, 3 are some small parts of the whole t-SNE graph. From Figure 2, we find that the system is able to correctly cluster countries' names together. Figure 3 shows that the system can correctly cluster the name of months together.

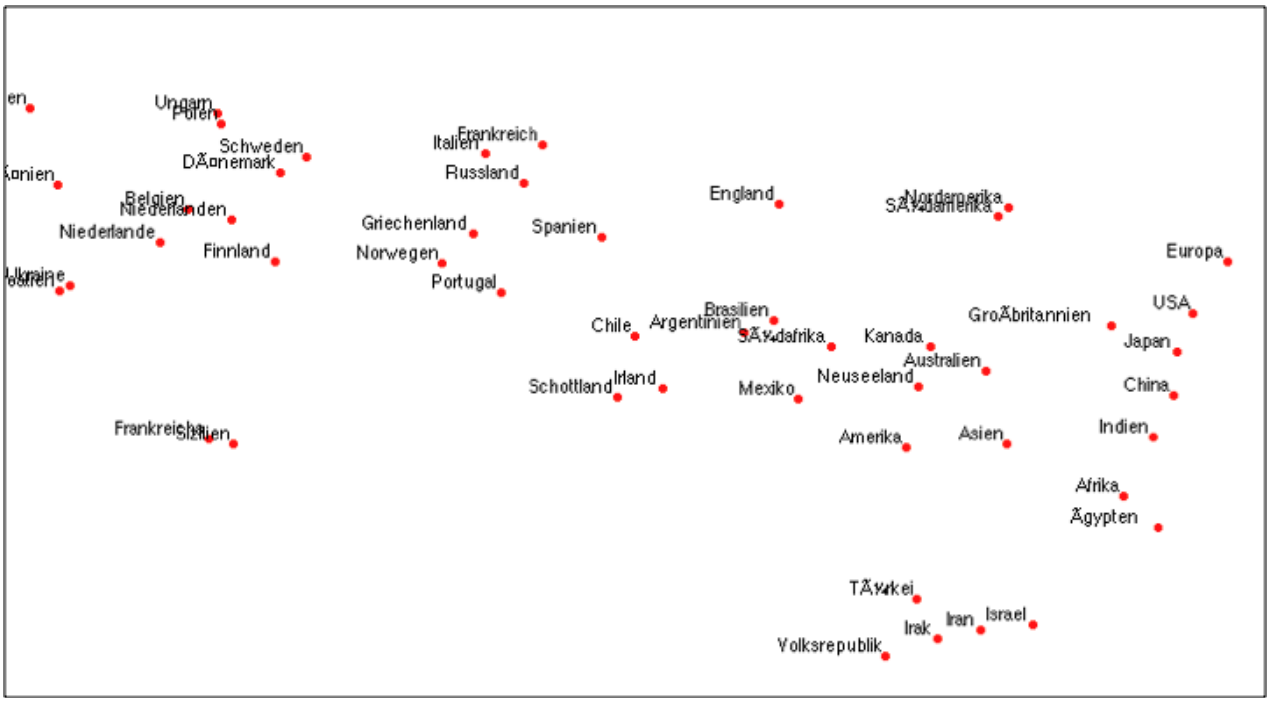

Figure 2

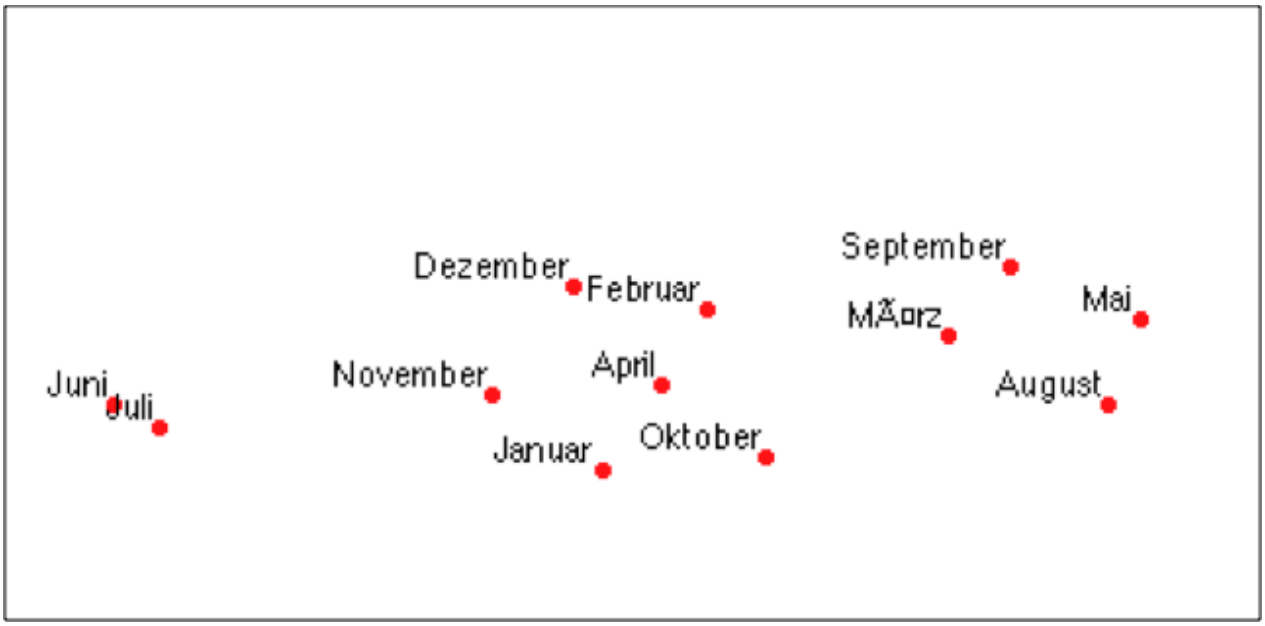

Figure 3

4.3 Window Error Trend

The training objective is to minimize the *rank loss* function for each sentence-document pair (s,d). For each iteration in the learning stage, we output the *rank loss* value per iteration and present them in the following figures. Figure 4 shows the window error for exactly every iteration. Figure 5 shows the average window error for every 50 iterations. Figure 6 shows the maximum window error for every 50 iterations. Figure 7 shows the minimum window error for every 50 iterations.

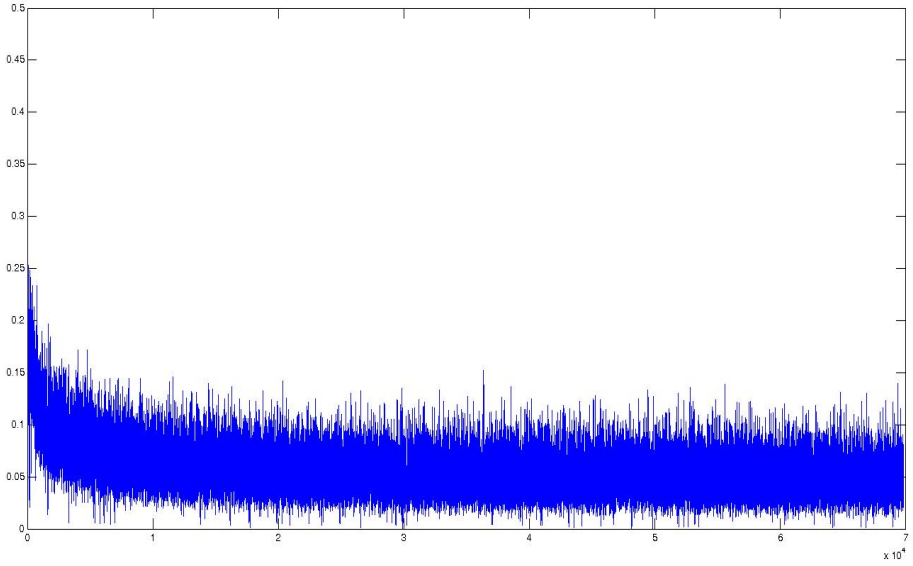

Figure 4: window error from iteration 1 to 70000

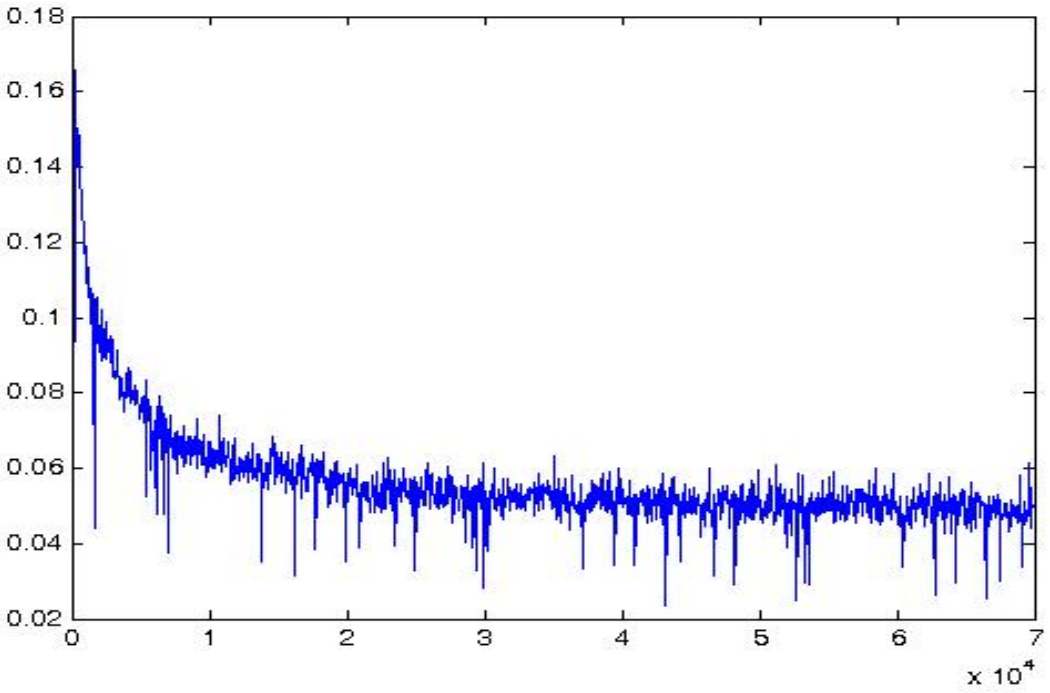

Figure 5: average window error for every 50 iteration

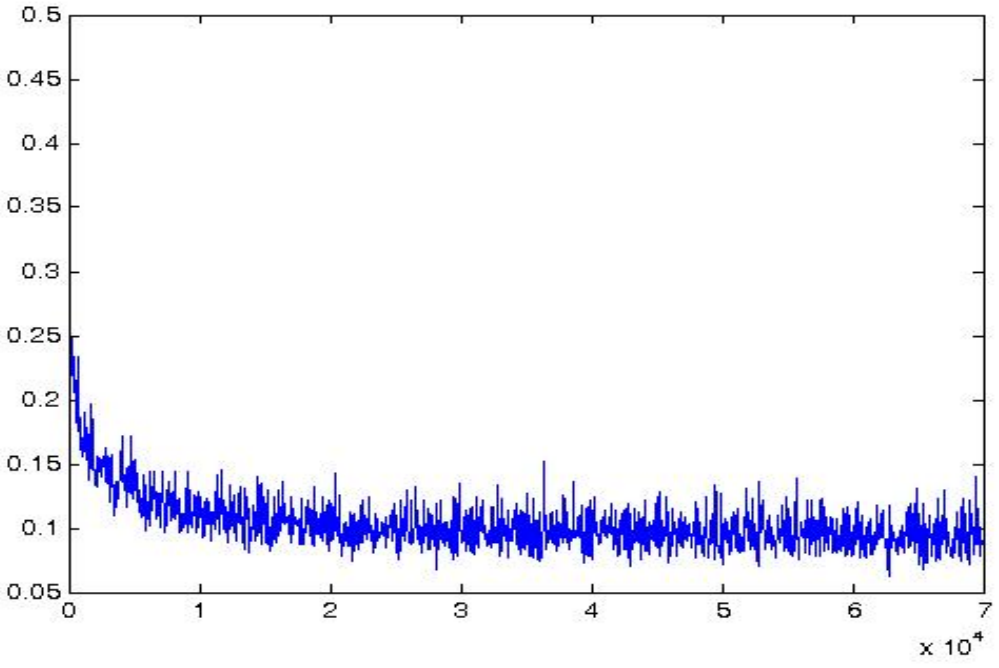

Figure 6: maximum window error for every 50 iteration

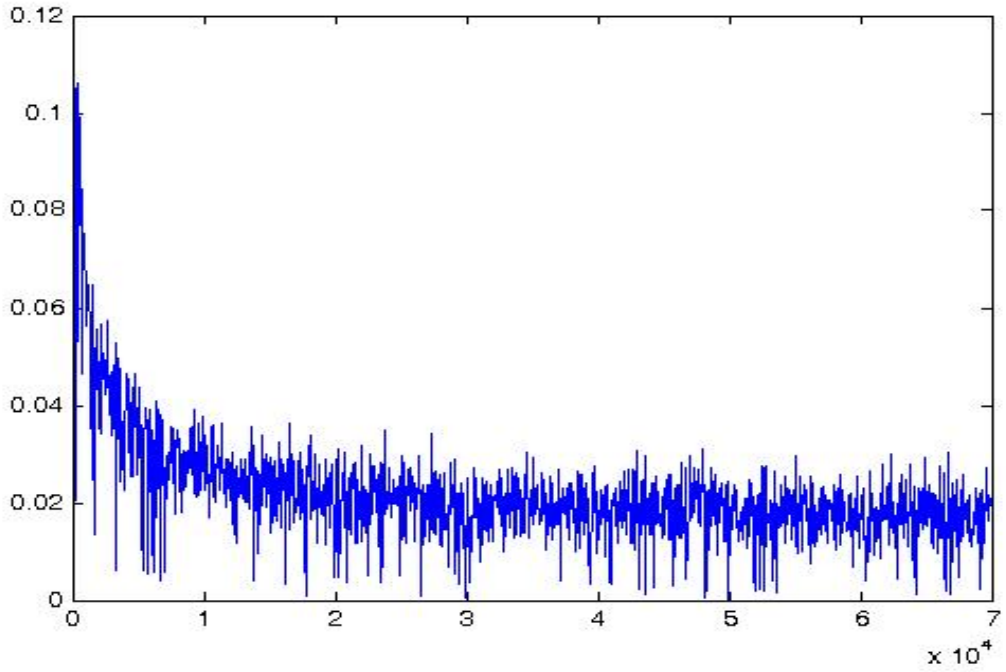

Figure 7: maximum window error for every 50 iteration

From the figures above, we find that the *rank loss* (window error) value decreases significantly from iteration 1 to iteration 10000. Then, it decreases slowly between iteration 10000 and 30000. After iteration 30000, the *rank loss* (window error) fluctuates around 0.06 on average. From the trends of the window error, it implies that most of the learning part, including updating the weights of the two neural networks and building the German word embeddings, are completed before the iteration 3000. Thus, little changes are done for learning after 3000, which might result in overfitting problem for training.

5 Conclusion

We reused the neural language model to generate the embedding matrix (including the semantic word representation for each word in German vocabulary), exploiting both the local and global context. From the results of our experiments, we conclude that our modification can find polysemy of German words reliably. Since our modification works for German and we did not introduce language-specific knowledge in the system, we believe our modification also works for other languages.

6 Package Manual

Our package contains several Python scripts and Matlab scripts. Here is the manual of how to use them.

6.1 Cleaning Wikipedia Dump File

First step is to clean the wiki dump file. This script will clean the raw data and cut the corpus into manageable sized chunks. We also remove wiki articles that have only a few words because they are usually redirect pages or auto-generated pages. The German wiki has more than 4 million pages. In the experiment, we cut them into 500k trunks to keep the size small.

Script	wiki_to_article.py
Arguments	start page num end page num output directory wiki dump file path
Input	Dump file downloaded from wikipedia
Output	Cleaned and tokenized wiki articles.
Sample Usage	python wiki_to_article.py 0 500000 ./ ../wiki_dump.xml

6.2 Generating Dictionary

This script generates the English-German dictionary. For other language, please change the source/target language at the beginning of the script. Some other parameters such as number of concurrent connections can also be set there.

Script	get_dict.py
Arguments	path to English vocabulary file
Input	A plain text file with English words; **one word** per line.
Output	stderr will show the status where each '.' represents a successfully crawled word and '!' represents a failed word. Failed words will be processed automatically. stdout will output the dictionary at the end.
Sample Usage	python get_dict.py english_vocab.txt > dictionary.txt

6.3 Generating Corpus File

For some article file, run get_vocab_from_article.py to generate the vocabulary file. After gathering the vocabulary files and the cleaned wiki articles, the next step is to generate the necessary data for the training algorithm. The following script takes the article and vocabulary as inputs, converts words into integers, and build the corpus and document frequency file. The output corpus will use the vocabulary and the words' id as provided. Any words not in the

vocabulary will be assigned id 1 which is UNK.

Script	gen_data_from_vocab_and_article.py
Arguments	first argument is the vocab.txt second is article.txt, cleaned wiki corpus third argument is the size for each chunk, in MBs
Input	vocabulary and cleaned wiki file
Output	corpus/1.txt: corpus file df.txt: document frequency file

6.4 Initializing German Word Embeddings

This step is to initialize German word embeddings based on the related English word embeddings. We use *GermanEmb_init.m* to do it.

Script	GermanEmb_init.m
Input	English-German dictionary Matrix of English word embeddings English vocabulary German vocabulary
Output	Matrix of German word embeddings

Basically, the script takes the English word embeddings, computes the average embedding and assign it to the related German words.

The output of this script serves as the initial embeddings for the training process.

The format of the dictionary file should follows this format:

English_Word1:English_Word2: ... | German_Word1:German_Word2: ...

Here, all the English and German words have related meanings, e.g.

the:das|dem|den|der|des|die|je

The matrix file of English word embeddings is the training output file from the package, e.g. *iter30000.mat.*

The English vocabulary and German vocabulary are the matlab files which contain cell vectors, and each cell contains one word.

6.5 Finding K-Nearest Neighbors and Evaluation

This step is to find the K-nearest neighbors for each word in the vocabulary based on their embeddings. We use *KNN.m* to do it, and user can specify the value of K.

Script	KNN.m
Input	Matrix of German word embeddings German vocabulary
Output	File which contains the K-nearest neighbors of each word

This script is responsible for finding the K-nearest neighbors for each word in the vocabulary based on their embeddings. We use Matlab's built-in KNN function in this script. And user can specify the value of K.
In each row of the output file, the first word is the subject word, and the remaining are its K-nearest neighbors.

After generating the KNN output file, we can use *GermanSynonymsEvaluate.py* to evaluate how many synonyms appear in the K-nearest neighbors of a subject word.

Script	GermanSynonymsEvaluate.py
Input	Output file from KNN.m
Output	The number of synonyms appearing in the KNN output

Our German synonyms set is from an online source[9].
The script needs to connect to the above website and send a subject German word in order to get the synonyms of this word from the website.
Since the training process clusters polysemous words, instead of synonyms, this script does not evaluate the results well.

6.6 Visualizing Results

This step is to visualize the KNN output. We adopt t-Distributed Stochastic Neighbor Embedding (t-SNE) to visualize it.

Script	drawTSNE.m
Input	Matrix of German word embeddings German vocabulary
Output	A figure of dimensionality reduction of word embeddings

This script uses *fast_tsne.m* and the binary implementation of t-SNE.

References

[1]Eric H. Huang, Richard Socher, Christopher D. Manning and Andrew Y. Ng, Improving Word Representations via Global Context and Multiple Word Prototypes

[2] Wikipedia Extractor, http://medialab.di.unipi.it/wiki/Wikipedia_Extractor

[3] Joseph Reisinger, Raymond J. Mooney, Multi-Prototype Vector-Space Models of Word Meaning (2010)

[4] Lecture notes (CS224N, Stanford University), Language Model: N-Grams, https://prod-c2g.s3.amazonaws.com/cs224n/Fall2012/files/cs224n-lecture5-language-models.pdf

[5] Daniel Jurafsky and James H. Martin. 2008. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Second Edition. Prentice Hall.

[6] Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986)
Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Volume 1: Foundations, MIT Press, Cambridge, MA.

[7] Neural net language models,
http://www.scholarpedia.org/article/Neural_net_language_models

[8] Language as a Foundation of the Semantic Web. Gerard de Melo, Gerhard Weikum (2008) *Proc. Poster And Demonstration Session of the 7th International Semantic Web Conference* (ISWC 2008), CEUR Vol. 401, Karlsruhe, Germany.

[9] WortSchatz, http://wortschatz.uni-leipzig.de/