DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Guided Research

Evaluating Semantic Linking Capabilities of Engineering-Specific Word Embeddings Across Languages

Peter Karl Weinberger

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Guided Research

Evaluating Semantic Linking Capabilities of Engineering-Specific Word Embeddings Across Languages

Evaluierung von Semantischen Verknüpfungsfähigkeiten ingenieurspezifischer Worteinbettungen in verschiedenen Sprachen

Author:	Peter Karl Weinberger
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Tim Schopf
Submission Date:	October 11, 2021

I confirm that this guided research is my own work and I have documented all sources and material used.

Munich, October 11, 2021 Peter Karl Weinberger

Abstract

In this guided research, we create bilingual engineering-specific word embeddings which are capable to aid domain experts during their own literature research and thus assist engineers to find new—for them unknown—techniques. To achieve this, we conduct experiments with several mapping algorithms, namely Transvec, VecMap, and MUSE, as well as, deep learning based word embedding techniques i.e. word2vec and fastText. To create the word embeddings, we exploit a data set of approximately $3,000,000$ engineering-specific articles in German and English. We also evaluate the yielded bilingual word embeddings inspecting both the coherence of the word embeddings and also the capability of mapping both languages into a common space. Based on our findings, we provide two possible approaches for creating bilingual document embeddings to allow engineers to find similar documents based on a query document.

Contents

1. Introduction

Word embeddings are astonishing tools as they embed the pure information about a word in a high-dimensional space such that computer scientists are capable to e.g. find similar words i.e. words that often share a similar surrounding. As the former example only depicts the tip of the iceberg one is able to find magnificent applications of word embeddings in literature [Lia15; WLL18; BR18].

In our guided research, we will exploit word embeddings to cover a certain use case which is described immediately in Section 1.1.

1.1. Use Case

Our use case is to aid engineers finding new—for them unknown—technologies, which are related to technologies they already know. One can imagine the scenario where an engineer of a small or medium sized business has to conduct literature research in order to develop a new product. Companies at that scale usually do not possess a research department, they rather rely on engineers which do both research and development. As a result, engineers experience often the situation that they are sitting in front of an empty search field of their favorite search engine and do not know what to look up. To be more precise, of course they have the knowledge about what they want to accomplish but one is not able to know the unknown new possibilities in the field to reach the known goal. Additional to that, literature in the engineering domain is both German and English, so as a result the engineer—who is also capable of both languages—has to search in both languages to get reliable results.

To tackle the problem, one might come up with a simple search query extension via exploiting thesauri. But such a query extension does not solve the pain of the engineer as the engineer wants to find words which are somewhat related to a query word but not a synonym nor an antonym. We display our desired outcome vs. the output of a thesaurus to illustrate the differences in Figure 1.1. One can see in the figure, that our desired outcome should yield neighborhoods of words which are pervaded of very different words, but which are all related to the query word. To elaborate, the neighborhood should ideally contain the direct translation of the query word as well as other related words in both languages.

(a) Desired example neighborhood of the word "welding" (blue).

(b) Screenshot of the query word "welding" on thesaurus.com [Phi13].

Figure 1.1.: Visualization of the results we want to achieve (left) and the results a thesaurus would be capable of (right).

So in order to full-fill the needs of the engineer, we have to create bilingual word embeddings such that the engineer can search in one common space containing query words of both languages and obtaining bilingual results highly related to the initial query word. This in particular is what we are aiming for as a result of this guided research.

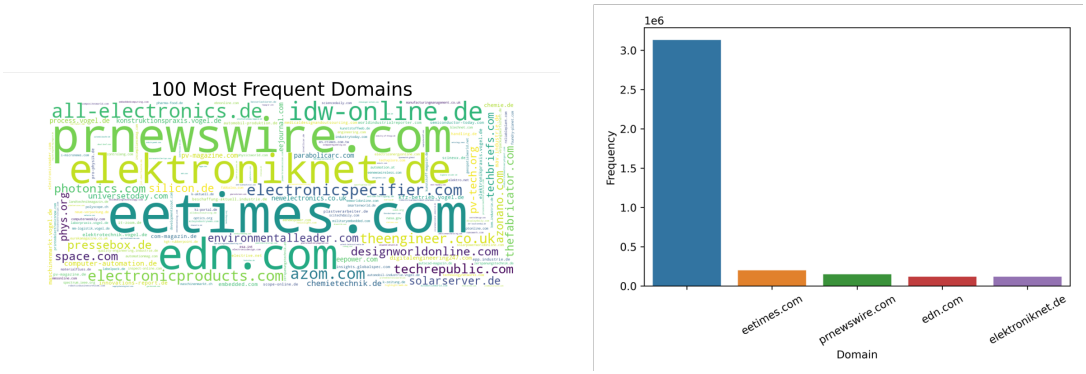
1.2. Research Questions

The two specific research questions, which this work addresses, are:

1. How to merge engineering domain-specific word embeddings of different languages to ensure suitable semantic word comparisons between different languages?
2. How can this approach be adapted for semantic comparisons between engineering domain-specific articles of different languages?

1.3. Data

We have access to approximately $3,000,000$ engineering-related German and English articles. As the articles are scraped and processed real-world data, we analyze at first the origin of the data which is depicted in Figure 1.2 to get an overview of the origin of the articles. In the figure, we can see that the majority of the article items do not contain the information of its origin. However, after contacting the experts who created the data set, we are told that the data is mostly from websites of research institutes like "Fraunhofer" or engineering online platforms like "engineering.com". For the



(a) Most 100 frequent domains visualized as word cloud. (b) Top 5 values of the source value within the data set.

Figure 1.2.: Illustration of the most frequent sources of the articles contained within the full data set.

experiments described in Chapter 4, we distinguish between a reduced data set and the full data set.

1.3.1. Reduced Data Set

We refer to a reduced data set which contains only a fraction of the full data set. The reduced data set contains 654,426 English and 443,558 German articles and was scraped in the year 2017.

1.3.2. Full Data Set

The full data set contains 1,895,025 English and 1,236,991 German articles which were scraped in the years 2017 to 2021 and therefore includes also the reduced data set.

1.4. Resources

Our industry partner—ROKIN GmbH—supplies us the data and one domain expert who is proficient in both languages English and German. Additionally, the industry partner granted access to the Google Cloud Platform and enabled us to work with cutting-edge high-performance computers.

1.5. Structure

First, we describe succinctly the methods used in this work in Chapter 2. Then we elaborate on our evaluation methods in Chapter 3. The experiments we conduct are described in Chapter 4 and discussed in Chapter 5. In Chapter 6, we illustrate possible future approaches to apply further research and present two approaches to tackle our second research question. Finally in Chapter 7, we conclude our guided research.

2. Methods

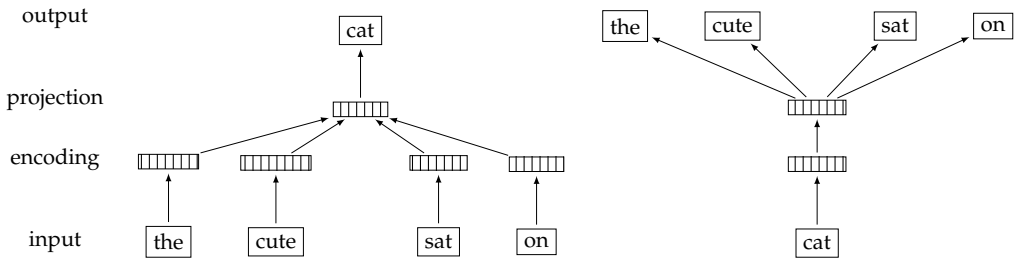
In this chapter, we briefly describe the methods we are using in Chapter 4.

2.1. Word Embeddings

For our experiments, we use word2vec (Section 2.1.1) and fastText (Section 2.1.2) word embeddings. Both are deep learning based methods which create an embedding for each word by using the context—i.e. the surroundings of the word. The most appealing advantage of both word embedding techniques is that one can compress the essential information of a word in a rather low-dimensional space in comparison to vocabulary size—enabling an efficient word representation [Mik+13a].

2.1.1. word2vec

The word embedding technique word2vec published by [Mik+13a] and [Mik+13b] is often referred as the most widely used and successful state-of-the-art word embedding technique in literature [VC19; Lin+15; Li+19].



(a) CBOW model (b) Skip-gram model

Figure 2.1.: Illustration of word2vec models applying both models on the sentence "The cute cat sat on the couch." with a window size of 2.

In a nutshell, there are two different word2vec models which can be applied to create word embeddings (see Figure 2.1). The continuous bag-of-words (CBOW) architecture aims to predict the pivot word from its surroundings, while the skip-gram

model predicts the surrounding words from the pivot word. Both architectures apply one-hot-encoding to the words. For the CBOW version one sums up the individual one-hot-encoded input vectors such that one input vector is present. Then one feeds the input vector into a deep neural network with one N dimensional hidden layer and aims to predict the one-hot-encoded pivot word vector, compares the output via a softmax layer with the pivot word, calculating a loss, and backpropagate the errors to adjust the embedding layer within the hidden layer. After training the model, the final embedding of the pivot word is the hidden N dimensional vector i.e. the weights of the neural network—for the example depicted in Figure 2.1, it is the projection. Of course in a real world setting one would have matrix operations and matrices accordingly i.e. the input would have dimensions of $2 \times \mathit{window\_size}$ and the projection would have dimensions of $\mathit{vocabulary\_size} \times \mathit{embedding\_size}$. One can analogously describe the skip-gram architecture.

2.1.2. fastText

The fastText word embedding technique is rather new and enriches the word embeddings with subword information [Boj+17]. So fastText is very similar to word2vec but unlike the latter fastText operates on subwords—so called n-grams. N-grams applied on words are word parts e.g. disassembling the word “think” in 2-grams would yield the word parts “_t”, “th”, “hi”, “in”, “nk”, and “k_”. Remark that one usually reserves a special character like “_” to have the ability to track whether the letters are the first or the last letters of the word. In particular, fastText takes advantage of the skip-gram architecture of word2vec and extends the scope to word level. According to [Boj+17], the main advantage of fastText over word2vec is that fastText trains faster than word2vec while it yields similar performance.

2.2. Mappings

After a literature research, we discover three easy to use libraries. All those presented methods tackle the problem very differently while each of them tries to find a translation matrix from one embedding space to the other embedding space via exploiting learning dictionaries and therefore minimizing the distance between similar words across languages. While Transvec and VecMap are originally developed to map word2vec embeddings, MUSE is developed for fastText word embeddings. However, also MUSE has the ability to be applied on text-based word embedding files, which enables us to apply it on word2vec word embeddings as well.

2.2.1. Transvec

Transvec uses ridge regression to create a translation matrix and aims for minimizing the least squares loss while predicting the destination vector (of the monolingual embedding one wants to use as target) from the source vector of the other monolingual word embedding multiplied by the translation matrix [big20]. After the training one can use the translation matrix to map unseen words from one language in another. The main pitfall is that the translation matrix is not necessarily the optimal mapping by any chance i.e. the mapping is not necessarily orthogonal nor an approximation since the underlying approach is rather oversimplified and just produces a mapping based on a least squares estimate. However, one will see in Chapter 4 if this first impression holds.

2.2.2. VecMap

VecMap tries to find an orthogonal projection which maps one monolingual word embedding into another via applying linear algebra and computing the singular value decomposition [ALA18a; ALA18b; Art19]. In more detail, VecMap is not only able to operate in a supervised mode, but is also able to create the necessary learning dictionary via semi-supervised, and unsupervised learning. First, the word embeddings get normalized. For the supervised approach the learning dictionary gets initialized and then one iteratively creates a re-weighted orthogonal mapping. The semi-supervised approach is a combination of the supervised and unsupervised approach, which means that the initial seed dictionary is the same as for the supervised approach but the dictionary changes after each iteration. In a nutshell, the unsupervised approach uses an intra-lingual similarity distribution for each word of one language and compares each word with each word of the vocabulary of the other word embedding via Kullback-Leibler divergence to search for most similar words and use them as word pairs for the seed dictionary which will be updated after each iteration.

2.2.3. MUSE

MUSE yields an approximate orthogonal projection after applying a generative adversarial network (GAN) with two objectives [Con+17; Lam+17; Fac19]: the discriminator and the mapping objective. The discriminator wants to identify a word's origin while the mapping objective aims for minimizing the differences of similar words in both languages yielding an approximate orthogonal projection. The learning algorithm is the vanilla GAN approach. At the same time as one wants to minimize the loss of the mapping objective one wants to maximize the discriminator's confusion about the origin of the word vectors. In more detail, they exploit cross-domain similarity local scaling (CSLS) to calculate nearest neighbors and the multilayer perceptron for

the mapping consists out of two hidden layers of the size of 2048, and Leaky-ReLU activation functions. Furthermore, MUSE can operate supervised and unsupervised. For the unsupervised mode [Lam+17], MUSE uses adversarial training and iterative Procrustes refinement, while the supervised approach only uses iterative Procrustes. While training, MUSE heavily relies on external validation data to find the optimal mapping.

2.3. Tools

Throughout the implementation of this guided research, we use several tools which we want to describe briefly.

2.3.1. Similarity Metric

For all experiments described in Chapter 4, we use the cosine similarity as similarity metric. The formula which is applied to obtain the cosine similarity of two n-dimensional vectors $\mathbf{a}$ and $\mathbf{b}$ is depicted in Equation 2.1.

$$sim(\mathbf{a}, \mathbf{b}) = cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}} \tag{2.1}$$

2.3.2. t-SNE

There are numerous methods to visualize high-dimensional data in a low-dimensional space. One method is t-distributed stochastic neighbor embedding (t-SNE) [HR02; VH08]. The strength of t-SNE is that it is a nonlinear dimensionality reduction technique which can be applied to a variety of high-dimensional data. For word vectors in particular it is well-suited because it constructs a probability distribution over word pairs such that similar words getting a higher probability assigned in comparison to dissimilar words. Then t-SNE defines analogously a probability distribution in the low-dimensional space and minimizes the Kullback-Leibler divergence between this probability distribution and the previously mentioned. As a result, t-SNE arranges similar high-dimensional data points close to each other in a low-dimensional space.

2.3.3. Python Libraries

To make the externally used software packages transparent, Table A.1 provides a list of the packages that are installed using pip with the corresponding version. All other software versions are described in the corresponding sections of Chapter 4.

3. Evaluation Methods

State-of-the-art neural word embeddings represent the relativeness of words via geometry. Words are encoded into a vector $\mathbf{w} \in \mathbb{R}^n$ with n as the number of dimensions. The neighborhood of such an vector $\mathbf{w}$ should commonly reflect the contextual similarity. Thereby, one can ask the question: How to evaluate a contextual similarity? In general such an evaluation is more of subjective nature, rather than objective [Sch+15]. In addition, one also has to consider the specific use case to find the right model for the particular application. Hence, the evaluation of word embeddings is not trivial.

One can divide the methods of word embedding evaluation into two categories: extrinsic and intrinsic evaluation [Bak18]. Extrinsic evaluation rates a model accordingly to a downstream task e.g. classification or other supervised learning tasks which utilize the pre-computed word embedding. Intrinsic evaluation on the other side, tests the quality of a representation independent of a specific natural language processing task.

An extrinsic evaluation has several pitfalls and is a controversial topic in literature [Sch+15]. One major disadvantage of extrinsic evaluation is the fact that this type of evaluation only examines the goodness of an embedding by the downstream task. As a result, the outcome of the evaluation is dependent on a task which is not necessarily meaningful for the original use case. Furthermore, one must have labeled documents or words to perform at least a semi-supervised downstream task.

Intrinsic evaluation techniques—on the other hand—test the quality of the word representations independently of a specific natural language processing task. Considering that this work—establishing and enabling engineers to discover individual unknown fields—has no real downstream task, because of our main goal and the gigantic data of three million mostly unlabeled engineering-specific articles in German or English—which will be exploited in this research—the evaluation will be intrinsic.

There can be numerous intrinsic evaluation metrics found in literature [Bak18; GD16; Wan+19; Cam+17]. However, in this research we focus on two evaluation techniques which are chosen to score the model's capability of certain language understanding tasks i.e. each of the discussed approaches in Section 3.1 and 3.2 covers different goals. The implementation of both evaluation methods is described in Section 5.1.1 and 5.1.2.

3.1. Comparative Intrinsic Evaluation

In general, comparative intrinsic evaluation as described in [Sch+15] takes representatively selected query words into account. The k most similar words to each query word will be calculated by a similarity metric e.g. cosine similarity. People are then asked to select the word they feel is most similar to the initial query word. Thus, we can subsequently calculate a score and know which of the trained models represents similarity relationships the best.

This approach will evaluate the model's ability to map multiple languages into one single model. The rough idea is that for our use case a good model would represent a query word in the way that the nearest neighbor of that word should be the plain translation of the query word. So we are adapting this evaluation technique and alter it to fit our use case. More precisely, we search for the k most similar words to each representatively selected query word. Then evaluate if the translation of the query word is the nearest neighbor, is in the set of the three nearest neighbors, if the translation is in the set of the five nearest neighbors, or in the set of the ten nearest neighbors. For each metric we count the number of appearances of the translation and divide it by the number of selected query words. To give more insights, a domain expert will evaluate if the translation is in the specified neighborhood and we calculate the distance between the word vectors via exploiting cosine similarity.

One problem to solve is the selection of representative vectors from a high dimensional vector space. From a mathematical point of view one could try to find uniformly distributed vectors in $\mathbb{R}^n$, or another suggestion would be to find vectors in the vector space which are orthogonal to each other. Another approach would be to sample word pairs from engineering-specific dictionaries randomly, and then let a binomial trial decide of which language the query word would be. Furthermore, from a practical way of thinking one could also just sample randomly from the word embedding vocabulary. However, we are of the opinion that specific words of various contexts chosen by a domain expert would be the best idea to ensure that the evaluation covers as much areas as possible. Therefore, an expert chooses 20 highly independent engineering-specific German and English query words.

3.2. Coherence

Coherence is another intrinsic evaluation technique which is capable to evaluate the semantic word representations [Sch+15]. The basic principle of this approach is that good word embeddings should have coherent neighborhoods for each word, so that inserting a word not belonging to this neighborhood should be relatively easy to spot.

To evaluate the coherence of a word embedding, one has to sample query words from the vocabulary. Then the k-nearest neighbors are determined, and afterwards an "intruder" word is sneaked into the list of k-nearest neighbors that has nothing to do with the other words. A human evaluator is of charge to find this "intruder" word and a score can be computed to evaluate different word embedding models.

This technique is perfect for our use case and we do not have to refine this method from the ground up. But to be more specific, we apply coherence to our models and evaluate them based on three, five, and ten nearest neighbors determined by using the cosine similarity. The intruder word will be chosen randomly for each evaluation instance, while the query words are chosen by a domain expert and are the same across the evaluation of different models. The exact approach is analogous to the selection of the query words in Chapter 3.1. In the original approach described in [Sch+15], they calculate a precision score by dividing the number of raters who discover the "intruder" word by the total number of raters. This is infeasible for us, so we decide to create the score based on the number of correctly classified "intruder" words divided by the number of over all query words. In the referenced paper, they mention that each Turker receives 25 to 50 query words at maximum. This is why we conclude that our domain expert could evaluate 30 instances of intrusion detection. Additionally, we ensure that the "intruder" word is indeed a word which has no close relationship to the query word by take only words into account which are ranked as the $100th$ nearest neighbor.

4. Experiments

In this chapter, we describe the conducted experiments in detail. For that very reason, we distinguish between initial experiments executed in the beginning of this work (Section 4.1) and final experiments which are conducted and evaluated to provide a final overview (Section 4.2).

4.1. Initial Experiments

For acquiring a better understanding of the underlying engineering-specific data and the use case—we are aiming to solve as good as possible—we have performed certain initial experiments which are limited in their expense. The main reason for the initial experiments are that we can easily evaluate many different approaches of both components—word embeddings and mappings—in a rather short period of time using a reduced number of epochs in training the word embeddings while at the same time using only a fraction of the full data set. In particular, we use the reduced data set already described in Section 1.3.1.

4.1.1. Learning Dictionary

Several mapping algorithms supply supervised or semi-supervised approaches to union two embeddings of different languages into one common embedding space via minimizing the distance between words which are having the same meaning in different languages. Therefore, we need to create a learning dictionary, but we also can not neglect the fact that we are focusing on engineering-specific word embeddings. As a conclusion, our learning dictionary must contain representative words from the engineering domain. To get a reasonable amount of representative engineering-specific training data, we ask a domain expert in the field for creating a corresponding training dictionary. This manually created training dictionary contains 170 word pairs which are direct translations of words from English to German.

4.1.2. Word Embeddings

In the phase of the initial experiments, we apply word2vec and fastText models to the reduced data set. Because of the nature of the used models, we also apply certain preprocessing to the data. After a first literature research, we choose to use word2vec because of the continuous bag of words approach which fits our use case very well. This is due the nature of our use case: We want embeddings which are meaningful representations of the domain specific words, which are represented by its surroundings i.e. the context. Furthermore, word2vec is widely used, tried and tested. As a second word embedding model, we choose fastText. The main difference between word2vec and fastText is, that fastText focuses on word parts. The latter implies that the word embeddings are rather different in their nature. This in particular will be shown in Section 4.1.4.

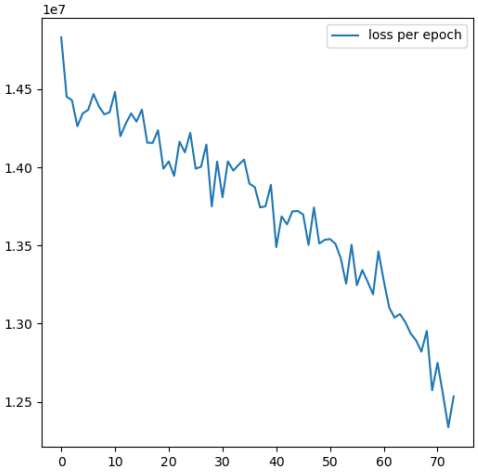
Preprocessing

The basic preprocessing is to transform each article into an array of sentences and each sentence into an array of words. We only alter the words in the way that they are in lower case letters, so that we do not change any information. As a result, we also do not lose information—as we would do using lemmatization or stemming. A downside of this is for sure that we have a comparable larger vocabulary. The English vocabulary size is for both models 338,668 words, and the German vocabulary size is for both models 530,070 words. The vocabulary size in both languages is quite large in comparison with literature which tries to use better scaling word2vec models apart from the original proposed model with a vocabulary size of 18,498 and states that it is common to use vocabulary size of 10,000 to 50,000 words [Li+19]. However, for our use case described in Section 1.1, we also need less frequent words because engineers can be eager to find related information for only rare or very new technologies. Therefore, we have to work with a relatively big vocabulary.

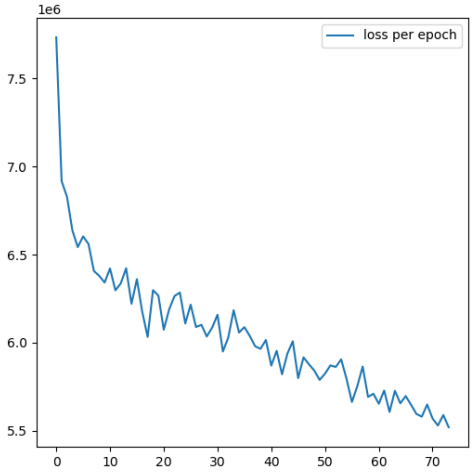
While we are only using simple preprocessing techniques, one has to keep in mind that the reduced data set for the English corpus alone is still 14,498,368 articles. For speeding up the prototyping and working with the full data set, we process the articles in parallel on 32 cores. This little enhancement results into reducing the time of preprocessing from multiple hours to about half an hour. To be more precise, one has to note that this preprocessing is applied to all data sets taken into account in this work.

word2vec

We use the word2vec implementation of the Python library gensim [RS11] in the version 4.0.1, which uses a slightly enhanced version of the original training algorithm of the



(a) Loss of the word2vec model trained on the reduced English corpus.



(b) Loss of the word2vec model trained on the reduced German corpus.

Figure 4.1.: On the left, the loss of the word2vec model trained on the English corpus is depicted. On the right, the loss of the word2vec model trained on the German corpus is displayed. The x-axis indicates the number of epochs and the y-axis is the loss per epoch. Both models are trained on the reduced data set.

vanilla word2vec published by [Mik+13a] and [Mik+13b]. As a matter of fact one has to note, that we are using for all occurrences the same version of gensim. We are using an embedding size of 256, a window of 10, and trained the models with 75 epochs on 32 cores. All other parameters are default and stated in the official documentation [Reh21c]. The first fact we notice is that the loss of the word2vec model is continuously decreasing for both languages—as depicted in Figure 4.1.

As a result of this analysis, we can opt for larger amounts of epochs. This is not necessarily the case in every machine learning task. Often one has to carefully adjust the number of epochs such that no overfitting is present. In our use case, we simply can not overfit because we want to have the best word embedding. So choosing the number of epochs is more a time factor, than an informed decision. We will apply this findings in Section 4.2.2.

fastText

For fastText we use a variety of implementations to get a glampse of experience using the model. First, we start with exploring the original implementation of fastText [Boj+17] (version 0.9.2), which is available on GitHub [Fac20] and investigate the binary and the Python library. After some inconveniences using the interface of the original implementation, we switch to gensim's implementation of fastText [Reh21b].

We apply the default parameters depicted in [Reh21b], but alter the size of the embeddings to 256, and train the model on 32 cores.

4.1.3. Mappings

First, we conduct a literature research for mapping multilingual word embeddings into a common vector space and find several "out-of-the-box" approaches namely Transvec, VecMap, and MUSE. For the preliminary mapping of the embeddings created by word2vec, we take Transvec, and VecMap into account. This is due to the fact that these approaches are explicitly developed to map word2vec embeddings. MUSE which is developed by Facebook is applied to the fastText models, because it is explicitly stated that MUSE is developed for fastText.

Transvec

We investigate Transvec and at first glance the project [big20] seems promising as the documentation shows rich examples, but the appearance is deceiving. After downloading the latest version (0.1.0) via pip it turns out that the package is incompatible with gensim in the version 4.0.1. So we dig into the code of the package and fix some points of failure. After fixing the code such that Transvec should work fine with the installed gensim library we discover that the library Transvec is in the best case astonishing bad in mapping two word2vec embeddings of different languages or does nothing useful at all.

VecMap

VecMap supplies three different modes: supervised, semi-supervised, and unsupervised [ALA18a; ALA18b; Art19]. We use the latest state of VecMap which is the commit from 7/1/2019. At first sight, we apply the word2vec models to the semi-supervised and unsupervised approaches. We try several values for the batch size (1000, 10000, and 1000000), but then decide to stick with the default parameters. The reason for this is that we dig into the code of VecMap and see that in the manual of VecMap it is

stated that one should not really tweak those parameters, and we can also observe no difference in the output of VecMap.

MUSE

MUSE is available in the modes supervised and unsupervised [Con+17; Lam+17; Fac19]. We use the latest state of the MUSE repository which is the commit from 4/23/2019. All parameters are set to default and we apply MUSE with and without a graphics processing unit (GPU) to see if the GPU speeds up the computation as promised in the readme file of the repository.

4.1.4. Findings

The preliminary experiments uncover rapidly pitfalls of the—at first glance—promising approaches namely fastText and Transvec. Transvec is not applicable at all and will be discarded for further experiments. FastText does not really fit our use case because of the nature of fastText—putting the parts of words in focus—we receive neighborhoods i.e. surroundings of words which are driven by the word parts. Table 4.1 illustrates this fact. While we observe that 100% of the nearest neighbors of the fastText model include the German search term "schweißen", we can see that in the word2vec model 40% of the neighborhood does not contain the search term and those are rather related to the original search term. We observe this pattern with multiple different query words in English and German and therefore—because of our use case—we discard fastText from further experiments.

We have already stated the fact that the loss of the word2vec models decreases continuously in Section 4.1.2. As a result, we will experiment with much higher number of epochs in the final experiments (see Section 4.2.2).

For VecMap, we see good first results in the common bilingual vector space. But we also notice a need for more training examples to conduct experiments with the supervised mode in VecMap. Therefore, we have to acquire a larger training dictionary.

MUSE is very promising and at first glance delivers better results than VecMap. However, this is only a subjective first impression, we will see if this feeling is indeed true after applying and conducting the final experiments and evaluating them in Chapter 5. We have applied MUSE to fastText because MUSE is developed by the same corporation—Facebook. That said, MUSE advertises to map fastText embeddings of two languages into a common space. However, it is also capable to work with text based word embedding files and the gensim library is also capable to store a word2vec model as a text file. A downside of text based word embeddings is that it seems that reading the word embeddings takes longer in comparison to word embeddings saved

Table 4.1.: Ten nearest neighbors of the German word “schweißen” (English: “welding” in the two different models (left: fastText, right: word2vec; in descending order of the cosine similarity).

fastText	**word2vec**
nahtschweißen	laserschweißen
heftschweißen	laserschneiden
einschweißen	roboterschweißen
reibschweißen	punktschweißen
kaltschweißen	löten
tiefschweißen	verschweißen
trennschweißen	entgraten
tandemschweißen	stanzen
anschweißen	laserstrahlschweißen
handschweißen	biegen

in a binary format. As a conclusion, we will apply MUSE to word2vec models and describe the process in Section 4.2.3.

In general, we see the need to mark the words based by their origin. For example in the German corpus, there are many English words included. This is due to the fact that there are many English words present in the German engineering domain. So in an evaluation one could not distinguish if the word embeddings are mapped properly in a common space or if the word embeddings just look that way due to its nature. With an explicit prefix or postfix i.e. altering the words before applying the mapping one can overcome this issue.

4.2. Final Experiments

After accomplishing the initial experiments and getting familiar with the models, mappings, and data set, we conduct the final experiments described in this section and evaluated in Chapter 5. We apply the findings from Section 4.1.4 and use the full data set described in Section 1.3.2.

4.2.1. Learning Dictionary

As we saw the need for a larger number of items in a training dictionary in the initial experiments, we decide to replace the hand-crafted initial learning dictionary containing 170 engineering-specific word pairs in English and German with additional

engineering-specific word pairs. After discussions with a domain expert, we discover an engineering-specific online dictionary [Onl]. So we extracted as many training examples as possible in the sense that we have downloaded the complete available dictionary and process it via extracting items with BeautifulSoup and then clean and extract the word pairs with regular expressions. As we can only process words and not group of words, we discard such lines in the original dictionary which contain multiple words in either German or English. Furthermore, we extract multiple lines from one line if there are alternatives present for a given word. For example we encounter pairs like "(klebrigkeit, stickiness/tackiness)", so we transform this line into two lines "(klebrigkeit, stickiness), and (klebrigkeit, tackiness)". We apply this technique in bidirectional ways such that we extract as much knowledge as possible, and have all perturbations included in the final training dictionary. In the end, we extract 655 engineering-specific word pairs from [Onl]. We use only these word pairs for creating the bilingual mappings of the final experiments.

Finally, we add the same prefixes accordingly to the words as described in more detail in Section 4.2.2.

4.2.2. Word Embeddings

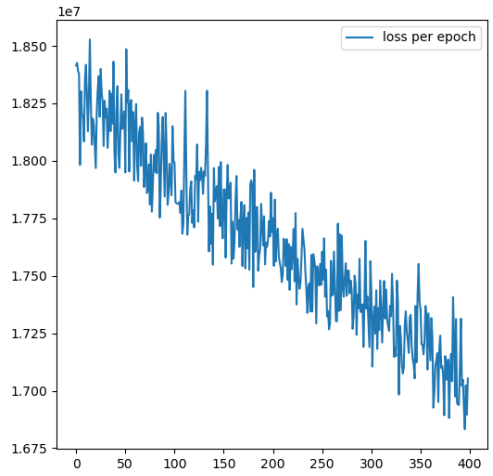
As mentioned in Section 4.1.4, we discard the fastText word embeddings and focus on word2vec word embeddings. Furthermore, we apply the lessons learned regarding the preprocessing (Section 4.2.2) and the mappings (Section 4.2.3).

Preprocessing

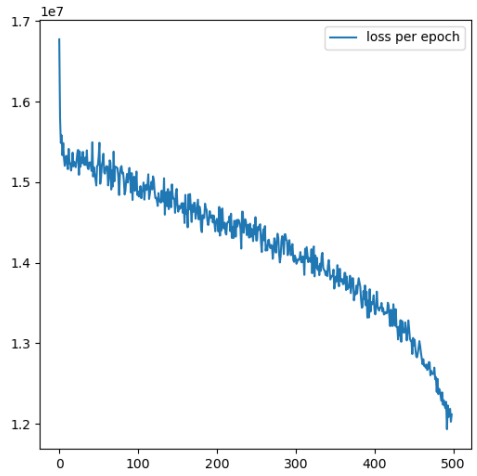
For training the final word embeddings, we apply the same preprocessing as already described in Section 4.1.2. Additionally, we add prefixes to each word. These prefixes enable us to identify the origin of a word—or with other words the original corpus—in a common vector space. For German words we add the prefix "de_" and for English words we add the prefix "en_" analogously.

word2vec

We basically apply the same parameters as in the initial experiments (see Section 4.1.2), but we change the dimensionality of the word embeddings from 256 to 300 because in the original word2vec paper [Mik+13a], the authors used 300 dimensions for their embeddings trained on Google News articles. As we have two large corpora, we are of the opinion that we can go with a rather large embedding size, so we choose 300 dimensions as our final embedding size.



(a) Loss of the word2vec model trained on the full English corpus.



(b) Loss of the word2vec model trained on the full German corpus.

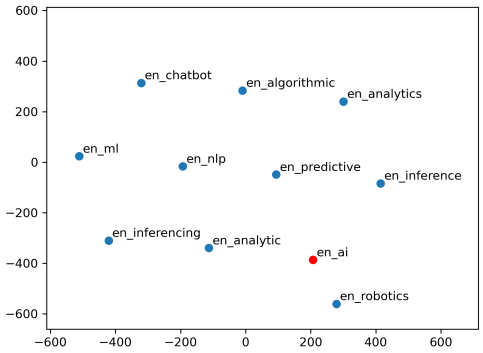
Figure 4.2.: On the left, the loss of the word2vec model trained on the English corpus is depicted. On the right, the loss of the word2vec model trained on the German corpus is displayed. The x-axis indicates the number of epochs and the y-axis is the loss per epoch. Both models are trained on the full data set.

As we saw a continuously decreasing loss in the initial experiments, we increase the number of epochs from 75 to 500, which means that the creation of the embeddings for each language takes approx. five days on 32 CPUs. To choose the most appealing word embedding, we store a snapshot of the model in different formats to have the ability to adapt to unsuspected possibilities after creating the word embeddings after every $100th$ epoch. The loss of both models can be viewed in Figure 4.2. One has to mention that we run out of storage while training the English word embedding model after the $475th$ epoch, so we had to enlarge the storage of the virtual machine and rerun the training algorithm from epoch 400 to epoch 500 to be on the safe side. We display the loss of the English model from epoch 400 to 500 in Appendix A.1. What we see is that the English model experiences noisier updates than the German model. But this effect can also be caused by visual effects because the loss does not decrease that heavily in the English model in comparison with the German model. Besides that fact, both losses are decreasing continuously as we already have observed during the initial experiments.

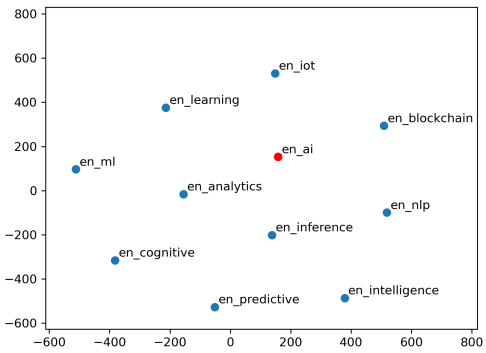
Table 4.2.: Ten nearest neighbors (nn) of the word "AI" with the assigned cosine similarity metrics (cos) observed over the training of the English word2vec model.

100 epochs		200 epochs		300 epochs		400 epochs		500 epochs	
nn	**cos**	**nn**	**cos**	**nn**	**cos**	**nn**	**cos**	**nn**	**cos**
en_ml	0.54	en_ml	0.55	en_ml	0.56	en_inference	0.54	en_analytics	0.61
en_nlp	0.48	en_inference	0.52	en_nlp	0.52	en_ml	0.54	en_learning	0.60
en_inference	0.47	en_nlp	0.49	en_inference	0.52	en_analytics	0.52	en_intelligence	0.59
en_analytics	0.47	en_learning	0.46	en_analytic	0.48	en_nlp	0.51	en_cognitive	0.56
en_inferencing	0.42	en_analytics	0.45	en_analytics	0.48	en_learning	0.51	en_iot	0.53
en_chatbot	0.42	en_algorithmic	0.43	en_learning	0.48	en_predictive	0.48	en_predictive	0.53
en_predictive	0.41	en_inferencing	0.43	en_predictive	0.46	en_analytic	0.47	en_inference	0.52
en_analytic	0.41	en_analytic	0.43	en_algorithmic	0.44	en_cognitive	0.46	en_ml	0.52
en_algorithmic	0.41	en_algorithms	0.43	en_chatbot	0.44	en_aiot	0.45	en_blockchain	0.52
en_robotics	0.40	en_aiot	0.41	en_automl	0.44	en_intelligence	0.45	en_nlp	0.50

In general—as far as we can judge from the loss—the models trained on 500 epochs seems the most appealing. But we want to answer the question how they differ in the neighborhood of words i.e. answering the question: How does the neighborhood alter with increasing number of epochs? To address the question, we examined the surroundings of words e.g. the German abbreviation for the word sequence "künstliche Intelligenz" which is "KI" and the English abbreviation for the direct translation "artificial intelligence" which is "AI". The development over time of the neighborhood of the word "AI" is displayed in Table 4.2.



(a) English word2vec model after 100 epochs.



(b) English word2vec model after 500 epochs.

Figure 4.3.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word "AI" (marked red) into two dimensional space.

One can observe that with increasing epochs the ten nearest neighbors of the word "AI" get consistently higher cosine similarity values. Additionally also from an subjective perspective, the neighborhood seems more consistent and coherent. Moreover, we also analyzed the t-SNE projection of the neighborhood of the word "AI" into a two dimensional space after every $100th$ epoch (see Figure 4.3 for the comparison between the output after the $100th$ epoch and the $500th$ epoch, and Appendix A.2 for the full overview). One can see, that the model after the $500th$ epoch yields a neighborhood which has two advantages over the neighborhood of the model observed after the $100th$ training epoch: The words have a higher similarity metric and t-SNE is agnostic to rearrange similar data points in a high-dimensional space very close to each other in a low-dimensional space [VH08] and even more important is the fact, that one can distinguish the words better from each other because of the larger differences in the cosine similarity scores. We observe those properties in multiple random samples and also in the models trained on the German corpus. Those effects also occur after 200, 300, and 400 epochs, but slightly less intense. It seems that those effects are slowly converging to a stable state. Nevertheless, we decide to take the models trained with 500 epochs into account for the final evaluation.

4.2.3. Mappings

To achieve a bilingual embedding in a common space, we evaluate three different approaches in the final experiments. First, we train a naïve bilingual model by simply extending the full English data set with the full German data set, taking this merged data set into account for training the word2vec model and creating the bilingual word embeddings for the vocabulary. The second approach is to create bilingual embeddings by using the word embeddings yielded from the word2vec models and mapping those embeddings into a common space via MUSE. Finally, we also evaluate the effectiveness to create bilingual embeddings via VecMap using the very same word2vec embeddings as for MUSE which are already described in Section 4.2.2.

Naïve Bilingual Model

For the naïve bilingual model, we extend the English corpus with the German corpus while keeping all preprocessing steps from the two monolingual word2vec models (see Section 4.2.2), and then train the bilingual model for 500 epochs with an embedding size of 300 and windows size of 10. We train the model on 16 CPUs, because we realize that gensim's word2vec implementation does not scale that well. The cores are more idle as fully used, while we have now a much larger vocabulary and the number of sentences is now $62,772,826$. Taking more cores into account would introduce more overhead

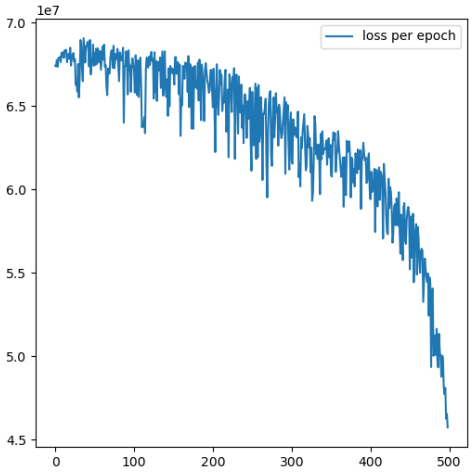


Figure 4.4.: Loss of the naïve bilingual word2vec model trained on both corpura. The x-axis indicates the number of epochs and the y-axis is the loss per epoch. The model is trained on both—English and German—full data sets.

and therefore the training time would be increased. To ensure that no side-effects can occur, we shuffle the sentences' order randomly. Finally, training the model on both corpora for 500 epochs takes about five days on 16 cores. We notice, that due to massive training examples and a larger vocabulary, the training loss is much larger than the loss of the monolingual models (see Figure 4.4).

MUSE

We apply MUSE in eight different ways to get a differentiated look at the performance of the algorithm. More precisely, we apply four different parameter sets for the supervised, and four different parameter sets to the unsupervised approach of MUSE. That means that the four configurations for the supervised approach are distinguishable in the number of refinements ("n_refinements") namely 5, 10, 15, and 20. Furthermore, we exploit the created training dictionary without prefixes described in Section 4.2.1. We will elaborate on this choice later on. The unsupervised models are composed by all perturbations of the two configuration sets n_refinement $\in \{5, 10\}$ and number of epochs ("n_epochs") $\in \{5, 10\}$. For all eight models additional parameters are set: "src_lang"=en, "tgt_lang"=de, "cuda"=true, "emb_dim"=300, and "seed"=42. With other words, we want to create a mapping such that the English vector space is the

```
fencing Gymnastik 2.17
volcano Magma 2.21
lizard Krokodil 3.13
episode Kapitel 3.17
Times Guardian 3.17
Latin Deutsch 2.92
Clinton Obama 3.04
Jupiter Merkur 3.21
evolution Darwin 2.25
essay Hausaufgaben 2.33
```

Listing 4.1: First ten items of the word similarity evaluation set (English to German) of [Cam+17]. The columns are tab-separated and in order English word, German word, similarity score.

source word embedding space and the target embedding space is the German. We take advantage of a server with a GPU in order to speed up computation as MUSE inherits a deep neural network, the GPU shortens the time consumption of the computation by an approx. factor of 30. Furthermore, we describe the size of the embeddings ("emb_dim"), and configure a seed such that we can reproduce our results if we need to.

While we train the MUSE model to create the embeddings, we notice that MUSE relies heavily on third party similarity scores. One similarity metric used in the evaluation set of MUSE is for example [Cam+17]. Those similarity scores are hand-crafted. In a nutshell, people assigned similarity scores from 0 (totally dissimilar) to 4 (very similar) to word pairs containing words of different language and the outcome is usually averaged—Listing 4.1 illustrates such a word similarity evaluation data set. Then one wants to see how good the own similarity scores correlate with those hand-crafted ones via applying e.g. Pearson correlation coefficient. So because of the fact, that MUSE is designed in the way that after every refinement MUSE has to evaluate the current performance on evaluation data, we are not able to apply MUSE on the preprocessed data and have to remove the prefix from both corpora. Then we apply the MUSE algorithm and after that we have successfully created the mapping, we export the word embeddings in two distinguishable text files—for each language one—and add the original prefix to the words. Finally, we are able to combine both files into one common file via appending the German word embeddings line by line to the English word embeddings and adjust the number of words which are contained in the final output accordingly. One drawback is that the word similarity evaluation data sets used

by MUSE contain words with correct grammar. As a result, we lose a large amount of information in the evaluation step. This is due to the fact, that we already have lower-cased the words and lost the information about the original case.

Additionally, we also discover a bug in the code of the supervised approach. MUSE only stores snapshots of the mapping if the mapping outperforms all previous mappings based on evaluation data sets. However, we notice that after we continue training the model with higher number of refinements (more than five refinements), MUSE yields the very same bilingual embeddings as computed after the $5th$ refinement. We dig into the log output and see that the loss on the evaluation data set decreases continuously after every refinement. The conclusion is to store the last model with the highest number of refinements. So we changed a few lines of the code of MUSE to accomplish that.

After training all eight mappings, we conduct a naked eye comparison of the results, and decide to only evaluate the supervised mapping with a total number of refinements of 5, and the mapping with the number of refinements of 20. For the unsupervised models, we see only very small differences in the output, as a conclusion we stick with the default parameters, which means that we will evaluate the model trained with the number of refinements of 5 and the number of epochs of 5.

VecMap

The journey with VecMap is rather unspectacular. VecMap does not inherit evaluation while creating the model. Thus, it is easier to apply to our preprocessed data. According to the codes' documentation the default parameters should be rather used and not altered [Art19], as a result we stick to the default parameters and create three different mappings via the modes supervised, semi-supervised, and unsupervised. For the supervised and semi-supervised approach, we use the training dictionary described in Section 4.2.1. For all three models, we train the respective model analogously to the MUSE models i.e. we want to create mappings which minimize the distance of the English word embedding to the German word embeddings. To speed up the computations, we use GPUs and to be able to reproduce the outcomes we use for all three mappings the seed 42. After we have trained the VecMap models and applied the mappings to each of the two output word embedding files, we merge each of the two files into one file by concatenating the text files analogously to Section 4.2.3.

In the end, we have three different common bilingual word embeddings created with the three different approaches of VecMap. We also review the results and compare them by the naked eye and decide that we shall evaluate all three different outputs because the results of them are all looking good at first glance, while being completely different created.

5. Evaluation & Results

In this chapter, we apply the evaluation methods described in Chapter 3 and compare the different chosen bilingual word embeddings of the final experiments depicted in Section 4.2. To be precise, we evaluate seven final models. One naïve bilingual model, three different mappings created by MUSE, and three different mappings created via VecMap. We differentiate between comparative intrinsic evaluation (Section 5.2.1) and coherence (Section 5.2.2). But before we can evaluate the models, we have to create evaluation data, which is described in Section 5.1.

To give the reader a better overview about the used abbreviations, we elaborate on that. In the following sections, we refer to the naïve bilingual model as "NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model" interchangeably, "SUP" as an abbreviation of supervised, "UNSUP" as an abbreviation of unsupervised, "REF" indicates the number of refinements (parameter "n_refinement"), "EPOCHS" indicates the number of epochs trained (parameter name: "n_epochs"), and "EN_DE" may represent that we map the English word embedding to the German word embedding.

5.1. Creating Evaluation Data

To create a trustful evaluation data set one has to respect many different properties of the words one would like to choose. The first characteristic that should be mentioned is that the words used in an evaluation have not been used in any steps of the training—we will call them unseen words or unseen evaluation set. In traditional machine learning tasks this can be simply done via splitting the original data set into at least training and test data. However, on the one hand, we want as many training examples as possible, on the other hand, we also have to choose the evaluation words with respect to their representativeness for the holistic engineering domain, so as a result we have to choose evaluation words from the vocabulary and refine the choices by a domain expert and by asserting that the words have the correct properties according to the already addressed ones in Chapter 3. With this unseen evaluation set, we want to test the resulted mappings against the ability to perform on unseen, new, real world data. Additionally, we also want to see how well the mapping performs on data it might have seen due to the training—we refer to this data set as seen evaluation set. While unsupervised methods of course had not the chance of gaining knowledge about those

data points and maybe have similar scores as on unseen data points, the supervised and semi-supervised models should yield very good results on the seen data set. If we will see any different behavior, we might have a problem in training the models, so we see the seen evaluation set more as a sanity check and the unseen evaluation set as the data set that should provide results about the model's performance.

We explain the technical details of creating the evaluation data sets in Section 5.1.1 and Section 5.1.2.

5.1.1. Comparative Intrinsic Evaluation

For the unseen comparative intrinsic evaluation, we receive a list of 50 words from a domain expert, which should cover the whole engineering domain representatively. We preprocess the data analogously to Section 4.2.2 and ensure that none of the words is present in the training dictionary described in Section 4.2.1 nor in the training dictionary described in Section 4.1.1. Then we sample 20 evaluation words randomly. These 20 words will be the unseen evaluation words for the comparative intrinsic evaluation, the remaining 30 words will be available for the the unseen coherence evaluation data set. Now we ensure that those words are also present in the bilingual word embeddings yielded by MUSE and VecMap. To decide which word of the word pair is chosen for evaluation, we toss a fair coin to decide whether we choose the English or the German word. Finally, we explore for each of the 20 words the neighborhood in means of 1, 3, 5, and 10 for each yielded bilingual mapping and store the individual results into a comfortable comma-separated values (CSV) file such that the evaluation process is convenient for the evaluator.

Additionally—as already stated—we introduce also a seen comparative intrinsic evaluation. For this in particular, we choose 50 words from the learning dictionary described in Section 4.2.1 at random and ensure that we not reuse any randomly chosen word another time. Then we choose 20 words from the set of 50 words at random. Analogously to the unseen comparative intrinsic evaluation data set, we use the other 30 words for the seen coherence data set and proceed with choosing the word to be used for evaluation of each word pair with a fair coin, and explore the neighborhood of each word in each yielded bilingual mapping as stated for the unseen comparative intrinsic evaluation data set.

5.1.2. Coherence

As already described in Section 5.1.1, we take the remaining 30 word pairs for evaluating the coherence of the neighborhood of word embeddings into account. We also toss a fair coin to decide which word of the word pair should be finally chosen, and we

also ensure that all words are included in the vocabulary of every yielded bilingual word embedding. Now we have to find suitable intruder words. During the search of suitable words chosen randomly, we see that we have to limit the number of words in the vocabulary. We need to cut-off the vocabulary while searching for intruder words, because many words which are only rarely used are only abbreviations or combinations of letters and numbers. As a result, using such rare words would lead to even more trivial identification of the intruder word. So we search in each of the monolingual vocabularies for words randomly which have a word count $\pm 1,000$ the average word count in the $40,000$ most used words. While applying that, we see that the average word count within those $40,000$ words varies between both corpura. While the average count for German words is approx. $10,535.65$, the English average word count is approx. $23,731.14$. Then for each of the yielded models, we explore for each word of the list of unseen coherence words the neighborhood of 3, 5, and 10 and add a randomly chosen intruder word with the described properties to the neighborhood while the origin of the intruder word is also randomly decided based on a fair coin flip. Then we shuffle the neighborhood of each evaluation word and add the intruder word at a random position to the neighborhood. Finally, we store each evaluation set (for each neighborhood) and each mapping into a separate CSV file. To analyze the performance of the model, we store the index of each of the intruder words per evaluation file into a text file which is not shared with the evaluator.

We perform analogously the creation of the seen evaluation files as we explained for the unseen coherence evaluation.

5.2. Evaluation

We hand-over the evaluation files to a domain expert, who will evaluate the files described in Section 5.1.1 and Section 5.1.2. In more detail, for the comparative intrinsic evaluation he will add a 1 if the translation of the evaluation word is present in the set of the $1st, 3rd, 5th$, or $10th$ nearest neighbors. For the evaluation of the coherence of the mappings, he will indicate the index of the intruder word for each evaluation file and with that for every stated neighborhood and for each yielded mapping.

5.2.1. Comparative Intrinsic Evaluation

To analyze the results of the comparative intrinsic evaluation, we basically sum up the ones of each evaluation column assigned to a specific neighborhood and divide the sum by 20. This procedure yields an accuracy which describes how often we notice the direct translation of the evaluation word in each neighborhood. In other words, we show accuracy measurements which are $\in [0, 1]$.

Table 5.1.: Results of the seen comparative intrinsic evaluation. The highlighted row indicates the model which performed the best.

Mapping	1-NN	3-NN	5-NN	10-NN
NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model	0.00	0.00	0.00	0.00
MUSE_SUP_EN_DE_5_REF	0.25	0.30	0.35	0.50
MUSE_SUP_EN_DE_20_REF	0.25	0.30	0.35	0.50
MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS	0.00	0.00	0.00	0.00
VECMAP_SUP_EN_DE	0.60	0.75	0.80	0.85
VECMAP_SEMISUP_EN_DE	0.30	0.35	0.40	0.55
VECMAP_UNSUP_EN_DE	0.25	0.30	0.30	0.50

First, we discuss the seen comparative intrinsic evaluation depicted in Table 5.1 and compare it with the unseen comparative intrinsic evaluation available in Table 5.2. It is remarkable that the naïve bilingual model performs as bad as the unsupervised MUSE mapping. Both do not inherit a direct translation for any evaluation word in any examined neighborhood. For the supervised MUSE approach one can say that there are only little differences in the performance between applying 5 and 20 refinements. On the unseen evaluation data, the MUSE mapping with 20 refinements performs even worse than the model with 5 refinements. While the supervised VecMap mapping performs really good on seen data, we observe that the supervised VecMap mapping is not capable to abstract on unseen data. If one compares the supervised MUSE mappings with the semi-supervised or unsupervised VecMap mapping, one is able to conclude that they perform very similar on seen data. However, on unseen data the semi-supervised VecMap mapping outperforms every other method.

Table 5.2.: Results of the unseen comparative intrinsic evaluation. The highlighted row indicates the model which performed the best.

Mapping	1-NN	3-NN	5-NN	10-NN
NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model	0.00	0.00	0.00	0.00
MUSE_SUP_EN_DE_5_REF	0.40	0.65	0.75	0.90
MUSE_SUP_EN_DE_20_REF	0.40	0.60	0.70	0.85
MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS	0.00	0.00	0.00	0.00
VECMAP_SUP_EN_DE	0.00	0.00	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
VECMAP_SEMISUP_EN_DE	0.60	0.75	0.90	0.95
VECMAP_UNSUP_EN_DE	0.60	0.70	0.85	0.85

Table 5.3.: Results of the seen coherence evaluation. The highlighted row indicates the model which performed the best.

Mapping	3-NN	5-NN	10-NN
NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model	0.97	0.90	0.80
MUSE_SUP_EN_DE_5_REF	0.83	0.80	0.80
MUSE_SUP_EN_DE_20_REF	0.90	0.77	0.83
MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS	1.00	0.90	0.70
VECMAP_SUP_EN_DE	1.00	0.87	0.63
VECMAP_SEMISUP_EN_DE	1.00	0.87	0.77
VECMAP_UNSUP_EN_DE	0.90	0.87	0.77

5.2.2. Coherence

Now we elaborate on the evaluation regarding the coherence of the resulting bilingual word embeddings. We apply a similar scoring method as described in Section 5.2.1. We are counting the number of correctly classified intruder words for each examined neighborhood for each mapping and divide this number by 30.

The overall performance on this task is very similar across all models. We see that all yielded models are delivering higher scores on unseen data in comparison with the scores on seen data. We will try to explain this phenomenon in Section 5.3. On seen data, we observe that the unsupervised MUSE mapping slightly delivers the best scores. The same mapping is also very good on the unseen coherence evaluation data set. However, there are three models in total where the evaluator could identify all intruder words correctly added to the three and five nearest neighbors namely the

Table 5.4.: Results of the unseen coherence evaluation. The highlighted rows indicate the models which performed the best.

Mapping	3-NN	5-NN	10-NN
NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model	1.00	1.00	0.97
MUSE_SUP_EN_DE_5_REF	0.97	1.00	0.93
MUSE_SUP_EN_DE_20_REF	1.00	1.00	0.87
MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS	1.00	0.97	0.93
VECMAP_SUP_EN_DE	1.00	0.93	0.83
VECMAP_SEMISUP_EN_DE	1.00	1.00	0.93
VECMAP_UNSUP_EN_DE	0.97	1.00	0.90

naïve bilingual word2vec model, supervised MUSE mapping (with 20 refinements), and the semi-supervised VecMap mapping.

5.3. Discussion

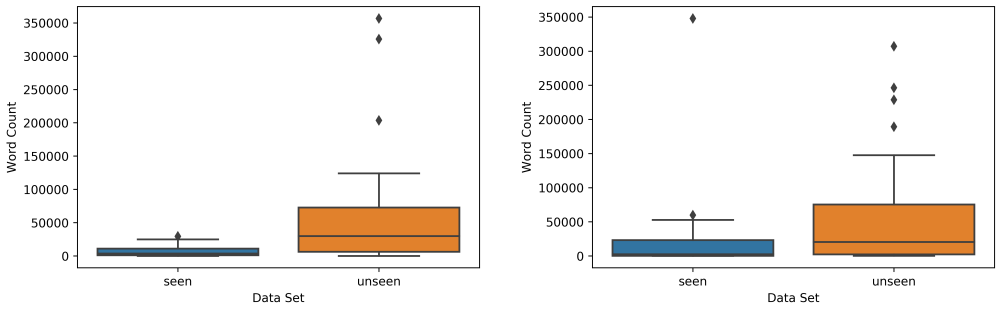
Having reviewed the findings of Section 5.2, we now elaborate on reasonable inferences.

5.3.1. Comparative Intrinsic Evaluation

The most astonishing result of the comparative intrinsic evaluation—which is described thoroughly in Section 5.2.1—is that the supervised VecMap mapping is heavily overfitted to the training data, while the semi-supervised approach is able to adapt on unseen data, yielding the best results on the unseen data set. It is odd that the model which had the chance to create in some sense knowledge about certain word embeddings performs better on unseen data.

An explanation for this observation might be to consider the number of evaluation words. Maybe the changes in the indicated accuracy is just noise, but that would mean that e.g. for the semi-supervised VecMap mapping, we would have a noise of 40% which is even with a number of evaluation words of 20 rather unlikely. As a result, we investigate the average word count for the unseen and the seen evaluation data set. We report that the average word count in the seen comparative intrinsic evaluation data set is $8,151.5$ while the average word count in the unseen data set is $72,531.05$—roughly $8.9\times$ higher than the average word count of the seen data set (see Figure 5.1 for an illustration and for further information). As a result, the fact that the mappings perform worse on the unseen data set than on the seen data set can origin by this phenomenon. Indeed, if one analyzes the amount of occurrences of all direct translations across the mappings, one can observe that the direct translation of high-frequent words are by far more present as the nearest neighbor and therefore also present in all other neighborhoods (see Figure 5.2 for details). In future work, we will investigate this further. It would be very interesting to see that the difference in accuracy of 40% of the semi-supervised VecMap mapping would vanish, if we have the same or rather similar word count distribution in both evaluation data sets. We elaborate on this in particular in Section 6.3.

However, one can also see this result in particular with more optimism. The semi-supervised VecMap model is able to find the direct translation in 40% of the cases within the set of the five nearest neighbors of words which have below average word count. Remark that the average word count in the English vocabulary is approx. $23,731.14$ and in the German vocabulary approx. $10,535.65$. But this comparison is problematic in the sense that we can not extrapolate the knowledge of the fact that the models perform



(a) Comparative Intrinsic Evaluation (b) Coherence

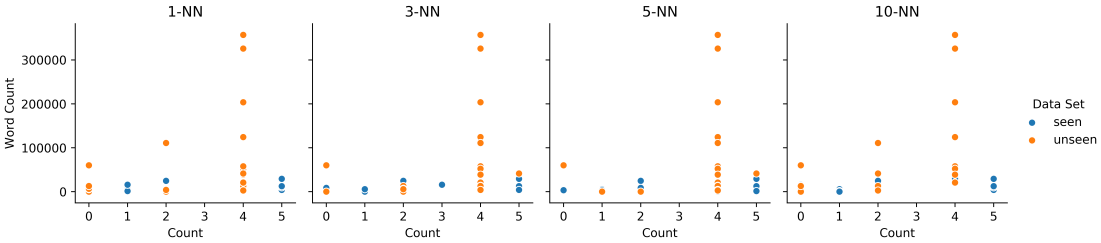
Figure 5.1.: Comparison of the word count in the data sets used for both evaluation methods.

well on the comparative intrinsic evaluation task applied on words with under-average word count. We will also propose to follow this first impression in future experiments described in Section 6.3.

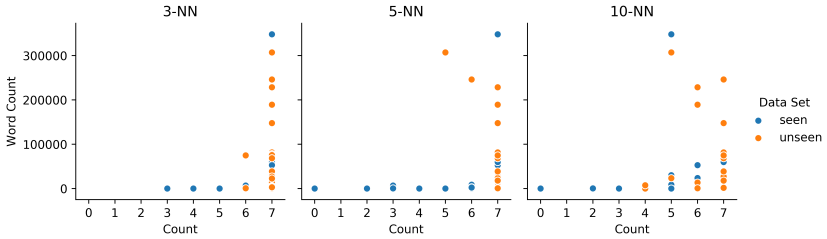
5.3.2. Coherence

We also see the same behavior analogously in evaluating the coherence of the word embeddings. While this effect is rather small in this context, it is also the case that the bilingual word embeddings yield higher accuracies on the unseen evaluation data set. So we investigate the average word count for both evaluation data sets and see that the average word count of the seen evaluation data set is $34,045.6$ while the average word count of the unseen evaluation data set is $88,131.0$. That means that the average word count of the unseen evaluation data set is roughly $2.6\times$ higher (see also Figure 5.1). So if our assumptions of the last paragraph are true and it turns out that this difference is originated by the word count, it would imply that—for the coherence evaluation task—a $2.6\times$ higher word count would make a difference of at max. 10% (observed while evaluating the naïve bilingual model).

Additionally, we also investigate the relationship between the word count of an evaluation word and observing a correctly classified intruder word for each neighborhood across all evaluated mappings (see Figure 5.2). We observe that it indeed holds, that it is easier for the evaluator to spot intruder words in the neighborhood of words with rather high word count and with that the evaluator is in the position to spot the intruders better on the unseen data set.



(a) The count represents the number of times a direct translation of an evaluation word with an assigned word count is present in the respective neighborhood across all evaluated models. (Comparative Intrinsic Evaluation)



(b) The count represents the number of times an intruder word is correctly classified in the respective neighborhood of an evaluation word with an assigned word count across all evaluated models. (Coherence)

Figure 5.2.: Illustration of the relationship between correct classifications and the word count of the evaluation words present in both evaluations.

5.3.3. Evaluation Data

Moreover, one has to mention that the number of evaluation words in each of the evaluation tasks is rather limited. The number of evaluation words is restricted by the amount of available time and particularly the number of domain experts who are in charge of processing the evaluation. In literature, one utilizes and averages the outcome of e.g. seven evaluators to evaluate the coherence of a monolingual word embedding [Sch+15], which is also not that many evaluators but surely better than relying on one evaluator.

The pitfall is that we can not easily use commercial providers of turking services like Amazon Mechanical Turk [Ama] as we are dependent on the domain-specific knowledge of engineers due to the fact that deciding which word is appropriate or the intruder word can be very difficult in this context. However, increasing the number of evaluation words in total would surely lead to more reliable and trustworthy results. We describe further possibilities regarding this in Section 6.2.

5.3.4. Techniques

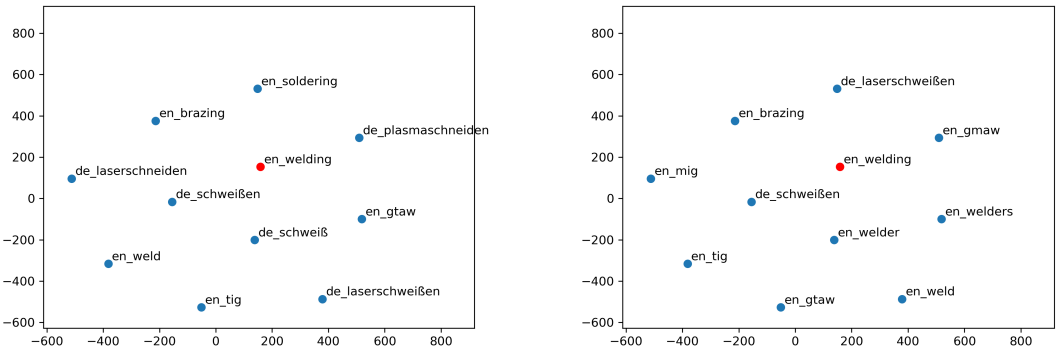
Overall, we observe that the semi-supervised VecMap mapping yields the best results in our evaluation. The second best model is the bilingual word embedding mapped by supervised MUSE (with 20 refinements). As a result, we can recommend applying further focus on those two methods in particular.

We have mentioned in Section 4.1.2, that we want to be capable to also have really rare words included in the bilingual word embedding. However, one limitation for real-world applications of both models is that both models cut-off the vocabulary by default. This means that VecMap uses only the top 20,000 most used words of each of the monolingual embeddings (see [Art19]), while MUSE uses the most used 200,000 words (see [Fac19]). Comparing VecMap with MUSE is rather difficult—as they tackle the problem of mapping two word embeddings into a common space very different (we elaborated on this in Section 2.2.2 and Section 2.2.3). But we recall that MUSE is heavily relying on external evaluation data while training the MUSE model, while VecMap operates independently of such external resources while training. Exactly this fact makes VecMap more flexible and therefore also more appealing for answering our second research question, which we like to address in Section 6.1.

5.4. Exemplary Outputs

After evaluating the results of the different mappings, we look at some outputs of the winning one—the semi-supervised VecMap, but also compare those with the supervised MUSE mapping (with 20 refinements). For this very reason, we display the t-SNE projections into a two-dimensional space of two selected query words—one of them originated from the German corpus and one English word accordingly. We display the outputs of all evaluated models additionally in Appendix A.3.1 and add the visualizations of four supplementary words to give the reader a bigger picture of the differences in the mappings. On some of the examples, both models perform equally poor, on other samples one would subjectively prefer the semi-supervised VecMap mapping over the supervised MUSE mapping (with 20 refinements).

First, we would like to revisit the example from Section 1.1—we explore the neighborhood of the word “en_welding” in the two most promising bilingual word embeddings, displayed in Figure 5.3. We can observe that both mappings are capable to yield useful bilingual results, which confirms our findings. We can also see that the semi-supervised VecMap mapping inherits more German words (five out of ten) in the neighborhood of the query word in comparison with the supervised MUSE mapping (two out of ten). Overall, in this comparison we can say that the semi-supervised VecMap mapping can not only yield the correct translation in the neighborhood of the query word,



(a) Semi-supervised VecMap.

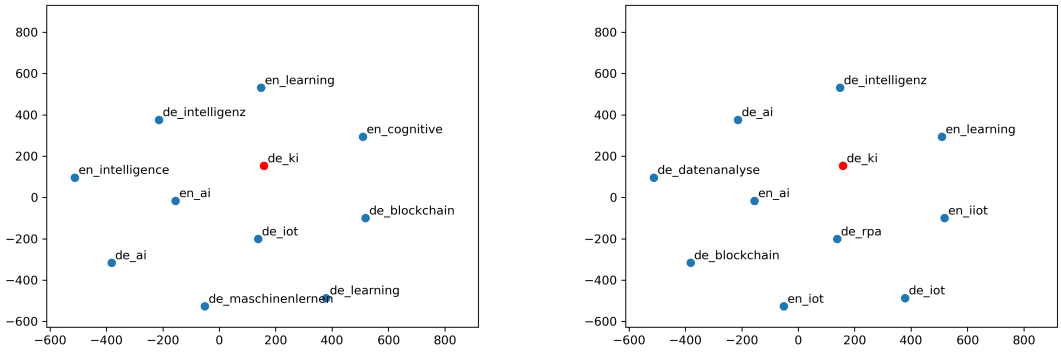
(b) Supervised MUSE (20 refinements).

Figure 5.3.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word "en_welding" (marked red) into two dimensional space.

but it is also capable to yield related methods to welding in both languages. The supervised MUSE approach yields also related methods in both languages, but—as mentioned—the neighborhood of the English word lacks in German words.

Furthermore, we also want to investigate the mapping of the query word "de_ki", which is the direct translation of the English abbreviation of artificial intelligence (AI). We already used the English abbreviation in Figure 4.3 to visualize the changes in training the word embedding models. For this example, the proportions of English and German words present in the neighborhood is equal for both mappings—four out of 10 words in the surroundings of the German query word are originated from the English corpus. Both mappings are on a subjective level appropriate and well-suited for our use case.

As a conclusion, we can endorse our findings from the experiments that both techniques yield an useful bilingual word embedding—while the supervised VecMap performs slightly better in total than the supervised MUSE approach (with 20 refinements)—which can be applied to help engineers to discover new techniques in both languages. From the evaluation, we see that the semi-supervised VecMap mapping scores a bit better than the supervised MUSE approach (with 20 refinements), and we also observe that the semi-supervised VecMap yields subjectively better results than the supervised MUSE approach (with 20 refinements) as we examining the neighborhood of additional example query words.



(a) Semi-supervised VecMap.

(b) Supervised MUSE (20 refinements).

Figure 5.4.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_ki” (marked red) into two dimensional space.

6. Future Work

First, we would like to address the second research question and outline possible approaches that we consider promising based on our experience. Additionally, we identify certain interesting challenges that can be explored in more detail in the future.

6.1. Bilingual Document Embedding

After conducting and evaluating our experiments to obtain bilingual engineering-specific word embeddings, we have seen that the supervised VecMap mappings are very promising for generating bilingual document embeddings. This is due to the fact that the supervised VecMap technique is independent of any external sources and also does not apply learning dictionary refinements. Thus, an easily imaginable scenario is to embed documents via e.g. doc2vec [LM14] (implemented in the gensim library [Reh21a]) and then to built a learning dictionary similarly to what we hand-crafted in Section 4.1.1. Of course this approach needs to have one-to-one translations of articles in the data set, such that the learning dictionary contains most similar document pairs. However, one could also translate texts via external translation service providers, which offer an application programming interface (API). An example for such a service is deepl [Dee21]. This would be one classical approach and one application of the findings of this guided research.

Additionally, one could exploit cutting-edge language models which are extremely powerful and capable of understanding multilingual documents right from the start without the need of mapping two monolingual embeddings into a common space. We are not able to apply such context-aware language model to the word-level use case as we want unique word vectors for each word independently of its context, but for document-level embeddings cutting-edge transformer-based models are perfectly suited. One example which is especially built for semantically aware document embeddings are pre-trained multilingual variants [RG20] of sentence-BERT models [RG19]. To obtain semantically aware multilingual document embeddings, the pipeline would be rather simple: one would have to apply one of the available pre-trained multilingual sentence-BERT models to the documents and then could search via cosine similarity in the embedding space with tailored lookup engines like Faiss [JDJ17] (available on GitHub [Fac21]). Such a tailored lookup engine will be necessary due to the fact that

the embeddings yielded by sentence-BERT models create document embeddings in higher dimensions in comparison with doc2vec e.g. 768 instead of 100 dimensions. A downside of this approach is the computationally expensive calculation of the document embeddings. Even though one just applies the model on unseen data, BERT models with billions of parameters are not easily to handle. Furthermore, one has to admit that those models have certain token limits e.g. 512 which has to be considered. Usually such a token is not a word but rather a word piece. So in the end one can only consider approximately 300 words from a document for the final context-aware document embedding. However, there are also different summarization techniques available which are able to extract only the important part of an article (e.g. extractive summarization [Mil19] or abstractive summarization [Zha+19]).

6.2. Evaluation

A clear restriction of this work is the available evaluation workforce. We can only take one domain expert into account and as we need domain-specific knowledge to gain a reliable evaluation, platforms like Amazon Mechanical Turk [Ama] are not applicable. What one can do in future work is to pay e.g. electrical engineering students for evaluation via an online survey format. Downsides are of course that one has to create the online survey and acquire external funding to pay the students, but one would have the chance to gain evaluation data of hundreds of domain experts and therefore create a more reliable evaluation.

6.3. Evaluation Data Set

As already mentioned in Section 5.3, for future experiments one has to conduct the evaluation with similar word counts across the evaluation data sets to get a more trustworthy result and to better compare the different evaluation data sets. Because it seems for now that the word count drives the probability to observe direct translations within the neighborhood of the bilingual word embeddings. It would be also interesting to have four instead of two evaluation data sets per evaluation method. One could evaluate the performance on under-average words and on over-average words regarding the word count. This would then also increase the time needed to evaluate the models, but we are of the opinion that such a comparison would be interesting for having a better overview of the different mapping techniques.

7. Summary

In this guided research, we conducted experiments in order to achieve bilingual word embeddings which include semantic linking capabilities across languages in Chapter 4. We have taken two different deep learning based word embedding techniques into account namely fastText and word2vec.

During initial experiments, we have noticed that fastText embeddings do not fit our expectations about similar words to a pivot word. Due to the nature of fastText, the embeddings are driven by the word parts and therefore are not ideal for our use case. We use three different mapping techniques (Transvec, VecMap, and MUSE) to create multilingual word embeddings in a common space. During first experiments, we saw that Transvec is not capable to yield meaningful results at all, so we have dropped this method and did not conduct further experiments with Transvec. Throughout the first experiments, we saw the potential of MUSE for creating bilingual word embeddings while we applied MUSE to fastText embeddings and found a way to apply MUSE also for word2vec embeddings.

With the findings of the first experiments, we conducted final experiments exploiting over 3,000,000 articles and evaluated—as well as—discussed the results. Thus, we can recommend the semi-supervised VecMap mapping and also the supervised MUSE mapping (with 20 refinements) for creating bilingual word embeddings.

In the end, we also gave an overview of possible approaches to introduce bilingual document embeddings which are useful for finding semantically similar articles based on a query article.

A. Appendix

A.1. External Python Libraries

Table A.1.: List of used external Python libraries.

Package	Version
absl-py	0.12.0
adal	1.2.7
aiohttp	3.7.4
aiohttp-cors	0.7.0
aioredis	1.3.1
ansiwrap	0.8.4
anyio	2.2.0
appdirs	1.4.4
argon2-cffi	20.1.0
arrow	1.1.0
asn1crypto	1.4.0
astunparse	1.6.3
async-generator	1.10
async-timeout	3.0.1
attrs	20.3.0
backcall	0.2.0
backports.functools-lru-cache	1.6.4
beautifulsoup4	4.9.3
binaryornot	0.4.4
black	21.4b2
bleach	3.3.0
blessings	1.7
blinker	1.4
brotlipy	0.7.0

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
cachetools	4.2.2
caip-notebooks-serverextension	1.0.0
certifi	2020.12.5
cffi	1.14.5
chardet	4.0.0
click	7.1.2
cloudpickle	1.6.0
colorama	0.4.4
conda	4.9.2
conda-package-handling	1.7.3
confuse	1.4.0
cookiecutter	1.7.2
cryptography	3.4.7
cycler	0.10.0
Cython	0.29.23
decorator	5.0.7
defusedxml	0.7.1
deprecation	2.1.0
docker	5.0.0
docker-pycreds	0.4.0
entrypoints	0.3
faiss	1.7.0
fasttext	0.9.2
filelock	3.0.12
flatbuffers	1.12
fsspec	2021.4.0
gast	0.3.3
gcsfs	2021.4.0
gensim	4.0.1
gitdb	4.0.7
GitPython	3.1.15
google-api-core	1.26.3
google-api-python-client	2.3.0
google-auth	1.30.0
google-auth-httplib2	0.1.0

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
google-auth-oauthlib	0.4.4
google-cloud-bigquery	2.15.0
google-cloud-bigquery-storage	2.4.0
google-cloud-bigtable	2.2.0
google-cloud-core	1.6.0
google-cloud-dataproc	2.3.1
google-cloud-datastore	2.1.2
google-cloud-firestore	2.1.1
google-cloud-kms	2.2.0
google-cloud-language	2.0.0
google-cloud-logging	2.3.1
google-cloud-monitoring	2.2.1
google-cloud-pubsub	1.7.0
google-cloud-scheduler	2.2.0
google-cloud-spanner	3.4.0
google-cloud-speech	2.3.0
google-cloud-storage	1.38.0
google-cloud-tasks	2.2.0
google-cloud-translate	3.1.0
google-cloud-videointelligence	2.1.0
google-cloud-vision	2.3.1
google-crc32c	1.1.2
google-pasta	0.2.0
google-resumable-media	1.2.0
googleapis-common-protos	1.53.0
gpustat	0.6.0
greenlet	1.0.0
grpc-google-iam-v1	0.12.3
grpcio	1.32.0
grpcio-gcp	0.2.2
h5py	2.10.0
hiredis	2.0.0
htmlmin	0.1.12
httplib2	0.19.1
idna	2.10

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
ImageHash	4.2.0
importlib-metadata	4.0.1
ipykernel	5.5.3
ipython	7.23.0
ipython-genutils	0.2.0
ipython-sql	0.3.9
ipywidgets	7.6.3
jedi	0.18.0
Jinja2	2.11.3
jinja2-time	0.2.0
joblib	1.0.1
json5	0.9.5
jsonschema	3.2.0
jupyter-client	6.1.12
jupyter-core	4.7.1
jupyter-http-over-ws	0.0.8
jupyter-packaging	0.9.2
jupyter-server	1.6.4
jupyter-server-mathjax	0.2.2
jupyterlab	1.2.16
jupyterlab-git	0.11.0
jupyterlab-pygments	0.1.2
jupyterlab-server	1.2.0
jupyterlab-widgets	1.0.0
Keras-Preprocessing	1.1.2
kiwisolver	1.3.1
kubernetes	12.0.1
libcst	0.3.18
llvmlite	0.36.0
Markdown	3.3.4
MarkupSafe	1.1.1
matplotlib	3.4.1
matplotlib-inline	0.1.2
missingno	0.4.2
mistune	0.8.4

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
msgpack	1.0.2
multidict	5.1.0
mypy-extensions	0.4.3
nb-conda	2.2.1
nb-conda-kernels	2.3.1
nbclient	0.5.3
nbconvert	6.0.7
nbdime	3.0.0
nbformat	5.1.3
nest-asyncio	1.5.1
networkx	2.5
nltk	3.6.2
notebook	6.3.0
notebook-executor	0.2
numba	0.53.1
numpy	1.19.5
nvidia-ml-py3	7.352.0
oauthlib	3.0.1
olefile	0.46
opencensus	0.7.12
opencensus-context	0.1.2
opt-einsum	3.3.0
packaging	20.9
pandas	1.2.4
pandas-profiling	2.11.0
pandocfilters	1.4.2
papermill	2.3.3
parso	0.8.2
pathspec	0.8.1
patsy	0.5.1
pexpect	4.8.0
phik	0.11.2
pickleshare	0.7.5
Pillow	8.1.2
pip	21.1.1

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
POT	0.7.0
poyo	0.5.0
prettytable	2.1.0
prometheus-client	0.10.1
prompt-toolkit	3.0.18
proto-plus	1.18.1
protobuf	3.15.8
psutil	5.8.0
ptyprocess	0.7.0
py-spy	0.3.6
pyarrow	4.0.0
pyasn1	0.4.8
pyasn1-modules	0.2.7
pybind11	2.6.2
pycosat	0.6.3
pycparser	2.20
Pygments	2.8.1
PyJWT	2.1.0
pyOpenSSL	20.0.1
pyparsing	2.4.7
pyrsistent	0.17.3
PySocks	1.7.1
python-dateutil	2.8.1
python-Levenshtein	0.12.2
python-slugify	5.0.0
pytz	2021.1
PyWavelets	1.1.1
PyYAML	5.4.1
pyzmq	22.0.3
ray	1.3.0
redis	3.5.3
regex	2021.4.4
requests	2.25.1
requests-oauthlib	1.3.0
retrying	1.3.3

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
rsa	4.7.2
ruamel-yaml-conda	0.15.80
scikit-learn	0.24.2
scipy	1.6.3
seaborn	0.11.1
Send2Trash	1.5.0
setuptools	49.6.0.post20210108
simplejson	3.17.2
six	1.15.0
smart-open	5.0.0
smmap	3.0.5
sniffio	1.2.0
soupsieve	2.2.1
SQLAlchemy	1.4.13
sqlparse	0.4.1
statsmodels	0.12.2
tangled-up-in-unicode	0.0.7
tenacity	7.0.0
tensorboard	2.5.0
tensorboard-data-server	0.6.1
tensorboard-plugin-wit	1.8.0
tensorflow	2.4.1
tensorflow-estimator	2.4.0
termcolor	1.1.0
terminado	0.9.4
testpath	0.4.4
text-unidecode	1.3
textwrap3	0.9.2
threadpoolctl	2.1.0
toml	0.10.2
tomlkit	0.7.0
torch	1.8.1
tornado	6.1
tqdm	4.60.0
traitlets	5.0.5

Continued on next page.

Table A.1 Continued from previous page.

Package	Version
transvec	0.1.0
typed-ast	1.4.3
typing-extensions	3.7.4.3
typing-inspect	0.6.0
ujson	4.0.2
Unidecode	1.2.0
uritemplate	3.0.1
urllib3	1.26.4
visions	0.6.0
wcwidth	0.2.5
webencodings	0.5.1
websocket-client	0.57.0
Werkzeug	1.0.1
wheel	0.36.2
whichcraft	0.6.1
widgetsnbextension	3.5.1
wordcloud	1.8.1
wrapt	1.12.1
yarl	1.6.3
zipp	3.4.1

A.2. Final Experiments

A.2.1. Word Embeddings

word2vec

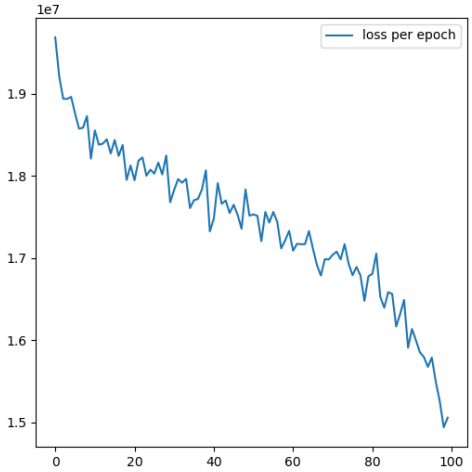
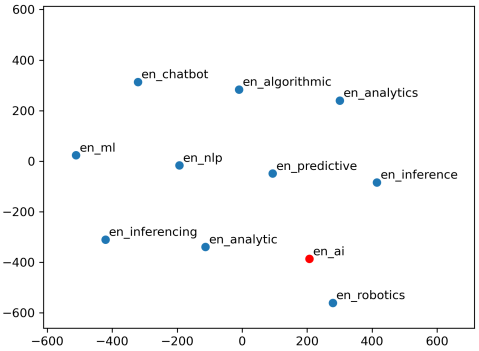
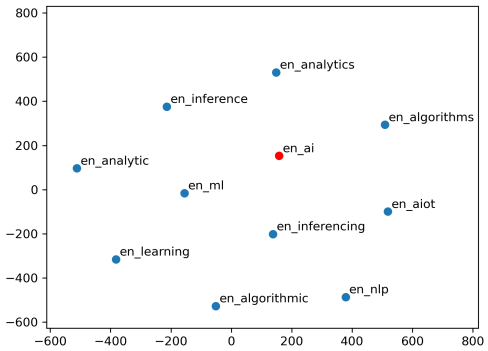


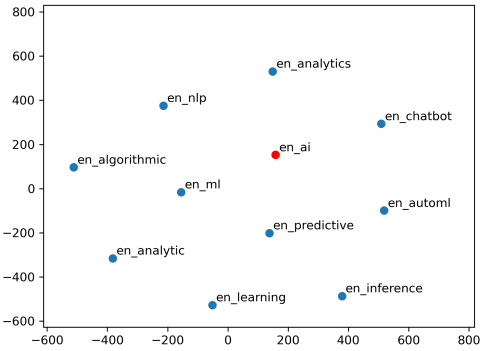
Figure A.1.: Loss of the word2vec model trained with the English corpus is depicted. The x-axis indicates the number of epochs and the y-axis is the loss per epoch. The model is trained on the full data set. Epoch 0 is indeed epoch 400 and epoch 100 in the diagram is the $500th$ epoch.



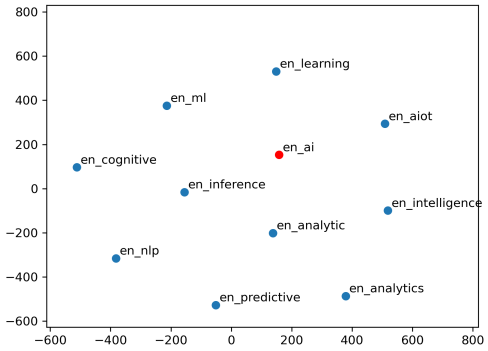
(a) English word2vec model after 100 epochs.



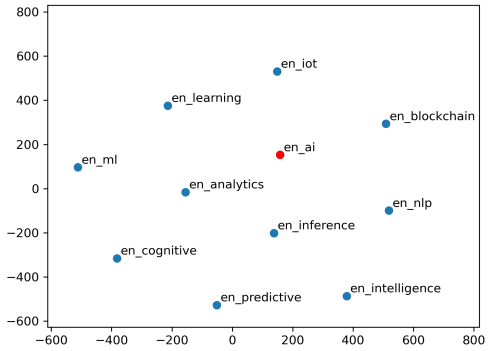
(b) English word2vec model after 200 epochs.



(c) English word2vec model after 300 epochs.



(d) English word2vec model after 400 epochs.



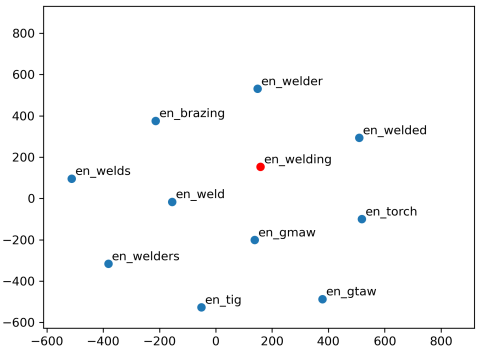
(e) English word2vec model after 500 epochs.

Figure A.2.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “AI” (marked red) into two dimensional space.

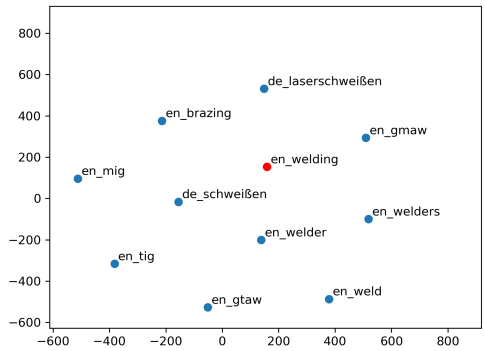
This page intentionally left blank.

A.3. Evaluation & Results

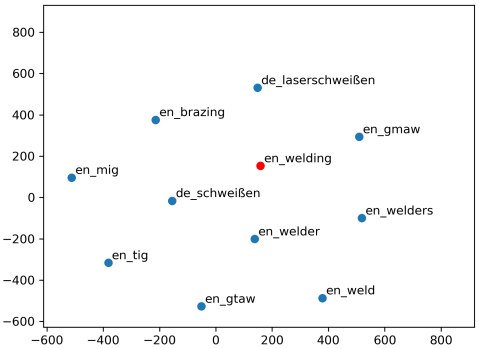
A.3.1. Exemplary Outputs



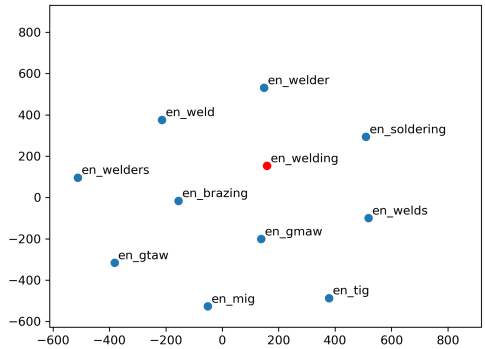
(a) NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model



(b) MUSE_SUP_EN_DE_5_REF

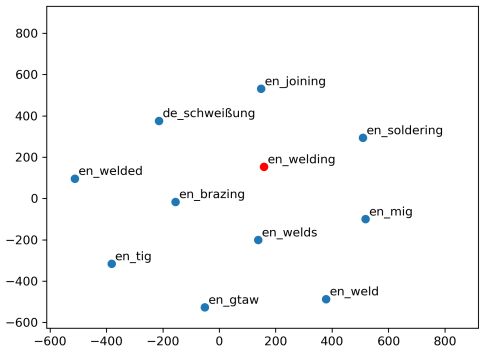


(c) MUSE_SUP_EN_DE_20_REF

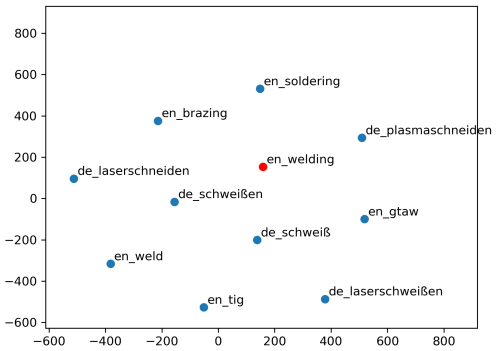


(d) MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS

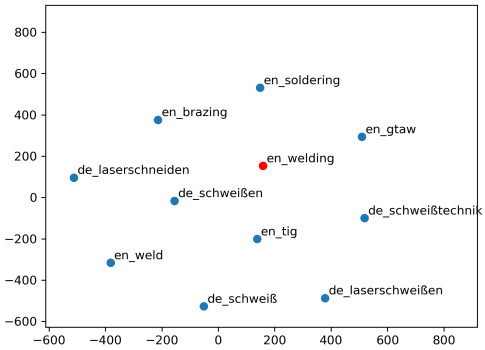
Figure A.3.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “en_welding” (marked red) into two dimensional space.



(a) VECMAP_SUP_EN_DE

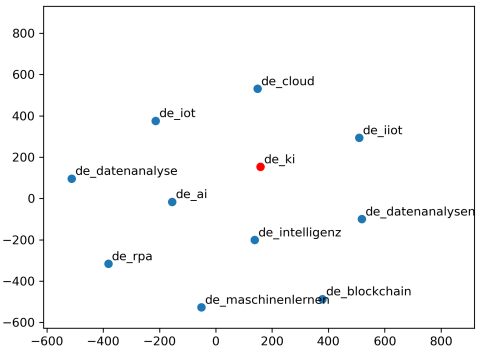


(b) VECMAP_SEMISUP_EN_DE

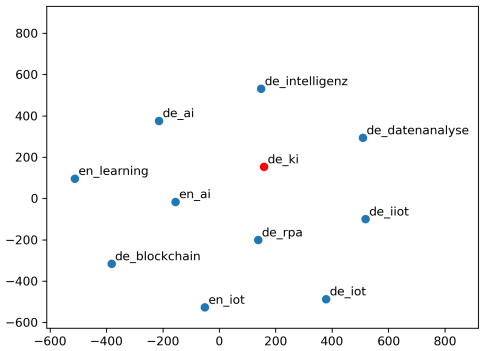


(c) VECMAP_UNSUP_EN_DE

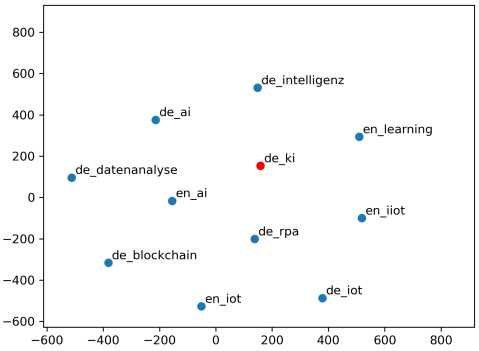
Figure A.3.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “en_welding” (marked red) into two dimensional space. (cont’d)



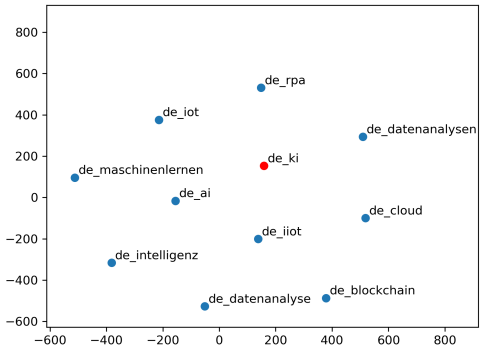
(d) NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model



(e) MUSE_SUP_EN_DE_5_REF

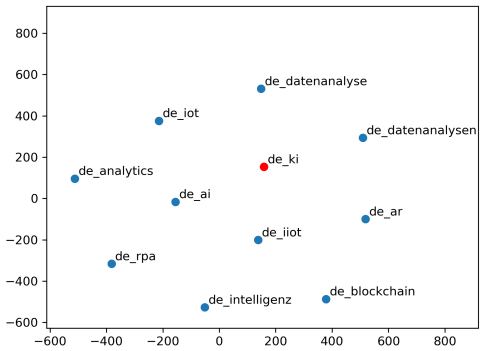


(f) MUSE_SUP_EN_DE_20_REF

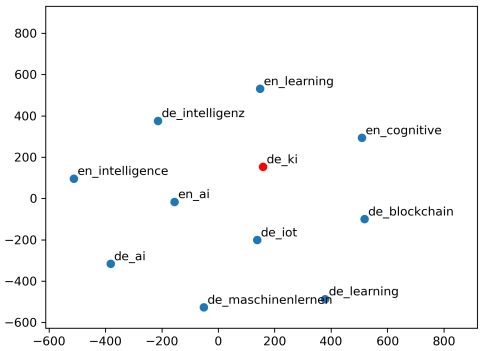


(g) MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS

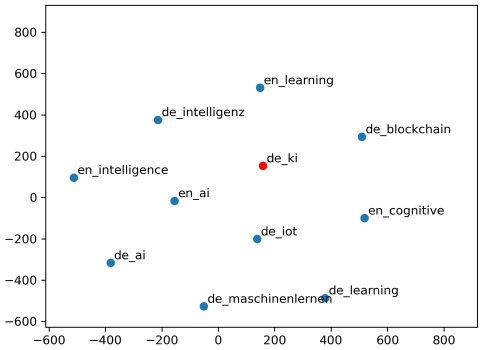
Figure A.4.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_ki” (marked red) into two dimensional space.



(a) VECMAP_SUP_EN_DE

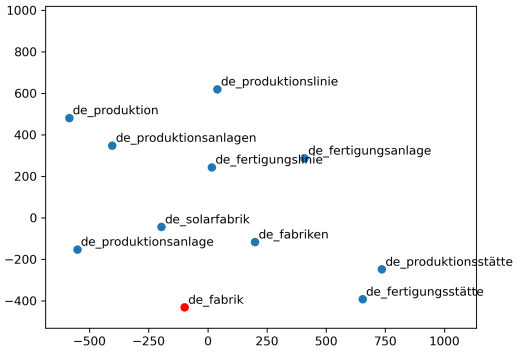


(b) VECMAP_SEMISUP_EN_DE

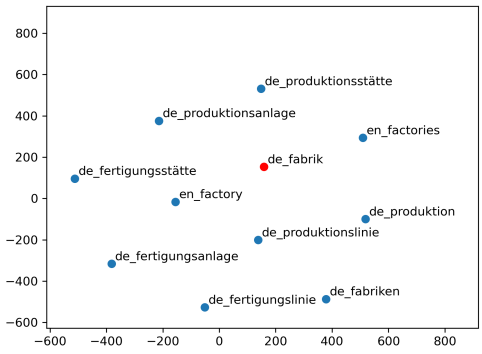


(c) VECMAP_UNSUP_EN_DE

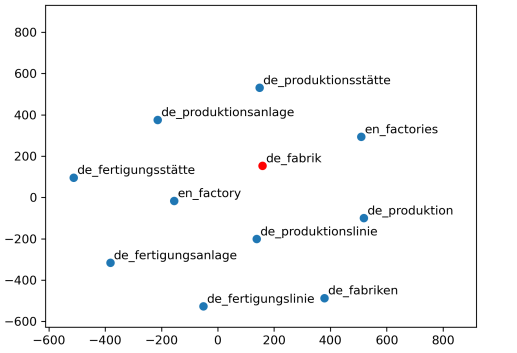
Figure A.4.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_ki” (marked red) into two dimensional space. (cont’d)



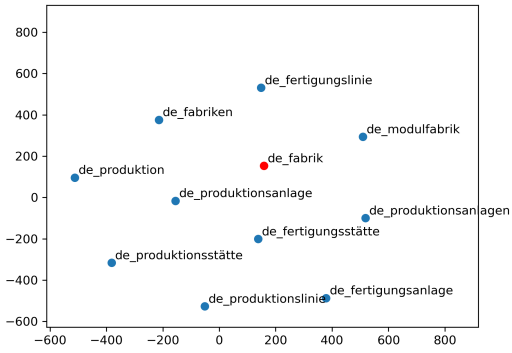
(d) NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model



(e) MUSE_SUP_EN_DE_5_REF

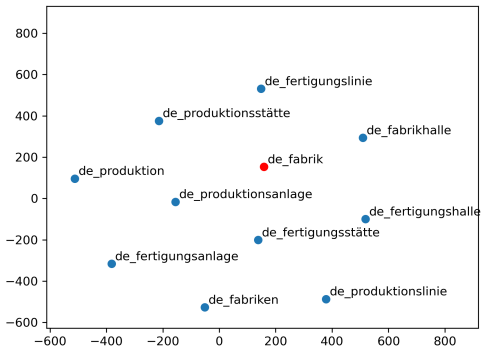


(f) MUSE_SUP_EN_DE_20_REF

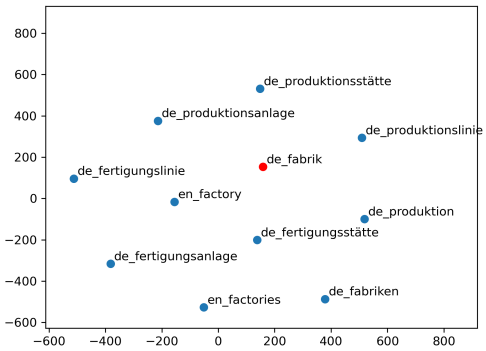


(g) MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS

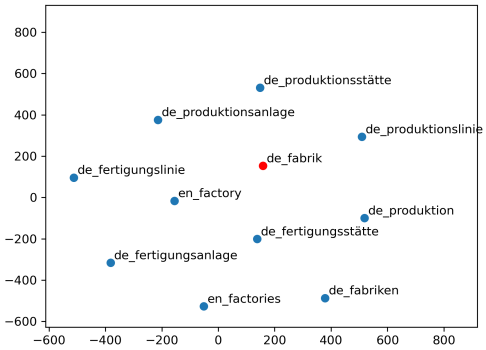
Figure A.5.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_fabrik” (marked red) into two dimensional space.



(a) VECMAP_SUP_EN_DE

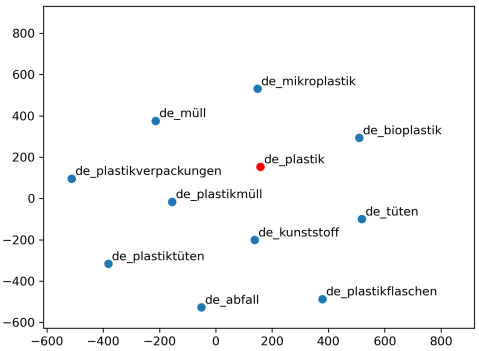


(b) VECMAP_SEMISUP_EN_DE

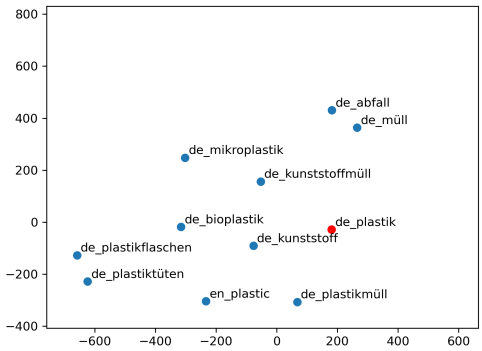


(c) VECMAP_UNSUP_EN_DE

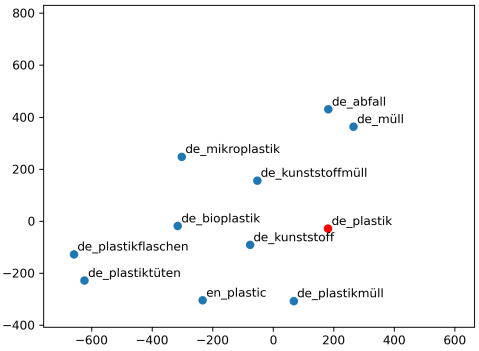
Figure A.5.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_fabrik” (marked red) into two dimensional space. (cont’d)



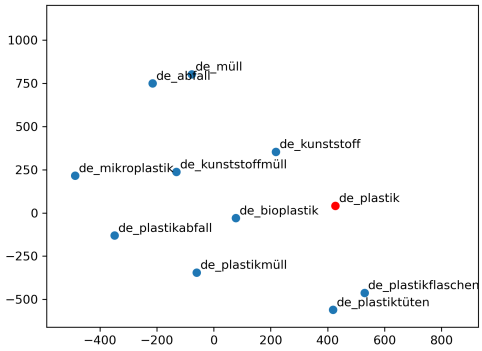
(d) NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model



(e) MUSE_SUP_EN_DE_5_REF

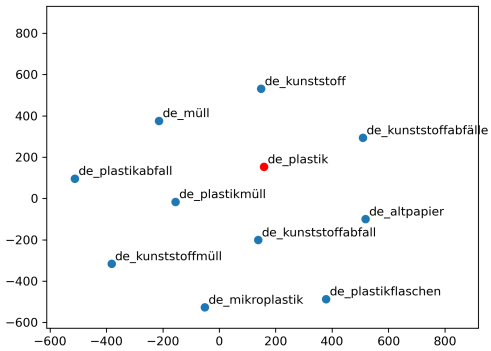


(f) MUSE_SUP_EN_DE_20_REF

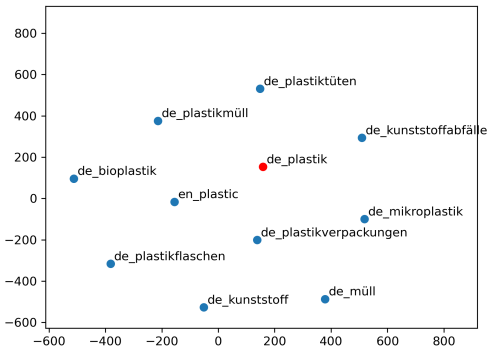


(g) MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS

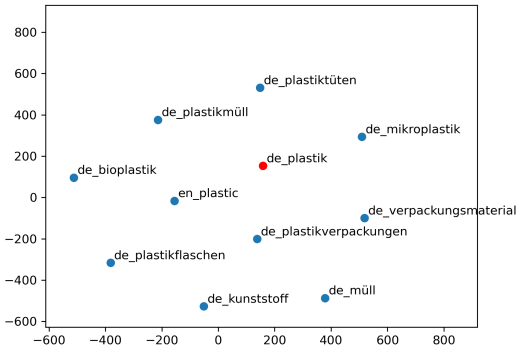
Figure A.6.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_plastik” (marked red) into two dimensional space.



(a) VECMAP_SUP_EN_DE

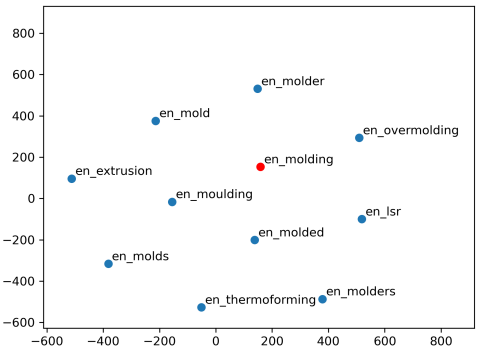


(b) VECMAP_SEMISUP_EN_DE

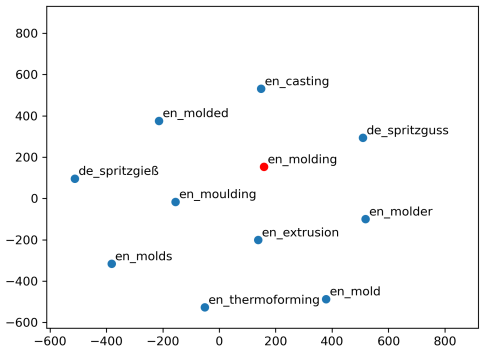


(c) VECMAP_UNSUP_EN_DE

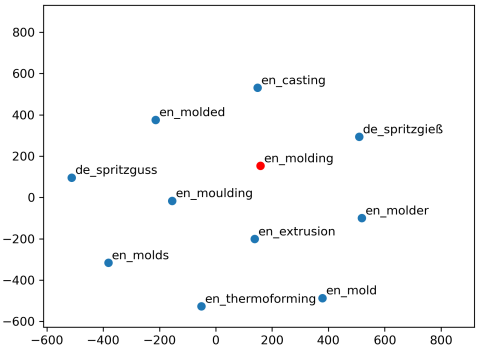
Figure A.6.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “de_plastik” (marked red) into two dimensional space. (cont’d)



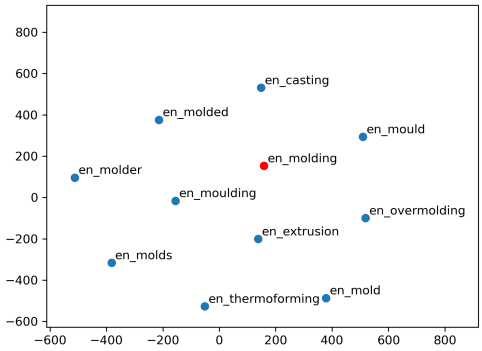
(d) NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model



(e) MUSE_SUP_EN_DE_5_REF

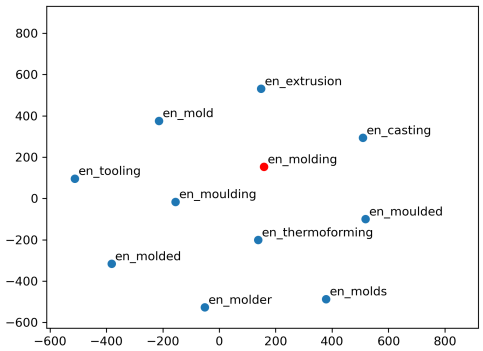


(f) MUSE_SUP_EN_DE_20_REF

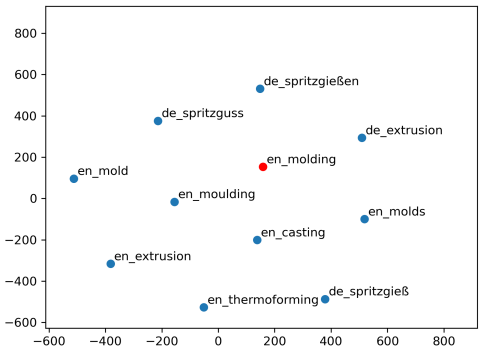


(g) MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS

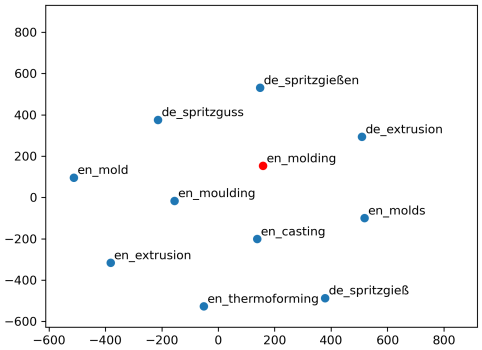
Figure A.7.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “en_molding” (marked red) into two dimensional space.



(a) VECMAP_SUP_EN_DE

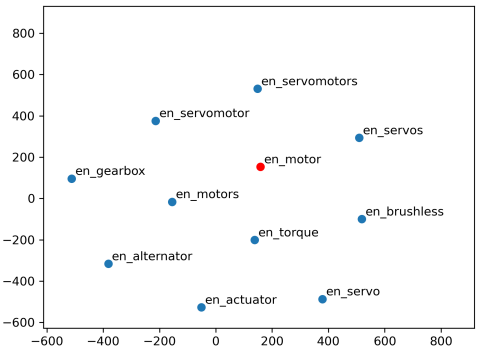


(b) VECMAP_SEMISUP_EN_DE

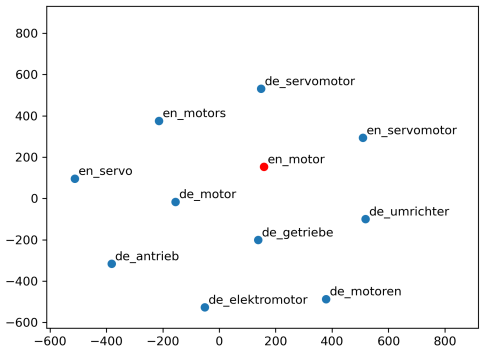


(c) VECMAP_UNSUP_EN_DE

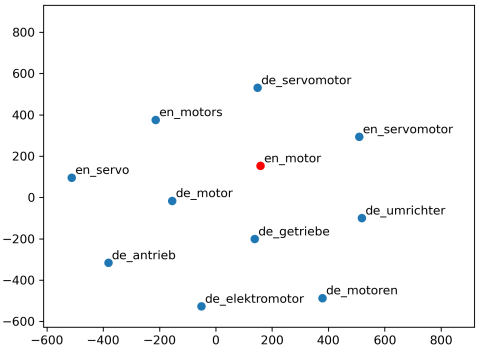
Figure A.7.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “en_molding” (marked red) into two dimensional space. (cont’d)



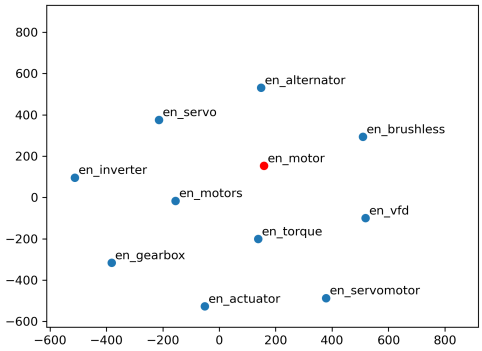
(d) NAIVE_w2v-eng-ger-full-500-epochs-300-dim-model



(e) MUSE_SUP_EN_DE_5_REF

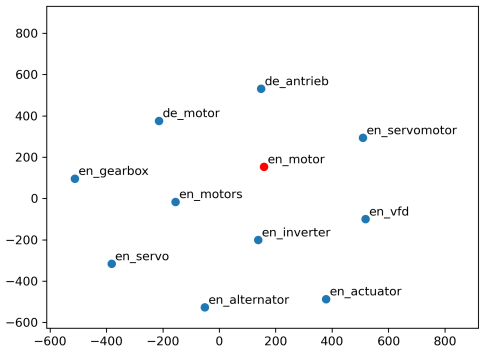


(f) MUSE_SUP_EN_DE_20_REF

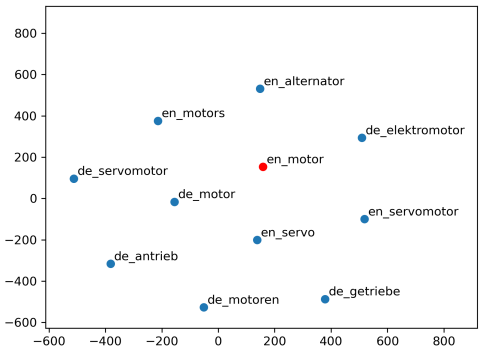


(g) MUSE_UNSUP_EN_DE_5_REF_5_EPOCHS

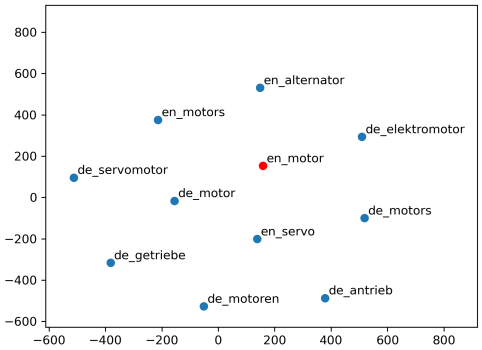
Figure A.8.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “en_motor” (marked red) into two dimensional space.



(a) VECMAP_SUP_EN_DE



(b) VECMAP_SEMISUP_EN_DE



(c) VECMAP_UNSUP_EN_DE

Figure A.8.: t-SNE projections of the embedded ten nearest neighbors (marked blue) of the word “en_motor” (marked red) into two dimensional space. (cont’d)

List of Figures

List of Tables

Bibliography

[ALA18a] M. Artetxe, G. Labaka, and E. Agirre. "A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 789–798.

[ALA18b] M. Artetxe, G. Labaka, and E. Agirre. "Generalizing and improving bilingual word embedding mappings with a multi-step framework of linear transformations." In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 2018, pp. 5012–5019.

[Ama] Amazon Mechanical Turk, Inc. *Amazon Mechanical Turk*. `https://www.mturk.com`. Online. Accessed on 9/11/2021.

[Art19] M. Artetxe. *VecMap*. `https://github.com/artetxem/vecmap`. Online. Accessed on 9/2/2021. 2019.

[Bak18] A. Bakarov. "A survey of word embeddings evaluation methods." In: *arXiv preprint arXiv:1801.09536* (2018).

[big20] big-o. *Transvec*. `https://github.com/big-o/transvec`. Online. Accessed on 9/1/2021. 2020.

[Boj+17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching Word Vectors with Subword Information." In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146. ISSN: 2307-387X.

[BR18] A. Budhkar and F. Rudzicz. "Augmenting word2vec with latent Dirichlet allocation within a clinical application." In: *arXiv preprint arXiv:1808.03967* (2018).

[Cam+17] J. Camacho-Collados, M. T. Pilehvar, N. Collier, and R. Navigli. "SemEval-2017 Task 2: Multilingual and Cross-lingual Semantic Word Similarity." In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 15–26.

[Con+17] A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou. "Word Translation Without Parallel Data." In: *arXiv preprint arXiv:1710.04087* (2017).

[Dee21] DeepL GmbH. *DeepL API*. https://www.deepl.com/de/docs-api/. Online. Accessed on 9/22/2021. 2021.

[Fac19] Facebook Inc. *MUSE*. https://github.com/facebookresearch/MUSE. Online. Accessed on 9/2/2021. 2019.

[Fac20] Facebook Inc. *fastText*. https://github.com/facebookresearch/fastText. Online. Accessed on 9/1/2021. 2020.

[Fac21] Facebook Inc. *Faiss*. https://github.com/facebookresearch/faiss. Online. Accessed on 9/22/2021. 2021.

[GD16] A. Gladkova and A. Drozd. "Intrinsic Evaluations of Word Embeddings: What Can We Do Better?" In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin, Germany: Association for Computational Linguistics, 2016, pp. 36–42. DOI: 10.18653/v1/W16-2507.

[HR02] G. Hinton and S. T. Roweis. "Stochastic neighbor embedding." In: *NIPS*. Vol. 15. Citeseer. 2002, pp. 833–840.

[JDJ17] J. Johnson, M. Douze, and H. Jégou. "Billion-scale similarity search with GPUs." In: *arXiv preprint arXiv:1702.08734* (2017).

[Lam+17] G. Lample, A. Conneau, L. Denoyer, and M. Ranzato. "Unsupervised Machine Translation Using Monolingual Corpora Only." In: *arXiv preprint arXiv:1711.00043* (2017).

[Li+19] B. Li, A. Drozd, Y. Guo, T. Liu, S. Matsuoka, and X. Du. "Scaling word2vec on big corpus." In: *Data Science and Engineering* 4.2 (2019), pp. 157–175.

[Lia15] Z. Lian. "Exploration of the working principle and application of word2vec." In: *Sci-Tech Information Development & Economy* 2 (2015), pp. 145–148.

[Lin+15] W. Ling, C. Dyer, A. W. Black, and I. Trancoso. "Two/too simple adaptations of word2vec for syntax problems." In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2015, pp. 1299–1304.

[LM14] Q. Le and T. Mikolov. "Distributed representations of sentences and documents." In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[Mik+13a] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[Mik+13b] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL].

[Mil19] D. Miller. "Leveraging BERT for extractive text summarization on lectures." In: *arXiv preprint arXiv:1906.04165* (2019).

[Onl] Online Übersetzungsbüro eLengua. *Technisches Fachwörterbuch Deutsch – Englisch*. `https://elengua.de/blog/technisches-woerterbuch-deutsch-englisch/`. Online. Accessed on 9/3/2021.

[Phi13] Philip Lief Group. *welding*. `https://www.thesaurus.com/browse/welding`. Online. Accessed on 9/23/2021. 2013.

[Reh21a] R. Rehurek. *Doc2vec paragraph embeddings*. `https://radimrehurek.com/gensim/models/doc2vec.html`. Online. Accessed on 9/22/2021. 2021.

[Reh21b] R. Rehurek. *FastText model*. `https://radimrehurek.com/gensim/models/fasttext.html`. Online. Accessed on 9/1/2021. 2021.

[Reh21c] R. Rehurek. *Word2vec embeddings*. `https://radimrehurek.com/gensim/models/word2vec.html`. Online. Accessed on 9/1/2021. 2021.

[RG19] N. Reimers and I. Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019.

[RG20] N. Reimers and I. Gurevych. "Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2020.

[RS11] R. Rehurek and P. Sojka. "Gensim–python framework for vector space modelling." In: *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3.2 (2011).

[Sch+15] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims. "Evaluation methods for unsupervised word embeddings." In: *Proceedings of the 2015 conference on empirical methods in natural language processing*. 2015, pp. 298–307.

[VC19] V. Vargas-Calderón and J. E. Camargo. "Characterization of citizens using word2vec and latent topic analysis in a large set of tweets." In: *Cities* 92 (2019), pp. 187–196.

[VH08] L. Van der Maaten and G. Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[Wan+19] B. Wang, A. Wang, F. Chen, Y. Wang, and C.-C. J. Kuo. "Evaluating word embedding models: methods and experimental results." In: *APSIPA Transactions on Signal and Information Processing* 8 (2019). ISSN: 2048-7703. DOI: `10.1017/atsip.2019.12`.

[WLL18] Y.-H. Wang, H.-y. Lee, and L.-s. Lee. "Segmental audio word2vec: Representing utterances as sequences of vectors with applications in spoken term detection." In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 6269–6273.

[Zha+19] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. 2019. arXiv: 1912.08777 [cs.CL].