UNIVERSITÄT KOBLENZ · LANDAU
Faculty 4: Computer Science

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Evaluation of Model and Hyperparameter Choices in word2vec

Bachelor's Thesis

in partial fulfillment of the requirements for
the degree of Bachelor of Science (B.Sc.)
in Informatik

submitted by
Jan Dillenberger

First supervisor:	Prof. Dr. Steffen Staab Institute for Web Science and Technologies
Second supervisor:	Lukas Schmelzeisen Institute for Web Science and Technologies

Koblenz, February 2019

Statement

I hereby certify that this thesis has been composed by me and is based on my own work, that I did not use any further resources than specified – in particular no references unmentioned in the reference section – and that I did not submit this thesis to another examination before. The paper submission is identical to the submitted electronic version.

	Yes	No
I agree to have this thesis published in the library.	☒	☐
I agree to have this thesis published on the Web.	☒	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐
The source code is available under MIT License.	☒	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐

....

(Place, Date) (Signature)

Abstract

When processing natural language on the computer, vector representations of words have many fields of application. They enable a computer to score the similarity between two words and allow to determine missing words for an analogy. In 2013 [Mikolov et al., 2013a] published an algorithm that is called word2vec to create such vector representations. It was able to create vector representations that far exceeded the performance of earlier methods. This thesis explains the algorithm's the popular skip-gram model from the perspective of neural networks. Furthermore a literature study is performed, examining the literature qualitatively and quantitatively figure out how word embeddings can and should be evaluated and to show the difference between proposed and actually applied evaluation methods. Thus, the thesis provides insights that enables us to identify and suggest best practices for the evaluation of these word vector representations. We identify the *similarity task*, *analogy task* and *tasks based on a downstream machine learning algorithm* that uses word vectors as data input representation as the three most important task categories for evaluating our word representations. In addition, the thesis shows which data sets are used for the evaluation of the similarity and the analogy task and further breaks down the downstream machine learning tasks used. The identified best practices are used to evaluate our own experiments to evaluate the effects for some small model and hyperparameter changes for the word2vec algorithm. The experiments results reveal that word representations for very rare words often are of bad quality and suggest vocabulary-sizes from 100,000 to 1,000,000 as reasonable choices. The evaluation also shows that embeddings created using a lowercase text corpus perform excellent on the similarity task but show a very poor performance on the analogy task. Weak performances are also shown by our experiments modifying word2vec's neural network model with the Adam optimizer or with dropout on the word2vec's embedding layer. Finally, we show that with word2vec especially rare words benefit from a high learning rate.

Zusammenfassung

Bei der Verarbeitung natürlicher Sprache am Computer haben Vektordarstellungen von Wörtern viele Anwendungsbereiche. Sie ermöglichen es einem Computer, die Ähnlichkeit zwischen zwei Wörtern zu bewerten und die fehlenden Wörter einer Analogie zu ermitteln. Im Jahr 2013 veröffentlichte [Mikolov et al., 2013a] einen Algorithmus zur Erstellung solcher Vektorrepräsentationen, der word2vec genannt wird. Dieser Algorithmus war in der Lage, Vektorrepräsentationen zu erstellen, die die Leistung früherer Methoden weit übertreffen. Diese Arbeit erklärt den Algorithmus des populären Skip-Gramm-Modells aus der Perspektive neuronaler Netze. Darüber hinaus wird eine Literaturstudie durchgeführt, die die Literatur qualitativ und quantitativ untersucht, um zu klären, wie Worteinbettungen bewertet werden können und sollten und, um den Unterschied zwischen vorgeschlagenen und tatsächlich angewandten Bewertungsmethoden aufzuzeigen. So liefert die Arbeit Erkenntnisse, die es uns ermöglichen, Best Practices für die Bewertung dieser Wortvektor-Darstellungen zu identifizieren und vorzuschlagen. Wir identifizieren die *Wortähnlichkeitsaufgabe*, die *Analogieaufgabe* und *Aufgaben basierend auf einem nachgelagerten maschinellen Lernalgorithmus*, die Wortvektoren als Dateneingabedarstellung verwenden, als die drei wichtigsten Aufgabenkategorien zur Bewertung unserer Wortdarstellungen. Darüber hinaus zeigt die Arbeit, welche Datensätze für die Bewertung der Ähnlichkeits- und Analogieaufgabe verwendet werden und gliedert die verwendeten nachgeschalteten Aufgaben des maschinellen Lernens weiter auf. Die identifizierten Best Practices werden verwendet, um unsere eigenen Experimente im Hinblick auf einige kleinere Modell und Hyperparameteränderungen für den word2vec-Algorithmus auszuwerten. Unsere Versuchsergebnisse zeigen, dass Wortdarstellungen für sehr seltene Wörter oft sehr schlecht sind und suggerieren Vokabulargrößen von 100.000 bis 1.000.000 als angemessene Wahl. Die Auswertungen ergaben auch, dass Einbettungen, die mit einem Textkorpus in Kleinbuchstaben erstellt wurden, bei der Ähnlichkeitsaufgabe sehr gut abschneiden, aber bei der Analogieaufgabe eine sehr schlechte Leistung zeigen. Schwache Leistungen zeigen die Experimente mit Modifikationen des word2vec-Algorithmus, die mit dem Adam-Optimierer oder Dropout auf der Einbettungsschicht von word2vec durchgeführt wurden. Darüber hinaus zeigen wir, dass besonders seltene Wörter bei word2vec von einer hohen Lernrate profitieren.

Contents

1 Introduction **1**

1.1 Tasks for Word Embeddings 2

1.2 Contributions 3

2 The word2vec Algorithm **4**

2.1 Training Data Representations 4

2.2 The Skip-Gram Model 5

2.3 The Neural Network 6

2.4 Extraction of Word Vectors 9

2.5 Subsampling for Frequent Words 9

2.6 Negative Sampling 11

3 Evaluation Approaches for Word Embeddings **12**

3.1 Evaluation Data 12

3.2 Methodology for the Literature Review 13

3.3 Similarity 14

3.4 Analogy 17

3.5 Downstream Machine Learning Tasks 19

3.6 Other Tasks 21

4 Model and Hyperparameter Choices in word2vec **22**

4.1 General Evaluation and the Comparability of our Implementation . . 23

4.2 Lowercase Embeddings 26

4.3 Impact of the Training Vocabulary 28

4.4 Noise Induced by Large Vocabularies 32

4.5 Hidden Layer Dropout 34

4.6 Impact of the Leaning Rate in word2vec 36

4.7 Adam Optimizer 40

5 Related Work **42**

5.1 How to Evaluate Word Embeddings 42

5.2 Model and Hyperparameter Modifications 42

6 Conclusion **44**

6.1 Evaluation 44

6.2 Experiments 44

6.3 Further Work 45

1 Introduction

One of the most powerful tools to express ourselves or interact with other people is natural language. While processing natural language is easy for humans, it can be challenging for computers. However, there is a lot of information a computer can try to extract from natural language, and even if computers don't really understand it, they can perform many useful tasks on a text-corpus, like sentiment analysis [Tang et al., 2014b, Liu, 2010] or machine translation [Zou et al., 2013, Koehn et al., 2007] giving the impression that they understand it.

Natural language encodes information or even knowledge in text using words. If you write a text in English, it seems to be clear how to represent them. For computers it can be helpful for some tasks to have a different word representation. Popular representations for computers are vector representations of words. These vectors encode information about the semantic and syntactic relationships of the words they represent [Mikolov et al., 2013c]. To get these vectors, we create so-called word embeddings m, which are mappings (equation 1) from a vocabulary of words V into vectors of a E-dimensional[1] space $\mathbb{R}^E$.

$$m : V \to \mathbb{R}^E \tag{1}$$

The reported origin of the first vector space models is not unambiguous in literature. [Turney and Pantel, 2010], for instance, wrote that these models were developed in [Salton, 1971]. But [Dubin, 2004] claimed that the idea of the vector space model was developed over a long period of time and was defined much later in [Salton and McGill, 1984]. In 2013, the word2vec algorithm was released, which can be used to create such word embeddings. With this algorithm it is possible to create word embeddings that by far outperformed the quality of previous attempts [Mikolov et al., 2013a]. Due to this breakthrough and the improvement of affordable computing power, popularity is growing rapidly and continues to shape research on this topic to this day.

In 2015 [Levy et al., 2015] showed that the massive quality improvement that word2vec was able to achieve compared to older approaches is not due to the innovative algorithm, but rather to well chosen hyperparameter choices. Even if the experience of the researchers working in this field has led to configurations that produce very high quality word embeddings, most hyperparameter and model choices are at best empirically motivated. Many researchers deal with this subtopic only in some side-note of their papers. However, changing hyperparameters is neither a very innovative idea nor is such a change expected to lead to a significant increase in performance, as the performance of modern embeddings is already very good. This could be the reason for the lack of scientific work that deals with this topic in depth.

[1]The dimensionality E of the vector space is normally located between 50 and 700 dimensions, which is usually much smaller than the size of the vocabulary.

This thesis evaluates embeddings resulting from different small word2vec modifications and hyperparameter configurations. To obtain meaningful evaluation results, a literature study is carried out, which examines how to best evaluate word embeddings.

1.1 Tasks for Word Embeddings

Word embeddings are used for a wide range of different natural language processing (NLP) tasks like sentence classification [Kim, 2014], named entity recognition [Pennington et al., 2014], machine translation [Zou et al., 2013], part-of-speech tagging [Al-Rfou et al., 2013] and even for recommender systems [Musto et al., 2016] or to rank web searches [Mitra et al., 2017]. Most of these tasks use word-vectors as input representation for downstream machine-learning tasks. Modern word embeddings encode the syntactic and semantic relationships of words to each other so, that much information can be calculated directly from them. The most popular tasks, which are used to extract information directly from a word embedding are:

Similarity Task Two words w_a and w_b are similar in a language if they are considered similar by people speaking that language. To estimate the similarity of two word-vectors the *cosine-similarity* of these vectors, which is defined by equation 2, is used [Schnabel et al., 2015]. Very similar words have a cosine similarity near to one and very dissimilar words have a cosine-similarity near to minus one.

$$\text{similarity}(w_a, w_b) \simeq \text{cosine\_similarity}(\vec{w}_a, \vec{w}_b) = \frac{\vec{w}_a \cdot \vec{w}_b}{\|\vec{w}_a\| \cdot \|\vec{w}_b\|} \tag{2}$$

For example, the similarity between the word vectors of the words *royal* and *queen* should be higher than the similarity of the words *royal* and *scissor*.

Analogy Task An analogy is a semantic relationship between two pairs of words. For the pairs (w_a, w_b) and (w_c, w_d) this means, as formalized in equation 3, that w_a relates in the same way to w_b as w_c relates to w_d. Analogy information in word embeddings is encoded through vector differences, see equation 4 [Mikolov et al., 2013a, Mikolov et al., 2013b].

$$(w_a, w_b) \cong (w_c, w_d) \tag{3}$$

$$\vec{w}_a - \vec{w}_b \simeq \vec{w}_c - \vec{w}_d \tag{4}$$

Word-vectors can be used to compute a missing word w_d in the analogy of equation 4, like it is shown in equation 5:

$$\text{nearest}(\vec{w}_b - \vec{w}_a + \vec{w}_c) = \vec{w}_d \rightarrow w_d \tag{5}$$

A popular example of an analogy is that the word *king* relates to the word *queen* in the same way the word *man* relates to the word *woman*.

1.2 Contributions

This thesis provides a variety of different contributions around word embeddings.

One of those is related to the question, how to evaluate word embeddings. [Bakarov, 2018] already summarizes the techniques, which can be used to evaluate word embeddings. However, it does not indicate which of these evaluation methods are of practical importance. In our opinion, however, this is key information to determine which evaluation methods should be used. Therefore we provide a frequency analysis on used evaluation methods.

Another question to clarify in this paper is how to choose hyperparameters. We offer experiments addressing this question, not only on the hyperparameters of the word2vec algorithm, but also on good hyperparameter choices in prepossessing and evaluation. However, our experiments do not only cover simple hyperparameter changes. We also modify the word2vec algorithm by applying popular ideas and building blocks for neural networks and evaluate how well these modified algorithms perform.

Furthermore, unlike the domain literature, we view the word2vec algorithm not primarily from the perspective of the language model, but from the perspective of neural networks. This approach should make it easier for people with experience in dealing with neural networks to familiarize themselves with the topic.

We provide the theoretical foundations for the theory of the word2vec algorithm from a perspective focusing on neural networks in Chapter 2. In Chapter 3, we then show what approaches and criticisms there are to evaluate word embeddings, and we compare them with our analysis of how word embeddings are evaluated in practice, which is in the same chapter. After that Chapter 4, describes our experiments in detail and provides the result on their evaluation and discusses the task specific findings. In Chapter 5, we provide a look on related literature and conclude our findings in the 6th and last chapter.

2 The word2vec Algorithm

In this chapter, the word2vec algorithm is presented. In Section 2.1 the trainings data representation for the algorithm is described. Section 2.2 describes word2vec's skip-gram model, which is in detail explained as a neural network in Section 2.3. Section 2.4 describes how the word vectors can be obtained from the trained word2vec model. Section 2.5 introduces a subsampling of frequent words as an optimization technique for the word2vec algorithm. And Section 2.6 explains the negative sampling optimization for the word2vec algorithm.

2.1 Training Data Representations

In order to create word represention vectors from a large amount of text, an appropriate word-representation is required that allows us to feed them into a neural network.

Definition 1. The *one-hot encoding* $\vec{e}_{w_i}$ of a word w_i in a vocabulary of words $V = \{w_1, \dots, w_{|V|}\}$ is a $|V|$-dimensional vector of only zeros, with its i-th component set to one [Rong, 2014].

A one-hot vector $\vec{e}_{w_i}$, as defined in *Definiton 1* is therefore used as a sparse, categorical vector representation for words [Goldberg, 2016]. Figure 1 shows a sentence and a possible representation of this sentence using one-hot vectors, with vectors created in the same order the words appeared for the first time in the sentence.

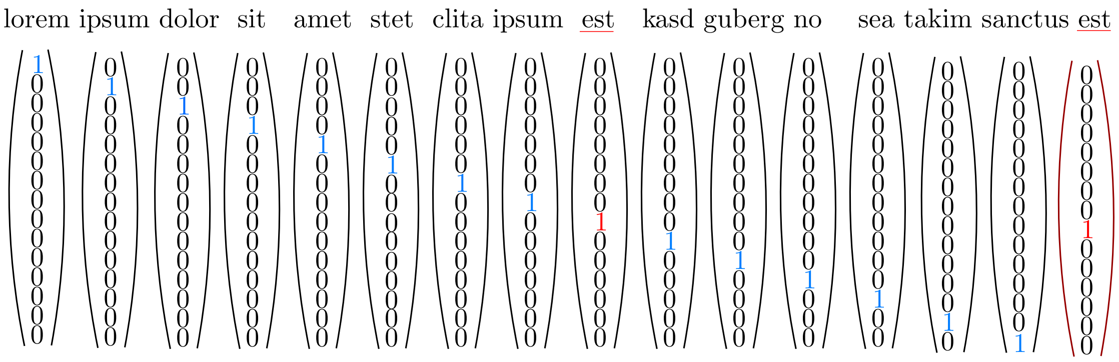
lorem ipsum dolor sit amet stet clita ipsum est kasd guberg no sea takim sanctus est

Figure 1: Example one-hot encoding representing the sentence above.

The exact quantity of different english words is not known[2]. In order to get an estimate number we counted the unique case insensitive tokens on the English Wikipedia after we extracted the article texts from a database dump[3] and we found more than 10 million unique tokens on this text corpus. This shows that it may not be a good idea to

[2]According to "How many words are there in the English language?" (Oxford Dictionary). Retrieved from `https://en.oxforddictionaries.com/explore/how-many-words-are-there-in-the-english-language` (accessed Jan 02, 2019).

[3]We used `https://github.com/attardi/wikiextractor` to extract the article texts from a wikipedia dump and deleted all remaining lines beginning with "<" from the corpus before we counted the tokens.

include all tokens in the vocabulary, because the memory consumption of the model we are going to train also grows with a larger vocabulary. It is therefore recommended to exclude the least frequent tokens from our vocabulary [Goldberg and Levy, 2014].

2.2 The Skip-Gram Model

The skip gram model for word2vec was introduced by [Mikolov et al., 2013a]. It is a kind of a neural network model used to generate meaningful word vectors. The neural network is trained to predict the surrounding words of a word t_r for each word in a text-corpus, which is a sequence of words $T = (t_1, t_2, \dots t_{|T|})$.

Definition 2. The *window size* ws is a hyperparameter for word2vec, which defines the size of a window of words. This window is composed of all words in a certain distance to a focus word t_r including this focus word. The window of words without the focus word is called the *context* of that word C_{t_r}.

The word2vec neural network requires labeled data for training. We provide this data as a sequence D of *(word, label)* tuples, which is defined by equation 6 .

$$D = ((t_r, t_s) \mid t_r, t_s \in T, |r - s| \leq \frac{\text{ws} - 1}{2} \wedge r \neq s) \tag{6}$$

The tuples in D are created for each word in our training text using its context words as labels (Figure 2).

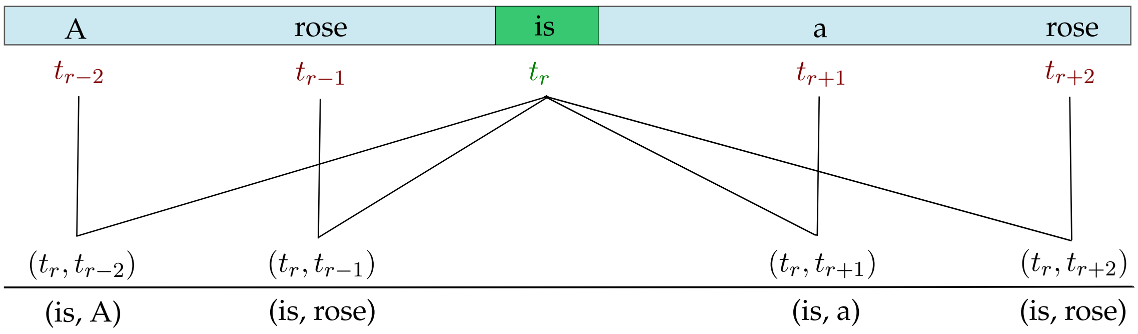

Figure 2: Illustration of a three layer neural network.

The conditional probability $P(w_j|w_i)$ for a word w_j to be part of the context window of a word w_i from our training corpus is described by equation 7. P is parameterized using two matrices W and W'. W is $M \times |V|$ dimensional and W' is $|V| \times M$ dimensional.

$$P(w_j \mid w_i) = \frac{\exp(\vec{w}_j'^{\top} \ \vec{w}_i)}{\sum_{k=1}^{|V|} \exp(\vec{w}_k'^{\top} \ \vec{w}_i)} \tag{7}$$

In word2vec the parameterization of W and W′ is searched, which minimizes the loss L (equation 8). $\vec{w}_i'$ is the vector of the transposed i-th row of the output weights matrix

W' of the trained neural network. Skip-grams goal is to maximize the average log probability L for all tuples in D.

$$L = \frac{1}{|T|} \sum_{(w_i,w_j) \in D} \log(\ P(w_j \mid w_i)\ ) \tag{8}$$

2.3 The Neural Network

In the literature, the word2vec algorithm is typically defined by equation 7 and not explained on the top of neural networks, as we will do in the following section. We explain this formula as a simple three-layer neural feed forward network. This neural network, which is schematically illustrated by Figure 3, consists of three fully connected layers of artificial neurons [Rong, 2014]. In the meanwhile, we give a short introduction to neural networks for NLP in general.

Definition 3. A *layer* $\vec{h}_j$ (equation 9) of a neural network is a group of artificial neurons. This group can be formalized for a feed-forward neural network using an $M \times N$ dimensional *weight matrix* W, an M-dimensional bias vector $\vec{b}$ and an activation function:

$$\vec{h}_j = \text{layer}(\vec{x})_j = \text{activation\_function}(W \cdot \vec{x} + \vec{b}) \tag{9}$$

The *activation function* is a non-linear function. Neural network layers use different activation functions such as ReLU, Sigmoid or TanH.

Biases
Bias vectors are very common for neural network layers, but in word2vec no bias vectors are used. For that we use the $\vec{0}$ as bias vector.

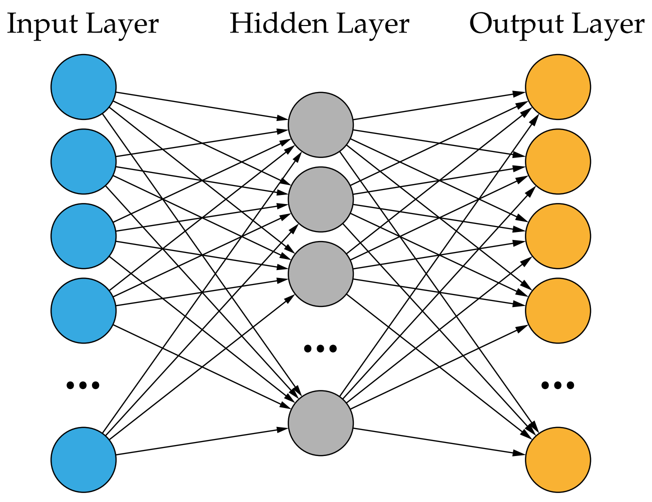

Figure 3: Illustration of a three layer neural network.

Input Layer The input layer $\vec{h}_1$, as formalized in equation 10, acts as a *passive placeholder* for data to be fed into the neural network. It performes no changes on the incoming data. Therefore equation 10 uses the identity matrix I as weight matrix and the identity function (id) as activation function. The dimensionality of the layer is $|V|$ for a vocabulary V. This allows to feed a one-hot encoded focus word $\vec{e}_{t_r}$ into this layer.

$$\vec{h}_1 = \text{layer}(\vec{e}_{t_r})_1 = \text{id}(I \cdot \vec{e}_{t_r} + \vec{0}) = \vec{e}_{t_r} \tag{10}$$

Input Data
In the case of word2vec, the data is fed into the neural network using one-hot vectors, but usual feed-forward networks are able to work with any type of vector if the dimensionality matches.

Hidden Layer The *hidden layer* $\vec{h}_2$ (equation 11) is, unlike the input layer, an *active layer* which makes calculations on the data-vector it receives from the previous layer. To calculate the result of the hidden layer a $|V| \times E$ dimensional *weight matrix* W is multiplied to the vector from the input layer. The dimensionality E of the hidden layer is much smaller than $|V|$, normally it lies between 50 and 700 dimensions [Al-Azani and El-Alfy, 2017, Goldberg, 2016].

$$\vec{h}_2 = \text{layer}(\vec{h}_1)_2 = \text{id}\left(W \cdot \vec{h}_1 + \vec{0}\right) = W \cdot \vec{h}_1 \tag{11}$$

Activation Function
Usually neural networks use activation functions in their active layers, but the hidden layer of the word2vec algorithm does not, because this way it can be trained much more efficiently. [Mikolov et al., 2013a]. We formalized this by using the identity as activation function.

Output Layer The last layer in the word2vec neural network is the *output layer* $\vec{h}_3$. This layer's result is one vector of conditional probabilities for each word that is part of the context window of the focus word entered into the input layer. The probability of the word w_i is represented by the i-th component of the output layers resulting vector $\vec{h}_3$. This *word probability vector* is calculated by applying a *softmax function*, see equation 12, to the result of the multiplication of a $E \times |V|$ dimensional output weight matrix W' and the hidden layers resulting vector [Goldberg, 2016].

$$\text{softmax}(\vec{z})_i = \frac{\exp(z_i)}{\sum_{k=1}^{|V|} \exp(z_k)} \tag{12}$$

$$\vec{h}_3 = \text{layer}(\vec{h}_2)_3 = \text{softmax}\left(W' \cdot \vec{h}_2 + \vec{0}\right) = \text{softmax}\left(W' \cdot \vec{h}_2\right) \tag{13}$$

Number of Layers
There is no explicit limit for the number of layers used in a neural network. In case of the word2vec neural network there are only three layers used.

The weight matrices of the hidden layer W and the output layer W' are randomly initialized and can be changed by a *stochastic gradient decent* optimizer, which adjusts all values of those matrices during the training of the neural network to minimize a *cross entropy loss function* H [Rong, 2014], see equation 14.

$$H(D) = -\sum_{r=1}^{|T|} p(D_{r,1}) \cdot \log(q(D_{r,2})) \tag{14}$$

The loss for one (word, label) pair is therefore calculated from the resulting probability distribution of the output layer $\vec{h}_3$ and the one-hot vector of the word used as label in the current training tuple D_r, see equation 15.

$$H(D) = -\sum_{r=1}^{|T|} \vec{e}_{D_{r,1}} \cdot \log(\mathrm{softmax}(W' \cdot W \cdot \vec{e}_{D_{r,2}})) \tag{15}$$

The neural network can be trained by feeding the focus word's one-hot vector $\vec{e}_{w_i}$ into the input layer. The calculation is passed on to the output layer, which leads to a prediction of conditional word probabilities. Then the cross entropy loss is calculated using the prediction vector and the one-hot vector of the word used as label. The optimizer then tries to minimize the loss by calculating a gradient for the entries of both mutable matrices. Afterwards, all these entries are adjusted according to the calculated gradient and a given learning rate α to achieve better prediction results. This process to feed a focus word's one-hot encoding into the input layer of the neural network and its label's one-hot vector into the calculation of the loss function to adjust the matrix elements using an stochastic gradient decent optimizer is called training of our model. This is now repeated for all the training word tuples *(focus word, label)* in a huge amount of training text. Empirically, the prediction accuracy increases after some training time.

2.4 Extraction of Word Vectors

During training, the stochastic gradient decent optimizer tries to adjust all mutable entries in the weight matrices to encode information about the *(focus word, label)* relationship for the context of each input word. The hidden layer has far fewer dimensions than the other layers, forcing the optimizer in the hidden layer to encode the relational information with appropriately fewer dimensions.

After the model has been trained on a huge text corpus, the D-dimensional, dense word representations can be extracted from the hidden-layer weight matrix W. The vector representation $\vec{w}_i$ of the word w_i is the i-th column of the matrix W, which contains all elements of our embedding-space $\mathbb{R}^D$ (equation 16).

$$W = \left(\vec{w}_1, \vec{w}_2, ...\vec{w}_{|V|}\right) \tag{16}$$

The resulting vectors $\vec{w}_i$ provide us with all the possibilities described above. This allows to make conclusions about word similarity by measuring the vector distances or the computation of analogies.

2.5 Subsampling for Frequent Words

Words like "*the*", "*for*", or "*of*" very often appear in a common text corpus. However they do not provide much information about the semantics of the words nearby. Therefore [Mikolov et al., 2013b] have shown that we can reduce some noise from our data and speed up our training by sometimes skipping those frequent words during training.

[Mikolov et al., 2013b] proposed to use equation 17, which works very well, to calculate the probability of a word to be skipped in training with $f(w_i)$ as the relative word frequency of the word w_i in the training corpus and t as threshold value, which they suggest to be chosen around 10^{-5}.

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{17}$$

This subsampling preserves the ranking of these words according to their frequency and significantly reduces the imbalance between very frequent and less frequent words [Mikolov et al., 2013b]. Figure 4 shows the frequencies of words according to their rank in the English Wikipedia and the frequency distribution of these words after a subsamling was applied.

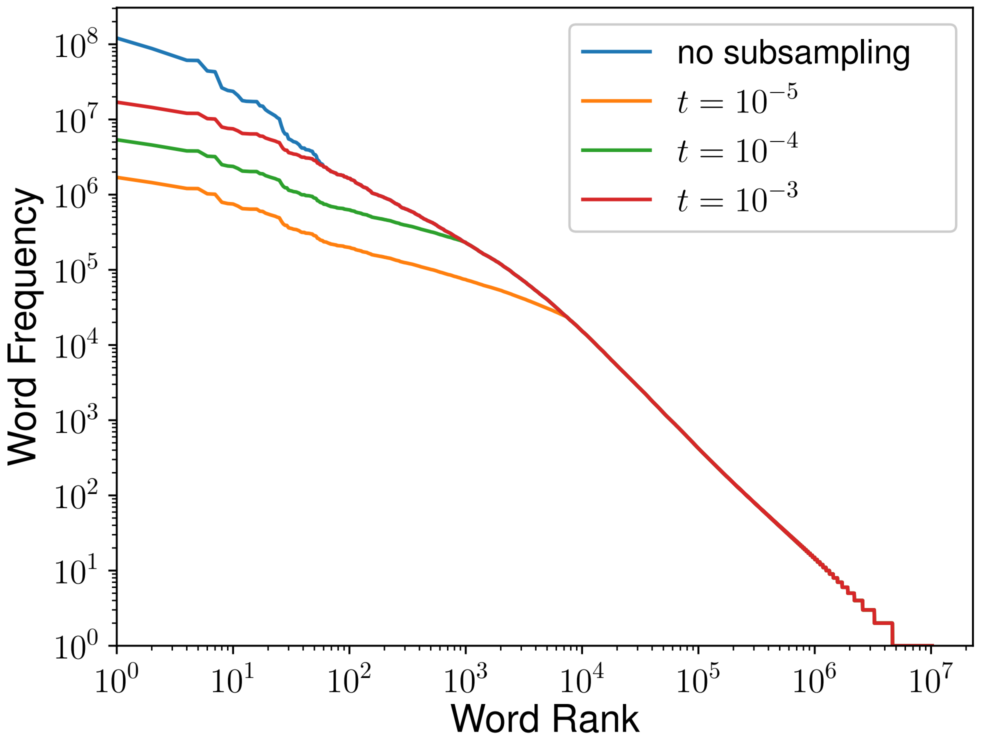

Figure 4: Word rank vs word frequency for the English Wikipedia (case-insensitive) on a logarithmic scale with and without subsampling.

2.6 Negative Sampling

The softmax function in word2vec, shown in equation 12, is calculated on vectors z of size $|V|$, which is usually a very large number. Since the softmax function takes into account every value in this vector, the calculation is very expensive and slows down the training.

To accelerate the training of the neural network, negative sampling has been introduced as optimization for word2vec [Mikolov et al., 2013b]. Using negative sampling, the calculation of the loss function including the softmax function is only performed for a subset $\vec{z'}$ of all entries in $\vec{z}$. This speeds up these calculations and results in a smaller number of elements that need to be adjusted in the weight matrices of our hidden and output layer, while stochastic gradient decent is performed.

The subset for the calculations $\vec{z'}$ is composed of z_a and n other entries of z, assuming that the neural network aims to predict the word w_a and the negative sampling rate is n.

3 Evaluation Approaches for Word Embeddings

Word embeddings can be used to solve many different problems, and there are just as many ways to evaluate them, making it impractical to apply all existing embedding evaluation methods. This is particularly problematic because the performance of an embedding depends heavily on the domain of the evaluation tasks. Furthermore, it occurs quite often that there is no strong correlation between embedding performance across different tasks [Chiu et al., 2016, Bakarov, 2018]. Therefore, we provide a literature study on which techniques are actually applied to evaluate word embeddings An outline of the evaluation critiques and best practices suggested in the literature focussing on often applied evaluation methods is also presented.

3.1 Evaluation Data

Most word embedding evaluations use labeled datasets for their evaluations. This is very practical as these datasets, once produced, can be shared and included in any other evaluation. On one hand this approach makes it very convenient to obtain evaluation data and it allows on the other hand a degree of comparability between evaluation results.

But no matter whether the data comes from available datasets or is even generated during the evaluation, the evaluation needs data and it has to be obtained somehow. According to [Bakarov, 2018], we divide the data collection process into four different categories:

Conscious The evaluation data is created by humans, which do perform the same or a somehow related task to what the word embedding should do.

Subconscious The evaluation data is created by measuring subconscious reactions of the human brain.

Thesaurus Based The evaluation data is generated from a thesaurus, which contains groups of synonyms and antonyms. This data about word relationships can be used to create evaluation datasets for word embeddings.[4]

Language Driven The evaluation data is like the word embedding generated from a text corpus, but it is not necessarily based on the co-occurrence of corpus- words.

[4]It may not be intuitively clear why a thesaurus-based approach is a category in its own sense, since it is also about man-made data collections. In our opinion, however, this is justified because a thesaurus is gradually improved by optimizations of different people and the data is therefore potentially much less biased by situational influences.

Most gold standard datasets are created with a conscious approach. However, this is heavily criticized because the human consciousness brings its biases such as priming effects[5] into the data [Bakarov, 2018]. This would probably be avoidable if the data were made using one of the other approaches [Bakarov, 2018]. Even if no biases were known about any of these methods, we could not say that there were none, which makes it hard to tell if one of these data-mining strategies is less biased for a particular task than the other ones.

Another problem with the consciousness approach arises when it comes to people's understanding of a particular task. The popular WordSim353 dataset (see Section 3.3.2) for similarity evaluation for instance was created by people who were instructed to rate the similarity for 353 pairs of words on a scale of zero to ten. The assessors rated the words coffee and cup as more similar than the words car and train, but the second pair is clearly more similar, even if the words are less related to each other than the words of the first pair [Faruqui et al., 2016].

3.2 Methodology for the Literature Review

To choose a series of papers from the literature, different search terms where used on *Google Scholar*, which are: *'word embeddings'*, *'word embedding evaluation'*, *'word representation'*, *'word representations'*, *'word vector'*, *'distributed representations'* and *'word vector space'*. The search performed in November 2018 selected scientific contributions. Only papers published in 2010 or later were included in the search results. Every paper in the results matching the criteria below was then included in the literature review[6].

1. The paper was in the first ten results for the search query.
2. The paper was an accessible scientific paper and was therefore not hidden behind a paywall.
3. The papers domain is related to natural language processing.
4. The paper evaluates word-embeddings and performs at least one evaluation in English only.

[5]Semantic priming describes the fact that people react differently to a word if they previously had contact with a related word.

[6]An overview of the papers we found for each search term including the ones, which do not match our criteria can be found here: `https://userpages.uni-koblenz.de/~jdillenberger/bachelorthesis/Frequency-Analysis-Results.ods`

These criteria are based on the following considerations:

Google Scholar sorts the results by the number of citations, often-cited scientific papers appear in Google Scholar on top of the search results. We assume that these papers have a particularly high impact on how word embeddings are evaluated. Thus these publications show what a state-of-the-art evaluation should look like. Therefore, we assume that the first ten papers of the Google Scholar result are the most relevant ones and show best what a state-of-the-art word embedding should look like.

Not all papers are freely available. Since we cannot examine any paper that we do not have access to, these will be excluded from our review.

Embedding technologies are also used in areas that have nothing to do with natural language processing. This causes our query results to contain, for example, some medical work [Nikfarjam et al., 2015, Tang et al., 2014a], which does not relate to our task, and therefore adds no value for our evaluations.

The literature study examines papers on which evaluation techniques are used in them. Therefore, papers without their own evaluation offer us no added value for our frequency analysis. In addition, all word embeddings of this work have been created for the English language. Therefore, evaluations in other languages can not add much value to our own.

3.3 Similarity

The similarity evaluation task defined in Section 1.1 is by far the most popular word embedding evaluation task. Typical datasets like *WordSim353*, *MEN*, or *Rare Words* are composed of two word columns and one column, which holds a similarity score, for each pair of words. The embedding's representation vectors for both words in a row are now used to calculate a similarity score for this pair. After this is done for each pair in the dataset, a correlation coefficient (e.g. *Spearman*, *Pearson*) is calculated between the similarity values given in the dataset and the calculated ones.

3.3.1 Similarity Evaluation Notes

One problem in the evaluation of word similarities is how to understand the term similarity[7]. Much literature as well as many datasets do not distinguish between similarity and relatedness of words [Faruqui et al., 2016]. Some researchers claim that there are more then fifty social, linguistic or psychological factors, which could introduce their biases into the human understanding of semantic similarity [Gladkova and Drozd, 2016].

[7]With an embedding, a relatedness score is calculated in exactly the same way, we would calculate a similarity score between two words. The difference is, that word similarity theoretically can only be scored if the words share a common category, while this is not required for relatedness.

Furthermore, we should be aware that many datasets for word similarity evaluation were produced in psycholinguistic studies without consideration of corpus statistics [Schnabel et al., 2015]. This indicates, for instance, that the data does not reflect the problems of particularly rare or common words.

3.3.2 Frequency Analysis

In order to get good comparable results with other publications and to get a better overview about how word embeddings should be evaluated, we analysed the scientific papers from the Google Scholar search from Section 3.2 and checked which datasets the papers use to evaluate their embeddings regarding similarity.
Table 1 shows, which paper uses which dataset. All discovered datasets and their sources are shown in Table 2. The discovered datasets are summarized below.

WordSim353 is by far the most commonly used dataset for the word similarity evaluation. It consists of 353 pairs of words and their similarity scores rated by participants of a psycholinguistic experiment. The evaluation is usually done using the average similarity score of all the participants ratings. The dataset was criticized because it does not differentiate between similarity and relatedness. Therefore, a split version of the dataset was created, dividing it into a similarity dataset and a relatedness dataset.

MEN is a dataset, which was first used in [Bruni et al., 2012] and then officially published in [Bruni et al., 2014]. The dataset was created using data gathered via Amazon Mechanical Turk. It consists of 3,000 pairs of words. The participants were asked to compare the similarity of word pairs instead of giving an absolute similarity score. This approach is hoped to be less biased by the human consciousness.

RG-65 is a very old dataset published by [Rubenstein and Goodenough, 1965] containing 65 pairs of words and their similarity scores, these similarity scores are the means of judgments made by 51 subjects.

RareWords is a dataset published by [Luong et al., 2013]. It uses absolute similarity ratings of Amazon Mechanical Turk participants. The words picked for the word pairs in the dataset, were divided into buckets according to their frequency. For each bucket an equal number of words was selected for the dataset. This ensured that rare words had a higher probability to be used in that dataset [Luong et al., 2013].

Miller&Charles is a subset of *RG-65* created by [Miller and Charles, 1991]. It contains a group of words with high similarity pairs, another group with medium similarity and a third one with low similarity. Each of these groups consists of 10 pairs of words and their similarity ranking. Sometimes a subset of this dataset is used, which only contains 28 of those word-pairs. This version of the dataset excludes the word-pairs which are not represented in WordNet [Resnik, 1995].

Evaluation Data / Paper	[Huang et al., 2012]	[Luong et al., 2013]	[Levy and Goldberg, 2014a]	[Levy and Goldberg, 2014c]	[Pennington et al., 2014]	[Faruqui and Dyer, 2014]	[Bansal et al., 2014]	[Neelakantan et al., 2015]	[Levy et al., 2015]	[Neelakantan et al., 2015]	[Schnabel et al., 2015]	[Faruqui et al., 2015]	[Ghannay et al., 2016]	[Faruqui et al., 2016]	[Bojanowski et al., 2017]	$\sum$
Chiarello			✗													1
MEN				✗					✗		✗	✗	✗	✗		6
MTurk-287						✗			✗					✗		3
MTurk-771														✗		1
Miller&Charles		✗			✗	✗								✗		4
RareWords		✗			✗				✗				✗	✗	✗	6
RG-65		✗			✗	✗					✗	✗		✗		6
SCWS		✗			✗			✗		✗						4
SimLex999									✗					✗		2
Verb-144														✗		1
WordSim353	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	15
Yang&Powers														✗		1

Table 1: All similarity evaluating papers from our search, and the datasets used for the evaluations.

Dataset	Literature Reference	Instances
Chiarello[8]	[Chiarello et al., 1990]	144
MEN	[Bruni et al., 2014]	3,000
MTurk-287	[Radinsky et al., 2011]	287
MTurk-771	[Halawi et al., 2012]	771
Miller&Charles	[Miller and Charles, 1991]	30 or 28
RareWords	[Luong et al., 2013]	2,034
RG-65	[Rubenstein and Goodenough, 1965]	65
SCWS	[Huang et al., 2012]	2,003
SimLex999	[Hill et al., 2015]	999
Verb-144	[Baker et al., 2014]	144
WordSim353	[Finkelstein et al., 2002]	353
Yang&Powers	[Yang and Powers, 2006]	130

Table 2: All publications with similarity evaluations we discovered in our frequency analysis, their literature reference and the number of training instances in the respective dataset.

SCWS is the accronym for the *Stanford Contextual Word Similarities* dataset. It was published by [Huang et al., 2012] and consists of 2,003 pairs of words. The dataset does not only provide pairs of words and their similarity scores but also a context for the similarity scores of those words.

MTurk-287 is a dataset of 287 human assigned word pairs and relatedness scores on a scale from zero to five. The dataset was created by [Radinsky et al., 2011] using Amazon Mechanical Turk.

SimLex999 is the most recently created dataset we found during our literature search. It was published by [Hill et al., 2015] and contains 999 word pairs and their similarity scores. In contrast to other datasets like MEN, they explicitly differentiate between similarity and relatedness, and therefore do not assign high similarity scores on words, which are highly related but not really similar. The authors also provide some additional features in the dataset, for example a part-of-speech tag for each pair of words.

Chiarello is not a classical word similarity dataset, because it does not provide similarity scores for the word pairs. Instead, the dataset published by [Chiarello et al., 1990] classifies words as similar, associative or both. But this kind of data can still be utilized for a word-similarity-like task by instructing the word embedding to rate the similarity of the word pairs and then using the dataset to check whether the similar words were rated as more similar than the associative ones.

MTurk-771 is dataset of word-pairs and human-assigned relatedness scores from zero to five published by [Halawi et al., 2012]. Like *MTurk-287* it was created using Amazon Mechanical Turk, but with more pairs of words.

Verb-144 is a dataset created by [Baker et al., 2014]. They used the same guidelines as used for the creation of WordSim353, but their dataset only contains verbs. The scores are created by ten annotators.

Yang&Powers is a dataset of 130 pairs of verbs and their similarity scores created by Amazon Mechanical Turk workers, with on average 23 similarity ratings for each pair of words. The dataset was published by [Yang and Powers, 2006].

3.4 Analogy

Word embeddings can be used to compute a missing word of a given analogy. [Mikolov et al., 2013b] have even shown that analogies can be computed without the usage of a downstream machine-learning task. This impressively shows how well the semantics of the words are encoded in a word embedding and gives us a technique to solve these analogies. This is particularly interesting as evaluations with this task did not contain the biases of the downstream task. Unfortunately this is not meaningful when judging

[8]Chiarello does not provide similarity scores. It classifies pairs of word as similar and/or related.

the overall performance of a word embedding, since there often is no strong correlation between the performance of a word embedding in different tasks. Nevertheless, the performance of a word embedding calculating analogies is a widespread evaluation method that makes it possible to show if an embedding is a useful representation of the syntax or semantics of a language.

Datasets for the evaluation of analogies will typically contain four columns, each containing a word in each row. Three of the words are now used as input for the calculation, the fourth word contains the expected calculation result. The accuracy between the calculations and the expected results in the dataset is calculated at the end as our accuracy score.

3.4.1 Frequency Analysis

We show the results of our frequency analysis for the analogy task in Table 3 and list the dataset along with their publishing papers and the number of trainings instances they contain in Table 4.

Google Analogy is by far the most used dataset for analogy evaluation. It was published by [Mikolov et al., 2013b] and consists of 19,544 quadruples of words, each representing one analogy. The dataset splits its quadruples in different subclasses of semantic or of syntactic analogies. The reason for its popularity may be, that it was the initial dataset published with the idea that analogies can be extracted from a word embedding. The dataset is also often called the Semantic-Syntactic Word Relationships Dataset.

MSR Analogy was published by [Mikolov et al., 2013c]. Its acronym *MSR* stands for Microsoft Research and it consists of 8,000 analogies out of 16 different classes.

SemEval2012 was published in [Jurgens et al., 2012]. It contains 10 014 questions created via Amazon Mechanical Turk. The dataset contains word pairs and numerical ratings. The ratings describe how well a pair fits into a specific relationship class. An example of such a class would be *ENTITY:SOUND*, and a pair like *'cat'* and *'meow'* would fit well into this class of relationships.

Evaluation Data Paper	[Mnih and Kavukcuoglu, 2013]	[Mikolov et al., 2013a]	[Mikolov et al., 2013c]	[Mikolov et al., 2013b]	[Levy and Goldberg, 2014c]	[Qiu et al., 2014]	[Levy and Goldberg, 2014b]	[Faruqui and Dyer, 2014]	[Levy et al., 2015]	[Schnabel et al., 2015]	[Faruqui et al., 2015]	[Ghannay et al., 2016]	$\sum$
Google Analogy	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	12
MSR Analogy	✗	✗			✗		✗		✗				5
SemEval2012			✗				✗						2

Table 3: All analogy evaluating publications from our literature search, and the used datasets[9] for the respective evaluations.

Dataset	**Literature Reference**	**Instances**
Google Analogy	[Mikolov et al., 2013a]	19,544
MSR Analogy	[Mikolov et al., 2013c]	8000
SemEval2012[10]	[Jurgens et al., 2012]	(10,014)

Table 4: All papers with analogy evaluations we discovered in our frequency analysis, their literature reference, and the number of training instances in the dataset.

3.5 Downstream Machine Learning Tasks

Most application possibilities for embeddings are only possible with the help of downstream machine learning techniques that use an embedding as input data representation for their training data. One problem of these techniques is that they introduce their own learned biases into the evaluation. However, since there is no clear correlation of the performance of an embedding in different tasks, the embedding would have to be tested in a variety of different tasks to get an impression of its overall performance. Therefore, downstream tasks are necessary for a proper evaluation of the overall performance of an embedding.

[9]There are much more datasets for analogy evaluation available. But our analysis discovered only these three.

[10]SemEval does not directly provide analogies, but it provides pairs of words and their relationship to each other. Therefore, we can easily create analogies from that dataset.

3.5.1 Frequency Analysis

Table 5 shows all the publications from our literature search matching our criteria and performing downstream tasks using an embedding as input data representation[11]. On the one hand, because this work is more about word embeddings than about the downstream tasks, on the other hand, because with such an overview there hardly would have been any overlaps between the methods used with different papers, the following table is about the tasks used in the publications and not about the datasets used.

Evaluation Task Paper	[Turian et al., 2010]	[Dhillon et al., 2011]	[Maas et al., 2011]	[Al-Rfou et al., 2013]	[Pennington et al., 2014]	[Bansal et al., 2014]	[Tang et al., 2014b]	[Le and Mikolov, 2014]	[Kiros et al., 2014]	[Kim, 2014]	[Faruqui et al., 2015]	[Kusner et al., 2015]	[Ling et al., 2015b]	[Bollegala et al., 2015]	[Clark and Manning, 2016]	∑
Blog Authorship Attribution									✗							1
Coreference Resolution															✗	1
Document Clustering												✗				1
Named Entity Recognition	✗	✗			✗											3
Part of Speech Tagging				✗		✗							✗			3
Sentiment Analysis			✗				✗	✗	✗	✗	✗			✗		7
Syntactic Chunking		✗														1

Table 5: All publications from our literature search, which do perform downsteam tasks for the embedding evaluation and the type of task they are doing.

In the following only the downstream machine learning tasks, with a frequency above one are described.

Sentiment Analysis is a popular classification task for sentences or even whole documents. Usually the classification goal is to determine, whether a piece of text conveys a positive, a negative or a neutral opinion concerning a subject or if a piece of text is more likely to be objective or subjective.

Named Entity Recognition The goal of this task is to find e.g. names, dates and locations in a text and to determine all the tokens, which are part of one of those.

[11]Our goal is to evaluate embeddings for the English language. Therefore, multilingual evaluations do not appear in the table. However, machine translation is also a popular task when embeddings are evaluated using multiple languages.

Part-of-Speech Tagging is the task to determine a word's function in a text. An example part-of-speech tagging task is, to determine if a word is a noun, a verb or an adjective and is it an indicator for singular or plural.

3.6 Other Tasks

There are some tasks for word embeddings other than similarity and analogy calculation that do not use the word embeddings as input representation for a downstream machine learning task. Evaluations on those tasks are much less frequent than those on similarity or analogy. Nevertheless, we found some other tasks during our frequency analysis and presented them below.

Synonym Detection is a word embedding task, which is closely related to the similarity task. The task is to determine which word in a selection of words is a synonym of a specific word.

Categorization Words can be categorized using word vectors. This categorization is done by clustering similar words into groups. Words of the same group are considered as words of the same category. [Schnabel et al., 2015]

Selectional Preference In the context of a specific word, some words appear more likely than others. To identify words as a context specific selectional preference, a measure of similarity is used on words which are grammatically appropriate for the given context. The word *'drives'* has a much higher selectional preference in context of the word *'car'* than the word *'eats'*, for example.

The synonym detection task was only performed by [Faruqui et al., 2015]. The only publication that performed the selectional preference as well as the categorization task is [Schnabel et al., 2015].

4 Model and Hyperparameter Choices in word2vec

Word embeddings have a wide range of current and potential applications. Unfortunately, it is often unclear in which scenarios which specific algorithm performs best. The huge amount of theoretically possible model decisions and parameters in algorithms like word2vec and the unclear implications of these make it very difficult for new researchers to get into the topic. Alongside their knowledge researchers need to rely heavily on their experience about the the effects specific model changes have in a word embedding.

The purpose of this chapter is therefore to obtain more insight on how to create and to use word embeddings. Unfortunately, it is often difficult to make a statement about the significance of an experiment in this field, since many modifications may only show their potential in combination with other changes, because different hyperparameters and model choices often strongly depend on each other. Nevertheless, for reasons of clarity, only small changes are applied in this chapter. Evaluations about the implications of the hyperparameter context concerning specific changes are left for future work.

Some design decisions lead to a different set of hyperparameters available in a word2vec implementation. There is a set of hyperparameters, which all our evaluations have in common. These hyperparameters are:

epochs The word2vec algorithm trains on a large corpus of text. Each training iteration on this text corpus is called an epoch. n epochs therefore means that the algorithm iterates n times over the available training data.

window-size The word2vec algorithm optimizes a neural network to predict whether a word w_i is part of the context of word w_t. The window-size (defined in Section 2.2) specifies the size of that context around the word w_t.

embedding-size The embedding-size is the number of dimensions for the neural network's hidden layer and therefore defines the vector-size for the word-vectors in the resulting word embedding.

vocabulary-size The vocabulary is a list of words sorted by their frequency in the text corpus. Its size is the size of this list and therefore defines how many words are contained in the resulting embedding of word2vec. All words that are not frequent enough to be part of this list are excluded.

batch-size The word2vec model trains on tuples of *(word, label)* training data (defined in Section 2.2). The batch-size is the number of tuples, on which we train the neural network in each training step.

learning-rate Usually only a *start-learning-rate* is defined, because the start-learning-rate for a certain training step is then dynamically calculated using the learning-rate and a decay function.

learning-rate-decay It is often helpful to reduce the learning-rate in a neural network during training. Therefore some kind of decay method can be applied to calculate the learning-rate for a particular training step.

negative-sampling-rate Negative sampling (defined in Section 2.6) is a popular optimization for word2vec, which is used in all experiments in this work. It introduces the negative-sampling-rate as hyperparameter for the algorithm, which defines the size of the subset of words $\vec{z}'$ for which the softmax function is computed.

subsampling-threshold Subampling of frequent words (defined in Section 2.5) is also a popular optimization for word2vec, which is used in all experiments in this work. It introduces a subsampling-threshold as hyperparameter for the algorithm, which reduces the probability of highly common words being used as training data.

In the next section, we first show the comparability of our algorithm with an embedding published by [Mikolov et al., 2013b]. The subsequent sections then evaluate different approaches for hyperparameter and model selection.

4.1 General Evaluation and the Comparability of our Implementation

For our experiments we created our own word2vec implementation[12] with Python and TensorFlow[13]. In this section an embedding created by using our reimplementation is compared with the *GoogleNews-vectors-negative300*[14] (GNV) published by [Mikolov et al., 2013b][15].

However, this embedding was created on the basis of other training data with other hyperparameters, which may have an effect on the resulting performance. Instead of using the Wikipedia corpus[16] the GNV embedding was trained on a Google-News dataset which was not available for us.

[Mikolov et al., 2013b] splits the corpus on every 'tab', 'space', and 'newline' to tokenize the corpus. We used the *word_tokenize* function from the Natural Language Toolkit[17]

[12]`https://gitlab.uni-koblenz.de/jdillenberger/word-vector-creator`

[13]`https://www.tensorflow.org/`

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

[15]The code of the original model is available on `https://github.com/tmikolov/word2vec`

[16]We extracted all articles from a Wikipedia dump using the WikiExtractor, which is available on: `https://github.com/attardi/wikiextractor`

[17]`https://www.nltk.org/`

(NLTK) instead, which is a lot slower, but leads to better tokenization results. GNV includes a vocabulary of three million words. We decided to use a smaller vocabulary of only one million words to reduce our algorithm's memory consumption for this experiment, as our fastest server, which we used for this experiment, had only eight gigabytes of working memory. For the same reason, we use an embedding-size of 200. The embedding size for the GNV was 300.

Furthermore, an unknown-token is used as word representation for all the very infrequent words, which are not included in our vocabulary. [Mikolov et al., 2013b] instead skip those words in their word2vec implementation.

Aside from these hyperparameters, we do not know which configuration [Mikolov et al., 2013b] used to create GNV. But as a subsampling-threshold for frequent words they proposed using a threshold of 10^{-5}, which we used in our embedding too. We dissect our training data into sections of 5,120,000 target and context word pairs. We randomize the elements of each section and then dissect the sections into training batches of 128 pairs[18]. Subsequently our algorithm is trained with one training batch per training step, using a start-learning-rate of 2.5, which decreases linearly and reaches zero as soon as the training is finished. Table 6 lists all hyperparameters for the creation of this embedding.

epochs	5
window-size	5
embedding-size	200
vocabulary-size	$1,000,000$
negative-sampling-rate	16
subsampling-threshold	10^{-5}
batch-size	128
start-learning-rate	2.5
decay	linear to zero

Table 6: Hyperparameters for the evaluation of our model.

Because of the time constraints of a bachelor thesis, we limit the scope of our evaluation to only consider similarity (Section 4.1.1) and analogy (Section 4.1.2). This means that no downstream machine learning tasks are evaluated.

[18]The training data is only randomized within sections to avoid huge memory consumption. Theoretically, it would be best to randomize the data across the entire dataset.

4.1.1 Similarity Evaluation

We evaluate the resulting word embeddings using all datasets that occurred at least five times in our frequency analysis. Furthermore, we decided to use the *SimLex999* dataset in our evaluation because we want to test our embedding as thoroughly as possible. SimLex999 is the newest dataset we found in our frequency analysis (Section 3.3.2). It is especially interesting because embeddings that perform well on this dataset tend to work well for some downstream machine learning tasks [Chiu et al., 2016], which are missing in our evaluation. The dataset explicitly prefers semantic similarity over association and relatedness. We believe that this may be especially valuable, because we do not provide any evaluation on downstream machine learning tasks.

We calculate the similarity scores of the word pairs in each dataset for both embeddings and compute Spearman's correlation coefficients between the similarity values provided by the datasets and respectively the corresponding calculated values. The results for both embeddings and the used datasets are provided in Table 7.

Dataset	**Our Result**	**GNV**	**Difference**
MEN	*0.672*	*0.699*	*-0.027* (-3.86%)
RareWords	*0.251*	*0.264*	*-0.013* (-4.92%)
RG-65	*0.737*	*0.749*	*-0.012* (-1.60%)
SimLex999	*0.375*	*0.437*	*-0.062* (-14.14%)
WordSim353	*0.681*	*0.659*	*+0.022* (+3.33%)
WordSim353-rel	*0.660*	*0.610*	*+0.050* (+8.20%)
WordSim353-sim	*0.695*	*0.741*	*-0.046* (-6.21%)

Table 7: Comparison between our embedding and GoogleNews-vectors-negative300 (GNV) for the word similarity task. The table shows the Spearman's rank correlation between the datasets and the calculated similarity score.

The results show that our embedding performs worse than GNV for most datasets. But the performance differences are small and probably explainable by the limitations, such as smaller embeddings and vocabulary sizes. Our embedding shows the worst performance for SimLex999 and WordSim353-sim compared to GNV. This may indicate that our limitations have more impact on word similarity than on word relatedness and associativity, since these two datasets explicitly target similarity evaluation in contrast to the other evaluation datasets.

4.1.2 Analogy Evaluation

Exactly as for the similarity datasets, we evaluate both embeddings on those datasets, which appeared at least five times. For the analogy evaluation, there are not as many popular datasets as for similarity evaluation, which leads to two datasets for analogy evaluation: the Google Analogy and the MSR Analogy dataset. Table 8 shows the evaluation results for our own embedding and GNV on both datasets. The performance for both embeddings is very similar in each analogy dataset.

Dataset	Our Result	GNV	Difference
Google Analogy	*0.715*	*0.735*	*-0.021* (-2.72%)
MSR Analogy	*0.578*	*0.568*	*0.010* (+1.76%)

Table 8: Comparison between our embedding and GoogleNews-vectors-negative300 (GNV) for the analogy task. The table shows the accuracy for all classified analogies.

4.2 Lowercase Embeddings

The English language uses mostly lowercase words. The reasons for a word to start with a capital letter is more often based on the structure of the sentence or the relevance of a part of text, than on a word's semantics. This leads to uppercase and lowercase representations of those words in our text corpus, which might not be very useful.

Therefore we converted all alphabetic characters in our corpus into lowercase characters before we trained our model. We used the model configuration from Section 4.1 (Table 6) on our lowercase text corpus.

However, it is problematic to correctly evaluate lowercase embeddings, as their evaluation requires the words in the evaluation datasets to be converted to lowercase as well. For an embedding that contains capital letters in its words, this has a negative effect on the performance of this embedding. But evaluating one embedding on lowercased datasets and one on the usual datasets on the other hand also results in an unequal comparison.

4.2.1 Similarity Results

Table 9 shows the evaluation results for our embedding from Section 4.1 *(Default)* and an embedding trained with the same hyperparameters but on lowercased words *(Lowercase)*. The Lowercase embedding was evaluated on lowercased datasets.

Dataset	**Lowercase**	**Default**	**Difference**
MEN	*0.701*	*0.672*	*+0.029* (+4.32%)
RareWords	*0.402*	*0.251*	*+0.151* (+60.15%)
RG-65	*0.749*	*0.737*	*+0.012* (+1.62%)
SimLex999	*0.411*	*0.375*	*+0.036* (+9.60%)
WordSim353	*0.625*	*0.681*	*-0.056* (-8.22%)
WordSim353-rel	*0.529*	*0.660*	*-0.131* (-19.85%)
WordSim353-sim	*0.745*	*0.695*	*+0.050* (+7.20%)

Table 9: Comparison between our lowercase embedding and our embedding from section 4.1 for the word similarity task. The table shows the Spearman's rank correlation for the dataset's similarity score and the calculated similarity score. The lowercase embedding is evaluated using lowercased datasets. The default embedding is evaluated using the usual datasets.

In our evaluation the lowercase embedding outperforms the default embedding for most of the used similarity datasets and surpasses our default embedding for the RareWords dataset by impressive 60.15%. We guess that the lowercase embedding improves the number of trainings steps for many infrequent words, because it trains the upper and the lowercase representations for those tokens. Additionally, it is more likely to include those rare words in the vocabulary, because stripping all the uppercase words from the vocabulary frees up space for other words in the vocabulary.

In our evaluation the lowercase embedding outperforms the default embedding for most of the used similarity datasets and surpasses our default embedding for the RareWords dataset by impressive 60.15%. We guess that the lowercase embedding improves the number of trainings steps for many infrequent words, because it trains the upper and the lowercase representations for those tokens. Additionally it is on more likely to include those rare words in the vocabulary, because stripping all the capital and uppercase words from the vocabulary frees up space for other words in it.

The lowercase embedding performs worse than the default embedding on the WordSim353 dataset and especially worse on the specialized WordSim353-rel dataset. In contrast it performs significantly better than the default embedding on SimLex999 and WordSim353-sim, which are more specialized on word similarity than on relatedness. We are currently not aware of what makes lowercase embedding better for these semantic similarity tasks and worse for the relationship task.

4.2.2 Analogy Results

Table 10 shows the evaluation results for the Lowercase and the Default embedding used for the analogy task.

Dataset	**Lowercase**	**Default**	**Difference**
Google Analogy	*0.603*	*0.715*	*-0.112* (-15.66%)
MSR Analogy	*0.445*	*0.578*	*-0.133* (-23.01%)

Table 10: Analogy evaluation result for our algorithm, for common hyperparameters.

The lowercased embedding performed much worse on both datasets for the analogy task.

4.2.3 Experiment Conclusion

Mixedcase embeddings and those trained on lowercase texts seem to differ greatly in their strengths. While the lowercase embedding performed better for our similarity tasks, the normal embedding performed better for the analogy task. We think that lowercase embeddings lead to poorly positioned embeddings. It is our expectation that similarity calculations are less sensitive to small deviations than analogy calculations. The lower case conversion of a text corpus simultaneously reduces the rank of the words in that corpus. Words with high rank are often expected to offer poor performance. By reducing the rank of the words, there will probably be an improvement in the positioning of these words in the vector space that is greater than the errors caused by the lowercase conversion. This effect could be further enhanced by the fact that a lowercase corpus contains more instances for many vocabulary words. This further enhances the effect of the subsampling thresholds.

4.3 Impact of the Training Vocabulary

While the English version of Wikipedia, for example, contains more than 10 million different tokens, 150 different words typically contribute to about half of the word occurrences in a large body of text [Powers, 1998]. This relationship that a few words are accountable for most word occurrences in a text corpus and most words occur very rarely is described by Zipf's law. This law enables us to approximate the number of occurrences of a word with rank k in a corpus of N words (equation 18). The parameter is a s characterization for the the distribution.

$$\text{frequency}(k, s, N) = \frac{1}{k^s \cdot \sum_{n=1}^{N} \frac{1}{n^s}} \tag{18}$$

Figure 5 shows that Zipf's law provides a good prediction for the frequency of a word in the Wikipedia text corpus.

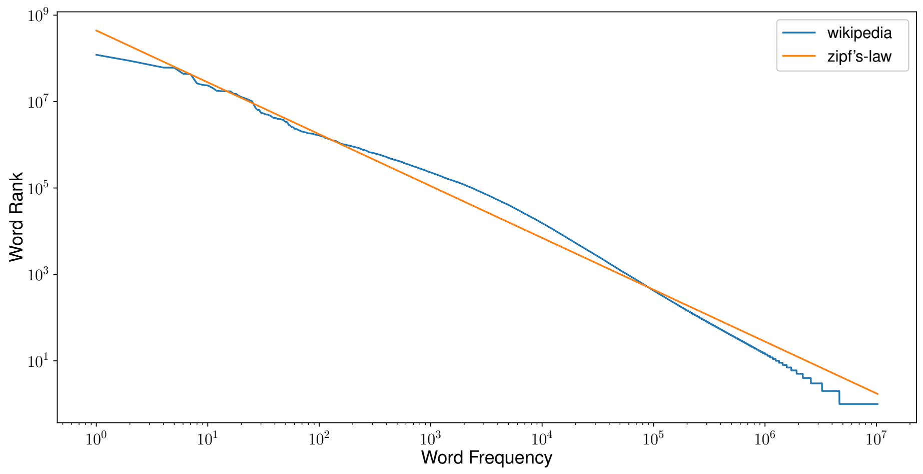

Figure 5: Plot of the real word frequencies (case-insensitive) in the English Wikipedia and the word frequencies according to Zipf's law for $s = 1.2$ and a corpus of Wikipedia's size.

This suggests that above a certain vocabulary size the performance of an embedding is not improved to a great extent, as it is very unlikely that our embedding has to deal with these words frequently. Additionally, smaller vocabulary sizes are often necessary to limit the vocabulary in order to limit the memory consumption, smaller embeddings, for example, enable the algorithm to calculate analogies much faster. Currently, however, it is not clear how large such a vocabulary should be during training in order to avoid significant performance losses. The following experiment is therefore intended to show to which extent it is helpful to increase the vocabulary-size.

4.3.1 Experiment

We created five embeddings using the configuration of Table 11 and vocabulary sizes of 50K, 100K, 500K, 1M, 5M with our algorithm. The five embeddings are evaluated using the similarity task (section 4.1.1) and analogy task (section 4.1.2).

epochs	1
window-size	5
embedding-size	200
negative-sampling-rate	16
subsampling-threshold	10^{-5}
batch-size	128
start-learning-rate	2.5
decay	linear to zero

Table 11: Configuration for the experiments described in Section 4.3 and 4.4.

4.3.2 Similarity Results

Figure 6 shows the evaluation results for each embedding and dataset. The x-axis of the plot shows the size of the embedding's vocabulary. The y-axis on the other hand the Spearman correlation for an embedding's result on the used datasets.

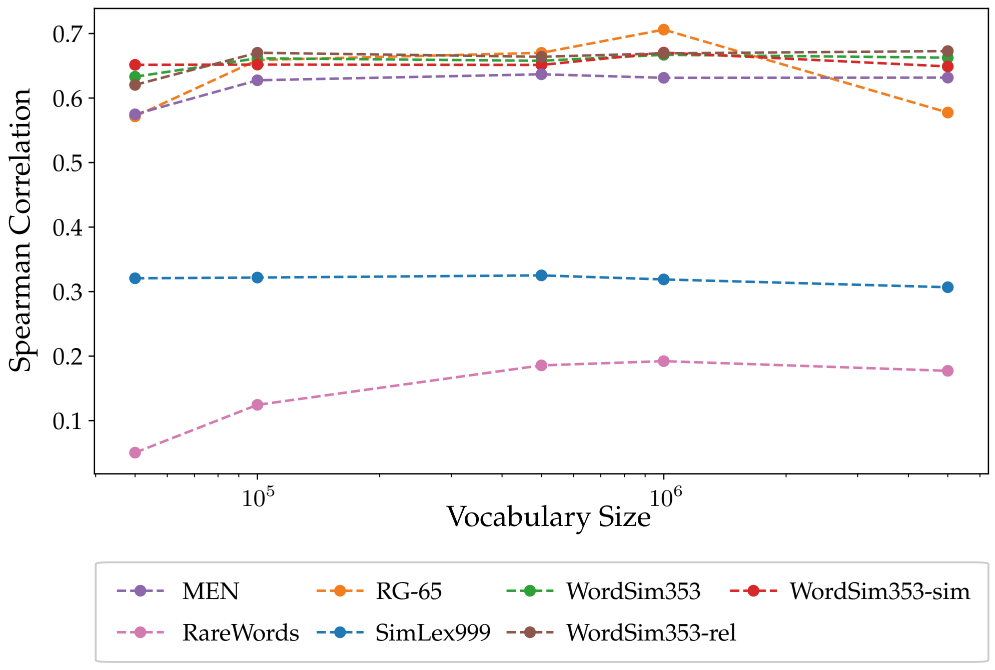

Figure 6: Similarity evaluation results for different vocabulary sizes.

Only on WordSim353-sim the 50K embedding perfomed better than on the 100K embedding and it performed only 0.05% better on that dataset. For SimLex999 the 100K embedding performs only a little bit better than the 50K embedding. The results for both datasets vary only by a small margin. On the other hand the 5M embedding only performs better than the 1M embedding in case of *WordSim353-rel*. This could be an indicator that a vocabulary size above a certain threshold generates more noise. Most of the datasets performed better with bigger vocabulary sizes between 50K and 1M. Vocabulary sizes around 1M should therefore be an advisable choice for most embeddings evaluating similarity.

4.3.3 Analogy Results

The five embeddings are also evaluated using the analogy task. The results are shown in figure 7. The x-axis of the plot shows the size of the embedding's vocabulary and the y-axis the embedding's accuracy on the used datasets.

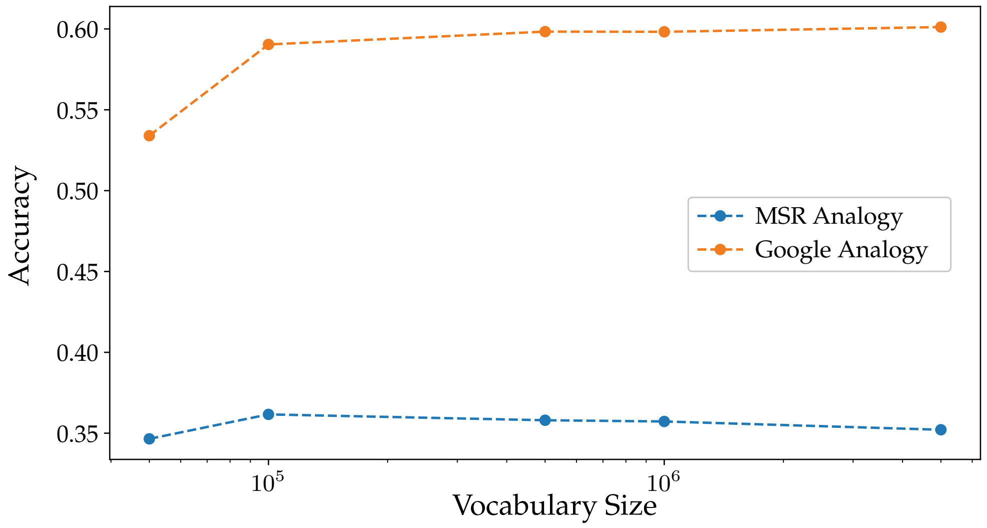

Figure 7: Analogy evaluation results for different vocabulary sizes.

On both datasets the embedding with a vocabulary size of 500K performs better than on the smaller ones. For the *MSR Analogy* dataset the 500K embedding performed best and the bigger the vocabulary the more the performance descends. The highest accuracy for the *Google Analogy* dataset was achieved with a vocabulary size of 5M. However, this 5M embedding is only 0.4% better than the 500K embedding. Therefore, a vocabulary of 500K seems to be sufficient for a good analogy evaluation.

4.3.4 Experiment Conclusion

For similarity evaluation the 1M embedding worked best for most datasets and for analogy evaluation the 500K embedding resulted in the best average performance. For most embeddings we would therefore recommend to use a vocabulary-size between 500,000 and 1,000,000. If a space-saving embedding is required, for most datasets the vocabulary can be reduced to a size of 100,000 without a massive performance drop. We do not recommend reducing the vocabulary size further than 100,000 because the performance for the 50K embeddings was worse than the performance of the embeddings trained with a larger vocabulary size.

In terms of training speed, the vocabulary size had comparatively small effect. The training time on our fastest server approximately doubled with an increase in vocabulary from 50,000 to 5,000,000.

4.4 Noise Induced by Large Vocabularies

The calculation of an analogy word requires to pick the word with the smallest distance to a calculated point in the embedding's vector-space. Therefore, the probability to pick a wrong word increases with the vocabulary size to be evaluated. This problem could be even bigger if it is taken into account that many very rare words are poorly positioned in our vector space due to a lack of training data for them. The following experiment therefore aims for a better understanding about the extent of the negative influence of a large vocabulary on an analogy evaluation.

We identified the ranks of the words contained in the analogy evaluation datasets according to their frequency in our text corpus[19] and we determined 677,710 as rank of the least frequent appearing word from Google Analogy and 4,696,052 as the rank of the least frequent appearing word from MSR Analogy.

Thus, MSR analogy uses words that at least almost exhaust our vocabulary size. However, Google Analogies does not use such rare words and a vocabulary that is significantly smaller than our basic vocabulary is more suitable to provoke a memorable difference. We therefore use the embedding with a vocabulary size of 5,000,000 from Section 4.3 (Table 12) as first embedding and a subset of this embedding, which includes only the 677,710 most frequent words, as second embedding for our comparison.

[19] All words that were not contained in our biggest vocabulary of 5,000,000 words were ignored (if one of the datasets contains such rare words).

epochs	1
vocabulary-size	5,000,000
window-size	5
embedding-size	200
negative-sampling-rate	16
subsampling-threshold	10^{-5}
batch-size	128
start-learning-rate	2.5
decay	linear to zero

Table 12: Training configuration of the used embedding.

4.4.1 Analogy Evaluation

The evaluation results for the 5M embedding and its subset of 667,710 words are shown in Table 13.

Dataset	**Subset**	**Full Embedding**	**Difference**
Google Analogy	*0.602*	*0.601*	*+0.001* (+0.09%)
MSR Analogy	*0.353*	*0.352*	*+0.001* (+0.31%)

Table 13: Comparison of the evaluation results for an embedding with 5,000,000 words and a smaller subset of that embedding with 677,710 words.

The results of the analogy evaluation for both datasets improved due to the reduction of the evaluation vocabulary. However, the improvement is slight and suggests that misclassifications are caused rather by a biased calculated analogy point rather than by badly positioned words in the embedding that have nothing to do with the actual calculation.

4.4.2 Experiment Conclusion

Interestingly, there is not only a slight improvement for Google Analogy, when reducing the vocabulary-size in the evaluation. There is also an improvement for the MSR-Analogy dataset, although it also contains words that are much rarer than those in the smaller embedding. This shows that rare words are badly positioned in the embedding space and thus questions the use of a very large vocabulary even more. And it reinforces our recommendation for a vocabulary-size from Section 1 by showing that vocabularies with more than 1,000,000 words do not need perform better than smaller ones.

4.5 Hidden Layer Dropout

Many supervised machine learning algorithms tend to overfit to given training data. This leads to poor performance if the task is to be performed with data or contexts other than what was used while training.

The word2vec algorithm is not a supervised machine learning appoach in the classical sense and the term overfitting is not appropriate for the algorithm, but a supervised machine learning task that may suffer from similar problems is used in its neural network to create the embedding. There are some regularization methods to prevent overfitting in neural networks, which can also be applied to the word2vec algorithm. [Mu et al., 2018] have shown that a regularization method which is called quadratic regularization can improve the performance of the word2vec algorithm.

There is another popular regularization method called *dropout* [Srivastava et al., 2014]. Dropout can be applied to a neural network layer and causes randomly selected elements in the resulting vector of that layer to be set to zero. This forces the algorithm to distribute information in such a way that it can still accomplish its task despite the missing information.

For this experiment, two embeddings created with the hyperparamters specified in Table 14 are used. One of them was created with a dropout rate of 50% on word2vec's hidden layer and the other one was created without usage of a dropout.

epochs	1
vocabulary-size	1,000,000
window-size	5
embedding-size	200
negative-sampling-rate	16
subsampling-threshold	10^{-5}
batch-size	128
decay	linear to zero

Table 14: Shared configuration for the dropout experiment embeddings.

[Srivastava et al., 2014] recommend using a learning-rate between 10 and 100 times higher than what would be used without dropout. However, all learning-rates that we tested higher than 5 only increased the loss while training. Therefore the embedding created with dropout uses a start-lerning-rate of five[20]. For the embedding created without dropout the start-learning-rate was 2.5.

[20]We have also tried dropout with a start learning rate of 2.5. The results for all datasets were slightly worse than with the start-learning-rate of five.

4.5.1 Similarity Evaluation

Table 15 shows the evaluation results for the embeddings created with and without use of dropouts in the hidden layer. The dropout embedding performed much worse for nearly all datasets. RG-65 was the only dataset were both embeddings performed nearly equally good.

Dataset	Dropout	No Dropout	Difference
MEN	*0.558*	*0.636*	*-0.078* (-12.26%)
RareWords	*0.155*	*0.179*	*-0.024* (-13.40%)
RG-65	*0.684*	*0.680*	*+0.004* (+0.59%)
SimLex999	*0.253*	*0.317*	*-0.064* (-20.19%)
WordSim353	*0.562*	*0.673*	*-0.111* (-16.49%)
WordSim353-rel	*0.564*	*0.674*	*-0.110* (-16.32%)
WordSim353-sim	*0.573*	*0.673*	*-0.046* (-14.86%)

Table 15: Similarity evaluation results the embedding which uses a dropout and the embedding without dropout.

4.5.2 Analogy Evaluation

Table 16 shows the analogy evaluation results for both embeddings. The embedding created with dropout performs worse than the version without for both datasets.

Dataset	Dropout	No Dropout	Difference
Google Analogy	*0.493*	*0.605*	*-0.121* (-18.51%)
MSR Analogy	*0.213*	*0.479*	*-0.266* (-55.53%)

Table 16: Analogy evaluation results the embedding which uses a dropout and the embedding without dropout.

4.5.3 Experiment Conclusion

The dropout embedding performed much worse for the similarity task and for the analogy task. Therefore it does not seem to be a good idea to use dropout for word2vec's hidden layer.

4.6 Impact of the Leaning Rate in word2vec

The word2vec algorithm consists of a neural network which aims to converge as close as possible to the point with minimum loss. A high learning rate helps to get quickly closer to that goal. But at the same time, however, it often leads to overstating the case.

Linear Decay was implemented by [Mikolov et al., 2013a] in the original word2vec algorithm. The algorithm starts with a base learning rate α, and decreases it linearly until it reaches a final learning rate α_f[21]. We calculate the learning-rate for a particular training batch α_t in a training set of T batches using the following equation[22]:

$$\alpha_t = (\alpha - \alpha_f) \cdot (1 - \frac{t}{T}) + \alpha_f \tag{19}$$

Exponential Decay is a decay method, which usually starts with a faster decay on the learning rate, but slows down after a while. We calculate the learning rate for a training batch α_t in a training set of T batches, depending on a decay rate d and a decay amplifier a as shown in the following equation[23].

$$\alpha_t = \alpha \cdot d^{\frac{a \cdot t}{T}} \tag{20}$$

Our evaluation makes a comparison between both decay methods and a constant learning rate against each other. We used a base learning rate of 2.5 for all three experiments and decayed it for the linear decay to 0.0001 as final learning rate α_f. For the exponential decay we used a decay rate d of 0.99 and a decay amplifier a of 500. The plotted learning-rate curve in relation to the current training step is shown in Figure 8.

[21]The final learning rate α_f for this is usually close to zero.

[22]We used `https://www.tensorflow.org/api_docs/python/tf/train/polynomial_decay` for our linear learning rate decay.

[23]We used `https://www.tensorflow.org/api_docs/python/tf/train/exponential_decay` for our exponential learning rate decay.

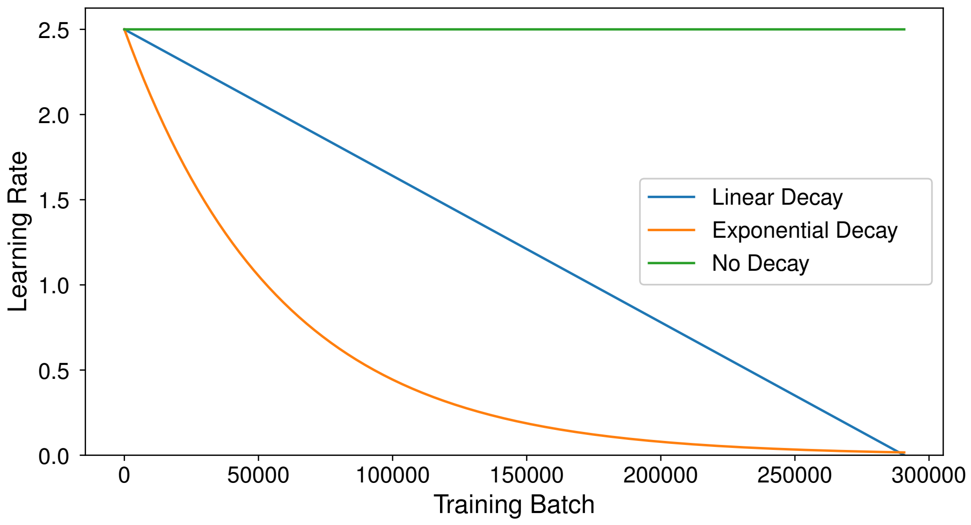

Figure 8: Plot of the decaying learning rates according to the training step for all three evaluated learning rate methods.

The hyperparameters for all experiments apart from the learning-rate and appart from the learning rate decay configurations can be found in Table 17.

epochs	1
vocabulary-size	$1,000,000$
window-size	5
embedding-size	200
negative-sampling-rate	16
subsampling-threshold	10^{-5}
batch-size	128
start-learning-rate	2.5

Table 17: Configuration for the experiments

4.6.1 Similarity Evaluation

Table 18 shows the evaluation results for the embeddings created with a linear and exponential decay and without learning rate decay. It shows that the embedding, created without a decay performed best for most similarity datasets, but the embedding created with a linear decay performed best on WordSim353, WordSim353-sim and WordSim353-rel.

Dataset	No Decay	Linear	Exponential
MEN	**0.639**	*0.636* (-0.4%)	*0.574* (-10.1%)
RareWords	**0.201**	*0.179* (-10.9%)	*0.156* (-22.4%)
RG-65	**0.699**	*0.680* (-2.7%)	*0.498* (-28.8%)
SimLex999	**0.319**	*0.317* (-0.6%)	*0.264* (-17.2%)
WordSim353	*0.633* (-5.9%)	**0.673**	*0.617* (-8.3%)
WordSim353-rel	*0.629* (-6.6%)	**0.674**	*0.643* (-4.5%)
WordSim353-sim	*0.641* (-4.8%)	**0.673**	*0.610* (-9.4%)

Table 18: Similartity evaluation results for the embeddings created with *Linear* or *Exponential* decay and with *No Decay* on the learning rate.

The results indicate that rare words in particular benefit if a high learning rate is maintained for a long time. Our results show that the relative difference between the embedding without learning rate decay and the embedding with linear decay is especially high for the RareWords dataset.

4.6.2 Analogy Evaluation

Table 19 shows the analogy evaluation results for the three embeddings. It shows that the embedding created with a linear decay performed best on both datasets.

Dataset	No Decay	Linear	Exponential
Google Analogy	*0.528* (-12.7%)	**0.605**	*0.463* (-23.5%)
MSR Analogy	*0.383* (-20.0%)	**0.479**	*0.300* (-37.4%)

Table 19: Analogy evaluation results for the embeddings created with a *Linear* or *Exponential* decay and with *No Decay* on the learning rate.

The results suggest that the embedding created without decaying the learning rate is less accurate in mapping the vectors and that the embedding created with exponential decay has not learned enough.

4.6.3 Loss Inspection

The algorithm shares almost the same implicit goal for the creation of all three embeddings, namely to minimize training loss on data that is equal except for a degree of randomization in order. Therefore, the training loss should be a good indicator of the expected performance of the resulting embeddings. We logged the loss of the neural network through its training steps and plotted it in figure 9 to get a deeper insight into the expected performance of an embedding per training step.

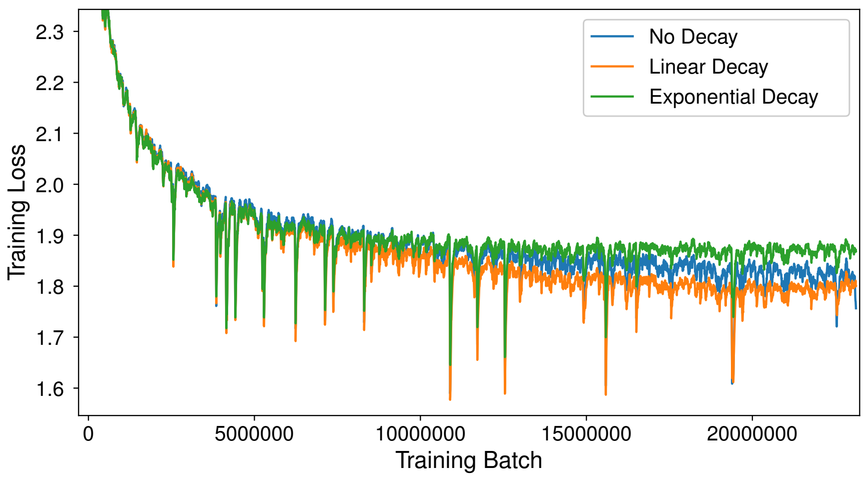

Figure 9: Plot of the decayed learning rates according to the training step for the three evaluated learning rate methods.

The best performing loss curve in the trainings loss plot is the one that uses a linear decay on its learning rate. The exponentially decayed algorithm has shown the worst performance. This does not mean that an exponential decay is generally a bad solution for embeddings. However, it shows that the learning rate was reduced too quickly in this case. For a larger dataset or training over several epochs, the results may prefer a different decay method. However, the results show that the training loss often is a good indicator of embedding performance, but they also show that the loss is strongly biased by the fact that it poorly represents the expected embedding performance for uncommon words.

4.6.4 Conclusion and Further Work

The results show that a rapid decay of the learning rate strongly reduces the performance for rare words. It could be advantageous to keep the learning rate high for a long time and then reduce it gradually, especially when training over several epochs.

4.7 Adam Optimizer

The weight matrices in an neural network are adjusted by an optimizer. A stochastic gradient decent optimizer (SGD) is typically applied on the word2vec weight matrices, but there are many other optimizers for neural networks, which can also be applied to word2vec and it is not clear which one performs best for the algorithm.

A very popular optimizer is called Adam (short for Adaptive Moment Estimation). In contrast to SGD it estimates an adaptive individual learning rate for different parameters based on mean and variance of the gradients [Kingma and Ba, 2014]. Adam's parameter specific learning rates exponentially decay[24] over time [Kingma and Ba, 2014]. This helps to reduce the probability for the optimizer to overstate the case.

The experiment compares the performances of embeddings created with Adam[25] on one hand and SGD on the other hand to determine which one is best suited for the word2vec algorithm. The common hyperparameters for both embeddings are listed in Table 20. For the embedding created with SGD a linear decay in learning rate is used. For the embedding created with Adam, there is no additional learning-rate decay is used, because Adam manages its learning-rate decay internally. The Adam optimizer performs much better with a start-learning rate that is significantly lower than one which performs good with SGD. Therefore a start-leaning-rate of 2.5 was applied to create the SGD embedding and a start-learning-rate of 0.005 was used for the embedding created with use of the Adam optimizer.

epochs	1
vocabulary-size	1,000,000
window-size	5
embedding-size	200
negative-sampling-rate	16
subsampling-threshold	10^{-5}
batch-size	128

Table 20: Basic configuration for the creation of the SGD Embedding and the Adam Embedding.

[24]Adam introduces the decay rates $\beta_1 < 1$ and $\beta_2 < 1$ as hyperparameters for the algorithm. We use the TensorFlow's default values as decay rates for Adam, which are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Adam applies those decay rates on the calculated mean and variance [Kingma and Ba, 2014].

[25]We use the Lazy Adam Optimizer from TensorFlow instead of normal Adam. Lazy Adam is much faster with sparse data. We first tried the usual Adam, but we stopped it after several hours without any significant progress (Training was performed on the CPU, for the GPU this may be a less significant problem). Lazy Adam processed the same set of training-data much faster than Adam but it needs nearly $\frac{1}{3}$ more training time than SGD on an comparable amount of training data.

4.7.1 Similarity Evaluation

Table 21 shows the evaluation results for the embeddings created with Adam and with SGD. The embedding created using the Adam optimizer performs worse for most datasets, but it performs significantly better than the embedding created using SGD on the RareWords dataset.

Dataset	Adam	SGD	Difference
MEN	*0.609*	*0.636*	*-0.027* (-4.25%)
RareWords	*0.219*	*0.179*	*+0.040* (+22.35%)
RG-65	*0.638*	*0.680*	*-0.042* (-6.18%)
SimLex999	*0.322*	*0.317*	*+0.005* (+1.58%)
WordSim353	*0.610*	*0.673*	*-0.063* (-9.36%)
WordSim353-rel	*0.601*	*0.674*	*-0.073* (-10.83%)
WordSim353-sim	*0.626*	*0.673*	*-0.047* (-6.98%)

Table 21: Similarity evaluation results for the embeddings created with use of the Adam optimizer and with SGD.

4.7.2 Analogy Evaluation

Table 22 shows the analogy evaluation results for both embeddings. The embedding created with use of the Adam optimizer performs worse than the SGD version for both datasets.

Dataset	Adam	SGD	Difference
Google Analogy	*0.485*	*0.605*	*-0.120* (-19.83%)
MSR Analogy	*0.353*	*0.479*	*-0.126* (-26.30%)

Table 22: Analogy evaluation results for the embeddings created with use of the Adam optimizer and with SGD.

4.7.3 Conclusion

Possibly, the performance of embeddings created using the Adam Optimizer can be further improved by fine-tuning the learning-rate decay parameters β_1 and β_2. However, we do not expect any performance improvement to justify using the Adam or Lazy Adam optimizer, as these were significantly slower than our implementation with SGD and potentially achievable performance improvements should also be achievable through more training epochs with SGD.

5 Related Work

This chapter presents previous research on topics similar to the two main contributions we provide. The first of these contributions is our investigation about how embeddings should be evaluated. The second is, how does the word2vec model react to certain model or hyperparameter modifications.

5.1 How to Evaluate Word Embeddings

The most comprehensive publication we have found that examines how to evaluate word embeddings best is [Bakarov, 2018]. It summarizes theoretical findings on different evaluation tasks on word embeddings, provides overviews and critiques of available datasets, and suggests some new evaluation methods. Thus [Bakarov, 2018] gives a good overview, but little help in deciding which of these evaluation methods are actually best.

Another paper, which tries to determine how to evaluate word embeddings is [Faruqui et al., 2016], which concentrates on the similarity evaluation of word embeddings. It also summarize the previous research but only for one task and does that in more depth. These authors heavily criticize the usual gold standard methods for similarity evaluation. Furthermore they claim that similarity evaluation, which is the most used evaluation task for word embeddings, is currently not meaningful for an embedding's overall performance and that much further research is needed to evaluate embeddings properly.

5.2 Model and Hyperparameter Modifications

The second major contribution topic, mentioned above, that we dealt with in this thesis is reflected in literature as well. Unfortunately many papers only deal with this topic in a side-note. This section presents papers devoted to small modifications of word2vec.

A journal article focussing on hyperparameter changes in algorithms for the creation of word embeddings is [Levy et al., 2015]. These authors showed that older, count-based models also perform well, if we turn their hyperparameters more towards those used in word2vec. Furthermore, these authors showed how important these small changes can be.

It's hard to determine which context window size is appropriate to best describe a words's semantic, because the perfect size may vary from word to word. [Huang et al., 2012] try therefore to provide local and global context information for embeddings.

word2vec's model does not reflect the order of the words in the context window. This may be appropriate for many tasks concerning a word's semantics but it is a problem when it comes to tasks with a more syntatic focus, because for many of those tasks the order of words is crucial. [Ling et al., 2015a] modify the word2vec model to better reflect the words order. They evaluate their word embeddings with a part-of-speech tagging task and show that their approach works well for this kind of problem.

An important factor for hyperparameter choices is not only their value, but also their combination. [Krebs and Paperno, 2016] identifies some combinations, and hyperparameters, which are highly dependent on each other.

Words are usually represented as sequences of characters. These sequences, even if we do not know the words, can often tell us a lot about these words. However, the word2vec model does not use this kind of information to create the word embeddings. Therefore [Bojanowski et al., 2017] modifies the word2vec to use character n-grams[26] for the neural network's training progress.

Another popular neural network optimization technique that can be applied to word2vec is regularization. [Mu et al., 2018] evaluate the performance of a word2vec model modified using quadratic regularization.

The word embeddings in word2vec are created using the weight matrix of the hidden layer. Alternatively, word2vec's output layer also has a weight matrix which can be used instead to create the word embedding. [Press and Wolf, 2017], therefore evaluates word embeddings created using the output layers and they manage to use this output layer matrix to improve their word embeddings performance.

There is also a paper focusing on changes to the analogy computation. [Levy and Goldberg, 2014b] proposed two new methods to calculate analogies, with use of a word embedding. They call those new methods *'3CosMul'* and *'PairDirection'* and show that these methods also work very well. They introduce a new name *'3CosAdd'*, for the conventional model for analogy computation, to differentiate between those three options.

[26]A n-gram of characters is a sequence of n characters. In this case these characters are a subset of all characters the word is composed of.

6 Conclusion

This thesis first explains the word2vec algorithm, which is used to create word embeddings. Subsequently, it was examined how to evaluate word embeddings, and how do these embeddings react to some hyperparameter and model changes.

6.1 Evaluation

For the evaluation of such word embeddings, the literature was analyzed for best practices. Recommendations for methods and datasets for the evaluation of word embeddings were extracted. We distinguish three common classes of evaluation methods.

Word Similarity The most commonly used datasets to evaluate the word similarity task we have identified were *WordSim353*, *RG65*, *RareWords* and *MEN*. We recommend the use of these datasets in particular also to ensure a certain comparability between different publications. The most recent dataset we found in our literature study was *SimLex999*. We recommend its use especially if no downstream machine learning tasks are evaluated, as this can give at least an indication for the expected performance.

Analogy The primary used datasets for the calculation of analogies using word embedding are *Google Analogy* and *MSR Analogy*. Here, we found that Google Analogy had by far the most uses. However, very rare words in particular are hardly represented in this dataset. In the literature study we found only one other dataset (*SemEval2012*) apart from these two datasets, which was used for the evaluation of the analogy task.

Downstream Machine Learning The most frequent used downstream machine learning tasks we found are *Sentiment Analysis*, *Named Entity Recognition* and *Part-of-Speech Tagging*.

In addition, we found some evaluation methods that do not belong in these three categories. However, such evaluations, were a rare exception in the examined literature.

6.2 Experiments

On the other hand, we have applied the identified evaluation techniques to our experiments to investigate how embeddings behave for different hyperparameters and model choices.

Lowercase We evaluated how word embeddings behave if they were created using an only lowercase text corpus. It was found found that these embeddings perform pretty good on the word similarity task. And their performance was especially remarkable for very rare words. But for analogy calculations, these embeddings have shown a very bad performance and we would therefore not recommend using these kind of embeddings, if the embedding should be used for the calculation of word analogies.

Vocabulary Size We evaluated embeddings created by word2vec with different vocabulary sizes. And we found that the embeddings show a pretty decent performance for vocabularies, which include the most common 100.000 words and more. But their performance gets better and better, with a bigger vocabulary until a certain threshold. After that the performance decreases for most datasets.

Hidden Layer Dropout It was evaluated how word2vec reacts to dropout applied to its hidden layer. And the results have shown, that dropouts on word2vec's hidden layer are not a good idea.

Learning Rate It was evaluated how word2vec reacts to the learning rate changes. We found that it is on one hand especially helpful for rare words, to keep a high learning rate for a long time while training. On the other hand the experiment has shown, that a learning-rate decay helps to create better word representations.

Adam Optimizer It was evaluated, how the the word2vec algorithm behaves if it was optimized using the Adam optimizer instead of stochastic gradient decent. We found that word2vec does not work well, with Adam.

6.3 Further Work

Our results from Section 4.4 show that rare words are very poorly positioned in word2vec resulting vector space. It would be interesting to create a plot, which visualizes the relative number of correctly classified analogies for fixed intervals of word ranks. Such a plot could help to get even more insights into the problems of rare words in the analogy evaluation of word embeddings.

In Section 4.2, lowercase embeddings were evaluated which performed particularly well in evaluating the similarity of rare words. However, the exact positioning of the vectors necessary for the analogy calculations could often not be achieved. To combine the advantages of lower case and mixed case embeddings, a hybrid solution of both approaches could be tested. In a first training epoch it could be tested to train all spellings of a word independent of upper and lower case which are part of the vocabulary simultaneously. And in the following training epochs, the training could be restricted exclusively to the word present in the corpus.

Another idea is to increase or decrease the subsampling-rate, which was proposed by [Mikolov et al., 2013b], while training. We belive that this should also have big effects on the performance of rare words.

References

[Al-Azani and El-Alfy, 2017] Al-Azani, S. and El-Alfy, E. M. (2017). Using word embedding and ensemble learning for highly imbalanced data sentiment analysis in short arabic text. In *The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017) / The 7th International Conference on Sustainable Energy Information Technology (SEIT 2017), 16-19 May 2017, Madeira, Portugal*, pages 359–366.

[Al-Rfou et al., 2013] Al-Rfou, R., Perozzi, B., and Skiena, S. (2013). Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*, pages 183–192.

[Bakarov, 2018] Bakarov, A. (2018). A survey of word embeddings evaluation methods. *CoRR*, abs/1801.09536.

[Baker et al., 2014] Baker, S., Reichart, R., and Korhonen, A. (2014). An unsupervised model for instance level subcategorization acquisition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 278–289.

[Bansal et al., 2014] Bansal, M., Gimpel, K., and Livescu, K. (2014). Tailoring continuous word representations for dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 809–815.

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL*, 5:135–146.

[Bollegala et al., 2015] Bollegala, D., Maehara, T., and Kawarabayashi, K. (2015). Unsupervised cross-domain word representation learning. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 730–740.

[Bruni et al., 2014] Bruni, E., Tran, N., and Baroni, M. (2014). Multimodal distributional semantics. *J. Artif. Intell. Res.*, 49:1–47.

[Bruni et al., 2012] Bruni, E., Uijlings, J. R. R., Baroni, M., and Sebe, N. (2012). Distributional semantics with eyes: using image analysis to improve computational representations of word meaning. In *Proceedings of the 20th ACM Multimedia Conference, MM '12, Nara, Japan, October 29 - November 02, 2012*, pages 1219–1228.

[Chiarello et al., 1990] Chiarello, C., Burgess, C., Richards, L., and Pollock, A. (1990). Semantic and associative priming in the cerebral hemispheres: Some words do, some words don't... sometimes, some places. *Brain and language*, 38(1):75–104.

[Chiu et al., 2016] Chiu, B., Korhonen, A., and Pyysalo, S. (2016). Intrinsic evaluation of word vectors fails to predict extrinsic performance. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, RepEval@ACL 2016, Berlin, Germany, August 2016*, pages 1–6.

[Clark and Manning, 2016] Clark, K. and Manning, C. D. (2016). Improving coreference resolution by learning entity-level distributed representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

[Dhillon et al., 2011] Dhillon, P. S., Foster, D. P., and Ungar, L. H. (2011). Multi-view learning of word embeddings via CCA. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 199–207.

[Dubin, 2004] Dubin, D. (2004). The most influential paper gerard salton never wrote. *Library Trends*, 52(4):748–764.

[Faruqui et al., 2015] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E. H., and Smith, N. A. (2015). Retrofitting word vectors to semantic lexicons. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1606–1615.

[Faruqui and Dyer, 2014] Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2014, April 26-30, 2014, Gothenburg, Sweden*, pages 462–471.

[Faruqui et al., 2016] Faruqui, M., Tsvetkov, Y., Rastogi, P., and Dyer, C. (2016). Problems with evaluation of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, RepEval@ACL 2016, Berlin, Germany, August 2016*, pages 30–35.

[Finkelstein et al., 2002] Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2002). Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131.

[Ghannay et al., 2016] Ghannay, S., Favre, B., Estève, Y., and Camelin, N. (2016). Word embedding evaluation and combination. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016.*

[Gladkova and Drozd, 2016] Gladkova, A. and Drozd, A. (2016). Intrinsic evaluations of word embeddings: What can we do better? In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, RepEval@ACL 2016, Berlin, Germany, August 2016*, pages 36–42.

[Goldberg, 2016] Goldberg, Y. (2016). A primer on neural network models for natural language processing. *J. Artif. Intell. Res.*, 57:345–420.

[Goldberg and Levy, 2014] Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722.

[Halawi et al., 2012] Halawi, G., Dror, G., Gabrilovich, E., and Koren, Y. (2012). Large-scale learning of word relatedness with constraints. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 1406–1414.

[Hill et al., 2015] Hill, F., Reichart, R., and Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.

[Huang et al., 2012] Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers*, pages 873–882.

[Jurgens et al., 2012] Jurgens, D., Mohammad, S., Turney, P. D., and Holyoak, K. J. (2012). Semeval-2012 task 2: Measuring degrees of relational similarity. In *Proceedings of the 6th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2012, Montréal, Canada, June 7-8, 2012*, pages 356–364.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kiros et al., 2014] Kiros, R., Zemel, R. S., and Salakhutdinov, R. (2014). A multiplicative model for learning distributed text-based attribute representations. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2348–2356.

[Koehn et al., 2007] Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*.

[Krebs and Paperno, 2016] Krebs, A. and Paperno, D. (2016). When hyperparameters help: Beneficial parameter combinations in distributional semantic models. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics, *SEM@ACL 2016, Berlin, Germany, 11-12 August 2016*.

[Kusner et al., 2015] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 957–966.

[Le and Mikolov, 2014] Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196.

[Levy and Goldberg, 2014a] Levy, O. and Goldberg, Y. (2014a). Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 302–308.

[Levy and Goldberg, 2014b] Levy, O. and Goldberg, Y. (2014b). Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning, CoNLL 2014, Baltimore, Maryland, USA, June 26-27, 2014*, pages 171–180.

[Levy and Goldberg, 2014c] Levy, O. and Goldberg, Y. (2014c). Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2177–2185.

[Levy et al., 2015] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225.

[Ling et al., 2015a] Ling, W., Dyer, C., Black, A. W., and Trancoso, I. (2015a). Two/too simple adaptations of word2vec for syntax problems. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1299–1304.

[Ling et al., 2015b] Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luís, T. (2015b). Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1520–1530.

[Liu, 2010] Liu, B. (2010). Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing, Second Edition.*, pages 627–666.

[Luong et al., 2013] Luong, T., Socher, R., and Manning, C. D. (2013). Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*, pages 104–113.

[Maas et al., 2011] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119.

[Mikolov et al., 2013c] Mikolov, T., Yih, W., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751.

[Miller and Charles, 1991] Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28.

[Mitra et al., 2017] Mitra, B., Diaz, F., and Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1291–1299.

[Mnih and Kavukcuoglu, 2013] Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2265–2273.

[Mu et al., 2018] Mu, C., Yang, G., and Yan, Z. (2018). Revisiting skip-gram negative sampling model with rectification. *arXiv preprint arXiv:1804.00306*.

[Musto et al., 2016] Musto, C., Semeraro, G., de Gemmis, M., and Lops, P. (2016). Learning word embeddings from wikipedia for content-based recommender systems. In *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, pages 729–734.

[Neelakantan et al., 2015] Neelakantan, A., Shankar, J., Passos, A., and McCallum, A. (2015). Efficient non-parametric estimation of multiple embeddings per word in vector space. *CoRR*, abs/1504.06654.

[Nikfarjam et al., 2015] Nikfarjam, A., Sarker, A., O'Connor, K., Ginn, R. E., and Gonzalez, G. (2015). Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. *JAMIA*, 22(3):671–681.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

[Powers, 1998] Powers, D. M. W. (1998). Applications and explanations of zipf's law. In *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning, NeMLaP/CoNLL 1998, Macquarie University, Sydney, NSW, Australia, January 11-17, 1998*, pages 151–160.

[Press and Wolf, 2017] Press, O. and Wolf, L. (2017). Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pages 157–163.

[Qiu et al., 2014] Qiu, L., Cao, Y., Nie, Z., Yu, Y., and Rui, Y. (2014). Learning word representation considering proximity and ambiguity. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1572–1578.

[Radinsky et al., 2011] Radinsky, K., Agichtein, E., Gabrilovich, E., and Markovitch, S. (2011). A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 337–346.

[Resnik, 1995] Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 448–453.

[Rong, 2014] Rong, X. (2014). word2vec parameter learning explained. *CoRR*, abs/1411.2738.

[Rubenstein and Goodenough, 1965] Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

[Salton, 1971] Salton, G. (1971). The smart retrieval system-experiments in automatic document processing. *Englewood Cliffs*.

[Salton and McGill, 1984] Salton, G. and McGill, M. (1984). *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company.

[Schnabel et al., 2015] Schnabel, T., Labutov, I., Mimno, D. M., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 298–307.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[Tang et al., 2014a] Tang, B., Cao, H., Wang, X., Chen, Q., and Xu, H. (2014a). Evaluating word representation features in biomedical named entity recognition tasks. *BioMed research international*, 2014:240403.

[Tang et al., 2014b] Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., and Qin, B. (2014b). Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1555–1565.

[Turian et al., 2010] Turian, J. P., Ratinov, L., and Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*, pages 384–394.

[Turney and Pantel, 2010] Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *J. Artif. Intell. Res.*, 37:141–188.

[Yang and Powers, 2006] Yang, D. and Powers, D. (2006). Word similarity on the taxonomy of wordnet.

[Zou et al., 2013] Zou, W. Y., Socher, R., Cer, D. M., and Manning, C. D. (2013). Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1393–1398.