A Neural Embedding Compressor for Scalable Document Search

Presented to obtain the Master of Science (M.Sc.)

Felix Hamann
October 22, 2018

Wiesbaden

Advised by
Prof. Dr. Adrian Ulges & Prof. Dr. Dirk Krechel
Hochschule RheinMain | DCSM

Deep Content Analytics

Abstract

Dense vectorial word representations trained on large text corpora encode semantic relatedness into a spatial property. Recent work has utilised this information to implement a similarity measure for documents: the Word Movers Distance (WMD) which formulates a transport problem based on these word vectors. However, the necessary pairwise term comparison of float embeddings is time consuming. The Relaxed Word Moving Distance (RWMD) lifts some of the tight constraints of the WMD to speed up the calculation. In this work a method for a fast calculation of document similarities based on the RWMD is developed: the Relaxed Hamming Word Movers Distance (RHWMD). Instead of calculating the spatial distance between word vectors in real-valued Euclidean space, the Hamming distance between binary hash representations of these vectors is used. These binary hashes are obtained by training a neural auto-encoder network on the embeddings. The encoder-decoder produces well suited approximations of the original embeddings and retains the spatial characteristics of the embeddings to one another. The RHWMD is evaluated on a German information retrieval data set and outperforms the de-facto standard Okapi BM25 for corpus subsets with less than 100k documents. The method also surpasses Term Frequency-Inverse Document Frequency (TF-IDF) and the WMD. When used in combination with BM25 for fast pre-selection and subsequent re-ranking at most baseline performance is achieved. The computation gains a 20-fold speedup increase over WMD standard implementations. This first proof of concept shows the potential of the approach for retrieval and clustering.

VNGOL

I would like to thank Prof. Dr. Adrian Ulges, Marco Wrzalik, Markus Eberts & Johannes Villmow for their helpful feedback and valuable discussions.

Erklärung gemäß BBPO

Ich versichere, dass ich die Master-Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Wiesbaden
22. Oktober 2018

Felix Hamann

Erklärung zur Verwendung der Masterthesis

Hiermit erkläre ich mein Einverständnis mit den im folgenden aufgeführten Verbreitungsformen dieser Abschlussarbeit:

Verbreitungsform	**Ja**	**Nein**
Einstellung der Arbeit in die Hochschulbibliothek mit Datenträger	×	
Einstellung der Arbeit in die Hochschulbibliothek ohne Datenträger	×	
Veröffentlichung des Titels der Arbeit im Internet	×	
Veröffentlichung der Arbeit im Internet	×	

Wiesbaden
22. Oktober 2018

Felix Hamann

LAVIS

Hochschule RheinMain

Contents

Chapter 1

Preface

The rise of information and communications technology in the 20th century transformed many societies, businesses and research in such way that information is now one if not even *the* central resource. Whole businesses, such as Alphabet, IBM, Oracle, SAP (to name only some of the largest) are primarily concerned with aggregating, filtering and processing information. The most successful tech companies apart from hardware manufacturers are those who offer services or products working with information. As hardware became more and more efficient and cheap, data even in large quantities could be stored and searched at any time. The manifestation of the Internet in the late 20st century offered everyone the possibility to become a content creator themselves and thus the amount of produced content grows at a much larger rate than can be conceived by any person or even indexing system. Now people, businesses, governments and even technical devices produce, store and retrieve information continuously using this infrastructure.

The amount of this data — be it text, images, video, audio or others — is hard to conceive. Google claims to have answered $1.2 * 10^{12}$ text search queries in 2012 [74]. Information Retrieval (IR) systems are devised for searching and sorting through the data heaps. The most successful ones are those who are best in selecting the most relevant documents and sorting them by their relevancy such that the needs of the user are satisfied. This, however, is a very difficult task with much research dating back to the very beginning of digitisation [40, 2, 5, 60]. The challenges include handling vast amounts of raw and at best only semi-structured data, the continuous addition of new data and of course devising a relevance scoring function itself. Computers are inherently good for processing statistical data but for a high quality text search it is often necessary to include knowledge about semantics in regards to content. Processing and analysing such textual content is the actively researched area of Natural Lan-

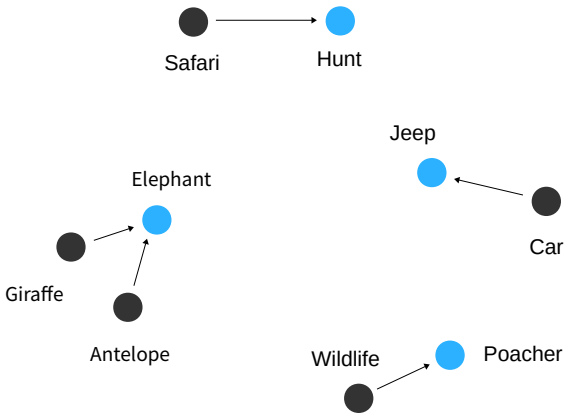


Figure 1.1: Illustration of determining the similarity of two documents (blue and grey) if their words are mapped into a space where their spatial distance encodes similarity. If the distance that each word needs to *travel* is small, both documents are very similar.

guage Processing (NLP). For businesses this becomes a critical part of operation as unused information is lost business value and might set them back against their competitors - regardless whether they offer the information as a service or use it internally to improve their production.

To efficiently process unstructured text many systems have been devised. One current approach is to *embed* words in a vector space [1, 55, 4, 52, 42]. These term embeddings can be used to determine the semantic similarity between word pairs by calculating their spatial distance. An IR system can work with these vectors by calculating the respective differences between all the words of a query and the candidate documents [35]. This, however, can not be done for large document collections because calculating all the distances is computationally too expensive. It also does not scale well when increasing the amount of candidate documents.

1.1 Motivation

This work introduces a novel relevance scoring function utilising both statistical data describing the distribution of words over documents and the term embeddings obtained using deep learning methodologies. This is motivated by the quite recent advances in research regarding the formerly addressed high quality word representations that are able to capture and encode semantics learned

from the domain they were trained on. With the new word embeddings there is much pioneering work done and many (most notably deep learning related) results are very promising. Calculations are computationally expensive because the embeddings are dense real-valued vectors in high-dimensional[1] Euclidean space. This challenge is addressed in this work by transforming the real-valued vectors into binary hash codes to enable a faster distance computation.

1.2 Overview

There are two major challenges concerning the devised system: First the production of binary hash codes for words and secondly the implementation of these codes for a practical search system: Therefore, the heart of this thesis concerns these two topics:

1) **The Neural Embedding Compressor:** In Chapter 3 the transformation of the Euclidean embedding space to Hamming space by use of a neural auto-encoder model is discussed in detail. After the formal definition a series of experiments is conducted using different parameter configurations to assess the models' ability to reconstruct the embedding vectors. The focus is then on using the model to transform real-valued vectors into binary hash codes for fast Hamming distance calculations between words. The chapter concludes with observations regarding the quality of the codes' nearest neighbours based on the Hamming distance.

2) **Information Retrieval:** Obtaining good nearest neighbours is the premise on which the subsequent Chapter 4 is built upon. The word pair distances are used in conjunction with statistical data regarding word distributions as the fundamental building blocks to implement a fast document similarity measure: the *Relaxed Hamming Word Movers Distance (RHWMD)*. A procedure is defined which utilises the hash codes to calculate a similarity score between documents. To use the RHWMD, a search system called the *Ungol Index* is implemented. The Ungol Index is used for ranking documents in a German information retrieval task. It is studied whether the original desired spatial properties of the original word embeddings are retained, if the novel scoring function is able to rank documents with high quality and whether utilising the Hamming distance offers the desired computational speed-up. Finally, the ranking performance of the RHWMD is compared to the Word Movers Distance (WMD), Best-Match 25 (BM25) and Term Frequency-Inverse Document Frequency (TF-IDF).

[1] Usually around 300 dimensions.

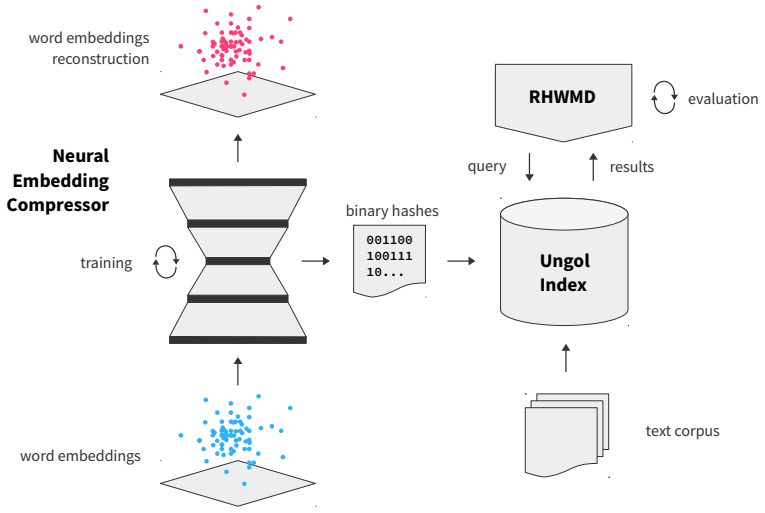


Figure 1.2: This is a high level overview of the system implemented in this work. The neural embedding compressor is used to produce binary files containing the hash codes. The Ungol Index is constructed for a free document collection by mapping the documents' contents to the hash codes. The search uses the index and the RHWMD for the evaluation of the ranking quality.

The following Chapter 2 compiles the theoretical foundation on which the systems' implementation is built upon. Additionally, related works and references for further reading are provided. The findings of Chapters 3 and 4 are discussed in Chapter 5. Here the strengths and weaknesses of the approach are reviewed and some attempts for improvement are formulated. The Appendix compiles data not further analysed or discussed and a lookup table for the most important Python implementations.

1.3 Notation

The mathematical notation used in the following tries to adhere to the usual conventions: Matrices M are capitalised. Vectors $\mathbf{v}$ are bold faced. Matrix multiplication is denoted as $A \cdot B$ or simply AB. A dot product either uses a dot $\mathbf{u} \cdot \mathbf{v}$ or is simply abbreviated: $\mathbf{uv}$. For concatenation the operands are simply concatenated and written in brackets, e.g. $[\mathbf{uv}]$ or $[AB]$. Sub-scripted characters $\mathbf{A}_{ij}$ are used for addressing and selection. To denote variation, characters are super-scripted: $t' \neq t$.

At the very end a glossary not only enumerates abbreviations or technical terms but also mathematical constants used throughout the whole thesis. This is designed to guide through the many formulas used without having to re-define the constants all the time. This is also helpful to find all other equations which involve the respective constant.

Examples are given in either English or German. English examples are meant as illustrations and are not based on real world data. Wherever examples or excerpts are given in German, they come from the data used for implementing the Ungol Index.

Chapter 2

Essentials

This section presents methods relevant to this thesis, namely various strategies towards estimating document similarities. Handling text on a computer poses the fundamental challenge of its representation. For example, character encoding is basically an arbitrary way of defining a table which maps a number to a symbol. Text is just the concatenation of these numbers and this way of storing textual data offers no useful information regarding the latent semantic structure that may exist there. Natural Language Processing (NLP) is inherently concerned with devising ways to unveil the grammatical, lexical and semantic information hidden in unstructured text. To determine the similarity of documents one approach is to work with their words as atomic data. Word order is generally not regarded; at best n-grams of words are considered. Hence, for this work, the relationship of words to one another and the distribution of words over document collections is very important.

This chapter is designed to provide all the necessary theoretical foundations for the following chapters. Relevant works are provided wherever suitable. First, an introduction to of methods working with Bag of Words (BOW) vectors, a common approach for information retrieval, is given. Both Term Frequency-Inverse Document Frequency (TF-IDF) and Best-Match 25 (BM25) are explained and used later in Chapter 4 for evaluation. In Section 2.2, approaches for mapping documents to real-valued vectors are presented. Following in Section 2.3 methods for the production of word embeddings are presented. Because they are used in Chapter 3, fastText and Global Vectors for Word Representation (GloVe) are explained in more detail. This chapter concludes in Section 2.4 with a detailed description of the Word Movers Distance (WMD). This is the origin of the Relaxed Hamming Word Movers Distance (RHWMD) developed in Chapter 4 and also used for evaluation later.

2.1 Bag of Words

One approach to design a document similarity measure is using Bag of Words (BOW) encodings. Here the idea is to work with vectors representing a text document where each component of the vector stands for a specific token. A document is then a (often very) sparse representation with all non-zero elements having a specific weight which expresses the importance of that word in the document. To obtain a measure of similarity, these vectors are compared without reducing their dimensionality. In case of a search application, this demands for transforming the query and the candidate documents to these vectors and ordering the candidate documents by similarity to the query. An example for such a term-document matrix was given in Section 2.2.1. This method is very well researched and used in popular document database systems such as Elasticsearch (ES) and Apache Solr [73].

2.1.1 Inverted Indexes

Main focus of this work is the application of such similarity measure to Information Retrieval (IR). To search through a large corpus of documents, every attempt at processing a query linearly on the raw data automatically leads to an unacceptable response time. Thus, for enabling fast searches, a common approach is to build an *inverted index* which offers a fast lookup based on terms shared by the query and candidate documents. Hence, the information retrieval task consists of two distinct phases: First the indexing phase where the inverted index is built from the document collection and secondly the retrieval phase where, based on the index, a score is calculated which ranks candidate documents by the relevance regarding the query. The former phase is further referred to as *offline* and the latter as *online*.

Both phases have very different constraints and performance requirements. The indexing procedure has often much more relaxed time constraints. To index a single document a few seconds is often acceptable. This only poses a problem if the rate of incoming documents exceeds the capacity of the indexer. This problem is then usually solved by building a distributed index over multiple machines — thus scaling the system horizontally — instead of fundamentally changing the approach. The online query response however has much higher demands. Here the system calculates a score for every candidate document and hence, to provide answers quickly, calculating the score must be fast.

The main goal of an inverted index is to provide a mapping of *tokens* to documents and provide metrics for calculating the relevance scores in the online phase. These tokens are usually preprocessed words gathered from the indexed

documents. Given a document collection D containing documents d who are sets of unique tokens t_d the index $\Pi : t \mapsto \{d_1, d_2, \dots\}$ knows which words originate from which documents. To use this mapping for a boolean retrieval task, consider for example a query with three tokens $t_1 \vee t_2 \vee \neg t_3$. The set of relevant documents is $(\Pi(t_1) \cup \Pi(t_2)) \setminus \Pi(t_3)$. Building an effective index is subject to handling data with greater size than can be loaded into memory, heterogeneous data sources and sometimes high availability contracts among other things. These requirements are not in the scope of this work and mainstream indexer such as Apache Lucene [72] are readily available. Boolean queries can be useful for pre-selecting candidates for a more fine-grained selection or offering statistics about a text corpus. The order of tokens is not considered.

2.1.2 Vector Based Scoring

However, boolean retrievals are usually not decent enough for a real world application. Consider for example the query *"where are the keys to my flat?"*. The boolean retrieval approach can only combine each token by binary $\vee$ or $\wedge$ and negate single tokens. This leads to a most likely empty result set for $\wedge$ because the constraint of having all these tokens in a document might be to tight. For $\vee$ the exact opposite might happen: As nearly all documents containing English natural text will contain words such as *are*, *the*, etc., the result set will be very large and uninformative. So two aspects are missing for a successful text retrieval: how informative every considered token is and a relevance measure to sort result sets by.

A simple approach to fulfil both requirements first encodes documents as BOW and Normalised Bag of Words (NBOW) vectors. Here each document d is assigned a vector $\mathbf{b}_d = (\#t_1, \#t_2, \dots) \in \mathbb{N}^N$ of the size of the vocabulary N. Each element of the vector holds the count of occurrences of the term t in document d. The NBOW normalises the vector such that each element of the vector is divided by the total count of tokens of the document and is further denoted as: $\overline{\mathbf{b}}_d$. The most common approach to have a measure of similarity is to compute the cosine distance of two documents [40]:

$$\begin{aligned} r_{\cos} &: D \times D \mapsto \mathbb{R} \\ r_{\cos}(d, d') &:= \overline{\mathbf{b}}_d \cdot \overline{\mathbf{b}}_{d'} \end{aligned} \tag{2.1}$$

2.1.3 TF-IDF Relevance Scoring

The formula given in Equation (2.1) is the basis for many document similarity measures who heuristically modify the weights of specific tokens. An approach for assigning relevance is the TF-IDF measure, first introduced by Sparck Jones [63] and Jones [29]. Here, each token's frequency of occurrence both locally for the candidate document and globally across the whole corpus offers the necessary statistical information to calculate a relevance score. Consider a query document d and a candidate document d' each containing a set of tokens.

$$\begin{aligned} r_{\mathsf{tfidf}} &: D \times D \mapsto \mathbb{R} \\ r_{\mathsf{tfidf}}(d, d') &:= \sum_{t \in d} \#t_{d'} \cdot \mathsf{idf}(t) \\ \mathsf{idf}(t) &:= \log \frac{|D|}{|\{d \in D : t \in d\}|} \end{aligned} \tag{2.2}$$

The higher the score, the more relevant is the candidate document d' for the query d. The Term Frequency (TF), further denoted by $\#t_d$ of token t in document d is the normalised Term Count (TC). The term frequency quantifies how important a term in the scope of the document is regarded. The assumption is that an important term also occurs more often in the document. This however does not hold for very generic words such as "the" or "it". As these terms occur many times in most English documents, the second part of the product handles the information content of the token globally. The Inverse Document Frequency (IDF) of a token is the logarithm of the fraction of the total count of documents divided by the number of documents the token appears in. This count of documents for a token is the Document Frequency (DF). Thus for a word contained in every document of the corpus the information value of that word is $\log 1 = 0$ and if any given word only occurs in one document, the idf value reaches it's maximum possible value of $\log |D|$.

2.1.4 Okapi BM25

Building upon the TF-IDF formula in Equation (2.2), the BM25 scoring as part of the Best-Match family of relevance schemes [54], introduces normalisation and adds two free parameters k_1 and b for domain specific tuning:

$$
\begin{aligned}
r_{\text{BM25}} &: D \times D \mapsto \mathbb{R} \\
r_{\text{BM25}}(d, d') &:= \sum_{t \in d} \text{idf}(t) \cdot \frac{(k_1 + 1)\#t_{d'}}{k_1((1-b) + bl) + \#t_{d'}} \\
l &:= \mid d' \mid / \left[\frac{1}{|D|} \sum_{d'' \in D} \mid d'' \mid \right]
\end{aligned}
\tag{2.3}
$$

The additional fraction introduces with l the ratio of the considered documents' (d') length to the average lengths of documents in the corpus. The basic idea is to take into consideration that terms with low frequency in large documents do not transport much information and as such must only contribute little to the relevance score. The two parameters control how the scoring is affected by the local term frequency and the documents' length in comparison to the average document length respectively. Assigning either k_1 or b a value of zero, the whole term becomes 1 and thus only the raw idf values contribute to the score.

k_1 serves as a saturation parameter which restrains the contribution of very frequent terms. For high k_1 values, the influence of the l is amplified which in turn leads to small results when solving the fraction. For small values of k_1 the influence of the documents' length decreases and high term frequencies contribute higher to the score.

b — chosen in the range of 0 and 1 — only concerns the influence of the documents' length in comparison to all other documents. The higher b is chosen, the higher is the denominator in conjunction with k_1 for large documents. For small documents d, the ratio is a value between $1/|D|$ and 1, which is also reduced by b. For large documents high values of b further increase the term value. As k_1 controls the influence of long documents locally (without knowledge about other documents), b controls the influence of abnormally sized documents globally. A common configuration assigns k_1 between 0.5 and 2 and b between 0.3 and 0.9. This similarity measure is developed heuristically and some efforts have been made to improve upon this [37, 64, 38]. In general, how these parameters are set and what kind of pre-processing works best usually depends on the target domain [66].

2.2 Document Embeddings

A way of encoding semantic information of words is following the so-called Distributional Hypothesis which states that semantically close words tend to share the same contexts [21, 13, 43]. This implies that to automatically find relatedness

amongst words, their immediate words contexts provide the vital information to do so. Several approaches exist and map words, paragraphs or documents to real-valued vectors: the *embeddings*. One of the most desired properties of the produced embeddings is their spatial closeness for semantic relatedness. This means if two terms or two documents are very close concerning their contents, than the corresponding embedding vectors are also very close in their space. An example is given in Table 2.1 where the nearest neighbours regarding cosine similarity of two word embeddings learned with fastText [4] are shown.

2.2.1 Topic Modelling

Some approaches to produce document embeddings, can be subsumed under the name *Topic Modelling*. Here a document is viewed as a mixture of topics and the similarity of documents can be determined by comparing their topics. A fundamental method is Latent Semantic Analysis (LSA)/Latent Semantic Indexing (LSI) by Deerwester et al. [10] where a matrix is constructed correlating word tokens' TF or TF-IDF (see Section 2.1.3) to documents. An example for such a TF-document matrix is given below where eight documents either cover forests or traffic:

$$
\begin{array}{l}
\\
\text{forest} \\
\text{stag} \\
\dots \\
\text{traffic} \\
\text{car} \\
\dots \\
\text{ist} \\
\text{und}
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 \end{array} \\
\left[\begin{array}{cccccccc}
5 & 1 & 2 & 3 & 1 & 1 & 0 & 0 \\
2 & 4 & 6 & 5 & 0 & 0 & 0 & 0 \\
 & & & & \dots & & & \\
1 & 0 & 0 & 0 & 9 & 3 & 4 & 3 \\
0 & 0 & 1 & 0 & 7 & 1 & 9 & 8 \\
 & & & & \dots & & & \\
9 & 8 & 4 & 9 & 3 & 6 & 9 & 9 \\
8 & 5 & 9 & 9 & 5 & 4 & 4 & 9
\end{array}\right]
\end{array}
\tag{2.4}
$$

Now each document can be represented by a column of this matrix and each words by a row. Working with these high dimensional vectors has several drawbacks, namely their sparsity and outlier values among others. Thus much effort is put into smoothing and transforming them into a lower-dimensional spaces.

To create document embeddings, the matrix is decomposed using Singular Value Decomposition (SVD) which leads to word tokens being summarised to *topics* which describe a document. The document embedding vectors can than be compared (e.g. using cosine similarity) to obtain a measure of relatedness

		Gott		Satan
Position	Word ($\mathbf{e}_2$)	Distance	Word ($\mathbf{e}_2$)	Distance
1	$\mathbf{e}_1 =$ gott	0.0000	$\mathbf{e}_1 =$ satan	0.0000
2	gottes	0.1723	satans	0.2139
3	„gott	0.2215	satana	0.2787
4	göttliche	0.2542	satanic	0.3126
5	göttlich	0.2651	satanas	0.3135
20	gottesvolkes	0.3212	dämon	0.4010
21	jahwe	0.3220	sünder	0.4026
22	jhwh	0.3247	dämonen	0.4030
23	gottesverehrung	0.3252	luzifers	0.4125
24	göttlichem	0.3273	teufels	0.4138
50	verehren	0.3586	darkthrone	0.4458
51	gottebenbildlichkeit	0.3607	teuflischen	0.4470
52	offenbarung	0.3607	slayer	0.4474
53	erlösung	0.3613	azazel	0.4474
54	schöpfung	0.3615	diaboli	0.4488

Table 2.1: Some nearest neighbours determined by cosine distance of the fastText embedding space. The values in the respective *Distance* column are $1 - \cos(\phi(\mathbf{e}_1, \mathbf{e}_2))$ where ϕ is the angle between the embeddings of two words.

based on the topics best describing the candidate documents. This concept is further developed as Probabilistic Latent Semantic Analysis (pLSA)/Probabilistic Latent Semantic Indexing (pLSI) by Hofmann [23]. Instead of using SVD, a latent class model which models the probability of word co-occurrence as a mixture of conditionally independent multinomial distributions is used for the dimensionality reduction. This is taken a step further by Blei et al. [3], Pritchard et al. [53] with Latent Dirichlet Allocation (LDA)/Latent Dirichlet Indexing (LDI) where the probabilistic model is enriched by the assumption that the topic distribution follows a Dirichlet prior. The produced document embedding vectors can now be compared, for example using their cosine distance, Jensen-Shannon distance [11, 48, 14] or Hellinger Distance [22].

2.3 Word Embeddings

In the following sections some approaches for the production of word embeddings are presented. Most of the explained methods *learn* word representation vectors on large text corpora. This works by optimising a machine learning model

such that it maximises the log probability of it correctly predicting a *centre word* based on its surrounding *context words* or vice versa. One of the most famous approaches is Word2Vec [41, 42] and presented hereafter in Section 2.3.2. A challenging part of creating such a model is the amount of calculations necessary to tune the models' weights. The second part of this section explains two methods which create word embeddings based on global co-occurrence counts: the Positive Pointwise Mutual Information (PPMI) in Section 2.3.4 and GloVe in Section 2.3.5.

2.3.1 History of Neural Word Embeddings

Collobert et al. [7] first proposed a unified neural network model to create intermediate word representations. Their main goal was to develop an alternative to the task specific feature engineering often used for tackling NLP related problems. *The Neural Probabilistic Language Model* proposed by Bengio et al. [1] follows this idea and is improved by *The Hierarchical Probabilistic Neural Language Model* by Morin and Bengio [45] which tackles the scaling problem by not computing the conditional probability for centre word with the whole vocabulary at once but, following the work of Goodman [18], decomposing words in a hierarchical set of classes. The probability is then computed by following a path through the hierarchy to the respective word. Huang et al. [24] combine global context information provided by the respective document with the local context surrounding a word and use a ranking approach for scaling based on Collobert and Weston [6] for training which differs from optimising the log likelihood of the next word. In the following some more recent approaches will be revised in greater detail.

2.3.2 Word2Vec

Continuous Bag of Words (CBOW) and Skip-Gram (summarised as Word2Vec) are two neural network models proposed by Mikolov et al. [41, 42] that are trained for predicting words in a context window. Given a sequence of word tokens, the contexts are all sub-sequences of a freely defined but fixed width of w or smaller. Sequences are clipped on the sentence boundaries. The word in the midst is denoted as *centre word*, the surrounding words are named *context words*. CBOW predicts the centre word based on the sum of the context words, whereas Skip-Gram predicts the context words of the centre word. More formally, given a vocabulary $V = \{t_1, \ldots, t_N\}$ of size N, a centre word t and a sequence of context words $(t'_1, \ldots, t'_w)$ the goal of Skip-Gram is to maximise the logarithmic probability of predicting the correct context words:

$$\arg\max \sum_{i=1}^{w} \log P(t'_i|t) \tag{2.5}$$

Now each word t is assigned two vectors $\mathbf{e}_t, \mathbf{e}'_t$. Two matrices $X, X' \in \mathbb{R}^{N \times E}$ contain the vector representations with E denoting the freely chosen word embedding dimensionality. Thus a row of the embedding space $\mathbf{e}_t$ is the embedding vector of token t. Both matrices are initialised randomly. To obtain the final embeddings, both X and X' are combined by summation. The conditional probability $P(t'|t)$ is calculated by applying a softmax on the product of one context word vector with the centre word for all context word vectors:

$$P(t'|t) := softmax(\mathbf{e}_t, \mathbf{e}_{t'}) = \frac{\exp(\mathbf{e}_t \cdot \mathbf{e}_{t'}^T)}{\sum_{i=1}^{N} \exp(\mathbf{e}_t \cdot \mathbf{e}_{t_i}^T)} \tag{2.6}$$

The CBOW model works analogous. However, the computational overhead for computing the softmax for the whole vocabulary is too big and is reduced by either implementing a hierarchical softmax [45] or negative sampling [42]. Here the training using negative sampling is explained: In each training step a centre word embedding $\mathbf{e}_t$ and the surrounding context word representations $\mathbf{e}_{t'}$ are selected. The goal is now to increase the log-probability that the model correctly predicts the context words. To delimit the correct context words from all other potential words of the vocabulary, *negative* samples $\mathbf{e}'_i$ are selected randomly using a noise distribution from the embedding space. This is based on the work of [20, 62]. The noise distribution which is reported to work best is a unigram distribution raised to the power of $3/4$. A free parameter Z controls the amount of negative samples. The function σ is a sigmoid used for introducing non-linearity to the model and mapping the dot products to probability values. The model is then trained using gradient descent to increase the value of the following objective:

$$\arg\max_{X,X'} \log \sigma\Big(\mathbf{e}_t \cdot \mathbf{e}_{t'}^T\Big) + \sum_{z=1}^{Z} \log \sigma\Big(- \mathbf{e}'_z \cdot \mathbf{e}_t^T\Big) \tag{2.7}$$

2.3.3 FastText

Bojanowski, Grave, Joulin, and Mikolov [4] propose an extension to the Skip-Gram model. Most methods to produce word embeddings assign each word a distinct vector. This leads to the problem that for languages with a rich morphology many vectors are assigned to many words which rarely occur in the training data. Methods such as Word2Vec have no notion of words having different forms

but the same meaning. Consider the German word *gehen*: There are many forms and compounds of this word such as *geh*, *geht*, *gehe*, *Gehstock*, *ausgehen*, *aufgehen*, etc. Each of these words is considered independently in a method such as Word2Vec. The fastText approach includes *sub-word* information; i.e. the internal structure of words represented by their character n-grams; i.e. some successive character sub-sequence. Each character n-gram is then embodied by a vector and the sum of all sub-word character n-gram embeddings is the word embedding.

Given a word token t and a fixed number $n > 0$, the words are enclosed by a special delimiter characters and separated into their character n-grams. Consider for example the German word *beispiel*, an n-gram size of three and the delimiter tokens <, > then the resulting representation looks like this:

$$G_{\textit{Beispiel}} = \big(\textit{<be}, \textit{bei}, \textit{eis}, \textit{isp}, \textit{spi}, \textit{pie}, \textit{iel}, \textit{el>}, \textit{<beispiel>}\big) \tag{2.8}$$

The word itself is added to the n-gram sequence. Note that the character n-gram *bei* and the German word *<bei>* are represented by two different vectors. Now, contrary to Word2Vec, a word is no longer represented by a single embedding vector $\mathbf{e}_t$ but is constructed from a new embedding matrix X which contains all embeddings for all encountered character n-grams by summation: Given a set of character n-gram embeddings $\mathbf{z}_g \in X$ for an n-gram g and G_t a set of character n-grams for a token t, then:

$$\mathbf{e}_t := \sum_{g \in G_t} \mathbf{z}_g \tag{2.9}$$

One great feature of the model is the ability to create new embeddings on-the-fly by constructing the new embedding from the sub-word information of the regarded word. The training is changed such that the model optimises the new embedding vectors (again using a training based on negative sampling):

$$\underset{X, X'}{\arg\max} \; \log \sigma\Big(\sum_{g \in G_t} \mathbf{z}_g \cdot \mathbf{e}_{t'}^T \Big) + \sum_{z=1}^{Z} \log \sigma\Big(- \sum_{g \in G_t} \mathbf{z}_g \cdot \mathbf{e}_z'^T \Big) \tag{2.10}$$

2.3.4 Pointwise Mutual Information

Another approach works with a global co-occurrence matrix for terms and is called PPMI [30]. A co-occurrence matrix $Q \in \mathbb{N}^{N \times N}$ is constructed by counting how many times a word occurs in close proximity to another word for a vocabulary of size N. Hence Q_{ij} denotes how many times term t_i occurs near t_j.

A simple example of such a co-occurence matrix is given in the following with a window size of three. The corpus, in this case, consists of four sentences with a total of eight unique word tokens:

So the FCC / Wont let me be
Let me be me / So let me see

Using these parameters, the sentence *"so let me see"* is counting the following co-occurrences (a, b), where a is the centre word and b is the co-occurring neighbour: *(so, let), (so, me), (let, so), (let, me), (let, see), (me, so), (me, let), (me, see), (see, let), (see, me)*. Doing this for all the four sentences, the matrix element Q_{ab} counts the number of times a relation (a, b) is encountered globally. The following equation displays the final co-occurrence matrix:

$$\begin{array}{c} \\ \text{so} \\ \text{the} \\ \text{fcc} \\ \text{wont} \\ \text{let} \\ \text{me} \\ \text{be} \\ \text{see} \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{so} & \text{the} & \text{fcc} & \text{wont} & \text{let} & \text{me} & \text{be} & \text{see} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 3 & 2 & 1 \\ 1 & 0 & 0 & 1 & 3 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \tag{2.11}$$

Now, given a term t_i and a random context term t_j: $P(t_i)$ denotes the probability that t_i occurs in a random local term pair and $P(t_j)$ calculates the chance that t_j appears as a context term:

$$P(t_i) := \frac{\sum_{j=1}^{N} Q_{ij}}{\sum_{i'=1,j'=1}^{N} Q_{i'j'}} \quad P(t_j) := \frac{\sum_{i=1}^{N} Q_{ij}}{\sum_{i'=1,j'=1}^{N} Q_{i'j'}} \tag{2.12}$$

$P(t_i, t_j)$ represents the probability that both terms occur in the same context:

$$P(t_i, t_j) := \frac{Q_{ij}}{\sum_{i'=1,j'=1}^{N} Q_{i'j'}} \tag{2.13}$$

The PPMI is now defined as:

$$r_{\text{PPMI}}(t_i, t_j) := \max\left(0, \log_2\left(\frac{P(t_i, t_j)}{P(t_i) \cdot P(t_j)}\right)\right) \tag{2.14}$$

$P(t_i) \cdot P(t_j) \approx P(t_i, t_j)$ applies if the respective probabilities are independent which leads to a PPMI score close to zero. If two terms occur many times in the same context but rarely in other contexts this score is high. Conversely for very frequent terms the logarithm is negative and clipped. The PPMI calculates a similarity score based on the co-occurrence of two words. Islam and Inkpen [27] introduce the *Second order co-occurrence PMI* which yields high scores for two words if they often share the same context.

2.3.5 GloVe

A machine learning approach to compute high-quality word representations using global co-occurrence counts is called GloVe by Pennington et al. [52]. Here, instead of sequentially processing word windows, the global co-occurence matrix Q of size $N \times N$ is used. based on weighted sub-sequences of the input word sequences. The columns or rows of this matrix could be used as word representations, but as the vocabulary is usually of much larger size sparsity issues arise when training downstream models. Also the input layers of these models would be to large to be effectively trained. Apart from that, the matrix is not very smooth, because some words occur much more frequently than others. To tackle this problem, the dimensionality of the vectors must be reduced while preserving as much encoded information as possible. To do so a model is trained in an unsupervised manner. The training part consists of minimzing a least square error of a regression model, given in Equation (2.15).

$$\arg\min \sum_{i,j=1}^{N} f(Q_{ij}) \cdot (\mathbf{e}_i^T \cdot \mathbf{e}_j - \log Q_{ij})^2 \tag{2.15}$$

The function $f : \mathbb{R} \mapsto \mathbb{R}$ serves to apply a penalty to very frequent words (such as *the*, *a* etc.). The empirically found function used in the original work is given below (with $x_{max} = 100$ and $\alpha = \frac{3}{4}$):

$$f(x) := \begin{cases} (x/x_{\text{max}})^\alpha & \text{if} \quad x < x_{\text{max}} \\ 1 & \text{otherwise} \end{cases} \tag{2.16}$$

The model is trained using gradient descent. In each training step a random non-zero element Q_{ij} is sampled from the co-occurrence matrix. Two randomly initialised embedding matrices X, X' offer corresponding embedding vectors $\mathbf{e}_i \in X$ and $\mathbf{e}_j \in X'$. The optimiser then adjusts the embedding vectors such that their dot product converges towards the total logarithmic co-occurrence of the respective tokens (see Equation (2.15)). After training, both matrices X, X' are combined by summation.

In summary, the skip-gram and CBOW models capture co-occurence information for each window at a time whilst the GloVe model collects the counts of the global co-occurence statistics of all words. Both models are heavily used downstream in a variety of applications of deep learners to NLP.

2.4 Word Movers Distance

Using the formerly described BOW models for retrieval tasks proved to be quite successful. These approaches have some drawbacks however as they always rely on having a unique representation of a token. Some effort must be put into preprocessing such that inflections, paraphrases etc. can be distinctly mapped to a single token. This includes stemming, compound splitting (helpful for Germanic languages) [12], the use of paraphrase databases [15] and the manual maintenance of synonym lists among others. Beyond this, paraphrasing and synonyms can not be handled by the BOW model.

Consider the following two example sentences:

> *Document 1:*
> Bei den Krawallen im Stadion von Dublin wurden 50 Menschen verletzt.
>
> *Document 2:*
> Am Mittwoch kam es in Irland bei einem Fußballspiel zu Ausschreitungen.

The two sentences obviously cover the same topic but paraphrase the content differently. The human reader is able to correctly associate *Krawalle* with *Ausschreitungen*, *Dublin* with *Irland* and *Fußballspiel* with *Stadion*. Methods working with term identity such as the formerly presented BM25 struggle to associate such documents. One way of handling this is using lists of synonyms for words. This, however, is a very *static* approach and can only relate words such that they are equally important for scoring. A different approach to tackle this problem is presented by Kusner et al. [35] and is called Word Movers Distance (WMD). The method is based on the Earth Movers Distance (EMD) and its manner of functioning is presented hereafter.

2.4.1 Document Similarity as a Transport Problem

Based on their training objectives, the word embedding vectors presented in Section 2.3 try to catch semantic relation through spatial distance. Shorter distance means closer semantic relation as this distance encodes the observation

of same or similar contexts. This inspires the Word Movers Distance as an implementation of the Earth Movers Distance — used for image retrieval tasks — for text data. The EMD (also known as Wasserstein or Kantorovich-Rubinstein Distance Metric) tries to formulate how much cost is associated with transforming one probability distribution into another. The cost, in this case, is the amount of change required to carry out the transformation. This distance metric was originally popularised for image retrieval [51, 57, 56]. For these tasks this is usually applied to the histogram of two images. The cost to transform the one histogram into the other is then the distance of the respective images.

This problem of finding the optimal transformation is a *transport problem*. This problem is concerned with moving some amount of goods — in this case a word embedding — from one place to another. Each movement is associated with a cost and the task is to find the optimal routes such that the cumulative cost of all movement is minimised. The implementation for the EMD for text documents based on word embeddings is proposed by Kusner et al. [35] as WMD. The WMD uses the Euclidean distance of the embedding vectors as the cost. Applied to two documents and the attempt to formulate a distance measure, each word of the query document is allowed to be transformed into any other words of the candidate document in total or in parts. It is auxiliary to think about each word having an associated mass and this mass has to be segmented in a way that all mass of the query document is moved such that it *fits* the candidate document. Moving mass is associated with a cost and the sum of these movement costs defines the quality of the movement plan. The best movement plan yields then the quantified distance of the two documents. The formal definition is as follows:

Given each document is encoded as a sparse NBOW vector $\mathbf{b} \in \mathbb{R}^N$ for a vocabulary of N words. This vector is normalised by dividing the frequency of the word by the total number of tokens of the document. Based on this frequency information a flow matrix $\mathbf{T} \in \mathbb{R}^{N \times N}$ with $\mathbf{T}_{tt'} \geq 0$ is to be found which encodes how much *mass* of a word is moved to a corresponding target word. This flow matrix is constrained by the requirement that the incoming and outgoing flows must match:

$$\sum_{t}^{N} \mathbf{T}_{tt'} = \mathbf{b}_t \quad \text{and} \quad \sum_{t'}^{N} \mathbf{T}_{tt'} = \mathbf{b}_{t'} \tag{2.17}$$

Each token t has a corresponding embedding vector $\mathbf{e}_t \in \mathbb{R}^{\mathbb{N}}$. The cost of transforming one token t into a corresponding token t' is their spatial distance. In the original paper, the Euclidean distance is used: $c(t, t') = ||\mathbf{e}_t - \mathbf{e}_{t'}||_2$. Thus the transportation problem is solved by finding the flow which reduces the overall transportation costs:

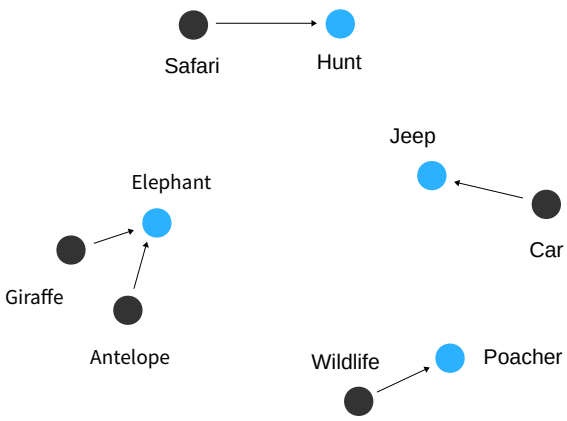


Figure 2.1: Illustration of how the transport plan might be solved for two documents when using the Relaxed Word Moving Distance (RWMD) for one direction. For each token of the query document (denoted in grey) the nearest neighbours are selected from the candidate document (blue). The weighted cumulative distance between all these word embeddings is then the total distance between the two documents.

$$\min_{\mathbf{T}\geq 0} \sum_{t,t'=1}^{N} \mathbf{T}_{tt'} c(t,t') \quad \text{subject to Equation (2.17)} \tag{2.18}$$

Finding the optimal transport plan is a difficult task. Pele and Werman [50] show that the best average time complexity scales $O(t^3 \log t)$, where t is the count of unique tokens in the documents. This can not be calculated in an acceptable amount of time online on standard hardware for large document collections. Even calculating the distances offline while indexing would take too much time as the number of possible combinations increases with each added document and thus — even when scaling the infrastructure horizontally — this will only work for small document collections. Also an inverted index can not be used and hence all documents have to be scanned linearly.

2.4.2 The Relaxed Word Moving Distance

Solving the whole transportation problem for all candidate documents is computationally too expensive for maintaining sane response times. The problem's complexity is to be accounted for by the constraints of the need to match the

mass of the documents exactly: The mass of a word can be split up and be distributed over several words of the target document and each mass of a target word has in turn to be distributable to the source word masses (see Equation (2.17)). Relaxing these constraints leads to a faster solution which is introduced by Kusner et al. [35] as RWMD. There one direction of moving the mass is no longer a constraint which greatly reduces the computational overhead. The RWMD is a lower bound to the exact WMD but still yields comparable results on several classification tasks. Formally, when removing one of the constraints, two vectors $\mathbf{u} = (u_1, u_2, \ldots, u_N) \in \mathbb{R}^N$ and $\mathbf{u}' = (u'_1, u'_2, \ldots, u'_N) \in \mathbb{R}^N$ are constructed containing the weighted distance to each corresponding nearest neighbour:

$$\mathbf{u}_t = \begin{cases} c(t, t') \cdot \mathbf{b}_t & \text{if } t' = \arg\min_{t'} c(t, t') \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

$$\mathbf{u}'_{t'} = \begin{cases} c(t, t') \cdot \mathbf{b}_{t'} & \text{if } t = \arg\min_{t} c(t, t') \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Finally the problem is reduced to finding the nearest neighbours for each token in both directions, weight their distance by the normalised TF and sum all weighted distances to obtain two scores: one per relaxed constraint. The scores are then combined by selecting the maximum of these two to obtain the tightest lower bound:

$$\begin{aligned} r_{\mathsf{RWMD}} &: D \times D \\ r_{\mathsf{rwmd}}(d, d') &:= \max\Big(\sum_{t}^{N} \mathbf{u}_t, \sum_{t'}^{N} \mathbf{u}'_{t'}\Big) \end{aligned} \tag{2.21}$$

2.4.3 Word Centroid Distance

Even the sped-up variant of the WMD, the RWMD, is too inefficient to linearly process large corpora. Thus another lower bound for pre-selecting candidate documents is introduced: the Word Centroid Distance (WCD). The authors show that the WCD is a lower bound to the WMD and thus suitable for a very fast candidate pre-selection. The drawback however is that this bound is not very tight and many candidates may be selected. Here, simply the p2-norm of the summed up and weighted word vectors of the two documents is used as a threshold. More formally, given a word embedding space X of size $N \times E$ and the two NBOW representations $\mathbf{b}_d \mathbf{b}_{d'}$, the WCD is defined as:

$$r_{\mathsf{WCD}} : D \times D$$
$$r_{\mathsf{WCD}}(d, d') := ||X\mathbf{b}_d - X\mathbf{b}_{d'}||_2 = \left|\left| \sum_t^N \mathbf{b}_{dt}\mathbf{e}_t - \sum_{t'}^N \mathbf{b}_{d't'}\mathbf{e}_{t'} \right|\right|_2 \tag{2.22}$$

The algorithm proposed by the authors for the WMD to obtain the k-nearest neighbours (K-NN) for a document is to first sort all documents by their WCD. For the first k documents the WMD is calculated. Then the remaining documents are traversed and their much tighter RWMD is calculated. If this bound exceeds the current k-th neighbours' distance, it is not regarded. If it falls below this threshold its' exact WMD is calculated and the list of K-NN is updated. For an approximate search, not all remaining documents have to be considered.

2.5 Hashing

The following chapter covers how a neural auto-encoder network is trained to compress word embeddings such that binary hash codes are produced for the fast retrieval with the RHWMD. A requirement for these hashes is that they preserve the spatial characteristics of the original embeddings such that it allows fast vector comparisons. This section provides some related works regarding hashing methods.

Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [26, 16, 9, 33, 34] is an approach where the high dimensional space of the input data is separated into smaller sub-spaces. These partitions of the space are called *buckets* and each data point is associated to a bucket. The goal of this method is to produce hash collisions for data points close to one another in metric space. This is done by assigning a code to the bucket and adjacent buckets have close codes. The space is separated differently and independently multiple times. Gionis et al. [16] show that this method leads to a convergence towards the preservation of locality. A drawback of this approach is that multiple hash tables (per separation) have to be built which multiples the required storage space.

Spectral Hashing

Weiss et al. [68] define the *optimal* hash codes. They must adhere to three constraints: First, they must be easily computed for novel input; Secondly, they must be as short as possible; Finally, similar items must have similar hash codes. The authors show that the problem of finding the optimal hash codes for real-valued data points such that their Hamming distance best approximates the original spatial distance is NP-hard. Their method, called *Spectral Hashing*, relaxes the requirement to find the optimal codes and operates on the input data variance: First the principle components of the data are identified using Principle Component Analysis (PCA) [49]; secondly identifying the smallest eigenvalues along the principle components axes and thus defining the single-dimension eigenfunctions; and thirdly thresholding these eigenfunctions at zero to obtain binary codes.

Semantic Hashing

A different approach is proposed by Salakhutdinov and Hinton [59] and called *Semantic Hashing*: There histograms of word counts form a probability distribution and are used to train a stacked Restricted Boltzman Machine (RBM) such that the weights of the RBM model the joint distribution of the word counts and the trained latent variables. Their approach assumes a conditional Poisson distribution for the word counts and a conditional Bernoulli distribution for the hash codes. Following the initial training where each RBM is trained independently using the input of the respective larger RBM there is a fine-tuning step where the RBM is used as an auto-encoder fine tuned via backpropagation. A similar approach is pursued for image classification by Salakhutdinov and Hinton [58]. For image retrieval, Morere et al. [44] propose a RBM where alternating Gibbs sampling is used to approximate the joint distribution. These machine learning approaches proved to be very successful: Semantic Hashing outperforms LSH significantly for an IR task [59] and for Image Retrieval [65] both qualitatively and regarding speed.

Word2Bits

Spectral Hashing and LSH build hash codes based on data points in a continuous space. Word vectors as formerly introduced in Section 2.3 are embedded in such space and can thus be transformed to hash codes by these methods. The former machine learning approaches, however, expect probability distributions as input. For text NBOW vectors are used to create hash codes representing whole documents. This means that those methods are a different approach to solve the

retrieval task altogether. A novel machine learning approach called *word2bits* by Lam [36] tries to unify the two-step process of creating real-valued word embeddings and afterwards transforming them into Hamming space. Here, based on the work of Courbariaux et al. [8] and Mikolov et al. [41], a virtual quantization function is introduced into the CBOW function. This quantization function maps each element of the real-valued vector to either $-\frac{1}{3}$ for elements smaller or equal to zero or $\frac{1}{3}$ otherwise. The final word embeddings are still full-precision float vectors but as they can only contain at most two different values they can directly be mapped to binary hash codes. The quantized word vectors are compared to Word2Vec and GloVe and perform competitively on Question Answering, Word Analogy and Word Similarity tasks.

Chapter 3

Neural Compressor

Determining the document similarity based on the Relaxed Word Moving Distance requires many word-wise spatial distance calculations. These distance calculations of word embeddings are computationally expensive as they involve many real-valued operations. The computational overhead increases with higher dimensionality of the underlying embedding space. This work aims at an implementation of a fast search and one aspect of this approach consists of calculating the word-wise distance in Hamming space with binary hash codes representing a word. Thus a way must be found to transform the Euclidean space into a Hamming space while approximating the original embeddings' spatial relation.

Based on the paper by Shu and Nakayama [61], a model for compressing word embeddings is implemented and evaluated using different hyperparameter configurations. The work is motivated by the observation that - based on the objective functions of Skip-Gram [42, 41, 4] and Global Vectors for Word Representation (GloVe) [52] - semantically close words are located in near proximity to each other in the latent embedding space. Thus, when persisting word embeddings independently, the inter-similarity of these words is ignored and much redundant information is added to the set of embedding vectors. Motivated by the idea that similar embeddings can be encoded by similar codes, this work tries to explore whether these codes can be used for fast nearest neighbour searches in hamming space.

Their approach achieves high compression rates without worsening the performance of the downstream models they evaluate. The embeddings are compressed by around 98% for a sentiment analysis task and 94% for a machine translation task. When reducing the compression rate, even a performance gain can be observed. In this context *lossless* means without impairing downstream models. This, of course, is bound to the complexity of the evaluated models and can not be considered independently.

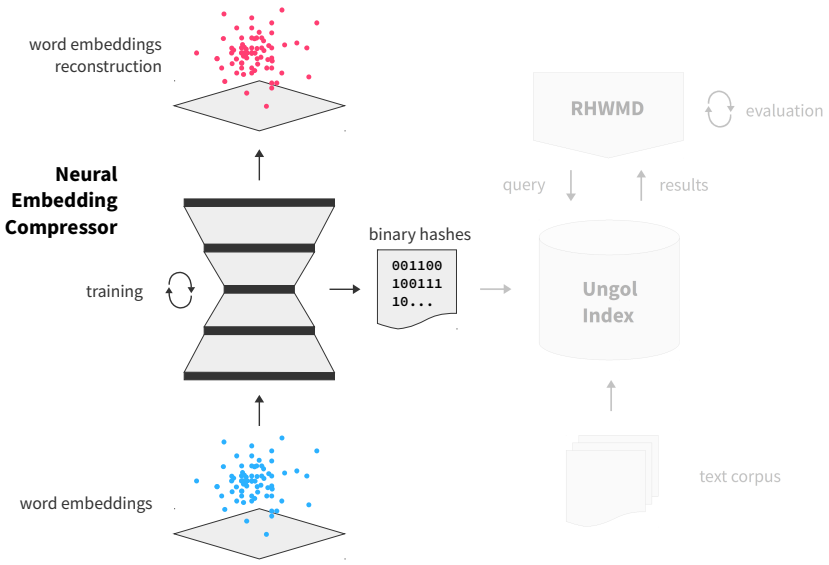


Figure 3.1: This chapter describes the process of extracting discrete hash codes for words based on real-valued word embeddings. These hash codes are then later used for a fast document retrieval.

3.1 The Neural Network

The network type is an auto-encoder which receives a single embedding vector as input and generates an output vector of the same dimensionality. Each embedding vector is trained prior and independently on large text corpora. For this work both GloVe and fastText embeddings are used. The training objective is to reduce the difference between the input sample and the generated output. For measuring this difference, simply the euclidean distance is used. Given an embedding $\mathbf{e}^w$ for word w sampled from the embedding space X of size $N \times E$ and the generated output vector $\mathbf{e'}_w$, the loss function is defined as:

$$\begin{aligned} L &: \mathbb{R}^E \times \mathbb{R}^E \mapsto \mathbb{R} \\ L(\mathbf{e}_w, \mathbf{e}'_w) &:= \frac{1}{2}||\mathbf{e}_w - \mathbf{e}'_w||_2^2 \end{aligned} \tag{3.1}$$

The model implements the compression algorithm which is based on additive vector quantization using multiple codebooks. There the idea is to use discrete hash codes to select from a set of basis vectors to create a reproduction of the compressed input through vector composition. Per word one vector

is selected from each codebook. Thus the compressed data consists of both a discrete hash code per compressed object and a collection of codebooks, each containing an enumeration of basis vectors per hash component.

3.1.1 Network Architecture

The model consists of the encoder, which is a fully connected feed forward network with one hidden layer and a discretization step to learn a one-hot encoding. The decoder uses the encoders output to select from a set of codebooks which contain basis vectors whose linear combination result in the reconstruction of the given embedding. The model's performance is targeted towards minimising the loss globally over all words from the vocabulary. The word frequency is ignored. An overview is given in Figure 3.2 and an in-depth discussion of the model's components follows in the next section. Three free parameters control the size of the model and subsequently the produced codes: M is the desired number of codebooks (which consequently is the number of code components) and K is the code domain. For example when producing a 128 bit binary hash code, the model uses $M = 128$ and $glsconst : K = 2$. To produce a code with three components and 64 possible values per component, the parameter combination is $M = 3, K = 64$. The third parameter is the dimensionality of the embedding space and is further denoted E. All experiments in the scope of this work were conducted on embedding spaces with 300 dimensions. In the following, Section 3.1.2 describes the encoder ϕ_θ and Section 3.1.4 explains the decoder ψ_θ. The model is the combination of encoder and decoder given a set of trainable parameters θ and the gradient descent optimises these on basis of the loss function defined in Equation (3.1).

$$\begin{aligned} \text{compress}_\theta &: \mathbb{R}^E \mapsto \mathbb{R}^E \\ \text{compress}_\theta(\mathbf{e}_w) &:= (\psi_\theta \circ \phi_\theta)(\mathbf{e}_w) = \mathbf{e}'_w \end{aligned} \tag{3.2}$$

3.1.2 Encoder Model

The model's encoder accepts a real valued vector of dimensionality E and produces a matrix of size $M \times K$. Each of the M produced vectors is a one-hot encoding of the component classes which defines the value of the code component. The one-hot encoded vectors are produced by a multilayer perceptron with one hidden layer. This serves as a feature detector for latent features of the embedding space. The hidden layer consists of $\frac{1}{2} \cdot M \cdot K$ neurons. Formally, the encoder is defined as follows:

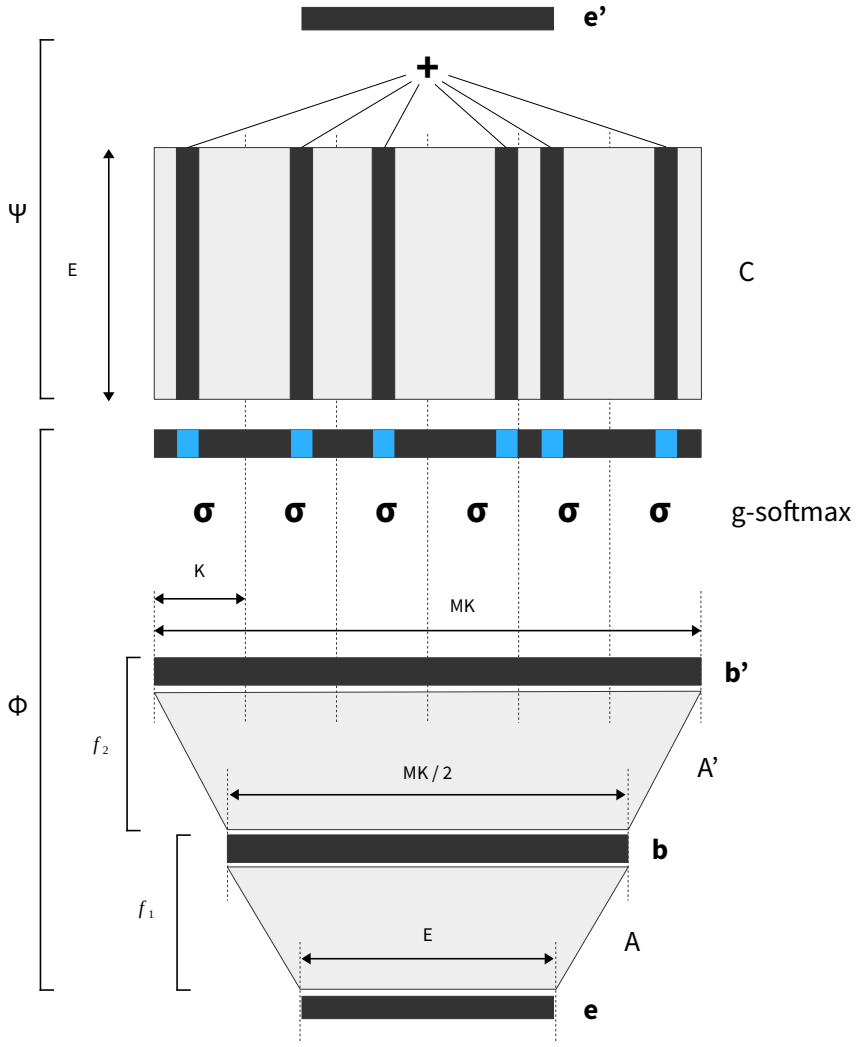


Figure 3.2: The auto-encoder architecture of Shu and Nakayama [61]. The encoder consists of two fully connected feed-forward networks and a discretization step to produce one-hot encoded embedding codes. These embedding codes are used by the decoder in conjunction with a set of codebooks to re-construct the original embedding. Each codebook contains the basis vectors needed for reconstruction.

$$\begin{aligned} \phi_\theta : \mathbb{R}^E &\mapsto \mathbb{R}^{M \times K} \\ \phi_\theta\left(\mathbf{e}_w\right) &:= \sigma(f_2 \circ f_1)(\mathbf{e}_w) \end{aligned} \tag{3.3}$$

Where f_1 and f_2 are the first and second fully connected layers. The first fully connected layer is given in Equation (3.4) and uses the hyperbolic tangent as the activation function. Equation (3.5) describes the second layer which uses the softplus function. The special function σ controls the one-hot encoding of encoder activation and differs for training and validation. This is discussed in the succeeding Section 3.1.3. The weight matrices A, A' and bias vectors $\mathbf{b}, \mathbf{b}'$ are added to the trainable model parameters θ:

$$f_1(\mathbf{x}) := \phi(A \cdot \mathbf{x} + \mathbf{b}) \qquad \text{with} \qquad \phi := \tanh \tag{3.4}$$

$$f_2(\mathbf{x}) := \phi'(A' \cdot \mathbf{x} + \mathbf{b}') \qquad \text{with} \qquad \phi'(\mathbf{x}_i) := \log(1 + \exp(\mathbf{x}_i)) \tag{3.5}$$

3.1.3 Gumbel Softmax

The bottleneck of the auto-encoder handles the code generation and is consequently responsible for the discretization of the encoder logits. This part of the model is crucial as the quality of the discretization step subsequently determines the quality of the produced codes. As formerly defined, the special function σ was used to refer to this transformation and it's properties are described in this chapter.

Two encoding functions σ are considered here: First the **training encoding function** which is based on applying a softmax to the encoder activation per codebook. Secondly the **validation encoding function** which creates a true one-hot encoding. This comes from the need to provide a continuously differentiable function while training to represent the categorical distribution of each code component. Since the model is trained using gradient descent selecting by maximum from the codebooks would not allow to propagate the gradient to the earlier layers of the network.

Given a activation vector for a codebook $m : \mathbf{x}_m = (x_1, \ldots x_{|K|})$ and a noise vector $\mathbf{g} = (g_1 \ldots g_{|K|})$ where each term is sampled from a Gumbel distribution, the Gumbel-Softmax is defined in Equation (3.6). The annealing parameter τ and the properties of the noise terms g_i are explained in Section 3.2.3.

$$
\begin{aligned}
&\text{g-softmax}_\tau : \mathbb{R} \mapsto \mathbb{R} \\
&\text{g-softmax}(\mathbf{x}_i) := \frac{\exp(f(\mathbf{x}_i))}{\sum_{j=1}^{N} \exp(f(\mathbf{x}_j))} \\
&f_\tau(\mathbf{x}_k) = \frac{1}{\tau} \cdot (\mathbf{x}_k + \mathbf{g}_k)
\end{aligned}
\tag{3.6}
$$

The second encoding type, the **validation encoding** only selects the strongest activation. Given the activation vector for a codebook: $\mathbf{x} = (x_1, \ldots, x_{|K|})$, a simple $i = \arg\max_{\mathbf{x}}$ is used to construct a one-hot encoded vector of zeroes per codebook where the i-th element is set to one.

These two encoding types operate on different premises: A low training loss is not necessarily an indicator for a good code generator because it is possible that many more and strongly non-discrete basis vectors are used for the reconstruction. To use the embeddings produced by the encoder it is necessary to choose the maximum activation. If the model fails to reconstruct the embedding using only one basis vector of the each codebook at a time, no meaningful codes can be extracted. Hence the proposed validation loss is the actual indicator for the model performance because it accurately reflects the quality of the model for reconstruction based on discrete codes. The relationship of the two losses is described in greater detail in Section 3.2.3.

The Gumbel-Softmax function as described in Equation (3.6) uses values $\mathbf{g}_i$ sampled from the extreme value distribution called Gumbel[1] distribution. This distribution is usually applied to describe extreme events. For example to build a dam high enough such that the highest flood of the next 100 years will not overflow it, the Gumbel distribution can be used to build a model forecasting the maximum flood height [19]. The probability density function $f(x)$ and the cumulative distribution function $F(x)$ with a mean of 0 and a variance of 1 are given in Equation (3.7). It can be shown that using multivariate independent and identically distributed samples $(x_1, x_2, \ldots)$ of the Gumbel distribution, the probability of any i-th component being the largest is exactly the softmax probability: $P(i \text{ is largest} \mid \mathbf{x}) = \text{softmax}(\mathbf{x}_i)$ [69].

$$
\begin{aligned}
&f(x) : \mathbb{R} \mapsto \mathbb{R} && F(x) : \mathbb{R} \mapsto \mathbb{R} \\
&f(x) := \exp(-(x + \exp(-x)) && F(x) := \exp(-\exp(-x))
\end{aligned}
\tag{3.7}
$$

[1] After Emil Julius Gumbel.

3.1.4 Decoder Model

The decoder handles the linear combination of the selected codebook vectors. Given a set of M codebook matrices $C = (C_1, \dots C_M)$ each containing K vectors with the embeddings dimensionality E, the respective vectors are selected by the output of the encoder $\mathbf{x}$. This leads to a combination of weighted basis vectors while training and a *hard* selection of M vectors for validation. The codebook matrices $C_i \in \mathbb{R}^{K \times E}$ are part of the trainable model parameters θ.

$$\begin{aligned} \psi_\theta &: \mathbb{R}^{MK} \mapsto \mathbb{R}^E \\ \psi_\theta(\mathbf{x}) &:= \sum_{i=1}^{M} C_i^T \mathbf{x} \end{aligned} \tag{3.8}$$

3.1.5 Implementation Details and Adjustments

The auto-encoder is trained using the Adam implementation for stochastic gradient descent [32]. The initial learning rate is set to 0.0005. Contrary to the reference implementation, the model's fully connected layers are initialised using the method proposed by Glorot and Bengio [17]:

$$P \sim U\Big[-1^{-\sqrt{d}}, 1^{-\sqrt{d}} \Big] \tag{3.9}$$

where d is based on the dimensionality of the matrices. The codebooks are initialised by uniformly sampling from the word embeddings.

Softmax: The former definition of the softmax differs from the original one traceable in the paper and the reference code. There, a logarithm is applied to the activation of the former layer $\mathbf{x}$. This is not feasible because the range of the last layer is defined by the softplus function, which is a continuous approximation for a rectified linear unit (ReLU) with some random noise added. It's range is in $(0, \infty)$ and given that the hyperbolic tangent of the previous layer provides values between $(-1, 1)$ — even with adding noise — it may yield many values close to zero, which in turn become close to ∞. This was found to result in *nan* values when implemented.

Loss: The loss function defined in the original paper is the mean over the L2-norm of the vector difference. However, the absolute values reported in the paper's tables indicate that in fact a function is used similar to Equation (3.1) which stems from the reference code fragment published in [77]. Thus to obtain comparable results, the given loss function is used.

3.2 Experiments

The experimental series is used to answer the following questions:

1. **Can the reported results be reproduced?** The results presented by Shu and Nakayama [61] indicate that the model's reconstruction performance improves when using more codebooks with a smaller code codomain. The objective is to find a set of hyperparameter configurations which reproduce, improve or falsify the reported results. The findings are presented in Section 3.2. No results are reported by the authors regarding binary hash codes. They are a special case with a configuration using $K = 2$. The objective of the original implementation is to maximise the compression factor whilst maintaining high quality embedding vector reconstructions using codes as short as possible. Codes with many components but binary codomain are not considered. The model's behaviour is evaluated in Section 3.2.4.

2. **Quality of the k nearest neighbours in hamming space.** For implementing a fast nearest neighbour search using hamming distances, the codes produced by a good performing $K = 2$ model are compared to euclidean and cosine distances in the original linear embedding space. The quality of nearest neighbours in the respective spaces is discussed in Section 3.2.5. Additionally, the distance distribution of those neighbours is discussed and the intersection of linear and hamming distances for different model configurations is calculated and analysed.

This section elaborates on the experiments conducted with the compressor model. The different M and K parameter configurations presented in the original paper are recreated. Additionally a full new experimental series is realised which only considers model configurations which produces binary codes (i.e. $K = 2$).

3.2.1 Loss Based Model Evaluation

Table 3.1 gives an overview over the different results obtained by training the model. All model parameters and their subsequent validation loss are selected at a point where no improvement in validation loss could be observed. The trained models are in line with the findings of Shu and Nakayama [61]. The table is sorted ascending by codebook count and it can be observed that adding codebooks indeed correlates with decreasing loss.

			GloVe (6B)		fasttext.en		fasttext.de	
M	**K**	$\lvert\Theta\rvert$	Loss	Entropy	Loss	Entropy	Loss	Entropy
		Baseline	20.17	-	12.106	-	11.468	-
8	8	19200	17.01	0.908	8.228	0.744	8.054	0.827
8	16	38400	16.12	0.935	7.282	0.854	7.060	0.888
8	32	76800	15.08	0.947	6.564	0.914	6.286	0.943
16	8	38400	15.48	0.933	7.203	0.787	7.008	0.811
16	16	76800	14.03	**0.951**	6.231	0.883	5.897	0.911
16	32	153600	12.73	0.921	5.504	**0.934**	5.127	**0.952**
32	8	76800	12.79	0.942	5.906	0.848	5.583	0.862
32	16	153600	**10.95**	0.907	5.001	0.914	4.622	0.930
32	32	307200	11.56	0.639	**4.763**	0.757	**4.199**	0.835

Table 3.1: Validation loss values for the parameter configurations used in the reference work. Additionally, loss values for fastText embeddings are also provided. The entropy value describes how well codes are distributed on average. There, a value of one describes perfectly distributed codes whereby a value of zero expresses that only one of the basis vectors is always selected from each codebook. Although the loss might attain the globally lowest values for larger models, the entropy worsens at some point. This indicates that the model is overfitting and not using the set of basis vectors optimally.

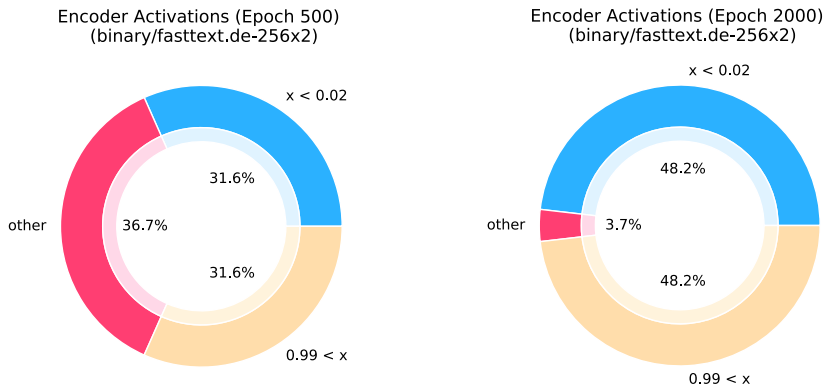
3.2.2 Entropy Measurement of Code Distribution

For the application it is of interest how well distributed the code components are over the codes domain. Because the codes are needed for searching in hamming space, well distributed codes are required for having many discriminatory features. It is imaginable that there can be a single value of a component expressing a specific feature. But given the small domain — with a maximum of 32 values — it is much more likely that features are different combination of code values. To express as many features as possible, a code distribution close to uniform is desirable.

To quantify this, entropy is provided as an additional measurement. The entropy value describes how well distributed the different code components are over the different component classes. This measures whether there are underused basis vectors inside the codebooks. In the ideal case, all vectors are queried more or less equally often and thus all spatial information of the original embedding space is distributed over the codes co domain. Low entropy either indicates that the model parameter count is bigger than necessary for catching all information, or worse: that the model is not able to properly train on the input set.

Entropy is an information theoretical measurement which describes how many bits are required at minimum to encode a message for any set of symbols. If the distribution of symbols is not known the entropy becomes exactly the $\log_2 K$ for an alphabet of K symbols. Given that symbols are not uniformly distributed but — for example — one of these symbols is very frequent and all other seldom occur, then the frequent symbol can be encoded with a very short code to reduce the overall size of the encoded message. The entropy value describing the symbols' frequencies thus becomes smaller. Given vectors $\mathbf{c} = (p(c_1), \dots p(c_K)) \in [0,1]^K$ with each component holding the partial activation count of the i-th basis vector, then there exist as many of these vectors as there are codebooks. The entropy per codebook is calculated as described in Equation (3.10). The entropy of the whole model as presented in the tables is the mean of these entropy values over the codebooks.

$$\begin{aligned} &\text{entropy} : [0,1]^K \mapsto [0,1] \\ &\text{entropy}(\mathbf{c}) := \frac{1}{log_2 K} \sum_{i=1}^{K} \begin{cases} 0 & \text{if } p(c_i) = 0 \\ -p(c_i) \cdot \log_2 p(c_i) & \text{else} \end{cases} \end{aligned} \tag{3.10}$$



(a) Values of encoder activations in epoch 500 *(b)* Values of encoder activations in epoch 2000

Figure 3.3: This figure and displays the encoder activation in epoch 500 (Figure 3.3a) and 2000 (Figure 3.3b) respectively. In this experiment in epoch 500 τ has a value of 1 and in epoch 2000 a value of 0.25. The red proportion are all activation values between 0.01 and 0.99. As the model is forced by the annealing parameter to converge towards a one-hot encoded categorical distribution, the amount of moderate values decreases.

Tables 3.1 and 3.3 show that the entropy values are very high for most configurations. This means that the codes exhibit a fair distribution over the code domain. When the parameter θ count exceeds a certain threshold both the loss increases and entropy decreases. This is an indicator for overfitting on the training data.

3.2.3 Gumbel and the Annealing Parameter

As described in Section 3.1.3, an annealing parameter is used in the Gumbel-Softmax function. This parameter τ controls the interpolation of the continuous categorical density towards a discrete one-hot encoding. The lower this parameter is set, the more is the model encouraged to train towards a categorical distribution. As τ approaches ∞, the samples become uniform.

Shu and Nakayama [61] report using a fixed value of 1.0 for τ for their training. Jang et al. [28] describe that for low values of $\tau \in \{0.1, 0.5\}$, the samples approach the categorical distribution, for $\tau >= 1$ the samples already converge towards a uniform distribution. Their recommendation is to start with a high value for τ and gradually decrease it towards a low value. The experiments conducted in scope of this work show that starting with a value of 1.25 and whilst training approaching 0.25 or 0.5 works best. Figure 3.3 shows how the activation of the encoder converges towards zero and one.

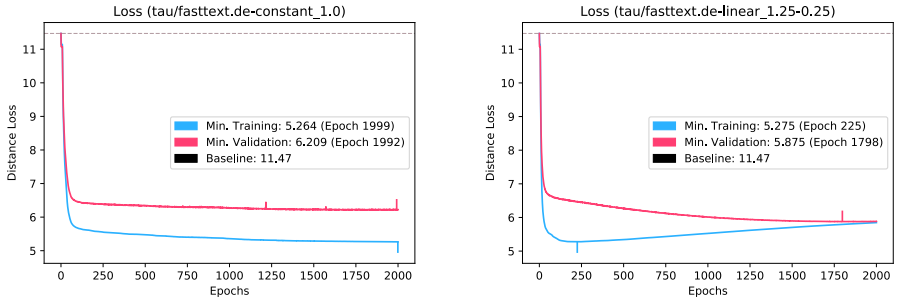
	GloVe		fasttext.en		fasttext.de	
Experiment	Loss	Entropy	Loss	Entropy	Loss	Entropy
baseline	20.17	-	12.106	-	11.468	-
linear	14.02	0.987	6.214	0.8757	5.875	0.907
fixed-1	14.91	0.988	6.537	0.893	6.209	0.922
fixed-0.5	14.16	0.943	11.806	0	11.176	0
fixed-0.25	15.18	0.707	11.806	0	11.177	0

Table 3.2: Model key figures for different τ configurations. The row denoted with *linear* change τ linearly from an initial value of 2 to 0 over 2000 epochs. Rows starting with *fixed-x* use a fixed value of x for τ. The model did not train properly for values $\tau < 1$ for fastText embeddings.

Figures 3.4a and 3.4b show how the annealing parameter affects the model performance while training. Higher values of τ lead to a very good training loss but have a negative impact on validation loss. This is the expected behaviour as described in the previous section. The less the model is forced to learn a categorical distribution, the higher is the penalty for choosing the $argmax$ whilst validating. As τ approaches lower values, the training loss and validation loss converge. For the final model performance whether τ is annealed or not does not change the overall loss much but produces slightly better code entropy. Table 3.2 summarises how the entropy changes based on the annealing strategy.

Baseline

To obtain a baseline performance, the encoder of the model is replaced by a module which samples uniformly from a normal distribution. This leads to a model which optimises it's codebook vector in such way, that the representation most likely converges towards the optimal average embedding vector representation. This baseline is used to define how well the activation of the actual encoder improve the codebook's basis vectors and subsequently the model performance. Different $M\,K$ combinations did not significantly change the baseline loss.



(a) *(b)*

Figure 3.4: Figure 3.4a displays the loss progression when using a fixed value of $\tau = 1$ throughout training. Figure 3.4b shows the relationship of validation and training loss when annealing τ linearly with an initial value of 1.25 and the final value of 0.25.

3.2.4 Binary Codes

In this section, experiments for creating binary codes — i.e. train models with $K = 2$ — are described. These codes are of higher importance in the scope of this work because fast online searches using Hamming distance can be expected to work best with binary codes. For non-binary codes, a bit-based comparison is possible too but, when encoding the classes one-hot, only a fraction of the bits can actually be used for discrimination.

Consider for example the two combinations $(M = 8, K = 2)$ and $(M = 2, K = 4)$ whose bit lengths are equal and use eight bits for encoding one-hot. For the first configuration the maximum Hamming distance is eight, for the second configuration only three. This effect worsens with fewer codebooks and greater code domain. Motivated by the observed tendency for good losses towards higher number of codebooks with smaller codomain, binary codes for various bit counts are produced and analysed. Table 3.3 gives an overview of the different loss and entropy values.

The most notable observation regarding the produced loss and entropy values is the very good validation loss on bit lengths of 256 and 512. Their loss is around 40% better than the best experiment of the former section while maintaining a very high entropy value.

3.2.5 K-NN Distance Evaluation

To be able to use the produced codes for fast exact nearest neighbour searches an analysis of nearest neighbours for different code configurations is presented in this section. For this work it is desirable to have the nearest neighbours in hamming space reproduce the nearest neighbours in the linear space. Thus it is

		GloVe		fasttext.en		fasttext.de	
M	$\lvert\Theta\rvert$	Loss	Entropy	Loss	Entropy	Loss	Entropy
Baseline		20.17	-	12.106	-	11.468	-
16	9600	17.88	0.929	9.250	**0.886**	9.005	0.922
32	19200	16.82	0.972	8.303	0.851	8.037	**0.960**
64	38400	14.99	0.981	7.332	0.729	6.809	0.932
128	76800	11.72	**0.985**	5.769	0.836	5.319	0.909
256	153600	6.986	0.961	4.064	0.838	3.682	0.873
512	307200	6.921	0.970	4.092	0.829	3.762	0.849
640	384000	3.651	0.958	1.975	0.768	1.762	0.832
768	460800	3.088	0.946	1.747	0.752	1.529	0.825
1024	614400	**2.368**	0.780	**1.406**	0.726	**1.208**	0.813

Table 3.3: Validation loss and entropy values for experiments with a fixed value of $K = 2$. The value of M also describes the bit-length needed for storing the codes. A global loss maximum for bit-lengths of 256 and 512 in combination with high entropy values can be observed. Based on this, the produced hash codes with $M = 256$ are used throughout the following chapters.

	Cosine - C				Hamming - H			
idx	**word**	**dist**$_C$	**idx**$_H$	**dist**$_H$	**word**	**dist**$_H$	**idx**$_C$	**dist**$_C$
1	compressors	0.694	2	60	turbine	60	2	0.693
2	turbine	0.693	1	60	compressors	60	1	0.694
3	supercharger	0.541	11	78	valve	72	25	0.440
4	high-pressure	0.541	8	75	engine	73	40	0.415
5	nozzle	0.524	19	82	combustion	74	8	0.503
6	turbocharger	0.513	16	80	cylinder	75	14	0.483
7	turbines	0.504	7	75	turbines	75	7	0.504
8	combustion	0.503	5	74	high-pressure	75	4	0.541
9	conditioner	0.501	54	89	engines	77	56	0.384
10	piston	0.495	23	85	cylinders	78	27	0.438

Table 3.4: Nearest neighbours of the word *compressor* for GloVe embeddings. The columns named *dist* contain the respective value for the cosine or hamming distance. The columns called *idx* denote where the word can be found using the other distance space. The red colour marks words which are not shared between both metrics.

explored how many neighbours are shared between the two spaces. All analysis is conducted on the 10, 100, 1000 and 4000 nearest neighbours of any given word sampled from the vocabulary.

The reference distance metric is the cosine distance of any given pair of input vectors. Figure 3.5 displays the average distance of the first k neighbours. It can be observed, that the peak distance does not vary much between the different classes of k. However, this does not necessarily impair the ability to find high quality nearest neighbours but is an example for the impact of sparsity in high dimensional spaces: i.e. the *curse of dimensionality*.

Figure 3.7 shows how many nearest neighbours are shared between the k nearest neighbours in the linear and hamming spaces. Although a great amount of embeddings and codes only share a small amount of neighbours, examples show that the neighbours in hamming space are different but not less intuitive. Some examples are given in Table 3.4. The underlying codes are produced by the $M = 256, K = 2$ model.

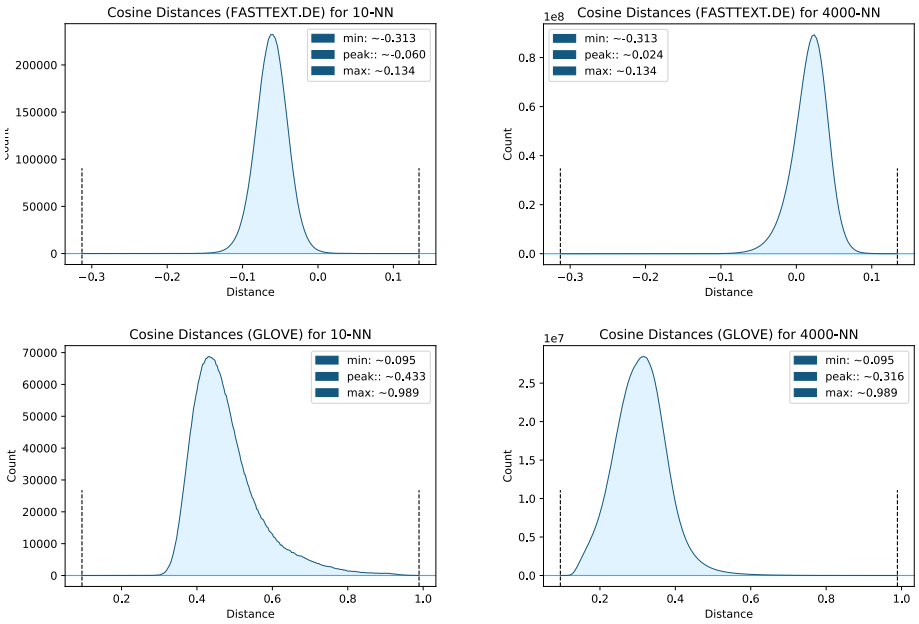


Figure 3.5: Distance distribution of nearest neighbours for GloVe and fasttext.de embeddings. It can be observed that the absolute difference between the ten and 100 nearest neighbours is very small.

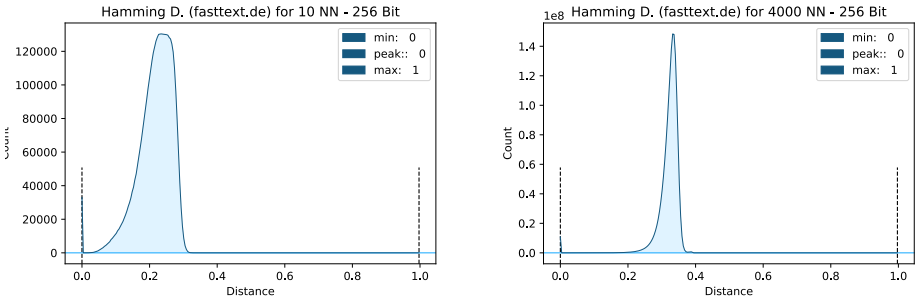


Figure 3.6: Distance distribution for nearest neighbours based on the produced codes for fasttext.de embeddings with 256 bit codes. The distribution matches the observed properties of the spatial distance distribution shown in Figure 3.5. Here, too, is the range of distance very small and the difference of the peak between the 10 and 4000 nearest neighbours reflects this.

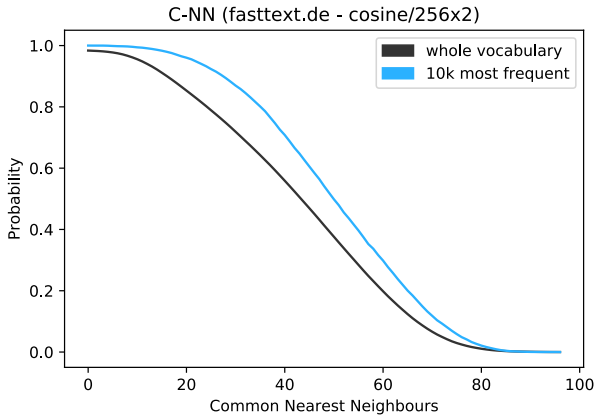


Figure 3.7: This plot shows how many nearest neighbours each word shares in the continuous space and the Hamming space. The x-axis denotes the k-th nearest neighbour and the y-axis the probability that a word shares k neighbours. Thus, nearly all words have at least one nearest neighbour in common but only two share a maximum of 96 neighbours (the words *"boraginaceae"* and *"mardinspor"*). For the whole vocabulary 6542 words have no common nearest neighbour. These plots show that the probability for finding common nearest neighbours between the two spaces is slightly higher for the more frequent words (i.e. words encountered more often when training the word embeddings).

Chapter 4

Information Retrieval

This chapter outlines how the formerly produced hash codes are used for the implementation of fast document similarity calculations. This similarity measure can be used for clustering or for ranking documents. The implementation of a search for a Information Retrieval (IR) task is the focus here. Since IR systems are aimed at being operated by humans, the response time is an important factor to take into consideration and will be studied in depth. The following Section 4.1 introduces the Relaxed Hamming Word Movers Distance (RHWMD). This algorithm is based on the Relaxed Word Moving Distance (RWMD) as theoretical foundation and uses the trained binary embedding codes for a computationally inexpensive similarity score calculation. The chapter concludes with the experiments' quantitative results and examples revealing where the approach succeeds and where it fails.

4.1 RHWMD Relevance Scoring

This section describes how — inspired by the RWMD — the binary embedding codes are used to devise and implement a scoring function. This scoring function uses a fast exact nearest neighbour search to choose the most related tokens from a document pair, chooses their distance as measure for relatedness, weights them by the Term Frequency-Inverse Document Frequency (TF-IDF) and calculates a relevancy score. This scoring function, which is further referred to as RHWMD, is then compared to Okapi Best-Match 25 (BM25) and the Word Movers Distance (WMD) in Section 4.4. This section describes both how the relevancy score is calculated and how a simple index is modelled for enabling a fast calculation of the score. Also, some modifications to the function are specified to obtain some insight into which component contributes to which amount to the final score qualitatively.

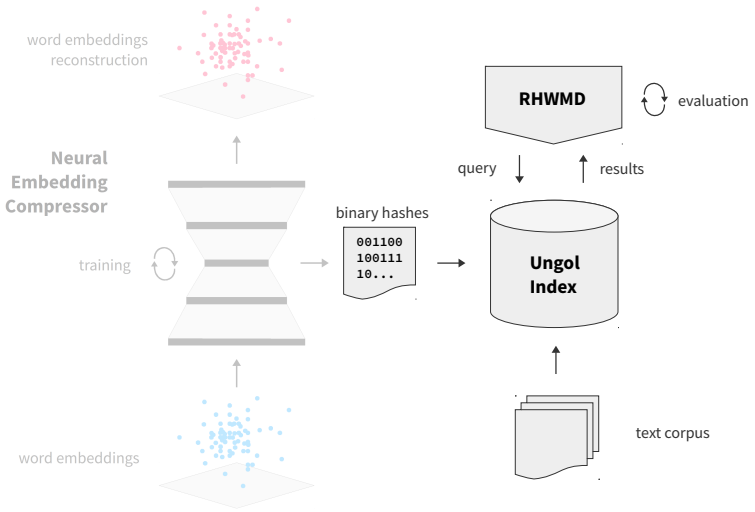


Figure 4.1: This chapter uses the formerly described codes to implement a search system. The system is evaluated using a German information retrieval data set.

4.1.1 The RHWMD Relevance Score Function

To calculate the RHWMD for a document d and a candidate document d', each word token t of the documents is mapped to a hash code $h_t \in \{0,1\}^M$. As discussed in Section 3.2.5, similar words tend to have similar codes and the more similar the words of two documents are to one another, the smaller is the average distance between the documents. The Hamming distance (further denoted as η) is used to quantify the difference between two hashes $h_t, h_{t'}$:

$$\begin{aligned} \eta &: \{0,1\}^M \times \{0,1\}^M \mapsto [0,1] \\ \eta(h_t, h_{t'}) &:= \frac{1}{M} \left| \left\{ i \mid \mathbf{x}_i \neq \mathbf{x}'_i \right\} \right| \end{aligned} \tag{4.1}$$

Here $\mathbf{x}_i$ is the i-th bit of hash h_t. The formula computes the percentage of unequal bits between the two hashes. Inspired by the RWMD, the RHWMD tries to associate each token with their respective nearest neighbour in the compared document. Finding the corresponding t' for every given t requires calculating the hamming distance $\eta(h_t, h_{t'})$ between all possible word combinations of the two documents and selecting the smallest value respectively. This leads to two sets of (t, t') relations, one of the size of the first document and one of the size of the second document. Formally, given any t in document d and a compared document d', the most similar token t' has the lowest Hamming distance to t:

$$t' = \arg\min_{t'' \in d'} \eta(h_t, h_{t''}) \tag{4.2}$$

Now, based on these nearest neighbours, a score can be calculated: Given two documents d, d' each $t \in d$ is assigned a corresponding $t' \in d'$ which is the closest nearest neighbour as described in Equation (4.2). This distance is flipped to obtain a measure of similarity and combined with an altered version of the Inverse Document Frequency (IDF) as defined in Section 2.1.3. The score s for the similarity of two documents is then:

$$\begin{aligned} s &: D \times D \mapsto [0, 1] \\ s(d, d') &:= \sum_{t \in d} \text{n-idf}(t) \cdot \big(1 - \eta(h_t, h_{t'})\big) \end{aligned} \tag{4.3}$$

The function n-idf was found to work better on the evaluation data than using the raw IDF value. Given a vocabulary V and a token t of a document d, then the n-idf value for that token is defined as:

$$\begin{aligned} \text{n-idf} &: V \mapsto \mathbb{R} \\ \text{n-idf}(t) &:= \frac{\text{idf}(t)}{\sum_{t' \in d} \text{idf}(t')} \end{aligned} \tag{4.4}$$

The score s is not symmetric. The final similarity measure applies a *fusion strategy* f to the two scores obtained when calculating both *directions* (d, d') and (d', d):

$$\begin{aligned} r_{\text{rhwmd}} &: D \times D \mapsto \mathbb{R} \\ r_{\text{rhwmd}}(d, d') &:= f\big(s(d, d'), s(d', d)\big) \end{aligned} \tag{4.5}$$

Introducing such a fusion f is motivated by the observation that for documents with very different size, large documents tend to produce smaller absolute score values. This is due to the constraint that for every word a corresponding neighbour *must* be found. Hence if two documents differ greatly in size, the large document selects from very few candidate tokens. This does not offer much information. Thus the average similarity is lower and the resulting sum decreases.

This phenomenon is very common as search queries tend to be much smaller than the candidate documents. It is examined in the experiments section for the following strategies selecting from $s_1 = s(d, d')$ and $s_2 = s(d', d)$ which fusion works best:

$$
\begin{aligned}
f_{\text{min}}(s_1, s_2) &:= min(s_1, s_2) \\
f_{\text{max}}(s_1, s_2) &:= max(s_1, s_2) \\
f_{\text{small}} &:= \begin{cases} s_1 & \text{if} \quad |d| < |d'| \\ s_2 & \text{otherwise} \end{cases} \\
f_{\text{big}} &:= \begin{cases} s_2 & \text{if} \quad |d| < |d'| \\ s_1 & \text{otherwise} \end{cases} \\
f_{\text{sum}} &:= s_1 + s_2
\end{aligned}
\tag{4.6}
$$

4.1.2 Example Application of the RHWMD

To further motivate the proposed formula, consider the example documents given in Table 4.1. Table 4.2 breaks the respective scores down to the atomic values. The sum of the elements in the *weight* column form the final score and the product of the *sim* and *idf(token)* divided by the sum of all IDF values. The score for comparing *wanze* to *teppich* is **0.642874** and *teppich* to *wanze* is **0.674287**; therefore lower than the scores for the more related documents. This is an example where the strengths of the RHWMD are revealed: Procedures such as BM25 and TF-IDF are not able to assign meaningful scores because none of the documents share words with high IDF values.

4.2 Implementation

This section describes the implementation of the RHWMD as runtime performance is an important requirement. The implementations for two critical challenges are discussed here: First the implementation of η for finding the hamming distance between terms and secondly the construction of a distance matrix between all possible term combinations and selection of closest nearest neighbours in both directions. For both challenges, three implementations are compared. One of the implementations is called *naïve* and is used as a baseline to obtain an intuition for the degree of improvement of the other candidates.

word	TC	1-NN	10-NN	100-NN	1000-NN	4000-NN
Document 1: *Auf der Mauer, auf der Lauer sitzt 'ne kleine Wanze.* (id=wanze)						
auf	2	0.219	0.230	0.262	0.289	0.309
der	2	0.168	0.211	0.246	0.281	0.305
mauer	1	0.180	0.223	0.262	0.316	0.336
lauer	1	0.266	0.285	0.309	0.332	0.348
sitzt	1	0.211	0.262	0.281	0.312	0.332
ne	1	0.227	0.262	0.293	0.324	0.344
kleine	1	0.137	0.246	0.277	0.305	0.320
wanze	1	0.188	0.242	0.293	0.340	0.359
Document 2: *Ein Marienkäfer schläft auf dem Zaun.* (id=marienkaefer)						
ein	1	0.148	0.203	0.230	0.277	0.309
marienkäfer	1	0.156	0.188	0.266	0.320	0.348
schläft	1	0.188	0.230	0.266	0.309	0.336
auf	1	0.219	0.230	0.262	0.289	0.309
dem	1	0.176	0.219	0.250	0.285	0.309
zaun	1	0.199	0.246	0.297	0.328	0.344
Document 3: *Ich möchte diesen Teppich nicht kaufen.* (id=teppich)						
ich	1	0.141	0.184	0.223	0.270	0.301
möchte	1	0.180	0.207	0.238	0.273	0.301
diesen	1	0.145	0.207	0.242	0.273	0.297
teppich	1	0.141	0.246	0.301	0.328	0.344
nicht	1	0.137	0.180	0.215	0.262	0.293
kaufen	1	0.133	0.188	0.258	0.305	0.332

Table 4.1: This table displays three example documents for later comparison. Apart from the Term Count (TC) column several distances of the k-th nearest neighbour for the specific tokens are shown. This is to call attention to a challenge: The first nearest neighbour for some terms is as far away as the 100th for other terms (e.g. *lauer*'s first nearest neighbour: 0.266 and *kleine*'s 100th nearest neighbour: 0.277). Further discussion can be found in Section 5.3.

token	nn	sim	tf(token)	idf(token)	idf(nn)	weight
comparing *wanze* to *marienkaefer* - score: **0.711667**						
wanze	marienkäfer	0.742	0.100	9.335	9.649	0.200
lauer	schläft	0.668	0.100	7.134	7.138	0.137
ne	auf	0.664	0.100	5.561	0.305	0.106
mauer	zaun	0.758	0.100	4.674	5.974	0.102
sitzt	schläft	0.727	0.100	4.361	7.138	0.091
kleine	auf	0.688	0.100	3.290	0.305	0.065
auf	auf	1.000	0.200	0.305	0.305	0.009
der	dem	0.785	0.200	0.033	0.370	0.001
comparing *marienkaefer* to *wanze* - score: **0.745616**						
marienkäfer	wanze	0.742	0.167	9.649	9.335	0.301
schläft	sitzt	0.727	0.167	7.138	4.361	0.218
zaun	mauer	0.758	0.167	5.974	4.674	0.191
auf	auf	1.000	0.167	0.305	0.305	0.013
dem	der	0.785	0.167	0.370	0.033	0.012
ein	der	0.758	0.167	0.325	0.033	0.010

Table 4.2: In this table it is shown how the score is calculated. The column denoted *weight* lists the similarity value weighted by the normalised IDF. The sum of the weights form the score.

4.2.1 Hamming Distance

The hash codes are encoded as arrays of one-byte values and thus for the used 256 bit encoding consist of 32 bytes. This implies that a single code can not be loaded into memory as a primitive type on a regular 32 or 64 bit OS architecture which enforces the implementation to use arrays. In the following, B denotes the number of bytes of a code which is $\frac{1}{8}M$. The three considered approaches for calculating the hamming distance are described below and the execution speed is listed in Table 4.3. Given are two codes $h_t = (b_1, \ldots, b_B)$ and $h_{t'} = (b'_1, \ldots, b'_B)$ as byte-vectors:

1) **hamming-naïve:** Using two loops and a bit-mask, shifting the *exclusive or* ($\oplus$) of the two bytes bit-wise eight times:

$$\eta(h_t, h_{t'}) := \frac{1}{M} \sum_{i=1}^{B} \sum_{j=0}^{7} \left((b_i \oplus b'_i) >> j \right) \wedge 1 \tag{4.7}$$

2) **hamming-bin:** This implementation uses the builtin Python routine *bin()* for obtaining a string representation of the binary number and counting the character '1' and using *count()* for obtaining the number of ones in the string. This number is then divided by B.

3) **hamming-lookup:** There are only 256 possible outcomes of $b \oplus b' \; \forall \; b, b'$. Therefore a lookup table is constructed as a vector $\Xi := (x_0, \ldots x_{255}) = (0, 1, 1, 2, 1, \ldots)$ is constructed with each x_i saving the number of ones of i for any byte. Consider for example $10011_2 \oplus 110_2 = 10_2 = 2_{10}$. The number 2_{10} has a bit count of 1 and to obtain this value, the vector $\Xi_2 = 1$. The inner loop of the **naïve** implementation becomes obsolete and the hamming distance is:

$$\eta(h_t, h_{t'}) := \frac{1}{M} \sum_{i=1}^{B} \Xi_{b_i \oplus b'_i} \tag{4.8}$$

Intriguingly the **hamming-naïve** implementation is a magnitude slower than **hamming-bin**. Even the overhead of allocating memory for the string representation is not slowing the use of the builtin functions so heavily. The lookup table solution — unsurprisingly — outperforms the other two implementations and adds a magnitude of execution speed.

1)	**hamming-naïve**	140μs
2)	**hamming-bin**	10μs
3)	**hamming-lookup**	4μs

Table 4.3: Execution speed for the hamming distance calculations.

4.2.2 Distance Matrix

For implementing the RHWMD it is necessary to select the nearest neighbour pairs of words between two documents. Therefore every token combination amongst the two documents needs to be computed to select the nearest neighbours. This inherently requires $O(|t| \cdot |t'|)$ computations. The time complexity of the **hamming-lookup** variant of η is $O(3B)$ (for the xor, memory lookup and summation). Hence the minimum, maximum and average time complexity for all implementations is $O(|t| \cdot |t'| \cdot B)$. Nonetheless by using the power of native routines implemented for numpy's vectorized operations much runtime performance is gained. The goal of the computation is to obtain a matrix $T \in [0,1]^{|d| \times |d'|}$ containing the normalized hamming distances.

Three implementations are compared with each other: The **t-naïve** implementation uses the **hamming-lookup** version of the μ function and **t-vectorized** applies **hamming-bin** for a comparison to the **t-lookup** variant.

1. **t-naïve:** Two nested loops iterating over each possible combination, applying the hamming function per pair:

$$T_{ij} := \eta(h_t, h_{t'}) \tag{4.9}$$

2. **t-vectorized:** This approach tries to facilitate numpy's vectorized operations for a faster native execution. This consists of constructing two matrices $H, H' \in \{0, \ldots, 255\}^{|d| \times |d'| \times B}$ with:

$$H := \begin{bmatrix} h_1 & \ldots & h_1 \\ \vdots & \ddots & \vdots \\ h_{|d|} & \ldots & h_{|d|} \end{bmatrix} \qquad H' := \begin{bmatrix} h'_1 & \ldots & h'_{|d'|} \\ \vdots & \ddots & \vdots \\ h'_1 & \ldots & h'_{|d'|} \end{bmatrix} \tag{4.10}$$

Now $Z := H \oplus^v H'$ creates a three dimensional tensor of byte-wise xor'ed code values. The operations labelled with the super-scripted v ($\oplus^v$, μ^v) accept whole matrices and apply the operation element-wise on the last

dimension. Numpy's *vectorize()* function is used to create η^v which accepts whole matrices of codes. The final matrix is created by calculating the Hamming distance per byte value and summing all the calculated bit counts:

$$T_{ij} := \sum_{k=1}^{B} \eta^v(Z)_{ijk} \tag{4.11}$$

3. **t-lookup:** This method uses the matrix $Z := H \oplus^v H'$ to retrieve the bit-count values from the lookup table Ξ:

$$T_{ij} := \sum_{k=1}^{B} \Xi_{Z_{ijk}} \tag{4.12}$$

A benchmark with several matrices reveals the great performance difference. The lookup-version (again unsurprisingly) outperforms the other approaches. Given that the documents used in this work usually have a mean token count of around 200, this speedup is essential for a sane retrieval time. Figure 4.2 displays the speed for different token counts. It is important to highlight that the speed increase stems from the ability to use lookup tables which is not possible when using spatial distance metrics of the original euclidean embedding space for obtaining exact matches.

The proof-of-concept is written in Python[1] which — as an interpreted language — inherently executes slower than a Ahead-of-Time (Compilation) (AOT) or Just-in-Time (Compilation) (JIT) compiled code. Also control structures such as loops or binary operations behave differently performance-wise in comparison to code compiled to machine code. This is an important fact to keep in mind for the observations as they would be much different in a language such as C. The execution time value is based on the minimum of $2 \cdot 10^5$ runs.

It is imaginable to also build a lookup table for all possible word combinations in the offline phase and combine that with an intelligent in-memory cache to reduce I/O time but this would require space for $(|N|^2 \cdot (8 + 4)$ bytes (8 bytes for a 64bit address and 4 Bytes for the 32 bit float value). Even when selecting only the 40k most frequent words, this would lead to an additional footprint of around 20 gigabytes with rarely or never used information. Building a clever caching infrastructure or devising approaches for horizontal scaling are not in the scope of this work.

[1] The Ubuntu pre-compiled packaged Python 3.6 using numpy executed by the CPython interpreter without any local modifications. All benchmarks are run on a single core of an Intel i7-6700 with 3.4GHz.

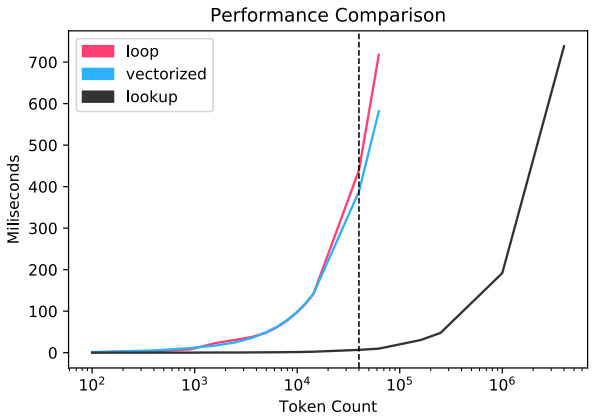


Figure 4.2: Benchmark for calculating the distance matrices for documents using the three different implementations. On the x-axis the total number of token comparisons is given. The y-axis denotes the time in milliseconds needed for the computation. For two documents with a combined token count of 400, 40000 distances are computed which takes around 8ms using the lookup table but 440ms using the naïve and 390ms using the vectorized implementation.

4.2.3 The Index Structure

To calculate the necessary metrics for the RHWMD, the following information must be provided: Each document is assigned the respective hash codes and the IDF values for each token. Additionally, for future work and for experimentation the count of each term per document is saved (both normalised and absolute). Such document-local information does not change when new documents are added to the index. Other metrics such as the amount of documents containing a specific term must be updated when new documents are added to the index.

All this data needed for computation should be stored as space efficient as possible. Additionally, for allowing the use of multiple cores of the CPU when calculating the score, the whole database must be accessible from multiple processes concurrently. To do so, the data model for the persistence is designed and presented in this section and further referred to as the *Ungol Index*.

Data Model

An Ungol Index instance contains both a reference to a meta object called *DocReferences* and a mapping of unique document id's to *Doc* instances. Each document of the corpus is transformed to a *Doc* such that the most important information — its tokens and associated term counts — are readily available. To

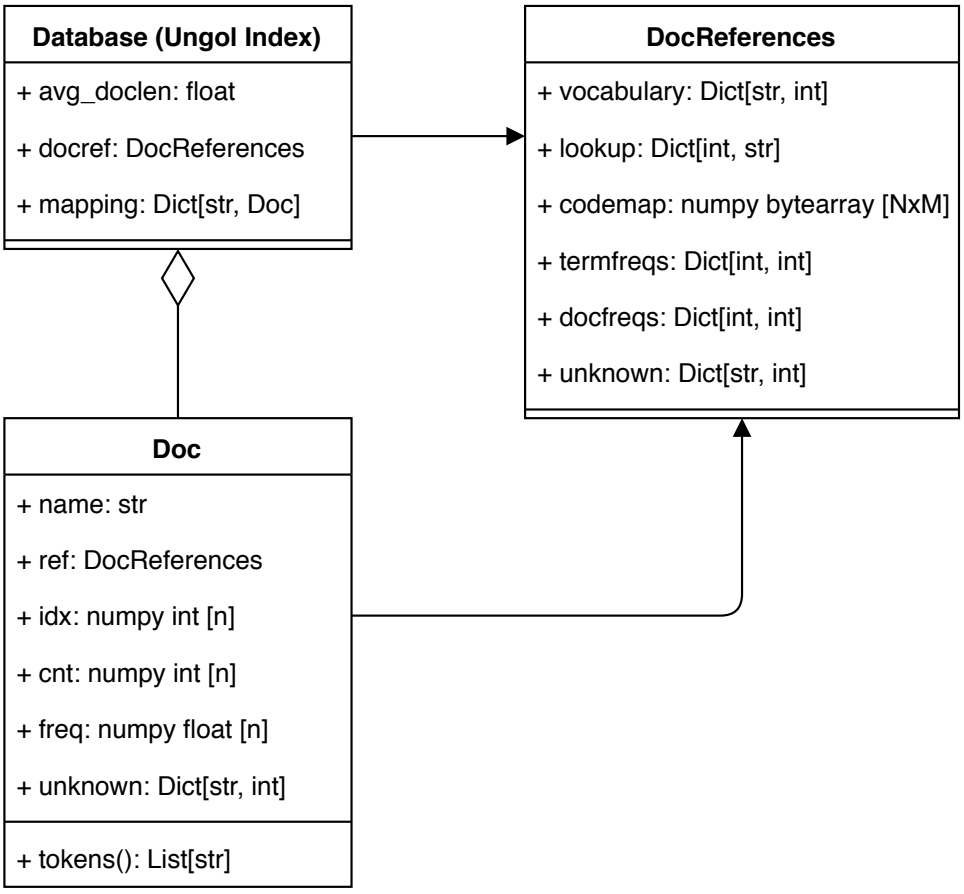


Figure 4.3: The Ungol Index consists of a data structure which collects *Doc* instances. Each *Doc* has an *idx* property for looking up its tokens in the *codemap* of *DocReferences*. The *cnt* property holds the TC for i-th token (enumerated over *idx*). The property *freq* holds the normalised TC.

token	index	TC	TF	DF	IDF	code
ribbeck	62043	2	0.286	40	8.905	`0x01 0xa6 0x21 0x51...`
herr	1568	1	0.143	6016	3.891	`0xa4 0xae 0x34 0x41...`
von	10	1	0.143	236283	0.221	`0xa8 0xa6 0x34 0x11...`
auf	21	1	0.143	217148	0.305	`0xa8 0x24 0x74 0x5b...`
im	13	1	0.143	225024	0.270	`0x60 0xec 0x10 0x5a...`
havelland	42441	1	0.143	15	9.886	`0x00 0x1c 0x46 0xc0...`

Table 4.4: Data readily available in the combination of a document instance and a reference instance. The document only saves the index, Term Count (TC) and Term Frequency (TF). The tokens are obtained by selecting from the lookup mapping, the DF stems from the document frequency mapping and the IDF is calculated based on the DF vector and the size of the index. The original text is: "Herr von Ribbeck auf Ribbeck im Havelland". The global metrics stem from a index fully populated by documents from the corpus introduced in Section 4.3.1

reduce the footprint as much as possible, only unique information about a document is saved to the *Doc* instance. Thus, all possibly redundant data is saved to *DocReferences*. This includes the $\{0, 1\}^{N \times M}$ shaped code map containing the hash codes for the vocabulary and the IDF information, which is a mapping of a token index to an absolute count of documents containing this token.

This mapping allows for the following data model: Every document contains an index vector where each element of the vector denotes the position of the respective hash code for that token in the code map. A detailed view of the data model is given in Figure 4.3. The count and frequency vectors also held by the *Doc* object contain the absolute and normalised count of the token's appearance in the document. The *DocReference* object additionally contains two mappings that resolve tokens to indexes (the vocabulary attribute) and vice versa (the lookup attribute). Table 4.4 illustrates the available data.

Indexing

Indexing a document is presented in this subsection. Figure 4.4 shows how the text is processed by the system. The document collection is read from disk and the raw text content is offered to a pool of worker processes. Every worker process awaits a raw text string per document to be processed. Each string blob is split by a German word tokenizer [75]. Each token is then sanitised by removing

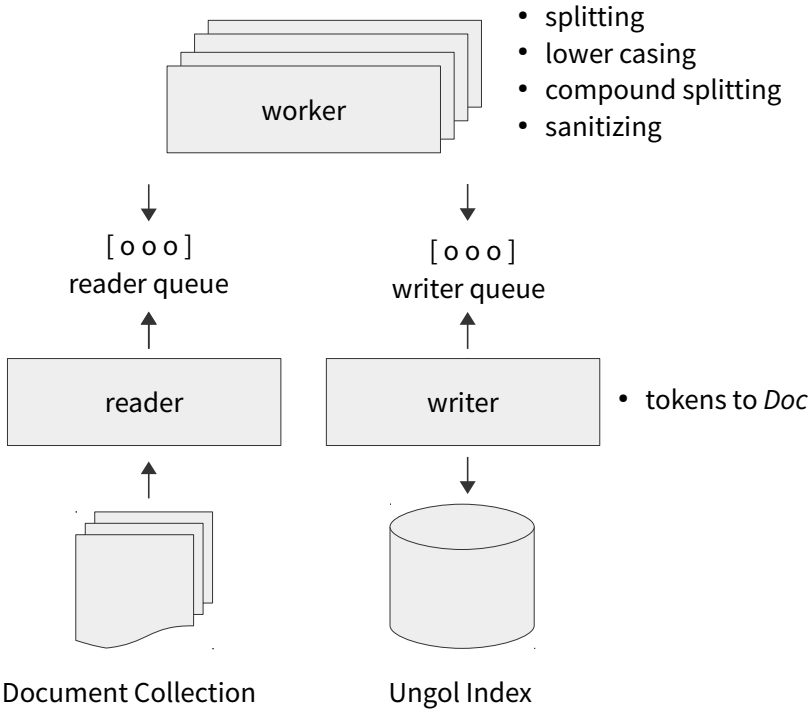


Figure 4.4: Text processing in the indexing phase. A pool of workers handle all heavy lifting while a reader continuously provides new data from disk. The writer process merges documents in the index as soon as they have been pre-processed.

most non-alphanumeric characters, lower-casing it and recursively splitting it into its compounds[2] if possible. The resulting token vector is then offered to the writer process. This process converts the token vectors into *Doc* instances and adds them to the index mapping. The index mapping uses a unique identifier for the document. Every update to the index mapping also updates the metrics inside the meta block: Most notably, the document frequencies which are used for the IDF calculation. This pipeline can be used to add new documents to the index at any time.

[2] The current implementation uses the *char_split* compound-splitter by dtuggener [71] based on the work of Tuggener [67].

Concurrency

The formerly explained implementation also allows for one important optimisation: distributing the work over the cores of the now established multi-core processors. The score calculations are independent from one another and thus can be computed independently. This leads to a system — given that the index can be made accessible read only from multiple processes — which has a minimal memory footprint for inter-process communication (namely two strings go in, one float comes out) and a minimal overall footprint (only one index in memory at a time). The implementation works reasonably well and was found to scale linearly with the number of cores used for processing.

4.2.4 Out of Vocabulary Words

There are words where no embeddings exist in the Ungol Index. These words — further referred to as Out of Vocabulary (OOV) words — can contain important information and should not be ignored. The current implementation follows a simple approach: If the two documents for which the RHWMD is calculated share OOV words, than naturally they have a similarity value of one assigned. The IDF value can be computed as formerly described, because the *docfreq* mapping keeps track of all encountered tokens regardless whether they appear in the *codemap* or not.

4.3 Evaluation Setup

For measuring the quality and comparing the different scoring algorithms a German information retrieval data set is used. This data set's content, ground truth labelling and the metrics used for measuring the ranking quality are introduced below in Section 4.3.2. The scoring functions are evaluated on a German news corpus introduced in the following section Section 4.3.1.

4.3.1 The Corpus

A corpus of sufficient size must be employed to adequately evaluate the scoring function performance. Only if there are enough negative samples at choice the quality of the scorer to decide which documents are relevant or not can be judged with confidence. For this task the corpus consists of around 300k news articles in German. The length of the documents varies and ranges from very short press releases to multi-page articles. The articles stem from "Frankfurter Rundschau", a German newspaper, "Der Spiegel", a German news magazine and

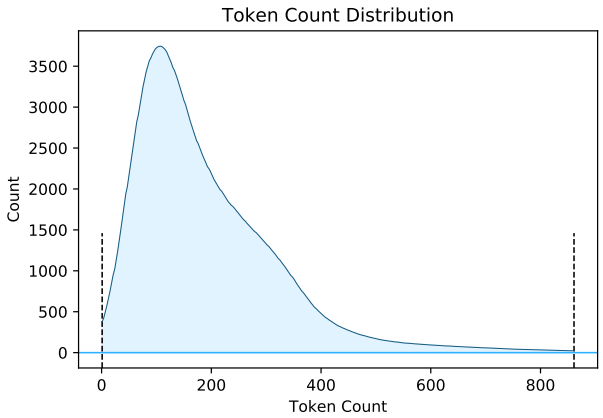


Figure 4.5: This plot shows the size distribution of the corpus described in Section 4.3.1. Most documents have up to 400 tokens and around 200 on average.

"Schweizerische Depeschenagentur (SDA)"[3] the Swiss national press agency. Figure 4.5 shows the distribution of unique tokens per document and Figure 4.6 displays the global distribution of term and document frequencies. Table 4.5 numerates some statistics after transforming the corpus as formerly described in Section 4.2.3.

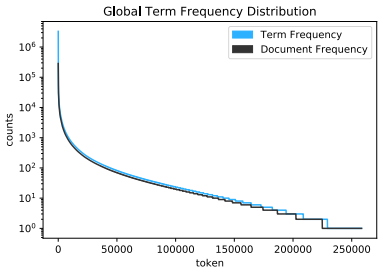
4.3.2 CLEF 2003 German Monolingual

For evaluation the 2003 ad-hoc German monolingual information retrieval task of the Conference and Labs of the Evaluation Forum (CLEF) is selected. The ground truth data consists of 56 *Topics*. These topics form an information retrieval task – a question to be answered by a set of documents. Below a sample topic description is given:

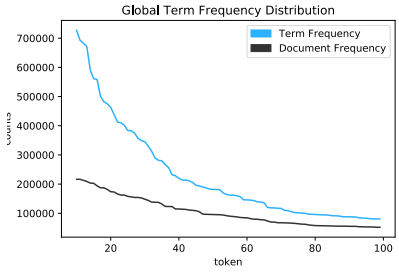
> *Topic 174: Finde Berichte über den Kruzifixstreit*
> *in bayerischen Schulen.*

The ground truth offers in this case 36 positives to be retrieved from the corpus. Additionally 432 samples are enumerated which are labelled incorrect. All documents not appearing in the numeration are also irrelevant. The amount of positives varies greatly. For some topics only one article is correct, other topics are assigned up to 226 articles. The following article is one of the 36 positives (truncated) for *Topic 174*:

[3] Since 2018 "Keystone-SDA"



(a) The whole corpus.



(b) The section of the 10th most frequent to the 100th most frequent token.

Figure 4.6: These two plots show the distribution of the TF and DF of each known token of the corpus. Note that the frequencies are sorted by total count independently. This means that the i-th DF might belong to a different token than the i-th TF. Also mind the logarithmic scale in Figure 4.6a.

Index	180921_ungol.db.pickle
Documents	294659
Vocabulary	400000
Code Size M	256
Unique Tokens	258391
Average Document Length	197
Skipped	8
Memory Footprint	$\sim$ 3 Gigabytes

Table 4.5: Ungol Index statistics after processing all documents and topics. This is the version of the index all evaluation runs were conducted on. Skipped documents are those which did not contain any usable tokens. The average document length is rounded and measured in tokens. Note that the size of the *codemap* is only around 100MB in size in contrast to the original real-valued embedding matrix which consumes around 1GB. The overall memory footprint is augmented by Python's internal memory maintenance.

München, 22. Sept. (sda/Reuter) Erstmals seit dem Kruzifix-Urteil des deutschen Bundesverfassungsgerichts sind in zwei bayerischen Schulen Kreuze abgehängt worden. Dies erklärte am Freitag in München der Sprecher des bayerischen Kultusministeriums. (...)

4.3.3 Evaluation Metrics

For determining the score quality a common approach is to measure the mean average precision μ_{AP}. Here recall and precision of a ranking system are combined such that the higher a system ranks the correct samples the higher the mean average precision becomes. Usually precision decreases while recall increases the more candidates are considered.

Formally, given a ranking system for Q classes returning k results the mean average precision is computed for every class by using two vectors:

$$\mathbf{p} = (p_1, \ldots, p_k), \mathbf{r} = (r_1, \ldots, r_k)$$

Each element p_i describes the precision value of the system when returning i elements. The vector $\mathbf{r}$ describes the change in recall which for the discrete case is simply the difference of the i-th value with the previous one: $r_i = r_i - r_{i-1}, r_o = 0$. Every r_i is greater or equal to zero because the recall is monotonously increasing. The mean average precision simply takes the mean over all obtained average precision values[4]:

$$\begin{aligned} \mu_{\text{AP}} &: [0,1]^{q \times k} \times [0,1]^{q \times k} \mapsto [0,1]^q \\ \mu_{\text{AP}}(\mathbf{P}, \mathbf{R}) &:= \frac{1}{Q} \sum_{q}^{Q} \sum_{i}^{k} \mathbf{r}_{qi} \mathbf{p}_{qi} \end{aligned} \tag{4.13}$$

4.4 Experiments

Two experiments are conducted and analysed here. The first one assesses the ranking performance by comparing different corpus sizes. It is explored how the average precision behaves both regarding the addition of negative candidates to the pool of possible candidates and in comparison of the different ranking algorithms. This experiment is further denoted as the *Baseline Comparison*. Secondly a set of documents is pre-selected by using an inverted index and BM25 and it is assessed whether RHWMD is able to increase the search result quality

[4] The resulting values are multiplied by 100 in the following.

Algorithm	250	500	1k	5k	10k	50k	100k
wmd	69.62	63.85	56.84	41.09	35.05	20.94	15.06
rhwmd.big	76.76	67.80	59.46	40.47	32.91	18.00	12.45
tfidf	77.75	68.96	60.44	42.17	35.07	19.98	13.91
rhwmd.min	84.12	76.41	70.02	50.40	41.16	22.88	16.33
rhwmd.max	94.10	90.54	86.45	73.22	66.41	50.59	42.87
rhwmd.small	94.10	90.54	86.45	73.22	66.41	50.62	42.90
bm25	93.07	89.99	86.69	76.36	70.64	56.94	**51.02**
rhwmd.sum	**96.03**	**93.44**	**90.27**	**78.91**	**72.80**	**57.49**	49.82

Table 4.6: μ_{AP} values for various retrieval configurations. Each column contains the performance of the employed retrieval algorithm for a specific data set size. With increasing set size, only the amount of negative samples increases. The smallest configuration of $k = 250$ already contains all positives for all topics because the largest amount of positives for a single topic is 226. The rows are ordered by their average. The standard deviation is below 1 for all experiment configurations.

by re-ranking the retrieved documents. This approach is targeted towards a real-world use case scenario where short retrieval times are usually a requirement. The experiment is referred to as the *Re-Ranking Experiment*.

4.4.1 The Baseline Comparison

To develop an intuition for the theoretical capacity of the RHWMD scoring and to gather data about the execution speed, subsets of the whole CLEF corpus are considered for retrieval. Each experiment is using a document collection of size $k \leq |D|$ per topic. Each subset is constructed by combining all relevant documents for the respective topic with randomly sampled non-relevant articles. The evaluation is run on three independent subsets per k and averaged for the final result. The total size of each data set is thus $k \cdot 56$ (the number of topics) but every topic only selects from k documents. This in principle emulates a perfect pre-selection algorithm with a recall of 100%.

The evaluated methods are:

1) **WMD:** Solving the transport problem on real-valued embedding words to transform one document into the other. Measure the transformation cost.
(see Section 2.4 on page 23)

2) **TF-IDF:** Calculating the Term Frequency-Inverse Document Frequency.
(see Section 2.1.3 on page 14)

3) **BM25:** Retrieve with the Okapi Best-Match 25 relevancy score.
(see Section 2.1.4 on page 14)

4) **RHWMD:** The approach developed in this work. (see Section 4.1 on page 47)

    - **sum:** Combine both scores by summation.
    - **big:** Selecting only the score from the larger document.
    - **small:** Selecting only the score from the smaller document.
    - **min:** Always selecting the smaller score.
    - **max:** Always selecting the bigger score.
    - **sum:** Combine both scores by summation.

Results

Table 4.6 summarises the promising findings. Always selecting the bigger document as the direction for score calculation or always selecting the smaller score of the two considered directions is not working well. Combining the directional scores by summation consistently albeit not grandly outperforms BM25. Taking the maximum or always the score of the smaller document is working reasonably well but their performance decreases with increasing candidate amount by a larger rate than BM25. Figure 4.7 shows a plot with the best and the worst ranking RHWMD implementations, BM25 and TF-IDF. Using TF-IDF as one of the employed algorithms serves as the baseline for the degree of information the similarity measures contribute. As prior defined, RHWMD uses TF-IDF as an important factor of the score calculation but weights it by nearest-neighbour similarity. The difference between TF-IDF and RHWMD is thus an indicator for the impact of the similarity measure. However, given the great difference in performance between the different RHWMD fusion strategies, this weighted similarity must be combined with some sensitivity. Overall, the decrease in difference between RHWMD and BM25 is noticeable and leads to BM25 outperforming RHWMD for k larger than 50k. BM25's hyper-parameters: $k_1 = 1.56$ and $b = 0.45$ are chosen based on a grid-search (see Table 4.7).

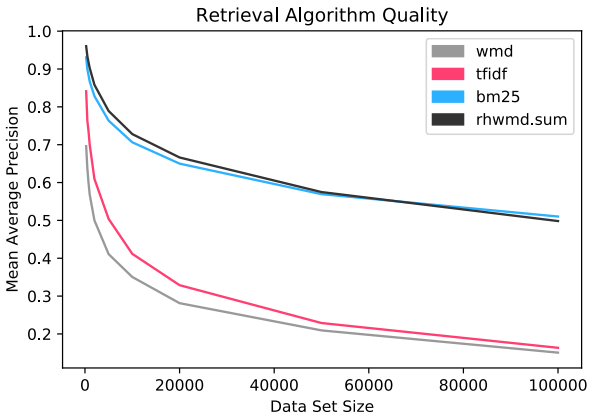


Figure 4.7: Mean Average Precision of the best and the worst ranking RHWMD implementations, BM25 and TF-IDF for a result list of $k = 250$

Problematic however, as formerly described in Section 2.1.2 is the retrieval time. This is primarily a result of searching data linearly which leads to unacceptable retrieval times for large data sets as shown in Figure 4.8. However, the RHWMD implementation outperforms the WMD runtime performance significantly (although the WMD implementation already uses all the optimisations such as using the Word Centroid Distance (WCD) and RWMD for candidate preselection).

4.4.2 The Re-Ranking Experiment

The former experiment showed that the RHWMD scoring has the potential to outperform BM25 for candidate sets of manageable size. In this section an experiment is conducted which uses an inverted index and BM25 for a fast ranking of document candidates. It is determined how well this ranking performs qualitatively using the μ_{AP}. A rank k is chosen on which the candidates are confined. This now smaller candidate set is given to RHWMD for re-ranking and the μ_{AP} is calculated again to examine whether the ranking was improved. The goal of the experiment is to determine whether BM25 is suitable for a candidate preselection and if RHWMD is able to improve the BM25 ranking.

Elasticsearch [70] is used for the fast retrieval of candidates. The inverted index is based on Apache Lucene [72]. Elasticsearch offers a production ready data store with a language agnostic interface, its own Okapi BM25 implementation (among others) and its own pre-processing pipeline for indexing.

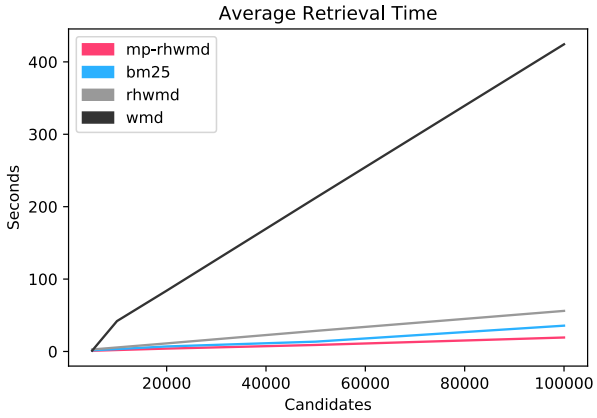


Figure 4.8: Average time needed to compute the score for a query and k candidate documents on a single core using BM25 and RHWMD linearly (i.e. not using an inverted index). The run-time of *mp-rhwmd* is using the multiprocessing version and eight cores. The implementation of WMD also uses all eight cores and the optimised pre-selection with the RWMD and WCD.

$k_1 \backslash b$	**0.00**	**0.15**	**0.30**	**0.45**	**0.60**	**0.75**	**0.90**	**1.00**
1.20	26.68	35.18	38.15	39.12	38.88	38.08	35.12	33.88
1.32	28.69	35.42	38.39	39.21	38.95	38.22	35.01	33.55
1.44	28.59	35.54	38.29	39.14	39.00	38.10	34.96	33.48
1.56	28.55	35.47	38.37	**39.23**	39.02	38.05	34.93	33.30
1.68	28.67	35.25	38.30	39.14	38.99	37.99	34.79	33.13
1.80	28.65	35.13	38.25	39.10	38.94	37.25	34.68	32.95
1.92	28.64	35.19	38.23	39.15	39.05	36.90	34.59	32.77
2.00	28.62	35.19	38.18	39.10	39.00	36.86	34.50	32.69

Table 4.7: Mean Average Precision (μ_{AP}) values of a 8×8 grid-search with $k = 2000$. It is noticeable that k_1 — which controls the weight of term frequencies — does not have much impact. Only the choice of b has a considerable effect on the scores which hints that the normalisation regarding document length is an important factor when working with this corpus.

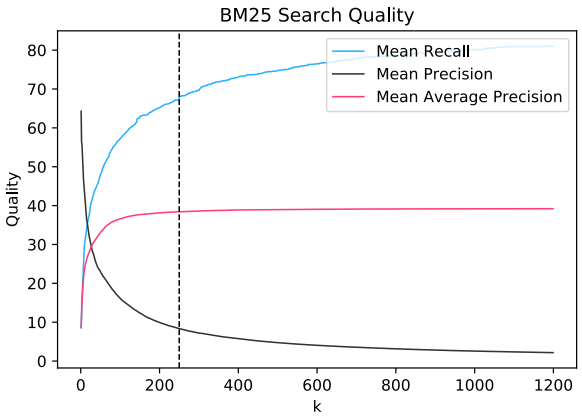


Figure 4.9: Evolution of Recall, Precision and Average Precision averaged over all topics when increasing the number of search results. The increase in μ_{AP} becomes negligible as soon as the Precision approaches zero. The dashed line marks the threshold chosen for pre-selection.

Results

The baseline experiment using only BM25 reached a μ_{AP} of around 39 with $k = 2000$. As displayed in Figure 4.9, this value is not changing much from 250 results on. Thus this is chosen as the pre-selection threshold. Table 4.8 shows the outcome. The BM25 (Ungol) implementation is the one also used in the formerly described pre-selection experiment and is used to ensure that it does not differ from the score calculated by Elasticsearch. The results show that the RHWMD is not able to re-rank the results such that they improve the final Mean Average Precision.

There are multiple reasons why the RHWMD is not able to improve the BM25 ranking: First there is a fundamental problem regarding the approach if word identity is the most important indicator for similarity. Consider the following sample query from the evaluation data and the corresponding relevant article:

> *Topic 161:* Finde Berichte, die Ernährungsprobleme von Sprue- bzw. Zöliakie-Erkrankten diskutieren.

> *SDA.950912.0104* Rom, 12. Sept. (sda/apa) Seminaristen, die unter Alkoholismus oder an Zöliakie (einer Mehl-Allergie) leiden und damit nicht die Eucharistie in der vorgegebenen Form mit Brot und Wein feiern könnten, dürfen nicht zur Priesterweihe zugelassen werden. (...)

Algorithm	μ_{AP}
rhwmd.big	20.49
rhwmd.min	20.49
rhwmd.max	35.05
rhwmd.small	35.08
rhwmd.sum	38.30
bm25 (Elasticsearch)	38.40
bm25 (Ungol)	**38.65**

Table 4.8: Mean Average Precision for the different scoring algorithms when re-ranking 250 documents pre-selected by Elasticsearch's BM25 implementation.

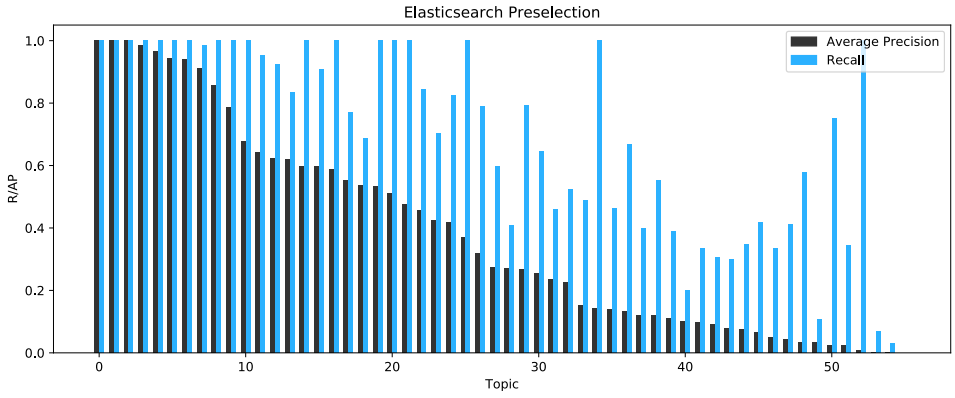


Figure 4.10: This plot displays the Recall and Average Precision per topic. Only some of the topics are promising candidates for re-ranking: Those with high or maximum Average Precision can hardly be improved. The topics with low or zero Recall do not offer any or many true positives to be ranked better.

token	nn	sim	tf(token)	idf(token)	idf(nn)	weight
comparing *Topic 161* to *FR940501-000387* - score: **1.510110**						
zöliakie	erkrankungen	0.738	0.100	11.900	5.614	0.187
erkrankten	erkrankten	1.000	0.100	6.363	6.363	0.135
diskutieren	diskutieren	1.000	0.100	4.760	4.760	0.101
finde	habe	0.758	0.100	5.330	1.469	0.086
ernährungs	gesundheit	0.754	0.100	5.999	4.569	0.096
bzw	und	0.770	0.100	5.027	0.095	0.082
berichte	berichtete	0.809	0.100	4.304	3.187	0.074
probleme	probleme	1.000	0.100	3.020	3.020	0.064
von	von	1.000	0.100	0.221	0.221	0.005
die	die	1.000	0.100	0.050	0.050	0.001

Table 4.9: This table shows the detailed score calculation for the highest ranked false-positive RHWMD search result for *Topic 161*. For brevity only the Query-to-Document calculation is shown. The corresponding candidate document has 326 unique tokens. Thus the probability of finding an exact match or a nearest neighbour with a great similarity value is high.

This topic only has one relevant document. As the evaluation reveals, this is ranked as the top result by BM25 but re-ranked by RHWMD such that some other results are ranked higher and the μ_{AP} drops to 33%. Table 4.9 displays the details for the incorrectly top-ranked RHWMD choice. This document covers the problems concerning the sarcophagus built to reduce the impact of the Tschernobyl nuclear accident and the ramifications for humans and wildlife. The highest contribution to the score of a single value stems from the mapping of *zöliakie* to *erkrankungen*. This is perfectly reasonable and actually an indicator that the algorithm works as intended. It impairs the evaluation however as the formerly well matched BM25 results are diluted by these more *fuzzy* results.

Doing a pre-selection with BM25 leads to a candidate set which heavily relies on word identity. If the candidates which would be ranked highly with RHWMD are eliminated a priori than the approach is impaired by the pre-selection in such a way that the re-ranking by RHWMD is only able to approach the BM25 baseline. To explore this reasoning further Figure 4.11 shows the difference between the Mean Average Precision of the BM25 and RHWMD ranking. There all negative bars show where the RHWMD ranked worse than BM25 and hence are candidates for a further analysis to improve the RHWMD performance on this data set.

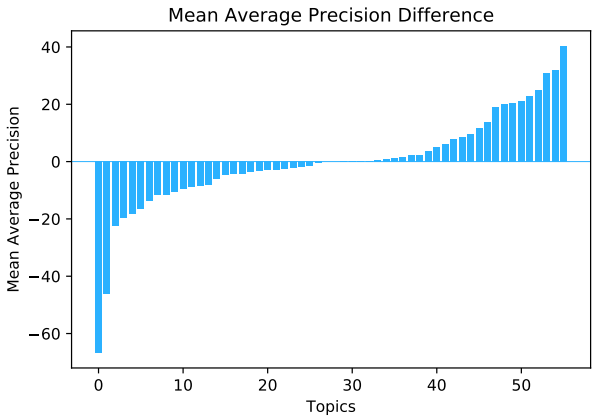


Figure 4.11: This plot shows the difference between the μ_{AP} of BM25 and RHWMD. The total difference evens out as they both achieve the same evaluation score. If the difference is negative, than BM25 ranked better and for positive bars RHWMD ranked better. This shows that the same evaluation performance is achieved although the algorithms largely score differently.

The findings reveal that the two algorithms rank the documents quite differently while exhibiting the same performance on the data set. If the two approaches would have ranked the same documents to the same places than this would have been an indicator that the similarity measure and fuzzy search for neighbours is not contributing to the score and the IDF dominates the search results.

Based on this observation, an example is selected which highlights the strength of RHWMD in Table 4.10 in contrast to BM25. The task in *Topic 161* is to find articles discussing a specific soccer game between England and Ireland. It was stopped throughout and the articles to be returned must explain why that happened. Here, the articles ranked high with BM25 also discuss the political relations of England and Ireland regarding the Irish Republican Army and the still ongoing national conflict at this time. This is reasonable as *Irland*, *England* and *Ausschreitungen* are often recurring terms in these documents.

token	nn	sim	tf(token)	idf(token)	idf(nn)	weight
comparing *Topic 161* to *FR940501-000387* - score: **1.510110**						
irisches	irland	0.742	0.008	9.226	5.042	0.022
dublin	dublin	1.000	0.024	6.225	6.225	0.020
freundschaftsspiel	fußball	0.738	0.008	7.884	3.715	0.019
ausschreitungen	ausschreitungen	1.000	0.008	5.782	5.782	0.019
krawallen	ausschreitungen	0.781	0.008	7.295	5.782	0.018
row	finde	0.688	0.008	8.211	5.330	0.018
randalierer	ausschreitungen	0.777	0.008	7.247	5.782	0.018
rowdies	ausschreitungen	0.660	0.008	8.450	5.782	0.018
landfriedensbruchs	ausschreitungen	0.699	0.008	7.757	5.782	0.018
birmingham	england	0.738	0.008	6.924	4.701	0.017

Table 4.10: The ten highest contributors to the score from the Document-to-Query perspective for *Topic 161*. The strengths of the RHWMD procedure are visible here: many words such as *rowdies*, *randalierer* or *krawallen* contribute highly to the score as they are all related to the query term *ausschreitungen*.

Chapter 5

Discussion

This work introduced the Relaxed Hamming Word Movers Distance (RHWMD) as an approach reduce the computational costs of the Word Movers Distance (WMD). This cost reduction is mainly owed to the transformation of real-valued word embeddings to binary hash codes using a neural auto-encoder model. It is shown that the neural model produces hash codes which preserve the spatial information of the original embedding space well. The RHWMD proof-of-concept implementation produces competitive results on a German information retrieval task. All in all it is demonstrated that this approach is worth exploring further as its' full potential is not yet uncovered: The introduction of free parameters to tune the scoring function, domain specific embedding vectors and a fitting approach for candidate pre-selection are the most promising directions for further research. Also only one Information Retrieval (IR) data set is used for evaluation. It might be interesting how well the RHWMD works on other IR data sets, for document clustering or classification using k-nearest neighbours (K-NN). However there is room for improvement and some possible extensions to the method are discussed in this chapter. Additionally, some of the experimental results are reviewed in more detail here to find explanations for the observed behaviour.

5.1 Online Code Generation

The Ungol Index can be updated by adding new documents at every time[1]. A challenge arises for words for which no hash code exists as outlined in Section 4.2.4. Not implemented in this work but theoretically possible is the ad-hoc creation of new hash codes for unknown words and a full replacement of the Out of Vocabulary (OOV) -Strategy currently applied. One requirement must be fulfilled:

[1] Multiprocessing concerns aside.

An embedding model must be used which can produce new codes (such as fastText as introduced in Section 2.3.3 on page 19). Then the procedure regarding OOV words looks as follows (prior to adding the document to the Ungol Index mapping):

1. Create an embedding $\mathbf{e}_t$ for the unknown word t using an embedding model such as fastText.
2. Transform the embedding to a hash code by using $\text{compress}_\Theta(\mathbf{t}_w)$ as described in Section 3.1 on page 31.
3. Update the Ungol Index by adding the new code to the *codemap* and update the Inverse Document Frequency (IDF) and *vocabulary* mapping.

5.2 Further Speed Optimisation

The current implementation of the Ungol Index is a proof-of-concept to show that the system works in principle. The achieved speed optimisations are hard to optimise further in plain Python. Also, the whole implementation currently works only with document pairs. This increases the total amount of redundant operations if token pairs appear in multiple query-document combinations for a single query. The following approach is suggested as the next step for speed optimisation:

First the critical parts of computation are transferred to cython [76]. The memory view offered by numpy is directly addressed to access the *codemap* of the Ungol Index. Now a whole batch of candidate documents can be processed at once: this works by selecting the union of all tokens of the documents and saving them to a vector. This vector is then used in combination with the query documents' vector to construct the distance matrix. After selecting the minima along each axis, for each document the relevant distances are selected.

The following example illustrates the algorithm: Consider a query document Q with a mapping $\mathbf{q} = (q_1, q_2, q_3)$ and two candidate documents D, D' who have some tokens in common: $\mathbf{d} = (t_3, t_1, t_2)$ and $\mathbf{d}' = (t_4, t_2, t_1)$. First a vector $\mathbf{u} := (d_i \mid D \cup D') = (t_1, t_2, t_3, t_4)$ and two mappings $\pi_\mathbf{d} = (2, 0, 1)$ and $\pi_{\mathbf{d}'} = (3, 1, 0)$ are constructed to associate the values of the distance matrix with the respective document. Given the function η computes the Hamming distance between two hash codes and the *codemap* offers the respective hash code h_{t_i} for token t_i, a matrix $H \in \mathbb{R}^{|\mathbf{q}| \times |\mathbf{u}|}$ is constructed where $H_{ij} = \eta(h_{q_i}, h_{u_j})$. Now, for every column the minimum distance is selected which defines the vector $\widetilde{\mathbf{u}} \in \mathbb{N}^{|\mathbf{u}|}$ with the nearest neighbours of all candidate document tokens to all query document tokens:

$$\widetilde{\mathbf{u}} := (u_1, \ldots, u_{|\mathbf{u}|}) \quad \text{with} \quad u_i := \min_i H_{ij} \tag{5.1}$$

Now for each candidate document a vector $\widetilde{\mathbf{q}}_d \in \mathbb{N}^{|\mathbf{q}|}$ is constructed with the nearest neighbours of the query token to the respective documents tokens. This is done by selecting from the matrix H those columns that are relevant for the document using the mapping π:

$$\widetilde{\mathbf{q}}_d = (q_1, \ldots, q_{|\mathbf{q}|}) \quad \text{with} \quad q_j := \min_j H[\pi_d]_{ij} \tag{5.2}$$

Everything is laid out now to obtain the document specific nearest neighbour distances to calculate the score. The following equation defines the lookup procedure for the distance of the relevant tokens to their respective neighbours for a document d. The distance of each query token to the respective nearest neighbour is found in the respective vector $\text{dist}(q_i) = \widetilde{\mathbf{q}}_{di}$. The distance of each document token to the nearest neighbour of the query document is selected from the vector $\widetilde{\mathbf{u}}$:

$$\text{dist}(t_i) := \widetilde{\mathbf{u}}_j \quad \text{with} \quad j = \pi_{\mathbf{d}}(t_i) \tag{5.3}$$

Regarding the implementation of η itself it has to be evaluated whether a lookup-table based approach is still faster than using bit-operations. To further speed up recurring computations, a cache that maps common token pairs to their distances can be used to avoid calculating the same Hamming distances redundantly.

5.3 Nearest Neighbour Distances

In Chapter 3, Figure 3.7 it is shown that, for at least the most frequent words, the Euclidean and the Hamming embeddings share many nearest neighbours. This is an indicator that the spatial properties are transferred well from one space to the other. However, no analysis is conducted regarding the distance of the k-th neighbour per word in comparison to the k-th neighbour of all other words. Consider Table 2.1 given in Chapter 2 on page 17 again. It can be observed that the 20th nearest neighbour of *Gott* is as far away as the 5th nearest neighbour for *Satan*. In combination with the observation made in Figures 3.5 and 3.6 on page 45 that the range of discriminatory distance is very small, the question arises how this affects the score calculation.

5.3.1 Rank Based Similarity Measure

The absolute difference of nearest neighbour distances affects the importance of different words regarding their relative contribution of the final score. Consider two documents $D = \{a_1, a_2, \dots, b_1, b_2, \dots\}$ and $D' = \{a'_1, a'_2, \dots, b'_1, b'_2, \dots\}$. The tokens a_i, a'_i are concerned with one topic, b_i, b'_i with another. a'_i are close nearest neighbours of a_i and b'_i are the first nearest neighbours of b_i. Now assume that the function *sim* calculates their similarity; $sim(a_i, a'_i) \mapsto [0.6, 0.9]$ and $sim(b_i, b'_i) \mapsto [0.4, 0.7]$. Hence the topic covering b is deemed less important than topic a for the score.

The WMD and consequently RHWMD fundamentally rely on the assumption that the embeddings express the semantic relationship through their absolute spatial distance. The experiments conducted by the authors of the WMD and the experiments in this work showed that this is working competitively in principle. However, it might be possible to construct a more reliable scoring mechanism by aligning these distances. Table 5.1 and Figure 5.1 motivate this further. It can be seen that there is no qualitative difference between the two neighbour groups but distance-wise they are very far apart. The distances of the first nearest neighbour spread over a large range. Figure 5.2 is provided to determine whether the nearest neighbour distances relate to on the term frequency encountered while training the embeddings. However, the plot shows that this distribution does not correlate with the word frequency. A different distance distribution for nearest neighbours would be desirable: The probability of the first ten nearest neighbours to be relevant is much higher than for the 100th nearest neighbour. The lower the rank of the neighbour, the higher should the influence of the effective distance be.

5.3.2 Similarity Normalisation

Another aspect of absolute similarity values is their distance to a perfect match. As shown in Figure 5.1 most of the first nearest neighbours (which are very good matches) have a similarity of around 0.75 to 0.87. This sets good matches far apart from words shared by both documents for whom a similarity of 1.0 is assumed. This is especially significant given that very bad matches (1000-NN+) exhibit a similarity of around 0.65 (see Figure 3.6 on page 45). So a normalisation for the similarity values might improve the scoring. Using Equation (4.3) for calculating the score given on page 49, a function $f : [0, 1] \mapsto [0, 1]$ needs to be found and plugged into the equation.

	High Absolute Similarity			Low Absolute Similarity		
Position	Word	1-NN	Distance	Word	1-NN	Distance
1	größeren	kleineren	0.0976562	steht	stehen	0.203125
2	französisch	spanisch	0.0976562	reihe	vielzahl	0.207031
3	metal	metalcore	0.0976562	liebe	mutterliebe	0.203125
4	heimatstadt	geburtsstadt	0.0976562	halten	hält	0.203125
5	schlechter	schlechterer	0.0976562	dienst	diensten	0.207031
6	güterverkehr	personenverkehr	0.0976562	wenigen	einigen	0.207031
7	eurovision	contest	0.0976562	kiel	kieler	0.203125
8	rowohlt	reinbek	0.0976562	lauf	läufe	0.207031
9	övp	spö	0.0976562	werner	heinz	0.203125
10	völlig	vollkommen	0.0976562	dieses	weiteres	0.207031

Table 5.1: Randomly sampled examples from two different distance ranges. The accompanying Figure 5.1 displays the distribution of these 1-NN among others for the most frequent words in Hamming space.

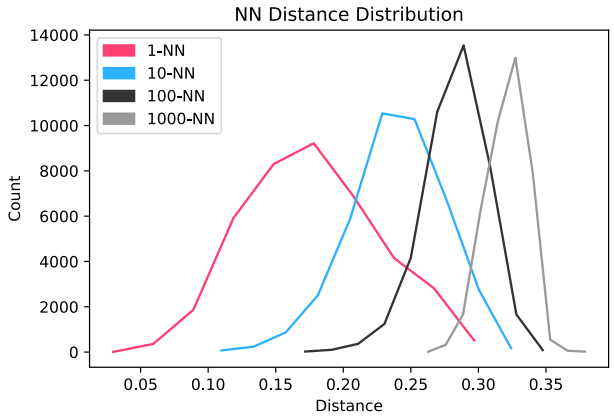


Figure 5.1: The distance distribution of some selected k-NN for the 40k words most frequently encountered while training the embeddings. The distance values are calculated in Hamming space based on the binary hash representations.

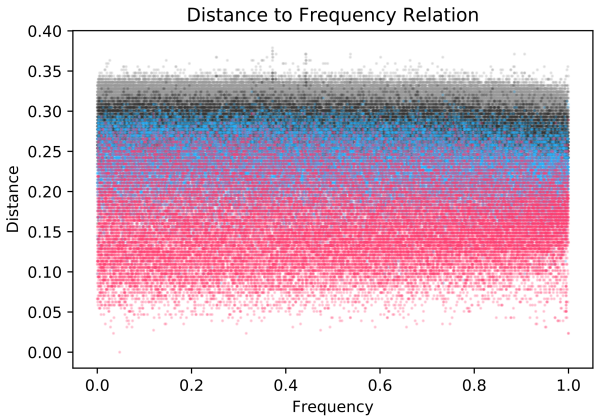


Figure 5.2: No obvious correlation recognisable: Frequencies of the 40k most frequent k-NN in Hamming space (see Figure 5.1 for colour reference) plotted with their respective distance to the output word. There are four data points per frequency value and the point at 1.0 is the most frequent word of the vocabulary.

$$
\begin{aligned}
s &: D \times D \mapsto [0,1] \\
s(d, d') &:= \sum_{t \in d} \text{n-idf}(t) \cdot f\big(1 - \eta(h_t, h_{t'})\big)
\end{aligned}
\tag{5.4}
$$

This similarity transformation does not need to necessarily move the high similarities further towards 1 - it has to be examined how much more important word identity is in regards to scoring. This might as well be a property specific to the respective purpose or domain the RHWMD is applied to. All the more, introducing such a function f as a free parameter might allow for increasing the performance specific to the respective needs.

5.3.3 Nearest Neighbour IDF

An observation was made for documents of unequal size: Calculating the score from the direction of the larger document requires finding a nearest neighbour in the smaller document for every word. This results — especially for documents which do not have much in common — to a situation where many words are mapped to the most generic words of the smaller documents. The training objective of the embedding trainers presented in Section 2.3 force very common words, such as pronouns and prepositions which occur in nearly every context, to be as close to every other word as possible. For unrelated documents, rare words contribute very highly to the score because the similarity between the *bad*

mapping is boosted greatly by the high IDF value. Combining both IDF values might be a promising approach. Another method is proposed by Huang et al. [25]. They extend the WMD with a supervised metric learning step and call their approach Supervised Word Movers Distance (S-WMD). The basic idea is to re-weight the words such that they influence the distance the WMD computes.

5.4 Pre-Selection using an Inverted Index

One of the main reasons Lucene and consequently Elasticsearch are so fast is the inverted index. As discussed briefly in Section 2.1.1 on page 12 this index uses token identities to assemble the set of candidate documents. The successive scoring mechanisms such as Best-Match 25 (BM25) or Vector Based Scoring (VBS) can work reliably on the candidate set because they too fundamentally work with token identity. To increase the set of candidate documents, synonym lists or paraphrase databases can be consulted. As the re-ranking experiment in Section 4.4.2 on page 66 et seqq. revealed this is not readily possible for the RHWMD. It is not even desirable as the strength of the RHWMD in contrast to methods such as BM25 should be the ability to score semantically close documents regardless of word identity.

A different approach may be promising: As discussed in Section 2.4.3 on page 26, the approach of Kusner et al. [35] uses the Word Centroid Distance (WCD) to effectively pre-select candidates because it forms a lower bound to the Earth Movers Distance (EMD) (Rubner et al. [56]). It may be possible to construct a set of clusters using the WCD and the inverted index returns documents assigned to these clusters. However, the authors of the WMD hint that the WCD is not a very tight bound and as such too many documents may fall into one cluster. Also, it is not explored how well the WCD is actually apt for clustering.

Instead of consulting the real-valued embeddings, some clustering directly in Hamming space may also be suitable for building the inverted index. The fundamental idea is to work out efficiently which code of the to be indexed document belongs to which cluster. Norouzi et al. [46] implement a fast nearest neighbour search algorithm relying on k-means clustering. A different approach is to implement a k-medoid clustering [31] using the Hamming distance as the similarity measure. The greatest challenge for this approach are the incremental updates to the index when new documents are inserted. This includes adding new clusters if necessary and the recalculation of the cluster centres.

Finally, instead of clustering, a fast K-NN search might better include promising candidates for re-ranking. The results of Section 4.4.2 showed that re-ranking a pre-selection using an inverted index and BM25 did not work well. To include

promising documents that are discarded because their word overlap is too small, a fast K-NN search for related words may augment the candidate set such that relevant documents are ranked high by the RHWMD. An implementation for such a fast exact K-NN search is proposed by Norouzi et al. [47].

5.5 Quality of Word Embeddings

All retrieval evaluation took place using German fastText embeddings. From exemplary evaluation there is noise. Consider for example Table 2.1 on page 17. There *„gott* is one of the first nearest neighbours of *gott*. It is completely unnecessary to have such a word in the vocabulary; especially given that some preprocessing takes place before documents are added to the Ungol Index (Section 4.2.3 on page 58). Reducing the vocabulary while training the embedding model may lead to better embeddings as there are automatically more distinct samples. This leads to the second point: Given that embeddings are also the product of a training procedure one of the greater strength of this whole approach is the possibility to train domain specific embeddings. It is both possible to fine-tune a pre-trained embedding model or train a model from scratch. It is not explored in this work whether this improves the retrieval performance but it is a promising direction to explore.

Chapter 6

Appendix

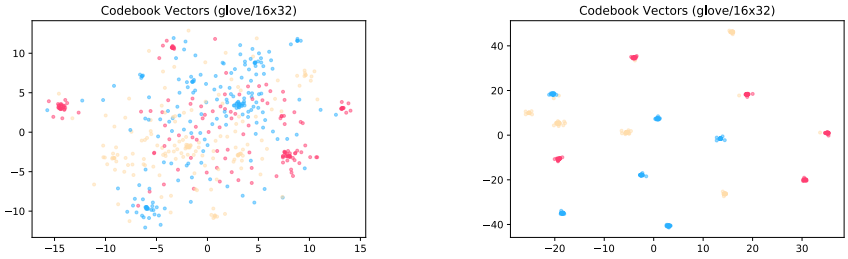
6.1 Observations Regarding Codebooks

In this section the basis vectors saved in the codebooks are analysed to obtain a deeper understanding of the model. The main points of interest focus on the basis vectors norms and distances to each other. Figure 6.1 shows a t-SNE [39] visualisation of the codebook vectors reduced to two dimensions. Here — without any deeper exploration — distinct clusters emerge which motivates an analysis of the vector's norm distributions. Figure 6.2 shows how the vector norms are distributed. The different bars are calculated as follows:

1. The **embedding vector norm** describes the mean p2-norm of each embedding vector. Given a vocabulary $E = (\mathbf{e}_1, \ldots, \mathbf{e}_N)$ of size N is simply obtained by calculating:
$$\frac{1}{N} \sum_{i=1}^{N} || \, \mathbf{e}_w \, ||_2 \tag{6.1}$$

2. The **codebook vector norm**, uses the same calculation rule as the embedding vector norm. The vocabulary consists of the $M \cdot K$ basis vectors.

3. The **cluster distance norm** describes the mean distance of each cluster to all other cluster. The cluster centre α^c is calculated given each cluster $c \in \{1, \ldots, M\}$ containing K basis vectors $\{\mathbf{v}_1^c, \ldots \mathbf{v}_K^c\}$. For obtaining the mean distance of all clusters to each other, the distance of all possible cluster combinations π_2^M is calculated:
$$\frac{1}{M!} \sum_{i,j \in \pi_2^M} || \, \alpha^i - \alpha^j \, ||_2 \quad \text{with} \quad \alpha^c := \frac{1}{K} \sum_{i=1}^{K} \mathbf{v}_i^c \tag{6.2}$$



(a) Distribution after 200 epochs.

(b) Distribution after 800 epochs.

Figure 6.1: Two t-SNE plots of two different training states of a $M = 16, K = 32$ model. Each data point represents a single basis vector selected from the codebooks. It can be observed how 16 distinct clusters are formed while training. The colours are assigned codebook-wise, indicating that vectors belonging to the same codebook also exhibit some relation in the latent vector space.

4. The **cluster radius** is simply the average distance of the vectors belonging to a cluster to it's cluster centre. As all cluster vectors are equidistant to the cluster centre simply selecting any of them and computing the distance suffices:

$$\frac{1}{M}\sum_{c=1}^{M} || \alpha^c - \mathbf{v}_i^c ||_2 \quad \text{for any} \quad i \in \{1, \dots K\} \tag{6.3}$$

The findings were found to be consistent over all different model configurations.

6.2 Implementation Lookup

To quickly find the implementation in the provided code base, a mapping of equation to the respective position is given in the following list:

Chapter 2: Essentials

- **Okapi BM25**: Equation (2.3) page 15
  - **ungol-wmd** / ungol.wmd.sim.bm25
- **TF-IDF**: Equation (2.2) page 14
  - **ungol-wmd** / ungol.wmd.sim.tfidf

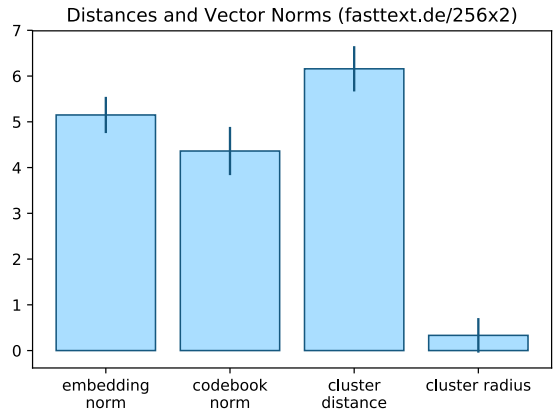


Figure 6.2: This plot displays the size of the different norms. It can be observed that the average norm of codebook vectors corresponds approximately to the average embedding vectors' norm. The cluster radius shows that the vectors forming a cluster are very dense while the cluster centres are very far apart.

Chapter 3: Compressor

- **Compressor**: Equation (3.2) page 32
  - **ungol-models** / ungol.models.embcompr.Compressor
- **Encoder**: Equation (3.3) page 34
  - **ungol-models** / ungol.models.models.Encoder
- **Gumbel Softmax**: Equation (3.6) page 35
  - **ungol-models** / ungol.models.models.Gumbel
- **Decoder**: Equation (3.8) page 36
  - **ungol-models** / ungol.models.models.Decoder

Chapter 4: chapter

- **RH-WMD**: Section 4.1.1 Equation (4.5)
  - **ungol-wmd** / ungol.wmd.sim.rhwmd
- **Hamming Distance**: Section 4.1.1 Equation (4.5)
  - **ungol-wmd** / ungol.wmd.rhwmd.hamming_bincount

  - **ungol-wmd** / ungol.wmd.rhwmd.hamming_bitmask
  - **ungol-wmd** / ungol.wmd.rhwmd.hamming_lookup
- **Distance Matrix Computation**: Section 4.1.1 Equation (4.5)
  - **ungol-wmd** / ungol.wmd.rhwmd.distance_matrix_loop
  - **ungol-wmd** / ungol.wmd.rhwmd.distance_matrix_vectorized
  - **ungol-wmd** / ungol.wmd.rhwmd.distance_matrix_lookup
- **Database Population**: Section 4.2.3
  - **ungol-es** / ungol.experiments.setup.do_ungol_setup_articles
  - **ungol-es** / ungol.experiments.setup.do_ungol_setup_topics
- **Mean Average Precision**: Section 4.3.3 Equation (4.13)
  - **ungol-es** / ungol.experiments.stats.TopicStats

Glossary

$\mu_{\textbf{AP}}$ Mean Average Precision. 66–71

AOT Ahead-of-Time (Compilation). 55

AP Average Precision. 69

B Number of bytes per code; $\frac{1}{8}M$. 53, 54

b One of two BM25 hyper-parameters. 14, 65

BM25 Best-Match 25. 8, 11, 14, 23, 47, 50, 65–71, 79, 85, 86

BOW Bag of Words. 11–13, 23

CBOW Continuous Bag of Words. 18, 19, 23, 29

CLEF Conference and Labs of the Evaluation Forum. 61, 64

D A corpus; collection of text documents. 13–15, 26, 27

DF Document Frequency. 14, 58, 62

E Word embedding dimensionality. 19, 26, 31, 32, 36

EMD Earth Movers Distance. 23, 24, 79

ES Elasticsearch. 12

fastText Word Embedding Model using Subword Information. 11, 16, 17, 20, 31, 38, 41, 74, 80

GloVe Global Vectors for Word Representation. 11, 18, 22, 23, 29–31

IDF Inverse Document Frequency. 14, 49, 50, 52, 56, 58–60, 71, 74, 79

IR Information Retrieval. 6–8, 12, 28, 47, 73

JIT Just-in-Time (Compilation). 55

K Code component domain. 32, 34–37, 39, 41, 42, 44

k_1 One of two BM25 hyper-parameters. 14, 65, 67

K-NN k-nearest neighbours. 27, 73, 79, 80

LDA Latent Dirichlet Allocation. 17

LDI Latent Dirichlet Indexing. 17

LSA Latent Semantic Analysis. 16

LSH Locality Sensitive Hashing. 27, 28

LSI Latent Semantic Indexing. 16

M Number of codebooks; the codes' length. 32, 36, 37, 41, 43, 44, 48, 53, 62, 85

N Vocabulary size; i.e. number of unique words. 13, 18–22, 24–27, 31, 55

NBOW Normalised Bag of Words. 13, 24, 26, 28

NLP Natural Language Processing. 6, 11, 18

OOV Out of Vocabulary. 60, 73, 74

PCA Principle Component Analysis. 28

pLSA Probabilistic Latent Semantic Analysis. 17

pLSI Probabilistic Latent Semantic Indexing. 17

PPMI Positive Pointwise Mutual Information. 18, 20–22

R Recall. 69

RBM Restricted Boltzman Machine. 28

RHWMD Relaxed Hamming Word Movers Distance. 1, 8, 9, 11, 27, 47, 48, 50, 54, 56, 60, 63–68, 70–73, 76, 78–80, 87

RWMD Relaxed Word Moving Distance. 1, 25–27, 30, 47, 48, 66, 67

Skip-Gram Part of Word2Vec. 18, 19, 30

SVD Singular Value Decomposition. 16, 17

S-WMD Supervised Word Movers Distance. 79

TC Term Count. 14, 51, 57, 58

TF Term Frequency. 14, 16, 26, 58, 62

TF-IDF Term Frequency-Inverse Document Frequency. 1, 8, 11, 14, 16, 47, 50, 65, 66

Ungol Project name for this works' implementation. Sindarin for "Spider".. 68, 69

Ungol Index Data structure holding all necessary data to compute the RHWMD. 8–10, 56, 57, 60, 62, 73, 74, 80

V Collection of all tokens of the vocabulary. 18, 49

VBS Vector Based Scoring. 79

WCD Word Centroid Distance. 26, 27, 66, 67, 79

WMD Word Movers Distance. 1, 8, 11, 23, 24, 26, 27, 47, 65–67, 73, 76, 79

Word2Vec Predictive Word Embedding Model Family. 18–20, 29

X Word embedding space. 19, 20, 22, 26, 27, 31

Literature

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[2] JD Bernal, David Chadwick, JE Holmstrom, and H Munro Fox. The royal society scientific information conference, 1948.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. corr abs/1607.04606. *URL http://arxiv. org/abs/1607.04606*, 2016.

[5] Vannevar Bush et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945.

[6] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[7] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[8] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[9] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[10] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[11] Dominik Maria Endres and Johannes E Schindelin. A new metric for probability distributions. *IEEE Transactions on Information theory*, 2003.

[12] Carla Parra Escartín. Chasing the perfect splitter: A comparison of different compound splitting tools. In *LREC*, pages 3340–3347, 2014.

[13] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[14] Bent Fuglede and Flemming Topsoe. Jensen-shannon divergence and hilbert space embedding. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 31. IEEE, 2004.

[15] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, 2013.

[16] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[17] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[18] Joshua Goodman. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pages 561–564. IEEE, 2001.

[19] Emil Julius Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series*, 33, 1954.

[20] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[21] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[22] Ernst Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271, 1909.

[23] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[24] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[25] Gao Huang, Chuan Guo, Matt J Kusner, Yu Sun, Fei Sha, and Kilian Q Weinberger. Supervised word mover's distance. In *Advances in Neural Information Processing Systems*, pages 4862–4870, 2016.

[26] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[27] Aminul Islam and Diana Inkpen. Second order co-occurrence pmi for determining the semantic similarity of words. In *Proceedings of the International Conference on Language Resources and Evaluation, Genoa, Italy*, pages 1033–1038. Citeseer, 2006.

[28] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[29] Karen Sparck Jones. Index term weighting. *Information storage and retrieval*, 9(11):619–633, 1973.

[30] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[31] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2130–2137. IEEE, 2009.

[34] Brian Kulis, Prateek Jain, and Kristen Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, 2009.

[35] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *ICML*, 2015.

[36] Maximilian Lam. Word2bits-quantized word vectors. *arXiv preprint arXiv:1803.05651*, 2018.

[37] Aldo Lipani, Mihai Lupu, Allan Hanbury, and Akiko Aizawa. Verboseness fission for bm25 document length normalization. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 385–388. ACM, 2015.

[38] Yuanhua Lv and ChengXiang Zhai. Adaptive term frequency normalization for bm25. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1985–1988. ACM, 2011.

[39] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[40] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

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

[44] Olivier Morere, Jie Lin, Antoine Veillard, Ling-Yu Duan, Vijay Chandrasekhar, and Tomaso Poggio. Nested invariance pooling and rbm hashing for image instance retrieval. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 260–268. ACM, 2017.

[45] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[46] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3108–3115. IEEE, 2012.

[47] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1107–1119, 2014.

[48] Ferdinand Österreicher and Igor Vajda. A new class of metric divergences on probability spaces and its applicability in statistics. *Annals of the Institute of Statistical Mathematics*, 55(3):639–653, 2003.

[49] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[50] Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *ICCV*, volume 9, pages 460–467, 2009.

[51] Shmuel Peleg, Michael Werman, and Hillel Rom. A unified approach to the change of resolution: Space and gray-level. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):739–742, 1989.

[52] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[53] Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2): 945–959, 2000.

[54] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241. Springer-Verlag New York, Inc., 1994.

[55] Douglas LT Rohde, Laura M Gonnerman, and David C Plaut. An improved model of semantic similarity based on lexical co-occurrence. *Communications of the ACM*, 8(627-633):116, 2006.

[56] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Computer Vision, 1998. Sixth International Conference on*, pages 59–66. IEEE, 1998.

[57] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[58] Ruslan Salakhutdinov and Geoff Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419, 2007.

[59] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[60] Mark Sanderson and W Bruce Croft. The history of information retrieval research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451, 2012.

[61] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*, 2017.

[62] Noah A Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 354–362. Association for Computational Linguistics, 2005.

[63] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[64] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593. ACM, 2006.

[65] Antonio Torralba, Rob Fergus, and Yair Weiss. Small codes and large image databases for recognition. 2008.

[66] Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to bm25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium*, page 58. ACM, 2014.

[67] Don Tuggener. *Incremental coreference resolution for German*. PhD thesis, Universität Zürich, 2016.

[68] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

Online Sources

[69] Ryan Adams. The Gumbel-Max Trick for Discrete Distributions.
https://hips.seas.harvard.edu/blog/2013/04/06/the-gumbel-max-trick-for-discrete-distributions.
Last access: 2018/08/24.

[70] Elasticsearch BV. elastic.
https://www.elastic.co.
Last access: 2018/10/01.

[71] dtuggener. CharSplit - An ngram-based compound splitter for German.
https://github.com/dtuggener/CharSplit.
Last access: 2018/10/15.

[72] The Apache Software Foundation. The Apache Lucene Project.
http://lucene.apache.org, .
Last access: 2018/10/01.

[73] The Apache Software Foundation. Apache Solr.
http://lucene.apache.org/solr, .
Last access: 2018/10/18.

[74] Google Inc. Google Zeitgeist 2012.
https://archive.google.com/zeitgeist/2012.
Last access: 2018/10/14.

[75] NLTK Project. Natural Language Toolkit.
http://nltk.org.
Last access: 2018/10/04.

[76] et al. Robert Bradshaw, Stefan Behnel. C-Extensions for Python.
http://cython.org.
Last access: 2018/10/13.

[77] zomux. Neural Compressor Reference Implementation.
https://github.com/zomux/neuralcompressor/blob/master/nncompress/embed_compress.py.
Last access: 2018/08/09.