Karlsruhe Institute of Technology

Keyword Based Document Retrieval via Document Embeddings

Bachelor's Thesis
of

Julian Brendl

KIT Department of Informatics
Institute for Anthropomatics and Robotics

Referees: Prof. Dr. Alexander Waibel
Prof. Dr. Tamim Asfour

Advisor: Dr. Sebastian Stüker

Duration: Febuary 16th 2018 – June 15th 2018

Erklärung:

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 15. Juni 2018

Julian Brendl

Abstract:

Many different kinds of document retrieval systems exist today using various approaches. Many rely on simple frequency statistics while others utilize neural networks to retrieve documents. In this thesis I present a document retrieval system called word2doc. Its purpose is to gain a semantic understanding of the query and indexed documents in order to improve retrieval results. In contrast to most existing retrievers, word2doc learns semantic meaning by combining several existing approaches to train its own document embeddings. The few retrievers that make use of document embeddings do not learn the embeddings themselves. I hypothesize that by training document embeddings myself, embeddings can be tuned to the query and to other contextually similar documents such that they aid in document retrieval. Furthermore, if document embeddings are trained in a similar fashion as word embeddings are, perhaps the success of word embeddings can be transferred to the document retrieval task. Having put forth this hypothesis, I tested it by implementing my ideas in word2doc and comparing results with a frequency-based document retriever developed by Facebook (Chen et al., 2017). In this thesis, both systems operate on the Wikipedia corpus. However word2doc was only trained on 1% of the full corpus. Yet results are promising, showing that word2doc outperforms Facebook's system when it comes to pure retrieval accuracies. However, word2doc is struggling with the retrieval of ranked document sets. While word2doc is adequate at identifying the target document, it is unable to retrieve document sets that are relevant to the target document. My two main explanations for this are as follows: Word2doc was only trained on 1% of the full Wikipedia corpus, whereas Facebook's system had the entire corpus at its disposal. Thus, word2doc had fewer relevant documents that could be retrieved. Furthermore, word2doc optimizes on accuracies and not on the ranking quality of a retrieved document set. Hence, it is possible that logits in the final softmax layer do not reflect a ranking and only reflect the single best document.

In conclusion, word2doc shows that document retrieval via document embeddings has potential. In order to fully test the performance, however, word2doc has to be trained on the entire Wikipedia corpus and not just on 1%.

Kurzzusammenfassung:

Es gibt heute viele verschiedene Arten von Dokumentenabrufsystemen mit unterschiedlichen Ansätzen. Viele verlassen sich auf einfache Frequenzstatistiken, andere nutzen neuronale Netze, um Dokumente abzurufen. In dieser Arbeit stelle ich ein Dokumentenabrufsystem namens word2doc vor. Ziel ist es, ein semantisches Verständnis der Abfrage und der indizierten Dokumente zu gewinnen, um die Suchergebnisse zu verbessern. Im Gegensatz zu den meisten existierenden Retrievern lernt word2doc die semantische Bedeutung, indem es mehrere existierende Ansätze kombiniert, um seine eigenen Dokumenteneinbettungen zu trainieren. Die wenigen Retriever, die Dokumenteneinbettungen verwenden, lernen die Einbettungen nicht selbst. Ich habe die Hypothese aufgestellt, dass durch das eigene Trainieren von Dokumenteneinbettungen die Einbettungen auf die Abfrage und auf andere kontextuell ähnliche Dokumente angepasst werden können, so dass sie bei der Dokumentensuche helfen. Wenn die Dokumenteneinbettungen in ähnlicher Weise trainiert werden wie die Wörtereinbettungen, kann der Erfolg der Wörtereinbettungen vielleicht auf die Aufgabe der Dokumentensuche übertragen werden. Ich testete meine Hypothese, indem ich meine Ideen in word2doc umsetzte und die Ergebnisse mit einem von Facebook entwickelten frequenzbasierten Dokument-Retriever verglich. In dieser Arbeit verwenden beide Systeme den Wikipedia-Korpus. Allerdings wurde word2doc nur auf 1% des gesamten Korpus trainiert. Doch die Ergebnisse sind vielversprechend und zeigen, dass word2doc das System von Facebook übertrifft, wenn es um reine Suchgenauigkeit geht. Word2doc kämpft jedoch mit dem Auffinden von geordneten Dokumentensätzen. Während word2doc für die Identifizierung des Zieldokuments adequat ist, ist es nicht in der Lage, andere für das Zieldokument relevante Dokumente abzurufen. Meine beiden Hauptgründe dafür sind folgende: Word2doc wurde nur auf 1% des gesamten Wikipedia-Corpus trainiert, während das System von Facebook über den gesamten Corpus verfügte. So hatte word2doc weniger relevante Dokumente, die abgerufen werden konnten. Darüber hinaus optimiert word2doc die Genauigkeit und nicht die Rankingqualität eines abgerufenen Dokumentsatzes. Daher ist es möglich, dass die Logits in der letzten Softmax-Ebene kein Ranking, sondern nur das beste Dokument wiedergeben.

Zusammenfassend zeigt word2doc, dass der Abruf von Dokumenten über Dokumenteneinbettungen Potenzial hat. Um die Leistung vollständig zu testen, muss word2doc jedoch auf dem gesamten Wikipedia-Corpus und nicht nur auf 1% trainiert werden.

Contents

1 Introduction

Free text documents, that is, documents comprised of unstructured text like it can be found in this thesis, make up a bulk of modern day information in the form of papers, articles and other documents. The utility of such documents is self-evident, and there are millions of them in existence. The question arises of how to quickly and accurately access these documents, and for this purpose document retrieval systems exist.

Document retrieval systems, or information retrieval (IR) systems, offer the ability to find free-text documents, given some user query. This query can be in the form of a keyword, or a phrase and usually describes what is being searched for, think of a Google search query. In fact Google's famous Page-Rank algorithm is a great example of such a document retrieval system for websites.

The aim of the thesis is to present word2doc, an ad-hoc document retrieval system to be used by the Lecture Translator (Müller et al., 2016) that is being developed by the Interactive Systems Lab at the KIT, along with a working implementation. This is motivated through the following use-case in the Lecture Translator: The overarching idea is to automatically highlight important keywords and phrases in the generated translations of the lecture translator, and to link these keywords and phrases to the document best describing the keyword or phrase. Thus, a need arises to provide relevant documents for highlighted keywords within the text. An implementation for word2doc can be found here: `https://github.com/jundl77/word2doc`.

Many different kinds of document retrieval systems exist already, with many different approaches. Many rely on simple frequency statistics while others use neural networks. However, word2doc gains a semantic understanding of the query and the document, and retrieves documents that contextually match the query. This too has been done to a certain degree, however most approaches only use neural nets to pre-process certain components, and retrieve results using these pre-calculated components. Word2doc intrinsically learns the semantic relations between documents, and it is, to my knowledge, the first system to do so.

Word2doc uses a word frequency pre-filtering followed by a document-embedding based neural network. Using a frequency based document retrieval system developed by Facebook (Chen et al., 2017) as a baseline, I show that this setup has potential. It outscores Facebook's system when it comes to measuring accuracies on the documents to retrieve. However, it falls short when it comes to measuring the quality of a retrieved ranked document set. There are two plausible explanations for this shortfall that I will discuss in chapter 6.

In the remaining chapter I cover related works and compare word2doc to what has previously been done. I explain what word2doc does differently, and where it improves upon other systems. In chapter 2 I discuss the theoretical foundations of word2doc, and explain word2doc's architecture in detail. Chapter 3 covers various datasets I used, including the training and evaluation datasets. Chapter 4 presents the evaluation techniques I use to measure word2doc's performance, as well as giving some implementation details and backing up several design decisions with empirical data. In chapter 5 I will present my final results, and in chapter 6 I will summarize and explore opportunities for future work.

1.1 Related Work

Word2doc falls into the category of ad-hoc information retrieval systems. Ad-hoc information retrieval systems are given a query and have to quickly and spontaneously choose one (or a few) documents out of a huge collection of documents. In word2doc's case this is the entire Wikipedia corpus. A common example of an ad-hoc information retrieval system would be any web search engine.

Many different approaches exist that perform information retrieval. Classic approaches use statistical methods to determine the best match, but given the recent success of word embeddings, there are more and more systems that turn to neural networks and embeddings to determine the result of a query. I will first present a few traditional approach to IR, and then delve into more relevant systems that use embeddings.

1.1.1 Statistical IR Systems

The most basic approach involves an inverted lookup index, like it is used in elastic search (Gormley and Tong, 2015). Elastic search is a common search engine used for example by Wikipedia or GitHub, and an inverted lookup index is a data structure that allows for very fast full-text searches. An inverted index consists of a list of all unique words that appear in any document, and for each word, a list of the documents in which it appears. It provides a fast way to find documents that contain words from the query.

Like elastic search, Facebook's document retriever from DrQA (Chen et al., 2017) also uses an inverted index. However, it improves on elastic search by additionally using ngrams and a hash function to more precisely index documents as well as the query. This is explained in detail in section 2.2.3. I use Facebook's document retriever as a baseline to measure wor2doc's performance.

Furthermore, Ponte & Croft propose QLM (Query Likelihood Model) (Ponte and Croft, 1998) in which a language model of each document is estimated. Each document is then ranked by the likelihood of the query matching it, according to the estimated language model. Robertson & Zaragoza present BM25 (Robertson and Zaragoza, 2009), a ranking function based on the probabilistic relevance model (Robertson and Sparck Jones, 1988). It uses TF-IDF to rank matching documents by their relevance for a query. Metzler & Croft propose a linear model (Metzler and Croft, 2005) that takes proximity between query terms into account through Markov random fields. Lavrenko in Lavrenko and Croft (2001) and Lavrenko (2004) proposes a pseudo relevance feedback (PRF) based method that performs well at the cost of executing two rounds of retrieval, instead of one. Terms from the ranked documents retrieved in round one are used to augment the query for round two. The ranked set from the second round is then presented to the user.

There are many more statistical approaches, notably Latent Semantic Indexing (Deerwester et al., 1990), Probabilistic Latent Semantic Indexing (Hofmann, 1999) and Latent Dirichlet Allocation (Blei et al., 2003) that utilize global statistical information of the corpus to calculate a ranking for a query. On the other hand, embedding techniques such as word2vec (Mikolov et al., 2013a) or doc2vec (Le and Mikolov, 2014) focus on local context information instead, which I talk about next.

1.1.2 IR Systems using Embeddings

There have been multiple works that use embeddings to improve the performance for IR systems in recent years, mainly due to the big success of word2vec (Mikolov et al., 2013a) in NLP. In general, these works

can be roughly divided into the following two categories.

Models in the first category reformulate the query using embeddings so that the query more accurately reflect a user's intent and thus yields better results. For example, Grbovic et al. (2015) map queries into an embeddings space and expand a given query with the K-nearest neighbors in the embedding space to augment the original query. Roy et al. (2016a) also use word embeddings and the K-nearest neighbor algorithm to obtain expansion words for the query. Kuzi et al. (2016) expand the query by adding terms from the corpus that are semantically connected to the query. They do this by computing cosine similarities between embeddings of terms occurring in the query and embeddings of terms that surround the original term in the corpus, similar to word2vec's Continuous Bag-of-Words (CBOW) model (Mikolov et al., 2013a). Zheng and Callan (2015) propose a framework to learn weights for individual terms occuring in the query. Weighting terms allows different words in the query to be differently emphasized. Zamani and Croft (2016a) create two query expansion methods to estimate query language models based on word embeddings and an embedding-based relevance model. The relevance model was developed by Lavrenko and Croft (2001) and it is a statistical model used to capture the contextual relevance of documents in the context of IR. Roy et al. (2016b) use kernel density estimation on the embedding space created by term embeddings to calculate a relevance model. Zamani and Croft (2016b) estimate embedding vectors for queries based on the individual embeddings of terms occurring in the query, something word2doc does as well. Diaz et al. (2016) explore the difference between global embeddings and corpus and query specific embeddings, applied to query expansion. Their results suggest that global embeddings the likes of GloVe vectors (Pennington et al., 2014) are underperformed by locally trained embeddings.

Models in the second category try to exploit semantic similarities between terms, queries and documents. Word2doc falls into this category. For example, Clinchant and Perronnin (2013) propose a document representation in which they non-linearly map word embeddings of words occurring in a document into a higher-dimensional space and then aggregate them into a document-level representation. Ganguly et al. (2015) construct a generalized language model, where the mutual independence between a pair of words no longer holds. Instead, they use word embeddings to derive the transformation probabilities between words. Their experimental results on TREC 6-8 ad hoc and Robust tasks show that their model significantly outperforms the standard language model and LDA-smoothed language model baselines. Zuccon et al. (2015) propose a translation language model that captures semantic relations between words in queries and those in relevant documents by using word embeddings within the well-known translation language model. A bit differently from these works, Mitra et al. (2016) present a documents ranking model named DESM in which a query document relevance score is computed by aggregating the cosine similarities across all the query-document word pairs.

There are also some models that use deep neural nets like MLPs or CNNs to directly learn embeddings for queries and documents themselves, however they use user generated data like click-through data. Microsoft's DSSM (Huang et al., 2013) or Microsoft's CLSM (Shen et al., 2014) are examples of those.

Word2doc combines these approaches. It performs two rounds of retrieval and augments the second round with information from the first round, similar to Lavrenko. The first round of retrieval is done through Facebook's document retriever, a purely statistical approach, while the second round is performed through a deep neural network. Furthermore, it reformulates the query for the second retrieval round by creating an embedding for it, and ultimately creates its own document embedding scheme to model semantic relations between documents and documents, and between documents and queries. It does this through the use of a deep neural net without the use of user generated data like DSSM or CLSM, and it is, to my knowledge, the first system to do so.

There are some other approaches that do not fall into the two categories above. Namely, Guo et al. (2017) who claims that a query-document match is not necessarily defined through semantic similarity, but more through relevance. They go on to present a deep relevance matching model (DRMM) for ad-hoc retrieval. The list of studies mentioned in this section is non-exhaustive. For a more detailed overview look at Mitra and Craswell (2017).

2 Theoretical Foundations

I will start off by discussing the underlying architecture and theoretical models behind word2doc. Word2doc is made up of three major components: First, InferSent, which is a sentence embedding method developed by Facebook (Conneau et al., 2017), second a frequency based document retriever also developed by Facebook and presented in (Chen et al., 2017), and finally the word2doc neural net architecture itself, that I developed and present in this thesis. InferSent and the document retriever are a pre-processing step for the neural network. Their results are combined and fed into the neural net, where the best fitting document is calculated. This process is depicted in the figure below.

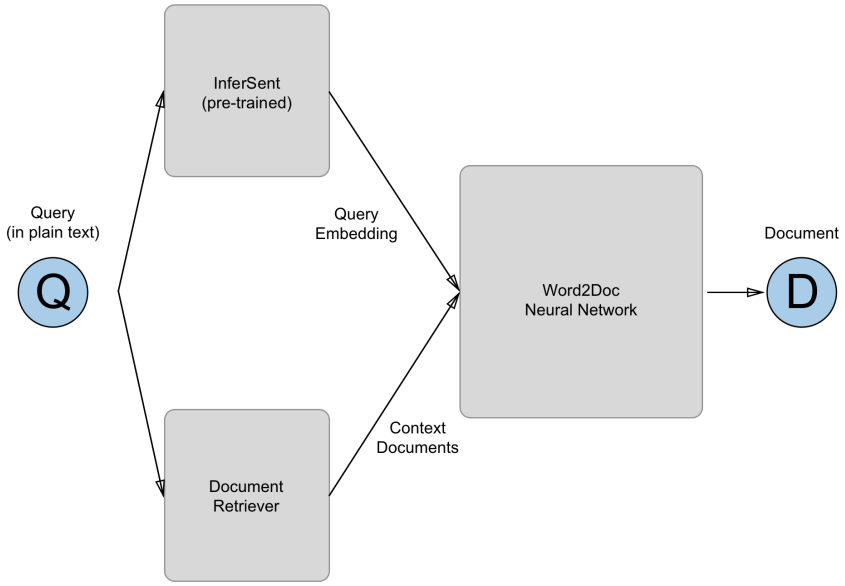


Figure 2.1: word2doc Overview

InferSent provides pre-trained sentence embeddings for the query. That means it generates an embedding out of the query, which is called a *query embedding*. Here, the query refers to the keyword or phrase we want to find a document for. I discuss this process in depth in section 2.2.2.

The document retriever generates a set of documents that are contextually similar to the one we are searching for. For example, say the query consists of "football". Then the document retriever will find the ten most similar documents to football, which could be "soccer", "goalie", etc. These documents provide the neural net with a context similar to the word contexts used to train word embeddings. These documents are called *context documents*.

Finally, the word2doc neural net takes both the query embedding and context documents as input to

generate a final result, which is the document best describing the initial query. In the following sections I will discuss each component in more detail. However, first I will explain how word2doc is trained.

Training

At prediction time, word2doc can take any query and return the best describing Wikipedia document, thus word2doc operates on the entire English Wikipedia corpus. To achieve this, word2doc is also *trained* on the entire Wikipedia corpus, similar to the way word embeddings are trained on a specific corpus, and only function within that corpus. That means overfitting is just as much a non-issue in word2doc as it is in word embeddings, since we want the model to represent as closely as possible the distribution we are modeling.

More specifically, training data consists of the Wikipedia document title as the query and the document ID as the label, which I also call the target document. Given the title, word2doc should learn to predict the document. However, as explained in section 3, the title is processed in such a way that word2doc doesn't memorize document-title pairs.

2.1 Relevant Concepts

Before I delve into more detail, I will explore some general concepts that will aid in the understanding of word2doc's full architecture. Different kinds of embeddings play an important role in word2doc, and that is why I will briefly review the concept of an embedding, and then explain sentence and document embeddings, which are more relevant for the thesis.

2.1.1 Embeddings

In the context of NLP, embeddings provide a means to compare the semantic similarity of two pieces of text. These pieces of text are often single words, in which case the embedding is called a *word embedding*. Their potential was arguably first demonstrated by Collobert and Weston in Collobert and Weston (2008) and made popular through word2vec in Mikolov et al. (2013a).

More specifically, an embedding is an n dimensional vector representation of a piece of text, where the vector represents the text's semantic meaning inside an n dimensional vector space. For example, take three words, say king, queen and apple, then the vectors for king and queen should lie closer to each other than the vectors for king and apple or queen and apple. "Closer" refers to the distance between two vectors and it is measured by the cosine similarity, which I talk more about later.

The following well-known example offers an intuitive way to look at word embeddings. If you take a word embedding for "king" and subtract the embedding for "man" and then add the embedding for "woman", the resulting vector will lie closest to the vector representation for "queen" in the vector space. This is illustrated in Figure 2.2.

Word embeddings are generally trained by scanning the context a word appears in. The context is defined as n words that surround the original word (a sort of window). If the network sees enough different word-context pairs, it will be able to generalize and eventually learn the meaning of the word. How this works in detail is not relevant for this thesis, and suffice it to say that there are two popular methods, one using the Bag-of-Words (Mikolov et al., 2013a) approach and one using the skip-gram approach (Mikolov et al., 2013b).

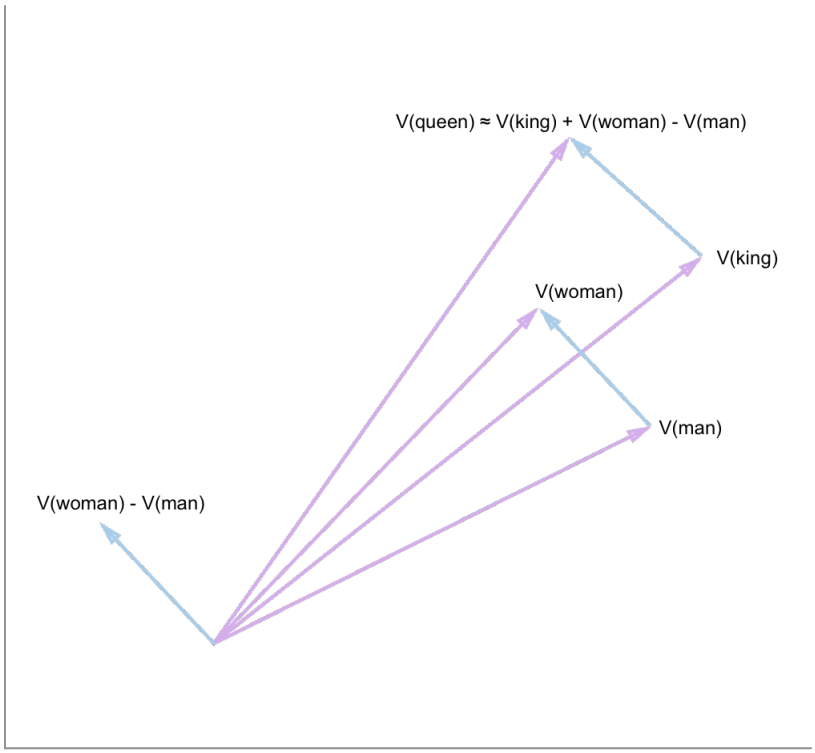


Figure 2.2: This figure shows possible calculations that can be done with word embeddings, and depicts in a simplified manner how embeddings represent semantic meaning. The V functions calculates the embedding for a given word.

It should also be noted that embeddings are low dimensional, meaning somewhere in the magnitude of 100s to 1000s. It turns out there is a sweet spot for the dimensionality of an embedding: neither too small nor too large. This sort of phenomenon, which can be found in many different machine learning problems, is often explained the following way: With too few parameters, the model cannot fit the signal, but with too many parameters, the model starts overfitting. Surprisingly, I could not find a good theoretical explanation to back up this intuition in the case of embeddings, so I will leave it at that. I mention this, because the low dimensionality of word embeddings does play a role in their success, even if it is hard to explain why, and thus I chose the magnitude for my embeddings accordingly.

Sentence Embeddings

Having reviewed embeddings, I will explain sentence embeddings. This is relevant in word2doc because the query from Figure 2.1 is represented through such an embedding. This provides the neural network with a fixed size semantic representation of the input query, which is easy for the network to work with.

The concept of a sentence embedding is very similar to the one of a word embedding. Instead of semantically representing a single word, the embedding represents the semantic meaning of multiple words, thus of a phrase or a sentence. This does become a significantly more complicated task and yields worse results than word embeddings, since much more goes into capturing the semantics of a sentence compared to the semantics of a word. Word order, punctuation, and inflection all play a role in determining the meaning of a sentence, which means sentences cannot be viewed as atomic elements, the way words can be. However, there are some approaches to go about creating sentence embeddings that I will now discuss.

Many different techniques exist to calculate sentence embeddings, with methods ranging from simple arithmetical composition of the word embeddings to more complex architectures using convolutional neural networks and recurrent neural networks. (e.g., Iyyer et al. (2015); Kiros et al. (2015); Socher et al. (2011); Kalchbrenner et al. (2014); Tai et al. (2015); Wang et al. (2016))

Simple Arithmetical Compositions for Sentence Embeddings The first approach I will discuss is a simple additional composition of word vectors. As presented in Arora et al. (2017), computing the weighted average of individual word embeddings in the sentence and then removing the projections of the average vectors on their first principal component, also known as common component removal, yields surprisingly promising results.

Arora et al. (2017) further write that "...this method achieves significantly better performance than the unweighted average on a variety of textual similarity tasks, and on most of these tasks even beats some sophisticated supervised methods tested in Wieting et al. (2015), including some RNN and LSTM models." While this may be true, techniques based on compositions of individual word embeddings have a drawback: they cannot handle out of vocabulary (OOV) words. If there is no pre-trained word embedding for a given word in a sentence, then it is no longer possible to calculate the correct weighted average of all word embeddings occurring in the sentence.

This is a cause of concern in the case of word2doc. It is likely that a word exists in the query for which there is no pre-trained word embedding, especially since the query is most likely composed of complex and infrequently used words rather than simple every day words. It is not an option to calculate pre-trained word embeddings for all possible queries because we cannot know all possible future queries, and thus it is more than likely that a query contains one or more OOV words at some point. As a result, the technique above is not a good option for word2doc, although it remains interesting for its simplicity.

Neural Net Approaches to Sentence Embeddings Another approach is presented in Kiros et al. (2015) as Skip-Thought Vectors, a method that uses an encoder-decoder architecture. The encoder is an RNN that is trained to encode sentences as embeddings, and the decoder is consists of two RNNs that decode the embeddings again into the two normal sentences that lie next to the original sentence. The idea is that sentences that lie next to each other share context, and that correctly decoding sentences requires the embedding to contain enough contextual information of the original sentence to be useful as an embedding. At the end, the decoders are thrown away and only the encoder is used to generate sentence embeddings. This method has become very popular, yet it still suffers from out of vocabulary words. To mitigate this problem, the authors present some methods of vocabulary expansion. For example, they propose working at character level instead of word level.

All in all Skip-Thought Vectors could work for word2doc, however, the technique is outperformed by Facebook's InferSent (Conneau et al., 2017) on various different tasks that measure the embedding's semantic representation.[1]. Furthermore, InferSent does not have any problems with OOV words.

Document Embeddings

Another form of embeddings are document embeddings. Just like other embeddings, document embeddings encode the semantic meaning of an entire document inside a fixed length vector. In word2doc, a document embedding is calculated for every document in the training data.

The document embeddings in word2doc are based on the paragraph vectors presented in Le and Mikolov (2014). Yet, there are some relevant changes that I made, however before I elaborate on those I will explain what paragraph vectors are.

Le and Mikolov (2014) present a way to calculate embeddings for paragraphs. They use a simple net that takes context words and a paragraph (indexed by IDs) in the form of vectors as input, and that predicts the next word in the context. For example, the model could take paragraph no. 42, "the", "cat", "sat", "on" as input and would try to predict "the". The architecture of that model is illustrated in Figure 2.3.

At training, a paragraph is chosen, and a sliding window is used to slide over the paragraph. Every paragraph and word is mapped to a unique vector, and as they are encountered again and again, these vectors are re-used. They slowly update their weights over the duration of the training to reflect the semantic meaning of the word, similar to a word embedding. The downside is that each word vector is kept the same across all seen paragraphs, which forces the net to generalize and thus learn.

At prediction time, the parameters for the model, the word vectors and the softmax weights, are fixed. Only the paragraph vector is not fixed, and via gradient descent the values of the paragraph vector are calculated. This way, the model can predict a paragraph vector for a never seen paragraph.

Word2doc does work differently than this as we will see later on, but it is inspired by this architecture.

Distance Metric for Embeddings

In this section I will talk about how I measure the distance between two embeddings. In word2doc, I use the cosine similarity metric, defined below.

[1] For more information, look at Table 4 in (Conneau et al., 2017)

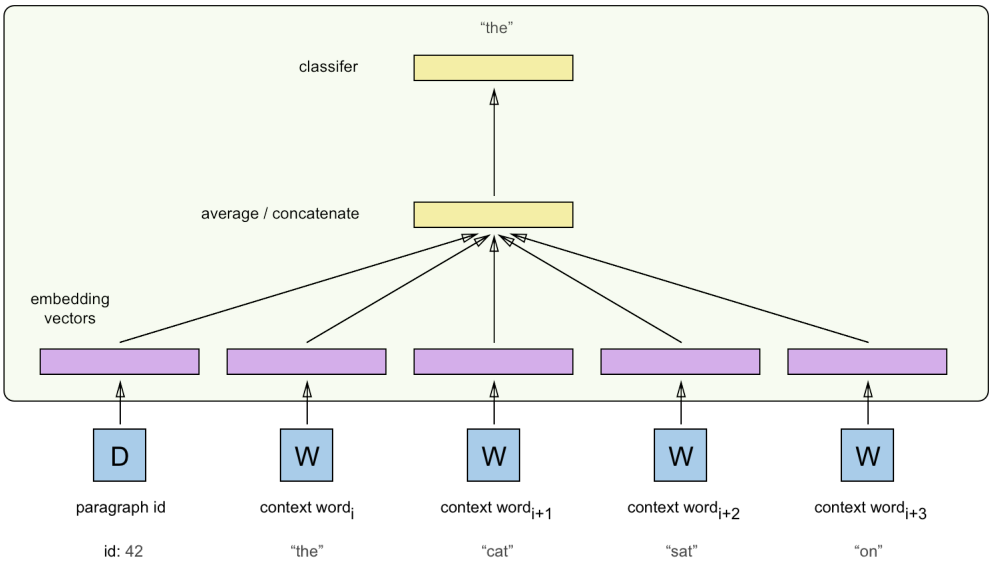


Figure 2.3: The PV-DM framework for learning paragraph vectors from Le and Mikolov (2014)

$$similarity = cos(\theta) = \frac{A \cdot B}{\| A \| \| B \|} \tag{2.1}$$

A and B are two vectors (the embeddings) and their dot product is divided by the product of their magnitudes. The result ranges from -1 to 1, where -1 indicates exact opposites, 0 indicates orthogonality (not correlated) and 1 indicates they are exactly the same.

It is an open discussion whether cosine similarity is the best metric to use for this task. Cosine similarity uses the norm of the vectors, and in doing so implicitly assumes the lengths of the vectors can be ignored, since they are normalized. The norm of the vectors is a nice thing to use, because it is somewhat related to the overall frequency of which words occur in the training corpus (in the case of word embeddings).[2]

However, in practice it is not clear whether the length of the vectors can be ignored, or whether it might actually hide important information and should be used. For example, Schakel and Wilson (2015) argue that length does matter, and suggest using the L2 norm in combination with term frequency to compare embeddings.

Despite this uncertainty, I chose the cosine similarity. Perhaps better results are possible with other methods, however cosine similarity is by far the most established, and thus I deem it less risky compared to other methods.

[2] I could not find any solid evidence for this claim, however it seems to be the general consensus among the community that it is so. This claim resurfaced on many GitHub issues and Stackoverflow posts again and again.

2.2 Architecture

Now that I covered some basic concepts used in word2doc, I will talk about each system in word2doc in detail. As shown in Figure 2.1 there are three major components, and I will start backwards, with the neural network itself, then explain how InferSent works in detail and last explain the document retriever.

2.2.1 word2doc Neural Network

In this section I will describe how the architecture of word2doc's neural net works. In a first step, I use a pre-trained model of Facebook's InferSent (Conneau et al., 2017) to create a query embedding. So unlike in word2vec, the network does not take a word as input, but rather a 4096 unit pre-trained sentence-embedding of the query (the query embedding). This allows the network to work with any word, because InferSent can work with any OOV word, as explained above.

The network then takes this 4096-unit-long query embedding and trains a new fully connected hidden layer. The hidden layer's purpose is to reduce the dimensions of the previous embedding, from 4096 to 512. This is done to reduce the amount of work backpropagation has to do, since the amount of weights is drastically reduced. For example, look at the second fully connected ReLu layer in Figure 2.4. If we have 5 million units (so documents) in the output layer, and we keep the 4096 units instead of 512, then we would have to train $4096 * 5,000,000$ weights, close to 20 billion. Using 512 units instead, we only have to train close to 2 billion weights, so considerably less.

Additionally, the network takes a list of context documents as input. This list of context documents is nothing but a list of document IDs, so a list of integers. For each ID a document embedding is calculated in the same manner a word embedding would be calculated. This implies that across all keywords, the document embeddings D as seen in Figure 2.4 stay the same for different keywords. It does not matter if the document 3234 appears in the context of the keyword "football" or "quarterback", in both cases the same embedding will be used. This is important because it forces the network to generalize. As the same document embeddings appear and re-appear in combination with different queries, their weights will ultimately shift in a direction that reflects the semantic meaning of their documents.

All document embeddings are then averaged along with the output of the hidden layer from above, similar to how it is discussed in the paragraph vectors from Le and Mikolov (2014). However, in the case of paragraph vectors, the authors concatenate instead of average, claiming that concatenation yielded better results. I also tested concatenation which I talk about in section 4.3.2, however, word2doc yielded better results when averaging. Furthermore, the query embedding replaces the paragraph id from Figure 2.3, and the documents fetched by the document retriever replace the context words used in paragraph vectors. That means instead of using a sliding window like it is done in paragraph vectors to identify context, word2doc uses the document retriever. Additionally, the paragraph id that is trained in Le and Mikolov (2014) is used to make a prediction that is kept instead of throwing it away. The interesting side effect is that that working document embeddings are calculated this way. I talk more about this in section 4.3.1.

The concatenation is then passed into another fully connected ReLu layer to reduce the dimensions again for the same reason as above. This reduced tensor is then passed to the output layer, where a one-hot encoding is calculated using negative sampling (a loss function) to find the best matching document. See Figure 2.4.

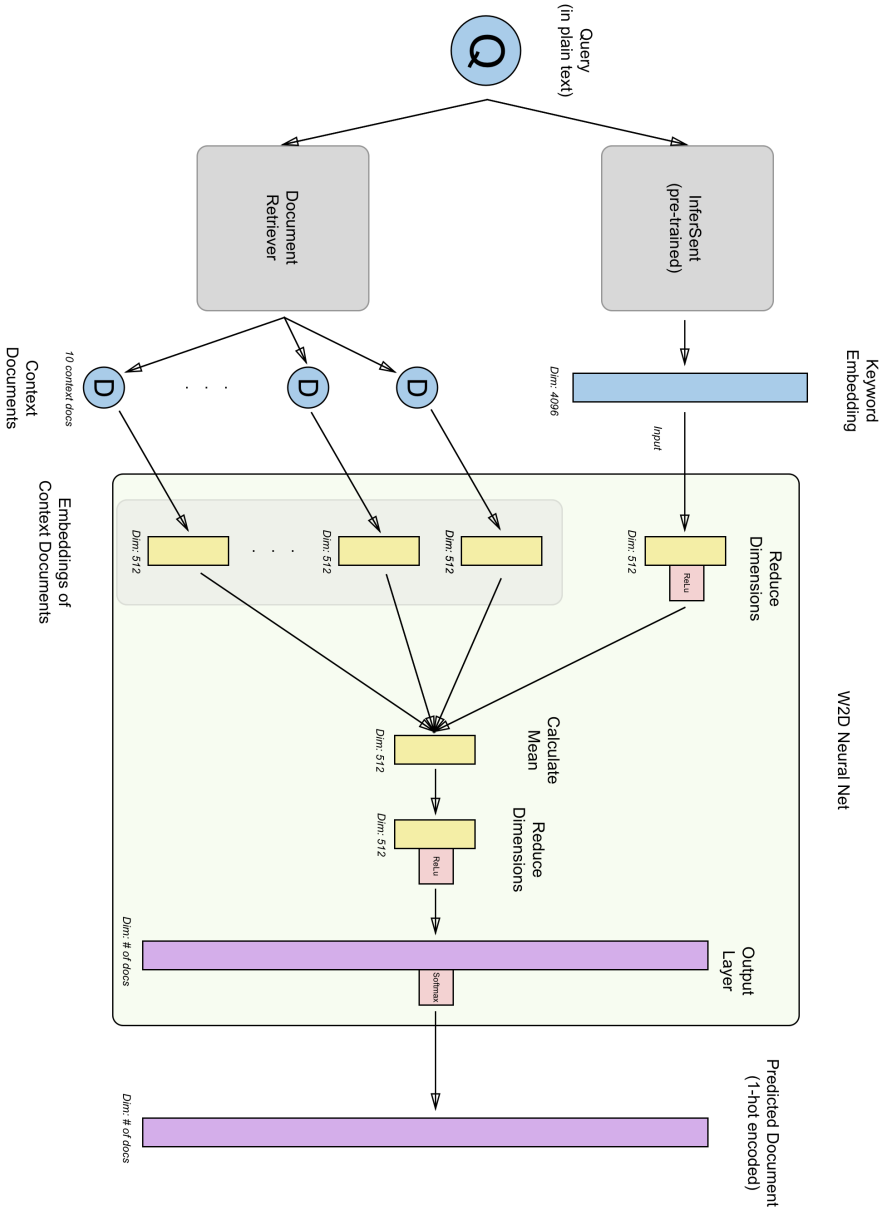


Figure 2.4: W2D Architecture

Negative Sampling

Word2doc uses negative sampling as a loss function. Negative sampling is a technique proposed by Mikolov et al. (2013b) to speed up training. Instead of adjusting all weights of the final output layer in each iteration, only a handful of weights are updated. A small amount of "negative" keywords are chosen, keywords that have nothing to do with the actual "positive" keyword. For example, if "neural network" is the positive keyword, then "gardening" and "football" might be negative keywords, because they have nothing to do with "neural network". The weights for the positive keyword are then updated in a "positive" direction, and the weights for the negative keywords are updated in a "negative" direction.

If done over many iterations, negative sampling will eventually update all weights while being much faster than the classic approach. Since word2doc has the total number of Wikipedia articles as output neurons, negative sampling is necessary, otherwise training would take a very long time.

Usually negative samples are chosen through the frequency of their occurrence, with more frequent words being more likely to be chosen as negative samples. In the case of word2doc however, there is no meaningful probability that a given query occurs. As a result, the network is handed a subset of documents (the context documents) from which not to sample, and it samples randomly from all other remaining documents.

2.2.2 InferSent

As mentioned before, Infersent is a tool used to calculate sentence embeddings developed by Facebook (Conneau et al., 2017). In word2doc it is used to create sentence embeddings from the query in order to get a fixed size semantic representation of all possible queries. The resulting embedding has a dimension of 4096, and is fed into the neural network where it is combined with context documents to choose a final, best matching document to the query. In this section I explain how this is done in detail.

Conneau et al. (2017) investigate seven different sentence encoding architectures, and find out that "...an encoder based on a bi-directional LSTM architecture with max pooling, trained on the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015), yields state-of-the-art sentence embeddings compared to all existing alternative unsupervised approaches like SkipThought or FastSent, while being much faster to train." Furthermore, they note, "...we establish this finding on a broad and diverse set of transfer tasks that measures the ability of sentence representations to capture general and useful information."

A transfer task is a problem in which knowledge gained in one task is applied to another. For example, knowledge gained while learning to recognize cars could be applied when trying to recognize trucks. This is what word2doc needs to be able to do. During training, queries are created through augmented Wikipedia titles and keywords, yet at prediction word2doc needs to handle more abstract and complicated queries created by people. Thus, Wikipedia titles are learned and need to be applied to a more general, real-life scenario. Granted, this is not as clear as the car and truck example from above, but it still qualifies nonetheless. Thus, if InferSent does well on transfer tasks, it should do well for word2doc.

Furthermore, InferSent uses the SNLI dataset as training data, which, the authors theorize, works well with sentence embeddings.

> The SNLI dataset consists of 512k human-generated English sentence pairs, manually labeled with one of three categories: entailment, contradiction and neutral. It captures natural language inference, also known in previous incarnations as Recognizing Textual Entailment (RTE), and constitutes one of the largest high-quality labeled resources explicitly

SNLI Sample Data		
Premise	Judgment	Hypothesis
A man inspects the uniform of a figure in some East Asian country	contradiction	The man is sleeping
A smiling costumed woman is holding an umbrella	neutral	A happy woman in a fairy costume holds an umbrella
A black race car starts up in front of a crowd of people	contradiction	A man is driving down a lonely road
A soccer game with multiple males playing	entailment	Some men are playing a sport

Table 2.1: SNLI Sample Data

> constructed in order to require understanding sentence semantics. We hypothesize that the semantic nature of NLI makes it a good candidate for learning universal sentence embeddings in a supervised way. That is, we aim to demonstrate that sentence encoders trained on natural language inference are able to learn sentence representations that capture universally useful features.

(Conneau et al., 2017)

The sample data from the SNLI corpus is shown in Table 2.1, taken from the SNLI website. [3]

InferSent Architecture

The basic architecture is illustrated in Figure 2.5 on the next page. The architecture has two sentence encoders that output a representation for the premise u and the hypothesis v. Once the resulting sentence vectors are generated, relations between u and v are calculated using three methods: concatenation, element-wise product and absolute element-wise difference. The result is fed into a 3-class classifier consisting of multiple fully connected layers culminating in a softmax layer. The network is trained to predict the judgment between the premise and the hypothesis.

Next, I will talk about the sentence encoder. Like I mentioned in the beginning of this section, the authors of InferSent tested seven different sentence encoder models, and found out that the bi-directional LSTM architecture with max pooling performed the best. Thus, I will ignore the other methods and only focus on the bi-directional LSTM here.

Two LSTMs read a sequence of words from the premise and hypothesis in opposing directions, generating the vectors h_t by concatenating the forward and backward LSTMs in the following way (Conneau et al., 2017):

$$\overrightarrow{h_t} = \overrightarrow{LSTM}_t(w_1, ..., w_T) \tag{2.2}$$

$$\overleftarrow{h_t} = \overleftarrow{LSTM}_t(w_1, ..., w_T) \tag{2.3}$$

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}] \tag{2.4}$$

[3] Note that this is not the exact representation. Some details were left out for the sake of simplicity. For the exact representation, please visit https://nlp.stanford.edu/projects/snli

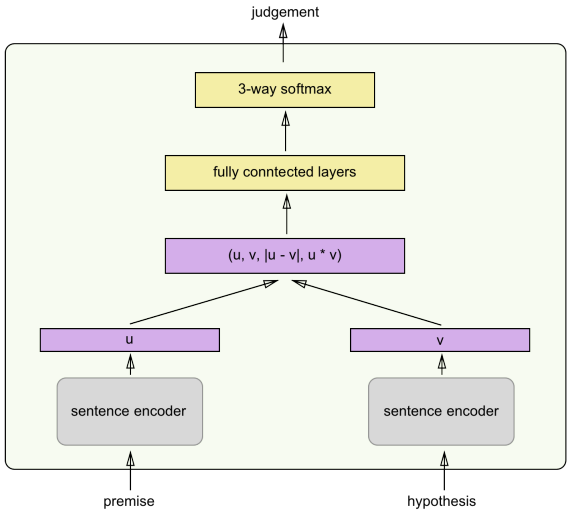


Figure 2.5: InferSent Natural Language Inference Training Scheme based on Figure 1 in Conneau et al. (2017)

That means h_t is the concatenation of $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$. The h_ts are then combined through max-pooling (Collobert et al., 2011), a technique in which the maximum value over each dimension of the hidden units in the LSTM is selected. This is illustrated in the Figure 2.6 from the original paper.

The result of this max pooling operation then yields the finished sentence encoding that is plugged back into the model seen in Figure 2.5.

2.2.3 Document Retriever

The document retriever is taken from Facebook's DrQA system (Chen et al., 2017) and it replaces the sliding window from paragraph vectors (Le and Mikolov, 2014) to generate a list of contextually similar documents that match the query. In this section I will explain how this is done.

The document retriever takes the query as input, and generates a list of ten documents that best match the query. Best in this context means documents that are closest to the query's topic. You can find an example of what this looks like in Table 2.2 on the next page.

Table 2.2 shows the actual results when submitting "neural nets" to the document retriever. They are listed in no particular order, because order does not matter in word2doc, as they are shuffled before training.

These documents are calculated purely through a non-machine-learning technique that is based on the frequencies of occurring words from the query. In a first step, an inverted index is created using all Wikipedia documents. More specifically, a sparse word to document count matrix M is created, such that M_{ij} is the number of times word i appears in document j. Word i is in fact not quite a word, but the hash of a bigram. This is done, according to the authors, to take local word order into account, which further improves the system by 1-2%. The authors note, that bigrams provided the best results out of all tried

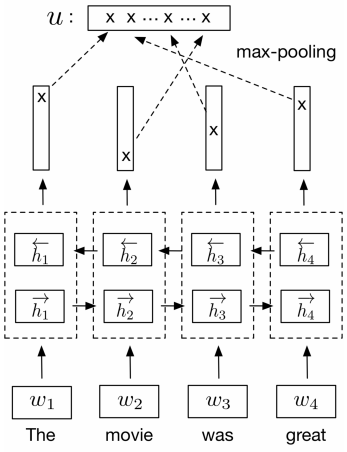


Figure 2.6: Bi-directional LSTM used as sentence encoder (Figure 2 in Conneau et al. (2017))

Document Retriever Sample Results	
Query	"neural nets"
Results (as Wikipedia document names)	Convolutional neural network Deep learning Alternative explanations of the "grandmother" cell The Age of Spiritual Machines Artificial neural network History of artificial intelligence Artificial consciousness Deeplearning4j Speech recognition Neurophilosophy

Table 2.2: Document Retriever Sample Results

n-grams. [4] An unsigned *murmur3* hash function (Weinberger et al., 2009) is used to quickly count the number of times a bigram appears in a document by dividing it into 2^{24} bins, which preserves speed and memory.

After this word count matrix is calculated, it is transformed into a TF-IDF matrix. TF-IDF is a numerical statistic that reflects to a certain degree how important a word is to a document within a corpus of documents. The TF-IDF value for a word changes proportionally to the frequency of the word in a document and is offset by the frequency of the word in the corpus, which takes into account that some words appear more frequently in general. The bigger this value, the more "important" the word is in the document, and the smaller the value, the less "important" it is. Thus, every element in the count matrix is transformed the following way to get a TF-IDF filled document matrix M'.

$$tfidf = log(tf+1) * log((N - N_t + 0.5)/(N_t + 0.5)) \tag{2.5}$$

tf is the term frequency, N is the number of documents and N_t is the number of times term t occurs in all documents. Thus, a term frequency based model is calculated, but note that term still refers to bigrams, and not unigrams.

Finally, a TF-IDF-weighted word vector V is calculated from the query, and the dot product between it and the TF-IDF document matrix is calculated in the vector space the of the TF-IDF-weighted word vector. To calculate V, the query is essentially seen as its own document, and the entire process from above is repeated for this new "document". Thus, we get a vector of TF-IDF weighted bigrams from the query.

The dot product between V and the TF-IDF document matrix M' is then formed the following way: Since there are 2^{24} bins, and since there are 5,316,954 documents in the Wikipedia dump I am using, M' has dimensions of $2^{24}x5,316,954$. The word vector V, like M, is also sparse and since its bigrams are subject to the same hash function system, its bigrams are also put into bins of size 2^{24}. Thus, it has the dimensions of $|V|x2^{24}$, where $|V|$ is the number of bigrams created from the query, so the length of V. Since V and M' share a dimension, the dot product can be calculated between the two, and the result is a sparse matrix R of size $1x5,316,954$, essentially a vector of document scores for each document. Finally, the top ten biggest scores from R are taken, and the documents belonging to the scores are returned as the best matching documents. Theses are the results in Table 2.2.

Taking the dot product between V and M' works because high TF-IDF scores from the query ("important" bigrams to the query) multiplied with high TF-IDF scores from a document ("important" bigrams to the document) yield a high product overall. If this happens with enough bigrams throughout the document, and is not offset by low TF-IDF score products, the sum of all of these scores (the dot product) will be rather high too, signaling that this document is a potential match for this query. If however, the query and the document do not contextually match well, there should be a considerable amount of bigrams whose product does not yield high scores, and thus the overall score is diminished compared to cases in which the query and document contextually do match.

[4] Unfortunetaly, the authors of Chen et al. (2017) do not provide data as to what other n-grams they tested.

3 Data

In this section I talk about different datasets word2doc uses. More specifically, I talk about training and evaluation datasets as well as pre-trained models that word2doc uses, and how these models might perform even better when custom trained.

3.1 Training Data

First, I will talk about how word2doc's training data is generated. I will talk a bit about overfitting and problems related to it, and then I delve into the details of generating the training data itself.

3.1.1 Overfitting

As mentioned before, word2doc operates on the entire English Wikipedia corpus, and is thus trained on the entire Wikipedia corpus, similar to the way word embeddings are trained on their own corpus. This is not an issue from an overfitting perspective, because we want overfitting to occur, as explained in section 2.

However, it is possible that word2doc memorizes document-title pairs instead of learning the semantic meaning behind each. What I mean is that word2doc could learn to match query "football" with document "football", not because it understands that both share a similar context, but because they mostly only occur together. When "soccer" is then used instead of "football", the net might not be able to identify that football and soccer share a similar meaning, because it only learned the title document associations which is pointless anyway (we know what title belongs to what document).

In word embeddings this problem solves itself, because the context a word appears in always varies (section 2.1.1). For example, the word "football" will occur together with so many different words, that there is little risk of building a one-to-one connection like described above. However, since word2doc is trained with Wikipedia document-title pairs, it is inherently so that a query (the title) is very likely to occur most often with its own document. In other words, documents do not have multiple titles, and titles do not have multiple documents, and this is a problem.

To counter this problem, I perform data augmentation. The title is dissected into important keywords (if the title is long enough) and the six most important keywords of the first paragraph of each Wikipedia document are extracted using rake (Rose et al., 2010), a keyword extraction tool. More specifically, I take the first two keywords from the very first sentence of a document, which is usually the topic sentence and thus it summarizes the document well. I then take four keywords from the remaining paragraph, which is the introduction paragraph. It should be noted, that not every query receives six keywords. Sometimes the first paragraph is not long enough, resulting in fewer than six keywords extracted. Thus, not every query has exactly six keywords.

The first (introduction) paragraph of a Wikipedia article is almost always a brief summary of the article itself, which is a nice feature I am exploiting. That means the six most important keywords of the first

paragraph are likely to be six important keywords describing the document. Using six keywords instead of two or three means that the later keywords fit the document less and less and thus they are more likely to also occur in different documents. Each of these six keywords is then paired with the target document, which remains the same. For example, let's say the "Artificial Neural Network" document (ID: 324923) has the following 5 keywords: neural, network, brain, programming and label. The data would be augmented with the following pairs: (neural, 324923), (network, 324923), (brain, 324923), (programming, 324923) and (label, 324923). Each pair will now be treated independently of each other, as its own separate data point.

If each individual keyword is paired with the original document in the way it is explained above, we should have conditions closer to those of word embeddings. Queries now share multiple documents since the extracted keywords are not unique to one document, and documents can share multiple queries. Ultimately, we will have gone from a one-to-one mapping between queries and documents to an n-to-n mapping. I call these keywords *pivots*, not to be confused with pivots in word2vec (Mikolov et al., 2013a). I call them pivots because the keywords pivot around the original document. When a sentence embedding is applied to them, I call them *pivot embeddings*.

Thus, my claim from section 2 that word2doc is only trained by using document titles as queries and the documents themselves as labels is not entirely accurate. For each document there are up to seven different data points. Of course augmented data points share the same context documents, since they all share the same label (the target document).

3.1.2 Wikipedia Dump

To obtain the English Wikipedia corpus, I downloaded a Wikipedia dump and processed it with Giuseppe Attardi's Wikiextractor tool [1]. The tool extracts text from the dump, along with metadata like the url and the title and saves the content to many JSON files, where it is then saved into a database to be processed by word2doc.

3.1.3 Generating Training Data

Once the entire English Wikipedia corpus is pre-processed, word2doc iterates through every single document to generate the necessary data for the neural network. That means for every document, context documents are calculated, and for every query, a query embedding is created. To be more efficient, the data is split into 100 bins of equal size, with 10 bins being processed at a time through a SLURM queue. Table 3.1 illustrates what a data point in the training data looks like.

[1] `https://github.com/attardi/wikiextractor`

Training Dataset Excerpt	
Document title	Hand crafted query
doc_index: doc_title:	1198816 Commonwealth Railways
pivot_embeddings	[[0.02357988, 0.01399379, ..., -0.03814263, -0.02892261], [0.05076515, -0.08939897, ..., -0.03814263, -0.02892261], [0.1395378 , 0.05702531, ..., -0.02987907, -0.02892261], [0.0112771 , -0.08939897, ..., -0.03814263, -0.02892261], [0.00092854, 0.00179368, ..., -0.03814263, -0.02830232]]
doc_window	[2678437, 2312586, 1198826, 5040959, 1198834, 1198831, 2198060, 1198816, 1198827, 1198817]

Table 3.1: Training Dataset Excerpt

Looking at Table 3.1, the document index is the ID of the Wikipedia article and serves as a label for the neural network. The pivot embeddings are all the pivots processed into 4096 long sentence embedding vectors. That means, for the document "Commonwealth Railways" there were 4 pivots plus the original title which is at position 0 of the array, resulting in a total of 5 pivot embeddings. And last but not least, the document window which are the context documents for *Commonwealth Railways*. This data is fed to the neural network, where each pivot embedding is extracted, paired with the label (document ID) and context documents, and added as an individual data point.

When this pre-processing step is complete with all 5,316,954 Wikipedia articles, the result is 100 bins, each with a size between five and six Gigabytes. In total, around 600 GB of data is generated this way, and then processes by the neural network.

3.2 Validation Data

Testing the performance of word2doc is not trivial (explained in section 4.1), and to do a better job, I generated two evaluation datasets consisting of 200 and 400 hand crafted data points each. In this section I talk about how I created these data points, and then I briefly mention how they are used in word2doc. The dataset of size 200 is called *W2D-VD-200* for word2doc validation data 200, and the dataset of size 400 is called *W2D-VD-400*.

To create these data points by hand, I built a testing module for word2doc (*word2doc-net-test.py*) that presents me with a random document from a sub-selection of 5000 documents. These data points were randomly sampled from the full dataset. For each random document, I am presented with the document title and the introduction paragraph, and from those I create a query by hand. This hand made query is then added to the evaluation dataset as an individual data point. An excerpt of this dataset is shown in Table 3.2.

Most of the time the query is a brief description of the document, similar to what I would type into a Google search. Since I created these queries myself, they are subject to a certain human bias, meaning I would not be surprised if I, without really knowing it, constructed the queries in a way that I deem them more likely to succeed, since I ultimately want them to succeed. To counter this, and to also make the test more realistic, I included tougher data points as well. More specifically, the model has to deal with typos (number 6.), it has to understand abbreviations (number 2.), and it has to turn numbers into their written form (number 3.). It did this surprisingly well, suggesting the model did learn something, but

Evaluation Dataset Excerpt		
No.	Document title	Hand crafted query
1.	Strike Back: Project Dawn	project dawn tv series
2.	National Operational Intelligence Watch Officer's Network	NOIWON secure conference call
3.	RARRES1	Retinoic acid receptor responder protein 1
4.	Medicine 8	Medicine eight
5.	Nigel Parry	new york photographer parry
6.	Guigues VIII of Viennois	guigues ofnvienne

Table 3.2: Evaluation Dataset Excerpt

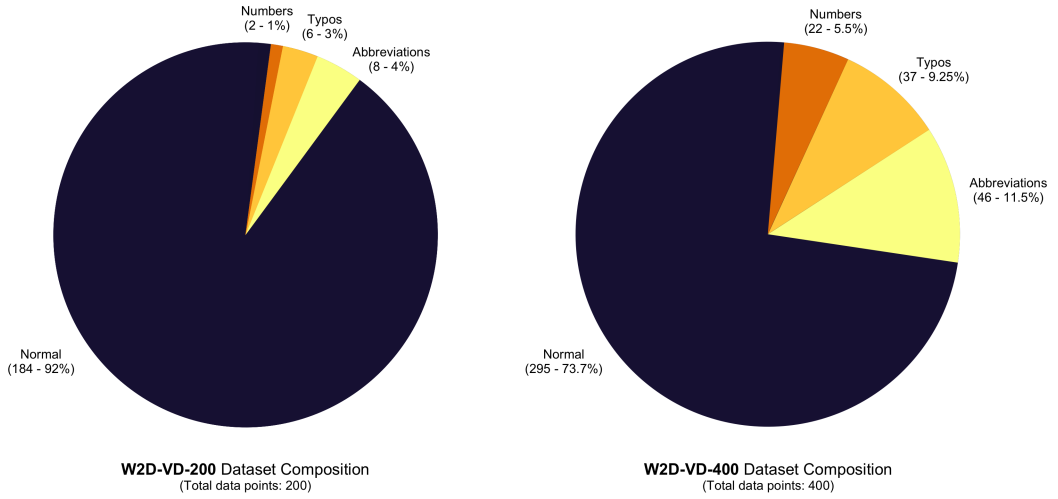


Figure 3.1: Composition of W2D-VD-200 and W2D-VD-400 datasets.

more to that in section 5. The exact composition of each dataset is found in Figure 3.1.

In Figure 3.1, "normal" refers to data points that have not been manipulated in any way. The "abbreviation" category is a broader term for cases in which one of the documents is refereed to by an alternative title. This can be a typical abbreviation, but it can also be a nickname or a title in another language. The abbreviation category is testing the net's ability to learn multiple terms for the same concept. The typos category is self-evident, it contains intentionally created typos. Finally, it should be noted that W2D-VD-200 is a subset of W2D-VD-400 with more emphasis on "normal" data points. W2D-VD-200 is used in section 4.3.2 to measure performance of different word2doc models, and W2D-VD-400 is used to evaluated word2doc in section 5.

3.3 InferSent

At last, I will talk about the trained InferSent model used in word2doc. I did not train it myself, instead I used Facebook's pre-trained model, trained on what Facebook called AllNLI, made up of the SNLI and MultiNLI datasets. I already talked about SNLI in section 2.2.2, and MultiNLI is the NYU's Multi-Genre NLI corpus, similar in structure to the SNLI corpus from Table 2.1. Both datasets are used because a

significant boost in performance was observed using both datasets, compared to just SNLI (Conneau et al., 2017).

The authors of InferSent note in their conclusion, that this work only scratches the surface and that they believe with more complete datasets significant improvements can be made to the quality of sentence embeddings. In the context of word2doc, this could mean that using more complete datasets, or domain specific datasets (if word2doc were to run in only a certain domain) could potentially improve final results.

3.3.1 GloVe

InferSent uses GloVe vectors during training. GloVe stands for Global Vectors for Word Representation, and is a collection of pre-trained word vectors part of Stanford NLP (Pennington et al., 2014). Through GloVe vectors, InferSent uses a vocabulary consisting of over 2.2 million of the most common english words to generate sentence embeddings. Instead of using pre-trained GloVe vectors, it could be an option to train word embeddings on vocabulary most likely to occur in word2doc's domain, if one were to use word2doc on a specific domain. Like I mentioned in the introduction, Diaz et al. (2016) claim that global embeddings like pre-trained GloVe vectors are underperformed by locally trained embeddings. However, GloVe vectors are very established and since they are very general and since the Wikipedia corpus is also very general, they are probably a pretty good match nonetheless.

More specifically, InferSent uses the *glove.840B.300d.txt* collection, a collection of pre-trained word vectors based on a Common Crawl [2] data. Common Crawl builds all kinds of datasets by crawling the internet. The Common Crawl GloVe model InferSent uses has 840 byte tokens, a vocabulary size of 2.2 million, it is cased, and has vectors of dimension 300.

If you go on the GloVe website and look at the models available, you might notice that there is actually a model available specifically trained on a 2014 Wikipedia dump. However, it has only has a vocabulary of size 400,000 and thus is much smaller than the Common Crawl dataset which makes it less suited for the task. Ultimately a much larger vocabulary is better, because it means the chance of encountering an OOV word is much smaller. InferSent can still process OOV words, as I explained in section 2.2.2 but it does a better job if the word is included in the pre-trained vocabulary.

[2] `http://commoncrawl.org/`

4 Experiments

In this section I will present methods used to measure word2doc's performance, as well as present certain implementation details and back up several design decisions with empirical data.

4.1 Evaluation Metrics

It is hard to accurately measure word2doc's performance because there is no such thing as a designated evaluation dataset working with Wikipedia articles, or at least I could not find one. There are various document retrieval test sets in existence, such as the TREC datasets[1], however these datasets do not work with the Wikipedia corpus, and must be purchased as well. The SQuAD dataset[2] does use the Wikipedia corpus, however it is a QA dataset and only has 442 unique documents. Most of the data comes from individual queries for different paragraphs in the documents, not the documents themselves. I judge 442 documents to be too few to contain enough documents to accurately create context documents, thus training the model on SQuAD will likely not work. Additionally, SQuAD's queries are QA queries, and because of that they often lack contextual information, or have misleading contextual information. For example, queries like "How has this foundation changed in recent years?" do not contain any context and are not uncommon throughout the dataset. For these reasons I chose not to use the SQuAD dataset.

Performing cross-validation is not an option either, because then the net would never have seen certain types of data. For example, if the net has never been exposed to football related documents, it will not be able to predict the best document associated with "quarterback". Instead, I have come up with a testing module to create a small sample of hand labeled keyword-document pairs as described in section 3.2. Using this dataset I measure performance on three tasks: the quality of the *top one* retrieved document, the quality of the *top five* retrieved documents and the quality of the *top ten* retrieved documents, including their ranking. I measure the quality of the top one document through simple accuracy, and the quality of the top five and top ten retrieved documents through mean average precision (MAP), as well as through accuracy. As an evaluation dataset I use W2D-VD-400, and I evaluate word2doc using random samples consisting of 1% of the Wikipedia corpus. More to that in section 5.

A lot of different evaluation metrics exist to measure the performance of IR systems (e.g. precision, recall, F measure, R-precision, ROC curves, NDCG, MAP). I will explain why I choose MAP instead of other metrics, but first I will quickly explain the concepts of precision and recall because they are relevant.

Precision Precision (P) is the fraction of retrieved documents that are relevant:

$$Precision = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved}) = \frac{tp}{tp+fp} \tag{4.1}$$

[1] `https://trec.nist.gov/data.html`
[2] https://rajpurkar.github.io/SQuAD-explorer/

where tp is true positive and fp is false positive.

Recall Recall (R) is the fraction of relevant documents that are retrieved:

$$Recall = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant}) = \frac{tp}{tp+fn} \tag{4.2}$$

where tp is true positive again and fp is false positive again, and fn is false negative.

F Measure F measure is a single measure that trades off precision versus recall. In fact, it is the weighted harmonic mean of precision and recall, and one of the most widely used performance metrics for IR tasks:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha} \tag{4.3}$$

where $\alpha \in [0,1]$ and $\beta \in [0,\infty]$. The most common F measure is the F_1 measure, which equally weights precision and recall, making $\alpha = 1/2$ or $\beta = 1$. The 1 in F_1 refers to the β value, and is short for $F_{\beta=1}$. When using $\beta = 1$, the above formula simplifies to:

$$F_1 = \frac{2PR}{P+R} \tag{4.4}$$

However, we don't have to use an equal weighting. When $\beta < 1$, precision is emphasized, and when $\beta > 1$, recall is emphasized. Thus, a value of $\beta = 3$ or $\beta = 5$ might be used if recall is more important, and a value of $\beta = 1/2$ when precision is more important.

F measure is popular because it trades off two opposing metrics. For instance, you can get a recall of one by returning all documents, however that yields a low precision, and vice-versa. Finding a trade off between these two metrics is a good way of measuring the overall performance of an IR system. However, it does require to calculate the recall, and that is often a problem in large data sets. Recall is the fraction of relevant documents that are retrieved, so in order to calculate recall you need to know the set of all relevant documents for a given query. That is very hard to calculate in large sets such as Wikipedia. Instead, it is possible to estimate recall by taking samples of the dataset, or by adhering to golden standards for a dataset, if such are available (Manning et al., 2008). However, since I am evaluating word2doc on a subset of the full dataset I have no golden standard available, and taking a sample of the sample to estimate recall does not seem to yield accurate results. Thus, I discarded all metrics using recall or that requires similar knowledge as recall, such as ROC curves or R-precision.

Furthermore, F measure does not take the order of a ranking into account, and applying it to a single returned document is useless (precision and recall assume there is a set of relevant documents greater than one). Thus, it does not make sense to apply F measure to either of my two evaluation tasks. Instead, I turned to MAP, which is a standard metric to measure the quality of a ranking.

4.1.1 Mean Average Precision

According to Manning et al. (2008), MAP is the most common metric to measure performance in the *TREC* community. Like the F measure, MAP also provides a single-figure measure of quality for all queries. It does this across all levels of recall and has been shown to have especially good discrimination and stability (Manning et al., 2008). For a single query, average precision is the average of the precision value obtained for the top k documents remaining, after each document is removed from the set one after the other. This value is then averaged over all queries. More specifically, if the set of relevant documents for a query $q_j \in Q$ is $d_1, ..., d_{mj}$ and R_{jk} is the set of ranked retrieval results from the top to document d_k, then

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}) \tag{4.5}$$

MAP is a rough estimate of the area under the precision-recall curve for a set of queries Q. The precision-recall curve is a graph in which precision is mapped against recall. It has often a saw-tooth shape which occurs the following way: If the n^{th} document retrieved is useful, both precision and recall increase, but if it is not, recall stays the same but precision drops, resulting in a saw-tooth like shape.

Furthermore, MAP weights higher ranked documents much more heavily than lower ranked documents, because of the way precision works. With only one document precision is either 0 or 1. If we add a second document which is not relevant, and assuming the first document was relevant (precision = 1), than precision drops to 0.5, and so on. Thus, the first document alone having a precision of one is weighted twice as much as the first and second documents combined, which have a precision of 0.5. MAP thus has a natural way to give more attention to more relevant documents.

MAP scores can strongly vary, but the higher the score, the better the system performs. A good score is usually somewhere between 0.5 and 0.7. However, it should be noted that MAP scores can vary widely across different queries, sometimes between 0.1 and 0.7 and that test data needs to be large and diverse enough to yield a representative score.

To calculate MAP scores, I built an evaluation system (*evaluation.py*) which presented me pairwise with the query and the document in question. I then judged whether the document in question was relevant to the query. I did this for all ranked documents for all queries, and the system evaluated MAP scores out of my judgments. It should be noted that my judgments are not bias free, and in order to gain more accurate results with less bias, a system like Amazon Mechanical Turk could be used.

4.2 Implementation Details

In this section I cover relevant implementation details. I present the hyperparameters used while training the final model presented in this thesis, as well as mention a few steps I took to improve final results. I also explain how I trained subsets of the full Wikipedia data, and talk about some problems that come with that.

4.2.1 Hyperparameters

As I will explain in section 5, I only trained word2doc on 1% of the full Wikipedia dataset. The hyperparameters used train the word2doc neural net on that 1% (58,170 documents) can be found in Table 4.1.

W2D Network Hyperparameters	
Parameter	Value
epochs	30
batch size	2096
input dimension	4096
embedding dimension	512
embedding vocab size	581700
num. context docs	10
dropout rate	0.3
loss function	negative sampling
negative sampling sample size	10
optimizer	adam
adam learning rate	0.001
adam beta1	0.9
adam beta2	0.999
adam epsilon	1e-08

Table 4.1: W2D Hyperparameters

The *input dimension* refers to the InferSent embedding, which has a dimension of 4096. The *embedding dimension* is the dimension of the document embedding vectors. The embeddings are relatively low dimensional as explained earlier in section 2.1.1. The *embedding vocab size* is the number of documents for which an embedding was learned. This is not equal to 58170 because for each of the 5000 documents, 10 documents were chosen as context, so $58170 * 10 = 581,700$. There is some overlap between documents, but since we don't know how much, I kept it at $581,700$. However, there is no impact if there are too many document embeddings, some will just not be used. The *num. context docs* refers to the number of context documents word2doc uses, which is ten. *negative sampling sample size* is the number of documents used to update the weights in a "negative" direction during negative sampling, as explained in section 2.2.1. Furthermore, Dropout (Srivastava et al., 2014) was used with a rate of 0.3 to further improve results. And last but not least, the adam values refer to the hyperparameters used for the adam optimizer (Kingma and Ba, 2014).

4.2.2 Shuffling Context Documents

Shuffling the order in which the network sees context documents in each run turned out to be very important. Without shuffling context documents before each iteration, the model dropped from a 0.99% accuracy on a 5000 document sample to a 22.46% accuracy on the same sample when it was presented context documents in a different order than that which it learned. This strongly suggests the model was memorizing the order in which it was presented context documents. Once context documents were shuffled before each iteration, the net stopped memorizing, and scored a 0.99% accuracy on the training set, regardless of the order the context documents were in.

4.2.3 Training on Subsets

When training on subsets of the full Wikipedia corpus a central problem arises: Not all context documents from the full Wikipedia corpus are contained in the random subsample. In fact very few data points contain all context documents. It is also not possible to add the context documents to the subsample without using the full Wikipedia corpus for the following reason: If I train on a sample of 5000 documents, then there are around $5000 * 10 = 50,000$ context documents, as explained above. If these 50,000 documents were then added to the sample, I would need ten documents again for each new document added, resulting in $500,000$ new documents to add, again. This will continue until the entire Wikipedia set has been added to the sample.

To mitigate this problem, I replaced missing context documents with duplicates of existing ones. This means that for all tests presented in this thesis, I was unable to exploit the full potential of context documents as I never trained on the full Wikipedia corpus. It is possible that results improve significantly when trained using all context documents.

4.3 Performance of Design Decisions

Now I will delve into the impact certain design decisions have on the model as a whole, give an intuition for what is happening wherever relevant, and back up my intuition with empirical data that I gathered while testing the model.

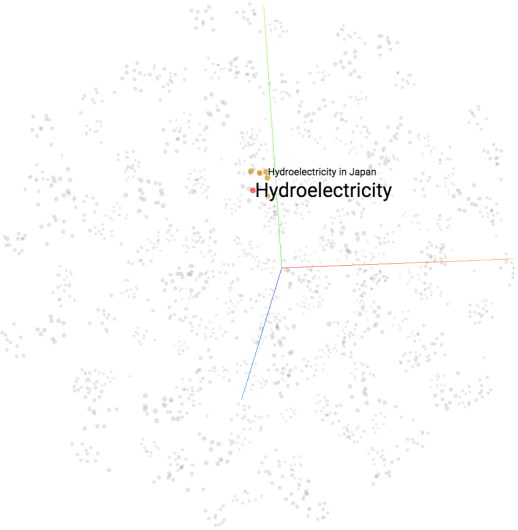
4.3.1 Document Embeddings

As I briefly mentioned in section 2.2.1, when training word2doc, document embeddings are automatically trained alongside. The context documents word2doc uses are turned into embeddings, and this can be visualized. It serves as a good illustration for what word2doc is learning. Figure 4.1 is a 3-D representation of 2000 document embeddings created using t-SNE (van der Maaten and Hinton, 2008), a visualization algorithm for high dimensional data. They originate from sample training data consisting of 200 documents. Each document has ten context documents, and since there is almost no overlap between context documents as it is extremely unlikely all context documents occur in a random 200 sized sample of Wikipedia, we can multiply 200 by ten, and get 2000.

As can be seen from the visualization, clear clusters form. Each point represents a document, and each cluster of documents shares similar context. Highlighted are the five documents closest to "Hydroelectricity" in the original space, measured by cosine similarity as explained in section 2.1.1. It is apparent that the top five documents all have something to do with hydroelectricity (Akimoto Lake is used for the generation of hydroelectricity).

It should be noted, that these documents are only chosen among 2000 documents, not the full Wikipedia corpus. This means the results are limited by the number of documents available. Unfortunately, when already as many as 50,000 documents are visualized, clear clusters disappear, and instead only a huge sphere remains. However, when scaling from 2000 to 50,000 documents, the performance is not impaired, both models reaching a training accuracy of 99.9%, and the later reaching a testing accuracy of 69.3%. Since visualizing embeddings for 50,000 documents is already a problem, visualizing the full Wikipedia dataset will probably not work very well either.

This visualization shows that word2doc is definitely learning something. The document embeddings are formed correctly and encapsulate semantic meaning of each document. Occasionally clusters form that



Nearest points in the original space	cosine similarity
Hydroelectricity in Japan	0.258
Hydroelectric power in the United States	0.242
Akimoto Lake	0.236
Renewable energy commercialization	0.221
World energy consumption	0.216

Figure 4.1: A visualization of 2000 document embeddings using t-SNE (van der Maaten and Hinton, 2008) trained on 200 data points, along with the five closest points to "Hydroelectricity".

are only very loosely connected through context, and sometimes have nothing to do with each other, but in general clusters that form share similar contexts.

t-SNE Visualization

There are a few points to make about t-SNE (van der Maaten and Hinton, 2008), the algorithm used to visualize the document embeddings, because it can be misleading. t-SNE performs a dimensionality reduction for modeling high-dimensional data faithfully in two or three dimensions, similar to PCA. However, unlike PCA, it is non-linear and adapts to the underlying data, performing different transformations on different regions. It has two hyper parameters, the learning rate (or "epsilon") and perplexity.

As the sklearn documentation puts it [3]: "The learning rate for t-SNE is usually in the range [10.0, 1000.0]. If the learning rate is too high, the data may look like a 'ball' with any point approximately equidistant from its nearest neighbors. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers." To create the graph above, I used a learning rate of 10 and a perplexity of 15.

The perplexity is a bit more tricky. It is a guide on how to balance attention between local and global aspects of the data. In a way it guesses how many close neighbors each point has. The authors of the paper say, "The performance of t-SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50." (van der Maaten and Hinton, 2008) However, the reality is more complex. Changes in perplexity can have huge effects on the visualization. Random data can, with the right hyperparameters, be shown to have certain clusters, which is impossible since it is random. Visualizations can differ from across different runs and cluster size is not always meaningful. Overall results offered by t-SNE are not straight forward to interpret.

[3] `http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`

To make sure the visualization from Figure 4.1 is somewhat accurate, I ran different versions, each for 5000 iterations as it was suggested in the guide presented by TensorFlow[4]. Fixing the perplexity at the default (12) and testing learning rates of 2, 10, 50 and 100, I noticed that at a learning rate of ten the model reached a point of stability. To determine perplexity, I fixed the learning rate at ten and tested multiple perplexities, starting at five and incrementing perplexity by five at every step. From 15 and on, I did not see a big change in the visualization, and thus I stuck with a learning rate of 10 and a perplexity of 15.

4.3.2 A Deeper Look

In order to get a better grasp of the impact certain design decisions have on the final model, I built different versions of the model, each version testing one design decision. More specifically, I tested the impact of averaging context documents and query versus concatenating them. I also tested the impact of the data augmentation scheme from section 3.1.1 and finally I tested the impact that the use of context documents has on the final results.

Word2doc as it has been presented thus far, that is with averaging, serves as a baseline for the tests to come. Each model was trained on a subset of 5000[5] randomly chosen documents out of the entire Wikipedia corpus, and evaluated on W2D-VD-200. All hyper parameters are the same as the ones presented in Table 4.1, except for the number of epochs and batch size which change on each run according to the table in Figure 4.2

Baseline All runs are compared with the baseline. For each run, every feature is fixed, except for the one that is being tested. For example, run no. 2 makes use of concatenation instead of averaging, but still uses data augmentation. Run no. 3 does not make use of data augmentation, but does use averaging, etc.

W2D with concatenation In this run I compare concatenation with averaging, in terms of which function to apply to all document embeddings along with the reduced query tensor (Figure 2.4).

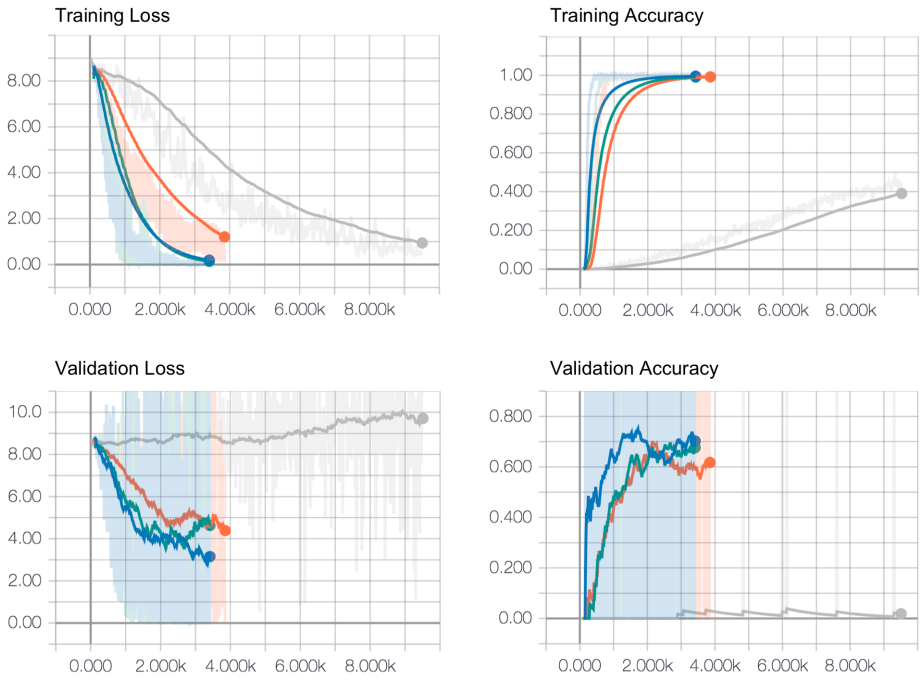
Word2doc using concatenation performs slightly worse compared to the baseline model. However, given that these tests are only performed on 5000 documents, it is not necessarily clear whether this slight underperformance is reproducible on the entire dataset. However, it is clear that the baseline model learns faster than the concatenation model. The baseline's training accuracy as well as its validation accuracy both have steeper slopes than the training and validation accuracies of the concatenation model, and that is mainly why I chose to average over concatenation.

A possible explanation for this speedup could be that averaging results in a much smaller tensor (size 512) compared to concatenation. Therefore there are a lot less weights to be learned, and thus the model trains faster. The question arises of whether the smaller number of weights is still sufficient for a much larger dataset, where the model does not have to choose between 5000 but between 5 million documents. Intuitively I would say yes, because 512^3 is already $134,217,728$, but since weights are real numbers and thus have many more than 3 possible configurations, 512 weights should be more than enough to semantically encode 5 million documents.

W2D without data augmentation In this run, I compare the baseline model that uses data augmentation with the baseline model that does not use data augmentation. Not making use of data augmentation

[4] `https://distill.pub/2016/misread-tsne/`

[5] Testing on a significantly larger data set would have yielded more accurate results, however it would not have been feasible.



Number	Color	Run	Epochs	Batch Size	Training Accuracy	Mean Validation Accuracy*
1		W2D with averaging (baseline)	40	256	0.998	0.693
2		W2D with concatenation	40	256	0.996	0.684
3		W2D without data augmentation	70	128	0.997	0.655
4		W2D without context documents	500	256	0.303	< 0.05

Figure 4.2: Results on four different models

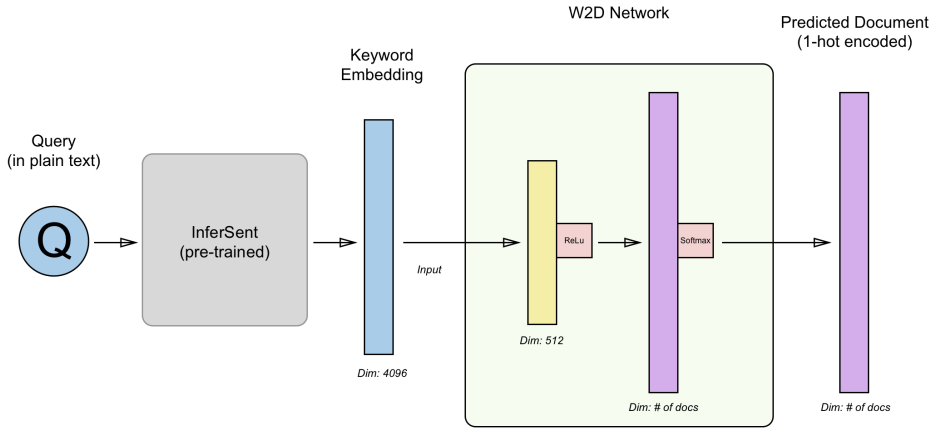


Figure 4.3: W2D Architecture without context documents

means that only the Wikipedia title is taken as a query. Therefore, without data augmentation there are exactly 5000 data points to train on, compared to the usual 24,488 data points that are obtained through data augmentation. In order to compensate, I increased the number of epochs and reduced the batch size. Given these changes, word2doc without data augmentation performed slightly worse. It is difficult to say to what degree this effect is reproducible on the full data set, and how much influence the change in hyper-parameters had but the effect does seem to be only small.

It is also possible that the full benefit of data augmentation only becomes apparent with very large corpora, as that is where documents and queries can intertwine the most, leading to better document embeddings. Perhaps with 5000 documents the data augmentation scheme does not create enough useful connections between queries and documents to better reflect the semantic meaning of a query or document, which then would only have little effect on the document embeddings.

W2D without context documents Now I test the impact that document embeddings as a whole have on the model. To do so, I removed the document retriever and everything to with it from word2doc, similar to PV-DM and PV-DBOW in paragraph vectors (Le and Mikolov, 2014). What is left behind can be seen in Figure 4.3

However, given this change, the model learns much slower. In fact after 500 epochs, the adapted model only reached a training accuracy of about 30%, and a validation accuracy of less than 5%. In fact the model did not seem to learn at all. This implies that the network is very reliant on the context documents. Either it memorizes the context documents despite dropout and fixed document embeddings, or it is able to generalize using the context documents. However, the fact that the other models achieve validation accuracies of around 68%, and the fact that the embeddings are clearly learned as can be seen from Figure 4.1 suggest that the model does in fact generalize, at least to a certain degree.

5 Results

In this section I present the results obtained while training word2doc repeatedly on different, random samples consisting of 1% of the full Wikipedia corpus (58,170 documents). Despite serious effort I was unable to train on a GPU and the largest subset I could train on a CPU in a feasible time was about 1%. Thus, I created ten random samples consisting of 1% of the full dataset and evaluated each. I now present my results averaged across all ten different samples. These results are compared to Facebook's document retriever from DrQA (Chen et al., 2017). It should be noted that Facebook's document retriever has access to the full Wikipedia dataset while word2doc only has access to 1% of the Wikipedia corpus. Keep that in mind while reviewing these results.

As already mentioned in section 4.1, I use accuracy and mean average precision (MAP) to compare Facebook's document retriever to word2doc. I compare both systems using all components (Full, Normal, Numbers, Typos, Abbreviations from Figure 3.1) of the W2D-VD-400 evaluation dataset.

5.1 Performance measured through Accuracy

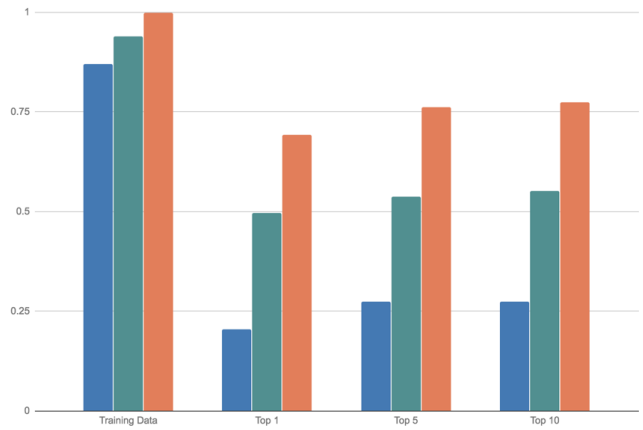
As mentioned above, all accuracies calculated for the 1% dataset are averaged over ten different 1% samples. This should alleviate the randomness factor to a certain degree, and give more reliable data than just a single 1% sample. All accuracies in this section are computed using the full W2D-VD-400 test-set as evaluation data.

In order to measure accuracy more effectively, I chose four different testing criteria. First, I compare the performance of both systems on the training set. This gives an indication of how well word2doc was trained and the potential of the document retriever. Then I compare accuracies across only the single best document, the top five and the top ten retrieved documents. This means if the target document is contained in the top one, top five or top ten returned documents, then the test case counts as a success, otherwise it counts as a fail.

As can been seen in Figure 5.1, word2doc with 1% of training data reached a training accuracy of 0.94%. Because all smaller samples that I trained eventually always reached a training accuracy of 0.999%, it suggests that the model is not fully trained after 30 iterations. That is why in this case I also included accuracies for much smaller training sample, consisting of only 5000 randomly chosen documents (the baseline from Figure 4.2). The smaller dataset possibly hints at the full potential if the 1% dataset had been trained to a training accuracy of close to 100%.

Facebook's document retriever performed at around 87% accuracy when it received the title of the target Wikipedia article (*Training Data* in Figure 5.1). This is somewhat consistent with the results offered in Chen et al. (2017), where the document retriever scores accuracies between 70% and 86% on various test sets. Word2doc performed at 0.94% and 0.998% accuracy under the same conditions, but that amounts to the conditions under which word2doc was trained. Thus, it received its own training data, and therefore, these high results are not surprising.

However, when it comes to the W2D-VD-400 dataset, I was not able to replicate the document retriever's



Color		Training Data	Top 1	Top 5	Top 10
	Document Retriever	0.873	0.205	0.273	0.273
	W2D with 1%	0.946	0.497	0.538	0.552
	W2D with 5000 Documents	0.998	0.693	0.761	0.775

Figure 5.1: Facebook's document retriever vs. Word2doc Performance [Accuracies] with the full W2D-VD-400 used as evaluation data.

high accuracies. Scoring accuracies between 20% and 30%, the document retriever might not have been able to deal with the more general and semantic based queries of the W2D-VD-400 dataset very well. While evaluating the MAP scores, I noticed that the document retriever was often able to retrieve documents from the general ballpark of the query. However, it was rarely able to nail the query on the head or retrieve documents that were truly relevant.

Word2doc performed much better on the same tasks, scoring around 50% to 55% in the case of the models trained on 1% of the data, and around 70% to 77% in the case of the model trained on 5000 documents. Both models performed significantly better, but the model trained on the 1% did perform worse compared to the 5000 document model. This might be because the model did not finish training, only achieving a training accuracy of 0.94%. It is possible that with more epochs these results also improve. In fact, it is probably likely that they do, because it is a more difficult task choosing one document out of 58,170 than out of only 5000. Of course due to the nature of the more difficult task, it could also be that the network is too small, or that a significantly better accuracy is not achievable with this setup. However, overall word2doc seems to perform at an acceptable level and better than Facebook's document retriever when it comes to accuracy.

It might be strange to see training accuracies of close to one. Often times in machine learning tasks achieving a training accuracy of almost one is a strong sign that the model is overfitting. I explain in section 3.1.1 that word2doc should be overfitting, and this is indeed what is happening.

5.1.1 Accuracies on Obfuscated Data

In order to better understand how robust word2doc is towards manipulated queries, I split up the W2D-VD-400 dataset into its components as shown in Figure 3.1. More specifically, I split them up into the

typos, abbreviations, numbers and the normal set (the set without typos, numbers and abbreviations).

It's important to note that when calculating the accuracies on the sub-components, they are often very small. In fact, the typos component has a size of 37, the abbreviations component a size of 46, the numbers component a size of 22 and thus the remaining "normal" component has a size of 295. That means that these calculated accuracies are a bit subjective and should not necessarily be taken for face value, but they should indicate a trend if one exists.

Accuracies are evaluated across the top one, the top five and the top ten retrieved documents, just like in the section above. In the case of word2doc, the accuracies are again averaged over ten random samples consisting each of 1% of the Wikipedia corpus. The results can be seen in Figure 5.2.

Word2doc performs significantly better than Facebook's document retriever when it comes to pure accuracies. On all components, word2doc outscores the document retriever, most noticeably when it comes to dealing with typos and turning numbers into their written form and vice-versa. This trend is persistent across the top one, top five and top ten graphs. In fact there does not seem to be much of a difference in the shapes of the three graphs. While the overall accuracies increase as the number of retrieved documents increases, the ratio between accuracies seems to largely stay the same. It is interesting to note that accuracies between the top five and the top ten retrieved documents do not change at all when it comes to the document retriever.

It is not surprising to see, that the "normal" component scored the highest (except for the "numbers" component) as the other components perform worse and thus negatively impact the entire dataset. However, it is strange that the numbers component scored so high. I explain this largely through the fact that there are only 22 data points in the numbers components, simply too few to report a representative result. Perhaps there were also enough relevant words in the query that helped word2doc find the correct document. For example, if the query was "King Henry the fifth", perhaps it had enough information in "King Henry" that it could identify the correct King Henry (assuming the previous four kings are not in the dataset).

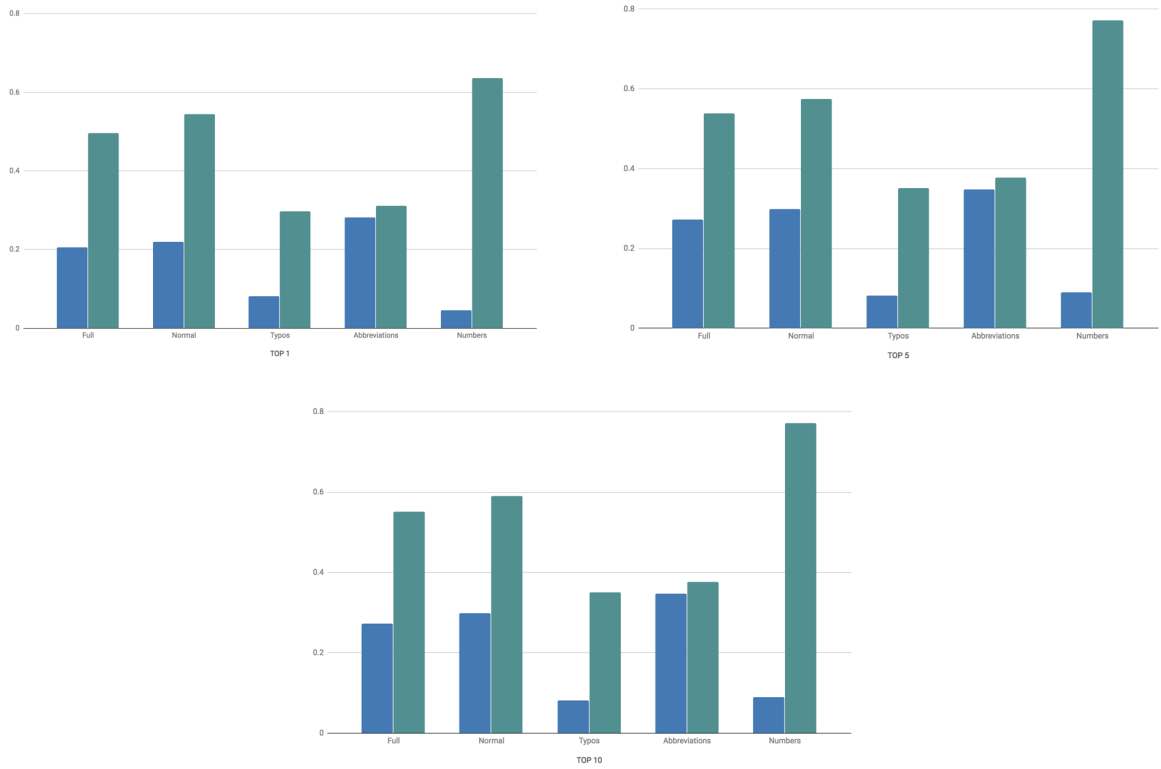
Overall, word2doc seems to be decently robust towards more complicated queries. It has the most difficulty with typos, which is not surprising. However, it still has less difficulty with typos than the document retriever. It is surprising to see that the document retriever performs better on abbreviations than on the complete test set or even just the "normal" component.

5.2 Performance measured through MAP

Word2doc and Facebook's document retriever can both return a list of ranked retrieval results. To measure the quality of these results, I use mean average precision on the top five retrieved documents, as explained in section 4.1.1. I only calculated MAP scores on one of the ten 1% samples.[1] The results presented in Figure 5.3 are thus not averaged across multiple samples but are only based on a single sample. All MAP scores are computed using the W2D-VD-400 test-set as evaluation data. Furthermore, to better understand the MAP score of the full W2D-VD-400 dataset, I also calculated MAP scores on all of its sub-components just like in the section above.

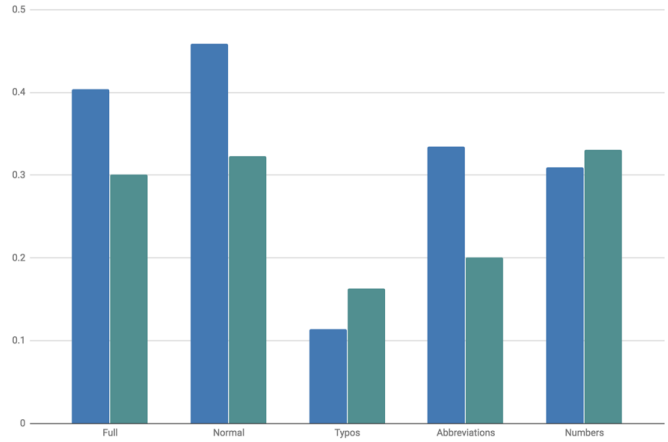
Looking at Figure 5.3, it becomes apparent that the document retriever performs better than word2doc. While creating the MAP scores, I noticed that word2doc only rarely returned other documents within the retrieved set of documents that were also relevant, assuming it did find the target document. In fact word2doc is very bad at predicting five or ten documents that are all relevant.

[1] Deciding by hand if 2000 documents are relevant to a query is a lot of work, and I did not want to evaluate 20,000 documents by hand.



Color	Top 1	Full	Normal	Typos	Abbreviations	Numbers
	Document Retriever	0.205	0.22	0.081	0.282	0.045
	W2D with 1%	0.497	0.544	0.297	0.311	0.636
	Top 5					
	Document Retriever	0.273	0.298	0.081	0.348	0.09
	W2D with 1%	0.538	0.574	0.351	0.377	0.772
	Top 10					
	Document Retriever	0.273	0.298	0.081	0.348	0.09
	W2D with 1%	0.552	0.591	0.351	0.377	0.772

Figure 5.2: Facebook's document retriever vs. Word2doc Performance [Accuracy] using all components of the W2D-VD-400 evaluation data. Accuracies are evaluated on the top one, top five and top ten retrieved documents.



Color		Full	Normal	Typos	Abbreviations	Numbers
	Document Retriever	0.404	0.459	0.114	0.335	0.31
	W2D with 1%	0.301	0.323	0.163	0.201	0.331

Figure 5.3: Facebook's document retriever vs. Word2doc Performance [MAP] with the full W2D-VD-400 used as evaluation data on the top five retrieved documents.

The document retriever performed much better in general. However, it severely lacks understanding of context. This makes sense since it is purely based on statistics. The few cases in which word2doc did return relevant documents (besides the target) were cases in which the other documents contextually and semantically made sense. This suggests that word2doc is able to group document's together based on their semantical similarity, at least to a certain degree. The fact that word2doc can only choose from 1% of all Wikipedia documents, whereas Facebook's document retriever can retrieve documents from the entire English Wikipedia corpus probably had a large impact on performance. My guess is that many documents that the document retriever found could not be found by word2doc because they were simply not included in the training data. Furthermore, word2doc trains on accuracy, not MAP scores. Thus, it is also possible that word2doc optimizes only to find the target document and not to yield logits which reflect a ranking in the documents. Hence, it might help to optimize for another metric than accuracy. MAP is not an option however, unless word2doc is trained on a dataset for which there is a golden standard so that MAP scores can be calculated automatically.

It should also be noted that word2doc's MAP scores were significantly improved by its base accuracies. Word2doc's accuracies are a lot higher compared to those of the document retriever. While the document retriever gets its MAP score from more or less all documents in the ranking, word2doc mostly gets its score by predicting the first document correctly much more often than the document retriever. Since higher ranked documents are weighted more heavily than lower ranked documents, predicting the first document correctly makes a big difference in the final MAP score.

Looking at the impact of individual components that make up W2D-VD-400, it is not surprising to see that the typos, abbreviations and numbers had a negative impact on the overall score. The test data component without any obfuscation (the "normal" component) performed better than the entire dataset combined. Except for the typos component, the individual components did not perform that much worse, suggesting that both systems can yield results that are somewhat relevant despite data obfuscation.

Furthermore, it is interesting that the spread in the graphs for word2doc's scores is less than that of the document retriever. This suggests that word2doc is overall more robust towards obfuscated queries than the document retriever, which is good.

It is also interesting to compare the accuracies on the top five documents retrieved from Figure 5.2 and compare them to their MAP scores. Word2doc's graphs have a similar shape, but those of the document retriever do not. The numbers component has a relatively high MAP score, but a very low accuracy. This suggests that the document retriever is able to retrieve documents that are relevant to the query, but it is unable to retrieve the target document.

One should not forget that in order to get reliable MAP scores, large and diverse test sets are required. That is hardly the case for the typo component (size 37), abbreviation component (size 46) and number component (size 22). Thus, these calculated MAP scores are subjective to a certain degree and should be treated with care.

6 Conclusion

In this thesis I present an information retrieval system called word2doc that makes use of document embeddings as well as query embeddings and a frequency based document retriever to retrieve documents. While each of these approaches have been explored individually, word2doc is to my knowledge the first system to combine these different methods and, in doing so, intrinsically train document embeddings to retrieve documents. With word2doc I try to build an information retrieval system that is able to gain a semantic understanding of documents, and thus retrieve documents not based on pure statistics but based on semantic meaning instead. Word2doc should understand the content of the query and the document, as far as that is possible.

The results show that word2doc's approach has potential, outscoring Facebook's document retriever from *DrQA* (Chen et al., 2017) when looking at accuracies. The authors of Chen et al. (2017) claim that their document retriever outscores the Wikipedia search, thus word2doc should do so as well in terms of accuracies. However, word2doc is unable to retrieve a ranked list of relevant documents pertaining to a query (see section 5.2), and thus it is unfit to work as a full document retrieval system at the present time. There is an easy explanation to this lack of performance however. Word2doc was only trained on 1% of the entire Wikipedia corpus, whereas the document retriever had the entire Wikipedia corpus at its disposal. I imagine performance would significantly improve if word2doc is trained on the entire Wikipedia corpus as well. While this fact has, without doubt, a big impact on performance, another issue could be that word2doc was built to optimize accuracy, and accuracy does not take into account the ranking within a set of retrieved results. Better results could likely be achieved, if word2doc optimizes on a metric that takes the ranking of the top ten documents into account instead of just the top one.

It should also be noted that information retrieval is a very subjective task. Different people have different opinions on which retrieved documents count as relevant. In fact, they might not even know which documents they are looking for, given their query. Word2doc was exclusively trained on queries that are based on Wikipedia titles, which could be interpreted as a weakness. It is possible that queries arise that are very far from any seen Wikipedia titles. However, to counter that (and for other reasons), I performed data augmentation as presented in section 3.1.1, which exposes the network to a range of different queries.

In conclusion, the approach taken by word2doc seems promising. The neural network was able to take the information it was given and retrieve the correctly labeled document to a certain degree. Additionally, the network was able to train document embeddings, showing that the network is generalizing and grouping documents by their semantic relevance. With a little bit of tweaking, I am confident that competitive results can be achieved.

6.1 Future Work

In a first step, future work should focus on training and evaluating word2doc on the entire Wikipedia corpus. All 600 GB of pre-trained data has already been computed, it only needs to be trained. There is a risk that the system does not scale to the entire Wikipedia dataset. In that case, an alternative option could be to train multiple different models, one for each topic in Wikipedia. The Wikipedia corpus could be

split into topic-based models using paragraph vectors (Le and Mikolov, 2014), which has shown success on that task. Using sentence embeddings the topic of the query could then be determined and matched with the relevant word2doc model. Of course, better accuracies are probably obtained when finding a way to use the entire Wikipedia corpus.

Furthermore, a different optimizing metric than accuracy should be explored for the reasons mentioned above. Another angle of attack could be to look into a different negative sampling technique proposed by Ai et al. (2016) specifically for paragraph vectors (Le and Mikolov, 2014) in IR systems. Unfortunately, I discovered their work only after I had completed mine, and was thus unable to incorporate it into my own.

Word2doc has three main components, the document retriever, InferSent and the neural network. Simplifying this pipeline and combining all of these components into one could be another source of improvement.

Bibliography

D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading Wikipedia to Answer Open-Domain Questions. *ArXiv e-prints*, March 2017. 5, 2, 3, 15, 17, 32, 38

Markus Müller, Thai Son Nguyen, Jan Niehues, Eunah Cho, Bastian Krüger, Thanh-Le Ha, Kevin Kilgour, Matthias Sperber, Mohammed Mediani, Sebastian Stüker, and Alexander H. Waibel. Lecture translator - speech translation framework for simultaneous lecture translation. In *HLT-NAACL Demos*, 2016. 2

Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2015. ISBN 1449358543, 9781449358549. 3

Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 275–281, New York, NY, USA, 1998. ACM. ISBN 1-58113-015-5. doi: 10.1145/290941.291008. URL http://doi.acm.org/10.1145/290941.291008. 3

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL http://dx.doi.org/10.1561/1500000019. 3

Stephen E. Robertson and Karen Sparck Jones. Document retrieval systems. chapter Relevance Weighting of Search Terms, pages 143–160. Taylor Graham Publishing, London, UK, UK, 1988. ISBN 0-947568-21-2. URL http://dl.acm.org/citation.cfm?id=106765.106783. 3

Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 472–479, New York, NY, USA, 2005. ACM. ISBN 1-59593-034-5. doi: 10.1145/1076034.1076115. URL http://doi.acm.org/10.1145/1076034.1076115. 3

Victor Lavrenko and W. Bruce Croft. Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 120–127, New York, NY, USA, 2001. ACM. ISBN 1-58113-331-6. doi: 10.1145/383952.383972. URL http://doi.acm.org/10.1145/383952.383972. 3, 4

Victor Lavrenko. *A Generative Theory of Relevance*. PhD thesis, 2004. AAI3152722. 3

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. 3

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM. ISBN 1-58113-096-1. doi: 10.1145/312624.312649. URL http://doi.acm.org/10.1145/312624.312649. 3

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944937. 3

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, January 2013a. 3, 4, 6, 19

Q. V. Le and T. Mikolov. Distributed Representations of Sentences and Documents. *ArXiv e-prints*, May 2014. 3, 9, 10, 11, 15, 31, 39

Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, and Narayan Bhamidipati. Context- and content-aware embeddings for query rewriting in sponsored search. In *SIGIR*, 2015. 4

Dwaipayan Roy, Debjyoti Paul, Mandar Mitra, and Utpal Garain. Using word embeddings for automatic query expansion. *CoRR*, abs/1606.07608, 2016a. URL `http://arxiv.org/abs/1606.07608`. 4

Saar Kuzi, Anna Shtok, and Oren Kurland. Query expansion using word embeddings. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1929–1932, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983876. URL `http://doi.acm.org/10.1145/2983323.2983876`. 4

Guoqing Zheng and Jamie Callan. Learning to reweight terms with distributed representations. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 575–584, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3621-5. doi: 10.1145/2766462.2767700. URL `http://doi.acm.org/10.1145/2766462.2767700`. 4

Hamed Zamani and W. Bruce Croft. Embedding-based query language models. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 147–156, New York, NY, USA, 2016a. ACM. ISBN 978-1-4503-4497-5. doi: 10.1145/2970398.2970405. URL `http://doi.acm.org/10.1145/2970398.2970405`. 4

Dwaipayan Roy, Debasis Ganguly, Mandar Mitra, and Gareth J.F. Jones. Word vector compositionality based relevance feedback using kernel density estimation. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1281–1290, New York, NY, USA, 2016b. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983750. URL `http://doi.acm.org/10.1145/2983323.2983750`. 4

Hamed Zamani and W. Bruce Croft. Estimating embedding vectors for queries. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 123–132, New York, NY, USA, 2016b. ACM. ISBN 978-1-4503-4497-5. doi: 10.1145/2970398.2970403. URL `http://doi.acm.org/10.1145/2970398.2970403`. 4

Fernando Diaz, Bhaskar Mitra, and Nick Craswell. Query expansion with locally-trained word embeddings. *CoRR*, abs/1605.07891, 2016. URL `http://arxiv.org/abs/1605.07891`. 4, 22

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`. 4, 22

Stéphane Clinchant and Florent Perronnin. Aggregating continuous word embeddings for information retrieval. 2013. 4

Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth J.F. Jones. Word embedding based generalized language model for information retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 795–798, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3621-5. doi: 10.1145/2766462.2767780. URL `http://doi.acm.org/10.1145/2766462.2767780`. 4

Guido Zuccon, Bevan Koopman, Peter Bruza, and Leif Azzopardi. Integrating and evaluating neural word embeddings in information retrieval. In *Proceedings of the 20th Australasian Document Computing Symposium*, ADCS '15, pages 12:1–12:8, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-4040-3. doi: 10.1145/2838931.2838936. URL http://doi.acm.org/10.1145/2838931.2838936. 4

Bhaskar Mitra, Eric T. Nalisnick, Nick Craswell, and Rich Caruana. A dual embedding space model for document ranking. *CoRR*, abs/1602.01137, 2016. URL http://arxiv.org/abs/1602.01137. 4

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. ACM International Conference on Information and Knowledge Management (CIKM), October 2013. 4

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. CIKM, November 2014. 4

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. *CoRR*, abs/1711.08611, 2017. URL http://arxiv.org/abs/1711.08611. 4

Bhaskar Mitra and Nick Craswell. Neural models for information retrieval. *CoRR*, abs/1705.01509, 2017. URL http://arxiv.org/abs/1705.01509. 4

A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *ArXiv e-prints*, May 2017. 5, 9, 11, 13, 14, 15, 16, 22

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*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL http://doi.acm.org/10.1145/1390156.1390177. 6

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. *ArXiv e-prints*, October 2013b. 6, 13

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-1162. URL http://www.aclweb.org/anthology/P15-1162. 8

R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-Thought Vectors. *ArXiv e-prints*, June 2015. 8, 9

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 801–809, USA, 2011. Curran Associates Inc. ISBN 978-1-61839-599-3. URL http://dl.acm.org/citation.cfm?id=2986459.2986549. 8

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014. URL http://arxiv.org/abs/1404.2188. 8

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015. URL http://arxiv.org/abs/1503.00075. 8

Yashen Wang, Heyan Huang, Chong Feng, Qiang Zhou, Jiahui Gu, and Xiong Gao. Cse: Conceptual sentence embeddings based on attention model. In *ACL*, 2016. 8

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2017. 8

J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Towards Universal Paraphrastic Sentence Embeddings. *ArXiv e-prints*, November 2015. 8

Adriaan M. J. Schakel and Benjamin J. Wilson. Measuring word significance using distributed representations of words. *CoRR*, abs/1508.02297, 2015. URL http://arxiv.org/abs/1508.02297. 10

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015. URL http://arxiv.org/abs/1508.05326. 13

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011. URL http://arxiv.org/abs/1103.0398. 15

Kilian Q. Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alexander J. Smola. Feature hashing for large scale multitask learning. *CoRR*, abs/0902.2206, 2009. URL http://arxiv.org/abs/0902.2206. 17

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. pages 1 – 20, 03 2010. 18

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715. 24, 25

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313. 26

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980. 26

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html. 27, 28

Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. Improving language estimation with the paragraph vector model for ad-hoc retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 869–872, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4069-4. doi: 10.1145/2911451.2914688. URL http://doi.acm.org/10.1145/2911451.2914688. 39