UNIVERSITY OF FREIBURG

BACHELOR'S THESIS

A review of word embedding and document similarity algorithms applied to academic text

Author:
Jon Ezeiza Alvarez

Supervisor:
Prof. Dr. Hannah Bast

A thesis submitted in fulfillment of the requirements for the degree of Computer Science

in the

October 22, 2017

Declaration of Authorship

I, Jon Ezeiza Alvarez, declare that this thesis titled, "A review of word embedding and document similarity algorithms applied to academic text" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

University of Freiburg

Abstract

Technical Faculty

Department of Computer Science

Computer Science

A review of word embedding and document similarity algorithms applied to academic text

by Jon Ezeiza Alvarez

Thanks to the digitalization of academic literature and an increase in science funding, the speed of scholarly publications has been rapidly growing during the last decade. This is obviously very positive, but it leads to a few logistical problems we have been tackling lately. Article availability is, for the most part, a solved problem, a scientist can get access to virtually any publication on demand. There's also a very comprehensive tool-set for retrieval of relevant articles by now and the last couple of years have seen a further surge, with a new wave of AI based retrieval tools.

However, the latest global surveys about reading patterns suggest that it is now harder than ever for a professional scientist to keep up. There is no human endeavor that is better publicly documented than scientific progress, yet, we are starting to loose the grasp of it. It is no longer viable for a single person to keep a global view of science, not detailed enough or updated enough to be useful at least. Conversely, recent advancements in natural language processing (NLP) and high volume distributed computation are starting to open the door to an alternative. We propose the long term goal of creating an automatic toolbox for indexing, understanding and interpreting all scientific literature. This work is intended to be a small early step in that direction.

Word embeddings, which facilitate working with abstract semantic concepts in numerical form, have become the foundation of modern NLP. Here we perform a thorough review of the most pivotal word embedding algorithms and perform an empirical evaluation on academic text to identify the best alternatives for the described use case. We also extend the concept of word embeddings to documents. We perform a similar review of recent document modeling algorithms and evaluate them on titles, abstracts and full articles.

University of Freiburg

Abstract

Technical Faculty

Department of Computer Science

Computer Science

A review of word embedding and document similarity algorithms applied to academic text

by Jon Ezeiza Alvarez

Literatura zientifikoaren digitalizazioari esker eta zientziaren esparruan egin diren inbertsioen ondorioz, nabarmen areagotu da argitalpen akademikoen kopurua azken hamarkadan. Oso datu baikorra da hori, noski, baina ekoizpenen ugaritzearekin batera zenbait erronka sortu dira. Artikuluak erabiltzaileen esku egon daitezen bermatzea da erronka horietako bat. Oro har esan liteke erantzun egokiak eman zaizkiola eskakizun horri, eta eskuarki zientzialariek badutela aukera beren intereseko argitalpenak eskuratzeko aparteko arazorik gabe.

Alabaina, irakurketa joeren inguruko mundu mailako azken txostenek adierazitakoaren arabera, gaur egun inoiz baino zailago dute zientzialariek haien jakintza-arloan egunez egun zabaltzen diren berriak eta berrikuntzak jarraitzea. Seguruenez, ez dago giza jardueraren alorrik zientziarena bezain ondo dokumentatuta dagoena; baina, paradoxikoki, horrek berak zailtzen du norberarentzat esangarri izan daitekeen informazioa atzitzea. Izan ere, gaur egun oso zaila da alor askotan zientziaren esparruan egiten denaren ikuspegi orokorra izatea; are zailago, jakina, informazio xehea eskuratzea. Zorionez, Hiztuntza Prozesamenduaren alorrean (HP) hainbat aurrerapen egin dira eta hasiak dira arazoari nolabaiteko erantzuna emateko bideak zabaltzen. Ildo horretatik, lan honetan proiektu bat aurkezten da, helburu duena erreminta-kutxa bat garatzeko literatura zientifikoa automatikoki indexatu, ulertu eta interpretatzea. Proiektua epe luzeko jomugan aurreikusten da, eta lan honetan urrats bat egin nahi izan da norabide horretan.

Jakina denez, Word embedding da HP garaikidearen oinarri sendoenetako bat gaur egun. Horri esker, kontzeptu semantiko abstraktuak zenbakizko errepresentazioen bidez adierazi ahal dira. Helburu horrekin hainbat algoritmo garatu dira. Hain zuzen ere, lan honetan word embedding algoritmo garrantzitsuenek testu akademikoak eskuratzeko zereginetarako eskaintzen dituzten emaitza enpirikoen balioespena egin da. Era berean, word embeddings kontzeptua dokumentuen eremura zabaltzen da, eta antzeko berrikuste-lana egin da aztertuta dokumentuak modelizatzen dituzten algoritmoek nolako emaitzak ematen dituzten algoritmo horiek aplikatzen direnean artikulu zientifikoen tituluen gainean, artikuluen laburpenen gainean eta artikuluen testu-gorputz osoaren gainean.

Acknowledgements

I'd like to start by thanking the University of the Basque Country (Euskal Herriko Unibertsitatea, EHU/UPV) for giving me the foundations I will be building on for the rest of my Computer Science career. I also thank EHU/UPV and Gipuzkoako Foru Aldundia for facilitating my stay at the University of Freiburg during my last year, where this project was developed. I am, of course, grateful to the University of Freiburg for hosting me and specially to Hannah Bast for supervising my Bachelor's thesis.

Special thanks to the IXA group at EHU/UPV and my supervisors there Eneko Agirre and Aitor Soroa. They gave me the opportunity to spend a few months on word embedding algorithms trained on the Wikipedia graph. This was the project that made me passionate about Natural Language Processing and introduced me to the new world of word embeddings, I am very grateful for that.

Last, but not least, I'd like to thank my partner in crime Mehdi and my friends at SCITODATE for giving me the room to work on this project in the middle of the storm that is building a startup from scratch.

I'll finish by dedicating this work to my sister, I hope she enjoys her university years as much as I have.

Contents

Chapter 1

Introduction

Thanks to the digitalization of academic literature and an increase in science funding, the speed of scholarly publications has been rapidly growing during the last decade. This is obviously very positive, but it leads to a few logistical problems we have been tackling lately.

The problem of availability is largely solved, getting access to an arbitrary article is no longer an issue for a scientist any more in most cases. This is thanks to metadata management standards and a special push from the Open Access movement. PubMed[1], ArXiv[2] and DBLP[3] are some of the leading indexers and aggregators for their respective fields, and act as the primary hosts of Open Access publications. All this coordination would not be possible without standards like OAI-PMH[4] or Dublin Core[5].

There is also a comprehensive set of tools for retrieval of relevant articles. Classic academic search engines like Web of Science[6] or Google Scholar[7] still stand strong and the last couple of years have seen a surge in progress in this front, with the launch of a new generation of retrieval tools based on AI like Semantic Scholar[8] from the Allen Institute.

However, the latest global surveys (Ware and Mabe, 2015) about reading patterns suggest that it is now harder than ever for a professional scientist to keep up. It is not uncommon to be confronted with a few hundred new publications on a daily basis in some fields, such as materials engineering or machine learning. Because of this, it is reported that each article is read only five times on average.

There is no human endeavor that is better publicly documented than scientific progress, yet, we are starting to loose the grasp of it. It is no longer viable for a single person to keep a global view of science, not detailed enough or updated enough to be useful at least. Conversely, recent advancements in Natural Language Processing (NLP) and high volume distributed computation are starting to open the door to an alternative. We propose the long term goal of creating an automatic toolbox for indexing, understanding and interpreting all scientific literature. A solution that goes beyond retrieval and that can dive into the semantic content, to identify overarching trends,

[1] PubMed: https://www.ncbi.nlm.nih.gov/pubmed/
[2] ArXiv: https://arxiv.org/
[3] DBLP: http://dblp.uni-trier.de/
[4] OAI-PMH: https://www.openarchives.org/pmh/
[5] Dublin Core: http://dublincore.org/
[6] Web of Science: https://apps.webofknowledge.com
[7] Google Scholar: https://scholar.google.com
[8] Semantic Scholar: https://www.semanticscholar.org/

understand how progress in science works and how to optimize scientific performance.

This work is intended to be one of the small first steps in this direction. Word embeddings, which facilitate working with abstract semantic concepts in numerical form, have become the foundation of Modern NLP. We have therefore focused on doing a thorough review of the mayor word embedding algorithms. We give an overview of the intuition behind the algorithms, their mathematical core and, most importantly, we do an empirical evaluation on academic text. This last task is especially relevant as most of the focus in the research community has been on general purpose text, such as news or social media activity. Scientific text is notably different, with a much higher density of specialized terminology. We check this assumption and see what algorithms perform best in this context.

It also stands to reason to extend the concept of word embeddings to documents, such as titles, abstract or full articles. We have also endeavored to perform a literature review on semantic document models and, more specifically, on semantic textual similarity (STS) based on word embeddings. Word embedding compositionality is still considered an open problem though and the size and type of documents plays a considerable role. Nevertheless, as with word embedding algorithms, we perform a thorough theoretical and empirical review of the few state-of-the-art algorithms.

1.1 Context

This work has been done under the overarching project of SCITODATE, a startup cofounded by the author. At SCITODATE we set out to build an AI toolbox and a large scale data infrastructure to extract business and research insight from large amount of academic articles. Currently SCITODATE finds customers for highly specialized providers of research equipment, materials and services. This offering is intended to be one of many business intelligence services in the scientific market that will fund our overarching goal of building a digitalized global high resolution view of all scientific progress.

This research work has been done in parallel and does not contain the core work that supports the main customer discovery service. The author declares that he has been the sole contributor to everything described in this document. However, both works are clearly aligned and SCITODATE has influenced some of the priorities of this sub-project. Particularly, it has led to the empirical review being focused on the biomedical field, where SCITODATE has most of its clients. Nevertheless, it also turns out that the biomedical field is the most developed in terms of metadata infrastructure and data availability, so it is the best choice, in any case.

1.2 Scope

This work has been divided in two review sections: one for word embedding algorithms and another one for document similarity algorithms. For each algorithm

we give an intuition of its rationale and inner workings, a summary of the mathematical foundations and an empirical benchmark of predictive and computational performance.

To perform such a review we have created our own training and evaluation datasets for the lack of standard datasets in the scientific domain. The creation of these dataset is also considered a core contribution of this work, so special attention has been given to the process of data collection and cleaning. The datasets are constrained to the biomedical field, where data availability is most developed. We have created a large corpus of 2M full articles and 26M abstracts from PubMed[9]. The word embedding evaluation dataset was created using the UMLS knowledge base[10] as a gold standard. The document similarity evaluation has been done using author linkage, with authenticated ORCID[11] author profiles.

This work only reviews corpus-based word embedding algorithms. There is a healthy ecosystem of word embedding algorithms trained on knowledge bases and there are big and very high quality knowledge bases in science to train on. However, corpus-based algorithms are more popular in the community, because their input is easier to acquire and they have better prediction performance overall. Therefore, knowledge base word embeddings are deemed outside the scope of this project and left as future work.

We also focus only on unsupervised document similarity models. Unsupervised models only rely on the knowledge mining performed by the word embedding algorithms, instead of training an end-to-end similarity estimator. The rational for this is data availability. Article linkage is weak by nature. Here we focus on author linkage, which ensures a similarity signal, but is somewhat loose. This could be improved by using authors in combination with publication dates. References can also be a good source of semantic linkage, although bibliographic references are often bloated and reference information is hard to acquire in any case. It may be possible to collect a big enough training gold standard, specially when centralized author IDs become more prevalent in the following years. However, in this work we deem that going to such lengths is out of the scope and we only use such linkages for evaluation.

Semantic textual similarity (STS) and word embedding compositionality is considered an open problem. It is still early days for STS and the state-of-the-art is fairly sparse compared to the word embedding scene. The STS problem is also fragmented, as not every word collection can be treated equally. The academia now distinguishes phrases, sentences, paragraphs and documents as different instances of text with independent solutions. The focus of this work is in the paragraph and document levels but the state-of-the-art here is fairly narrow. Because of this, it was also deemed interesting to explore algorithms that act at different levels, particularly on sentence similarity, which is a more active subfield. This diversity opens the door to do further experimentation with titles, abstracts and full articles, which gives a more complete picture.

In general, this is not intended to be a full comprehensive listing of the current state-of-the-art. We hand-pick a set of pivotal algorithms from the last five years, and explore their differences. Inspecting all the iterative improvements would be too

[9] PubMed: https://www.ncbi.nlm.nih.gov/pubmed/
[10] UMLS: https://www.nlm.nih.gov/research/umls/
[11] ORCID: https://orcid.org/

much work and its value would be questionable. Instead, we try to focus on algorithms that introduce new ideas and have, to some extent, influenced the community in a significant way. Finally, we have also limited ourselves to algorithms that are already implemented. There is a few interesting solutions only described on paper. Implementing such algorithms would steal too much time from doing a global review, though, and could be very error prone if not given its due attention.

Chapter 2

Foundations

The following chapter briefly covers the foundations needed to understand the rest of the work. We explain the basic classical notions of natural language processing (NLP) in a bottom-up approach, starting at character level and building up to more abstract understanding.

We also give a short primer on Neural Networks (NN), Deep Learning (DL), back-propagation and gradient descent optimization. Deep learning has become the core of modern NLP in the last five years. It has almost completely replaced three decades of rule based and statistical NLP work it terms of language understanding. This work is, in fact, focused on reviewing the lower level components of the DeepNLP ecosystem. Therefore, it is specially relevant to give a briefing on these topics too.

2.1 Preprocessing

Real world text is cumbersome to deal with. Language is highly inconsistent, even in the most formal settings. In this section we cover the standard NLP tool-set that is used to transform text from a raw sequence of characters to a form that is cleaner and easier to deal with computationally.

2.1.1 Tokenization

Tokenization is the task of splitting the text into more meaningful character sequence groups, usually words or sentences. It is almost always the first step in any NLP pipeline. Characters are hard to interpret by a computer on their own, but words mostly self-contained semantic units. It is a comfortable level of abstraction to work at, a lot of NLP operations act directly on words.

A naive approach to tokenization would be simply to split by spaces and remove any punctuation. It is a good baseline, but not ideal. Even for common English there is a number of tricky cases. For example, the use of apostrophe for contractions. Should "aren't" be a single token or two ("are", "n't"), meaning "are not". What about the different cases of hyphenation like "co-occurrence", "Bellman-Ford" or "five-year-old kid". Or composite borrowed names like "Los Angeles" or "in vitro". What about numbers, serial codes, mathematical notation or protein names.

Such corner cases are language and domain specific. It also is unclear how smart a tokenizer should be, if it should detect common phrases or composite words. The treatment of such corner cases is also ambiguous, depending on the desired use-case.

In practice, tokenization is largely considered a closed problem in academia. There is always some unavoidable mess in language, but there is good efficient tokenizers that output relatively clean results. The current approach is fairly pragmatic. NLP libraries like NLTK[1] offer a variety of language specific tokenizers that cover most of the corner cases. Each implementation has different priorities and one can choose the most appropriate one for each use-case. In the case where standard tokenizers are not sufficient, it is not too hard to implement your own.

Most tokenizers are rule based and tuned by humans. It is common to simply use a well designed regular expression. There is also more advanced implementations though. Simple statistical models are not uncommon, especially to take care of more complicated cases, such as proper sentence tokenization in corrupt text.

Our focus is in English, but each writing system has their own challenges. For example, Chinese writing has no spacing in it. There is also a fair amount of ambiguity on what characters should go together as words. In such cases, it is common to use a large word dictionary for tokenization instead.

2.1.2 Normalization

Isolation words is not enough for certain use-cases. Formatting and morphological variations can hinder matching, indexing or similarity tasks. A linguistic model should, for example, be able to identify that "fox" and "foxes" are the same substantive or "have" and "had" are the same verb.

Normalization, in general, is the exercise of removing undesired variation from text, so that slight linguistic differences do not get on the way of matching what are effectively the same concepts. This is usually done by removing or transforming some parts of the word so that only a common root is kept. However, normalization inherently leads to information loss, which is not always desirable.

An obvious first step is to lowercase or uppercase all characters, so that capitalization is not a problem. Removing morphological variation is more tricky though, we distinguish two types of techniques of doing so.

Stemming removes morphological variation by algorithmic means. Most languages have patterns on how they transform words based on morphology. Plurals, for example, seem to be relatively easy to normalize, just remove the trailing "s" or "es". It is true that there is a lot of ambiguity though. For "foxes" the "es" should be removed and for "chocolates" only the "s" should be removed. There is also words Latin words like "corpus" that have a different plural "corpora". These corner cases are not easy to deal with, but a fairly good baselines can be implemented with rule based methods. Granted, they cause significant information loss and the roots may not always be correct. However, such simple implementations are very efficient and scale well to most domains. This is usually the preferred method of normalization for large scale indexing and matching applications.

Lemmatization is an alternative method that uses human curated dictionaries to extract the correct lemmas or roots from known words. A common vocabulary used

[1] NLTK: http://www.nltk.org/

for this task is WordNet[2]. This approach is obviously more accurate than stemming, but such dictionaries can be fairly limited, specially in domain specific text. Lemmatization is also considerably heavier than simple rule based stemming. For maximum accuracy, it is not uncommon to use both methods in conjunction.

2.2 Parsing

After isolating and cleaning words, it is common to add yet another layer of processing to be able to work at higher abstraction levels. Parsing encompasses a set of annotation tasks to identify the role of each word with respect to its context in some text, usually a sentence.

Parsing is usually the core of NLP tasks that require deep understanding. Probabilistic language parsers where one of the big breakthroughs in NLP during the 1990s. Such parsers where one of the main components that enabled high quality machine translation for the first time.

POS tagging or Part-Of-Speech tagging is the computational equivalent of morphological linguistic analysis. It annotates each token with a morphological class, say substantive, adjective or verb. Some taggers also go deeper and specify substantive properties, such as plurality, or verb tenses.

POS tagging is usually implemented using a reference dictionary, a few morphological heuristics and a disambiguation component. Even using a dictionary, words can have ambiguous morphologic types. An example: "she sat on the back seat" and "he was hit on the back". Disambiguation is the most challenging part.

Historically disambiguation has been done probabilistically, by estimating co-occurrence probabilities. More recently, the state-of-the-art in disambiguation has been beaten by models using word embeddings or end-to-end deep models.

Syntactical analysis is the second step of parsing. Much like in linguistic analysis, syntax follows after morphological analysis. The task here is to determine the role of each word in the sentence and capture dependencies between each component.

This is an even harder challenge than POS tagging. Probabilistic methods where a good baseline but high quality results have not been achieved until recently starting to use deep supervised models. However, complete syntactical analysis is not required for many use-cases. For example, probabilistic models that extract subject-verb-object triplets have been effective for a while. Such triples are already extremely useful for most information extraction tasks.

Abstract meaning representation is the unification of POS tagging and syntactic parsing with real world information via Named Entity Recognition and word sense disambiguation, explained further below. It is the closest thing we have to real structured understanding of text. It is still early days for abstract meaning representation though. Even modern models struggle to put all these components together and

[2]WordNet Lemmatizer: http://www.nltk.org/api/nltk.stem.html#module-nltk.stem.wordnet

produce usable results. As a reference, it has been one of the tasks in SemEval only since 2015. Partial solutions do exist, however, as exemplified by the new natural language audio assistants that are starting to come out now to consumer markets.

2.3 Word senses

A sense or concept is the purest form of semantic unit. Words and senses are highly correlated, as there is a one-to-one mapping between both in most cases. However, this is not always the case. Some words have multiple senses (polysemy) and multiple words can mean the same sense (synonymy). Language is highly ambiguous and being able to distinguish and isolate senses is key to achieve complete understanding.

Knowledge graphs are a good resource to serve as reference for word senses. Each node in a knowledge graph represents a sense and the edges define relationships between these concepts. Knowledge graphs, also called knowledge bases, are a way to encode real world information in a structured machine understandable format. Some graphs focus on collecting common knowledge about the world, like Freebase or DBpedia. Other graphs such as WordNet[3] focus more on distinguishing word senses and defining concept hierarchies. An ontology such as UMLS fills both roles.

Word sense disambiguation is the task of annotating each word with their respective sense in a reference vocabulary or knowledge graph. Like POS tagging or syntactic analysis, sense disambiguation is a key part of tasks that require deep understanding, such as machine translation.

Sense disambiguation is performed much like POS tagging, which is also a disambiguation task. Classic probabilistic methods rely on collecting language statistics such as coocurrence probabilities. These prior probabilities are then used to compare a word to its context and predict the most likely sense. The same idea can also be implemented with word embeddings, which inherently contain such probabilities.

Named Entity Recognition or NER is an application of word sense disambiguation. The idea with NER is to annotate text to link words or word sequences to real world concepts. This is very interesting for retrieval and information extraction. With NER, search queries can use real world information instead of basing everything on word indexing. There are two parts to NER. First there is Recognition, matching the vocabulary to the text efficiently and in a robust manner, without being too sensitive to noise. Only once Recognition is done is Disambiguation applied.

Embeddings are numerical representations of semantic units. The most common form embeddings are word embeddings. Word embeddings assume that there is no ambiguity, so they end up capturing multiple senses in the same vector representation, which is not ideal. However, this is a good baseline and sense embeddings

[3]WordNet: https://wordnet.princeton.edu/

are much harder to train anyways. Word embeddings can be efficiently learned just by scanning big corpora. There are sense embedding algorithms that utilize some form of clustering to distinguish senses, either during or after training. In a way, word embeddings implicitly encode the same information as knowledge graphs. In fact, there is a healthy ecosystem of word and sense embedding algorithms that use knowledge bases as input data.

Semantic embeddings are the focus of this work. They will be explained more thoroughly in section 2.5 and thought the literature review.

2.4 Vector Space Model

The Vector Space Model (VSM) is a model for representing documents in algebraic form. It is a widely accepted standard that is used whenever a numerical representation of text is needed. The basic idea of the VSM is to represent text as a Bag of Words (BoW). In order to have a compact representation, the ordering of the words in a document is ignored and the document is represented by a vector of word frequencies. More formally, a vocabulary is established where each w_i word or term has a unique integer index i. A document d_j is represented by a column vector v_j where each element v_{ij} stores the $tf(i,j)$ frequency of word w_i in document d_j.

For the standard VSM model, BoW is generalized such that each v_{ij} value does not necessarily show the exact term frequency, but stores a weight that represents a relevance measure of the term in the document. Some of the most popular weighting schemes are Tf-idf and BM25.

Tf-idf stands for term frequency - inverse document frequency. The motivation behind Tf-idf is that the BoW weighting scheme gives more weight to naturally more frequent words, which are not necessarily more relevant, such as stop-words. The presence of such frequent words clouds the detection of truly relevant differences. Because of this, Tf-idf introduces the idea of document frequency, which is the number of documents containing a particular term. Using this additional measure, we can detect which words are naturally more common thought the whole dataset and are given less weight in favor of terms that stand out and mark the difference between documents.

Let $tf(i,j)$ be the frequency of a word w_i in document d_j and $df(i)$ be the document frequency of word w_i. Being N the total number of documents, the inverse document frequency is defined as $idf(i) = \log_2(N/df(i))$. Finally, Tf-idf is defined as $tfidf(i,j) = tf(i,j) \cdot idf(i)$. There are some variations of Tf-idf where the tf and idf components are normalized to reduce the influence of document size. It is also possible to use binary tf.

BM25 was also designed to polish the rough edges of the BoW weighting scheme. It expands the Tf-idf scheme and it is considered to be the ideal weighting for search, found after a long process of academic iteration. It keeps the idf component and parametrizes the tf component as $tf*$. The $tf*$ component depends on the free hyper-parameters k and b with default values of $k = 1.75$ and $b = 0.75$. It is defined like so: $tf^* = \frac{tf(k+1)}{k(1-b+\frac{b \cdot DL}{AVDL})+tf}$ where DL is the document length and $AVDL$ is the

average document length. The intuition behind the formula is that k makes sure that tf^* is monotonically growing and bounded and that b parametrizes the level of document length normalization. Finally, BM25 is defined like Tf-idf: $BM25 = tf^* \cdot idf$.

Document comparison is an important aspect of the VSM model. In fact, simple algebraic comparison in the VSM is the de facto similarity method for many use-cases and is a very good baseline if the weighting is well chosen. The standard comparison metric is cosine similarity, which is equivalent to dot product if the vectors are normalized. Euclidean distance can also be used as well as the less well known Tanimoto similarity, which is analogous to cosine similarity but with a different normalization factor.

The VSM is widely used in machine learning. It is a very convenient representation as it fits perfectly with the standard feature vector representation, which is the input format of most machine learning algorithms. VSM ignores word ordering but this is not a problem for most prediction tasks. In recent years new neural models such as CNNs and specially RNNs have enabled introducing ordering into the equation once again, but the VSM format has been the standard for most of NLP history.

VSM is also the standard representation of text in information retrieval. In fact, the model was devised to enable highly efficient similarity and search queries. Documents are commonly indexed as a big term-document matrix where each column is a VSM document vector and each row corresponds to a term. If a query is also encoded in the VSM form, a simple vector-matrix multiplication yields a ranking of all the documents in the index. Such linear algebra operations have very efficient implementations via low level instructions. These kind of operations are also very easy to parallelize and distribute at high scale. The ever present MapReduce framework was, in fact, born at Google to do such matrix multiplications at web scale.

2.5 Deep Learning

Deep Learning (DL) is a label given to a special kind of Neural Networks (NN) that has driven breakthroughs in the state-of-the-art of many predictive tasks. There is still no clear theoretical foundation that explains why Deep Neural Networks (DNN) are so unreasonably effective. Anyhow, DNNs conquered the field of computer vision with the introduction of Convolutional Neural Networks a few years ago. By now, DNNs have replaced most classical algorithms for pattern recognition tasks such as object recognition. The same thing has happened in NLP with the introduction of Recurrent Neural Networks.

Thanks to DNNs many old prediction problems are now are considered as solved in academia, leading to a new wave of deeper prediction tasks. Machine translation between common languages is now almost seamless. Translation is now being linked to speech recognition and audio translation devices are starting to enter the consumer market, which has been a long term dream. There is work on document and image question answering, where the DNN seems to understand the contents in great detail and can retrieve or generate short snippets that answer very specific details about the media. DNNs have also lead to breakthroughs in algorithmic stock trading and music processing, from classification to style transfer and composition.

Deep learning has kept breaking barriers during the last five years. It is only beaten careful feature engineering and large ensembles of Decision Trees or Support Vector Machines. However, probably because of it, saying that deep learning is hyped is an understatement. It is becoming a global phenomenon and it has started to enter mainstream media. People are starting to expect the world from deep learning, but its already starting to show its limitations. DNNs are highly effective at pattern recognition, but they do not have the capacity to do proper planning or reasoning. It is very possible that deep learning will soon start not meeting expectations. This might mean the end of another AI wave like the Expert System wave from the 1980s. Let us see how things progress.

Going back to the science, in this section will cover the basics of what neural networks are and how they are trained in a bottom-up approach. It is not intended to be a thorough explanation, though, it is just meant to build up the necessary vocabulary to understand the inner workings of the algorithms described in this work.

2.5.1 Neural networks

Neural networks have existed for a while. Perceptrons, the building blocks of NNs, where developed during the 1950s and 1960s by Frank Rosenblatt, following some earlier work by Warren McCulloch and Walter Pitts. Backpropagation, the algorithm to extract gradients from NNs and optimize them, has been around since the 1970s in different scientific domains and was introduced to machine learning by the famous 1986 paper (Rumelhart, Hinton, Williams, et al., 1988). NNs have stayed as competitive algorithms for machine learning for a while, but they have not been a stand-out until computational power and data availability have grown enough.

The original design for perceptrons has largely stayed unchanged. Neural networks are named as such because perceptrons are heavily inspired by the electromechanics of biological neurons. The field was in fact born from neurology. It started as an experiment to replicate brain functions through simulating small networks of neurons.

A perceptron is, in essence, a parametrized linear algebra operation that aggregates a set of input scalars and outputs another scalar value. More formally,

$$y = w^T x + b$$

In this equation x refers to an input column vector and y is the output scalar. The w column vector contains the weights, the coefficients of the linear combination of the input, and b is a scalar called bias. w and b are free parameters that are learned through a supervised optimization process.

In practice, though, there is a final element to a perceptron called an activation function a.

$$y = a(w^T x + b)$$

There is a series of valid activation functions. The original perceptrons for example made a binary decision, the output was 1 or 0 depending on the condition $w^T x + b > 0$. Modern activation functions are chosen based on three criteria: they are differentiable, they bound the output and they are non-linear. Bounding the output

is important to make sure that the optimization does not diverge and that unit scale is not a concern. Differentiability is key to be able to learn the parameters, as will be explained later. Finally, adding non-linearity to the model increases its capacity for generalization, this is a common practice in machine learning, it can make simple linear models very effective.

Activation functions are not designed on a case-by-case basis, there is a set of standard functions to choose from. For example, the sigmoid function ($\sigma(x) = \frac{1}{1+e^{-x}}$) compresses the input smoothly to the range $[0, 1]$ and the hyperbolic tangent ($\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$) compresses it to the range $[-1, 1]$.

Neural networks are created by stacking perceptrons or neurons into a graph, where an edge points out that the output of a neuron should be fed as the input of another one. The most common structure is called Feed-Forward Neural Network (FFNN). In FFNN neurons are organized in layers, where every neuron in a layer aggregates all the outputs of the previous layer. The first layer is called input layer, it is where a data point is introduced in vectorial form. The last layer is called output layer, it is the result of the computation. Any intermediate layers are called hidden layers.

It is proven that a FFNN with a large enough hidden layer can be trained to emulate any possible function. In practice, however, it is preferable to add more consecutive hidden layers. This is where the term deep learning comes from, a neural network with more than one hidden layer, where, the deeper you go into the network, the more abstracts representations you find.

2.5.2 Neural network optimization

Neural networks can be used to perform any machine learning supervised learning and prediction task. A learning task is supervised when there is a training dataset with pairs of example input and output pairs that are guaranteed to be correct. With a big enough training set and enough computation time, a machine learning model will then learn the implicit function defined by the pairs and hopefully generalize it to new data points.

We've already discussed that neural networks learn by optimizing their weights such that the complete network approximates the desired function. This is possible thanks to two algorithms: backpropagation and Stochastic Gradient Descent (SGD).

Backpropagation is a surprisingly simple algorithm. It has been reinvented several times in physics, maths and computer science. It is an efficient algorithm to compute the partial derivatives of the weights with respect to the accuracy of the output.

Let us start by defining loss. A neural network optimization algorithms needs to know how well it is doing. For this purpose, a function called loss is selected to compare the networks output to the gold standard output given in the training dataset. Much like activation functions, loss functions need to be differentiable and there is a set of standard functions to choose from, depending on the use-case. As the name suggests, the loss function informs about how bad are the predictions, so the objective is to minimize it.

Thanks to the loss, we know how well or badly we are doing, but we also need to know how we can improve. This is where backpropagation comes in. The whole neural network is composed by many simple differentiable operations. Given some

input, first, a forward pass is performed and a loss is computed. Then, on the backward pass, the partial derivatives of each intermediate result with respect to the loss are calculated by simply using the chain-rule. After the backward pass, we have the partial derivatives of all the weights with respect to the loss, which means that we know the direction to move in to reduce the loss.

$$w_{i+1} = w_i - \gamma \nabla F(w_i)$$

The idea is to make small steps towards the gradient until arriving to a minimum, this is called gradient descent. It is an iterative process because the gradient will not stay constant, it needs to be recomputed at each position. The step size is specified by the learning rate γ. Choosing a correct learning rate is one of the challenges of building a good neural network. If the learning rate is too small, it will take longer than necessary to arrive to the minimum; but if the rate is too large, the descent will be more chaotic and it may jump over minima. Modern gradient descent algorithms have dynamic learning rates based on the current and past gradients to achieve convergence faster.

We have explained how the optimization is performed based on a single input-output pair. Ideally, we would evaluate each gold standard pair on each iteration and average all the gradients. This is fairly costly however. A key discovery is that this is not strictly necessary. It is proven that by only using one data sample per iteration the algorithm will arrive to the minimum eventually. This is called Stochastic Gradient Descent (SGD). It is true that it may need more iterations than pure gradient descent to achieve convergence, but it is computationally cheaper and, in practice, much faster. The current trend is to use a compromise between both approaches, called mini-batch SGD. The idea is to use a few samples per iteration, a batch, instead of a single one. This is more robust and still fairly cheap.

An important point against gradient optimization is that we assume that the optimization space is convex. This is a strong assumption, many academics where skeptical of the idea in its beginnings because of this, there is no guarantee that the minimum that is found is the global minimum. It turns out, however, that this is a lesser issue than was expected. If the weights are initialized randomly, the learning rate is well managed and the gradient descent algorithm is further tuned to avoid local minima, by adding momentum to the descent for example, it is uncommon to get stuck in a sub-optimal setting.

2.5.3 Deep learning patterns

Neural networks have not been stand-out machine learning algorithms until vast computational power and have become readily available. However, those resources have not been the only factor of the success of neural networks. Simply stacking many neuron layers into a deep feed-forward network is not enough, these kind of architectures only beat the state-of-the-art if a few tasks. Proper regularization like drop-out is vital to avoid overfitting and make sure that the learning is propagated through the network correctly. The real breakthrough has been the introduction of more complex architectures that are designed to work with specific formats of data, such as Convolutional NNs or Recurrent NNs.

Convolutional networks or CNN where the first NN architecture to become a true phenomenon. In just a few years, CNNs have beaten the state-of-the-art in most computer vision tasks by a large margin. The naive way of working with images would be to simply take the image as an input vector and introduce into a big Feed-Forward network. This, in practice, is infeasible and wasteful. Even moder computational resources could not train a model with such high dimensionality. CNNs fix this my taking a page from classical computer vision, they train kernels.

A kernel in the context of image processing is a small matrix of weights. The general idea is that kernels are passed over an image to transform them in some way. Kernels have odd sizes and are usually two dimensional, but not always. When they are places over a set of pixels, the values of those pixels are multiplied by the kernel weight on top of them, summed and the central pixel's value is replaced by that sum. This way, pixels can be transformed with awareness of their neighboring pixels while keeping the whole process very efficient. Kernels have been heavily used in computer vision for tasks like edge detection.

In a CNN a neuron is a kernel that is passed over the whole image. Each neuron layer produces a set of alternative representation of an image, which are then further transformed by later layers. Labeling an image is essentially a dimensionality reduction task, so, to progressively reduce the dimensionality, CNNs also include pooling nodes. The most common pooling method is max pooling, where the size of an image is reduced by selecting the pixel with the larges value between its neighboring pixels.

This is an extremely powerful way to model images and other fixed size high dimensional data. A common visual example for the power of deep learning is showing the intermediate representation of a CNN trained to perform object recognition. It can clearly be seen that the first layers distinguish low level features such as lines and angles, later layers distinguish rough shapes and textures and the final layers show images of generic examples of the object they detect.

CNNs have also been used for other types of media with relative success, like language or music. However, both of those mediums have an inherent temporal dimension that CNNs do not capture well. They are also not fixed in size.

Recurrent networks or RNNs are the solution that was proposed to deal with temporal sequences and variable sized input. An RNN layer trains a single cell, which is an arbitrarily shaped NN that takes an element in a sequence represented as a vector and outputs another vector of the same size. The key is that an additional vector of the same size called state is transferred from one instance of the cell to another. A cell processes a sequence of input vectors in sequence, changes the state and it keeps it to process the next element. It is an implementation of temporal memory.

RNNs are used in a variety of ways. They can be used to read text and output a single labeling represented by the last state. They can also be used to take in a fixed size input in the first step and generate a sentence. They can take in a sentence and produce another one, like in translation. RNNs have been used for generating text too, but providing a random input at first and then feeding the last generated word to the next cell.

There are also more complex RNN architectures. A bidirectional RNN processes a sequence forwards and backwards. An encoder-decoder architecture has two RNNs,

the encoder takes a sequence and generates an abstract vectorial representation, the decoder then produces an output sequence based on this intermediate representation. Both of these elements are used to enable modern state-of-the-art translation.

RNNs have revolutionized natural language processing as CNNs revolutionized computer vision. However, a naive RNN architecture does not perform as well as one would expect. Keeping track of information over long sequences is challenging. The real breakthrough was the design of the standard Long-Short Term Memory (LSTM) cell architecture. LSTMs have several neural layers called gates that control what is forgotten from the state, what is added to the state and what is outputted from the state in each iteration.

2.6 Word embeddings

Word embeddings are vectorial representation of the meaning of words. This is a fuzzy notion; in practice, this usually means that word embeddings are placed in a high dimensional space where the embeddings of similar or related words are close to each other and different word embeddings are placed far from each other. Word embeddings also acquire more complex geometric structures as a side effect of some algorithms. A typical example for this are real world analogies that can be discovered using simple vector arithmetics: king - man + woman = queen.

Word or sense embeddings can be trained on knowledge graphs, but the most popular algorithms learn these vectorial representations just by scanning big corpora. All of these algorithms rely on a single assumption: words that appear in similar contexts have similar meanings. Most state-of-the-art corpus word embedding algorithms are variations of the original Word2Vec. Later more formal theoretical work has extracted a generalized framework that most of these algorithms fit in.

The task is always to factorize a word-word matrix that contains cooccurrance counts, Point-wise Mutual Information (PMI) or similar metrics. The factor matrices are usually called U and V, which define two distinct embedding spaces. U is a matrix that contains the final word embeddings and V is a temporal set of embeddings that contains the representations used for context words.

GloVe and a few other algorithms perform the matrix factorization explicitly. However, Word2Vec and most other embedding algorithms do the factorization implicitly. They do this by scanning the corpus with a fixed sized window. The window has a central word, called the target and a few neighboring words that are called the context. Both target and context embeddings are initialized randomly in U and V respectively. The goal is to minimize the distance, usually dot product, between words and their contexts. This is done by performing stochastic gradient descent, where each stochastic sample is a consecutive window in the corpus. Each time a target word is found with a context, their vectors are pushed together slightly, depending on the learning rate. These models are very simple and, therefore, very efficient. They can run through very big corpora and they only need a few scans to achieve convergence. This is precisely the key to the success of this new wave of embedding algorithms. Shallow models like these win over more complex deep neural models by being far faster and by consuming far more training data.

Although technically senses are the purest form of semantic unit, it is often assumed that there is a one-to-one correlation between words and senses and word embeddings are used as the basic input for many tasks. This is a reasonable assumption, as polysemy and synonymy are relatively rare. It is also far easier to train word embeddings using the framework described above.

Word embeddings can then be composed into more abstract structures, such as phrases, sentences, paragraphs or documents. Compositionality is still an open problem, the state-of-the-art is fairly narrow in this task. A lot of unsupervised sentence and document embedding algorithms still use a very similar framework to word embedding algorithms inspired by the original Word2Vec. In fact, some document embedding algorithms are specialized word embedding algorithms that optimize word embeddings such that just averaging them or performing a similar simple composition provides meaningful document embeddings.

Chapter 3

Evaluation framework

This chapter discusses the process building a framework for reviewing the state-of-the-art in semantic document similarity measures. This work focuses on reviewing document similarity in the medium of scientific texts or, more concretely, peer reviewed academic literature.

We will also primarily focus on abstracts as the target of similarity measurement. This is mostly due to availability, but it is also because abstracts are a condensed version of a publication's contents and have a good balance between length and amount of information. However, titles are also widely available and the sources that have been selected also provide preprocessed full-text content. Therefore, although it is not the main focus, we will also do some testing with titles and full-texts.

It is relevant to mention that to ensure generality and flexibility context information will be encoded in the form of word embeddings. During the process of offering service through SCITODATE, we have come to the conclusion that even the most complete knowledge bases in science are not broad enough for our use-cases. Coverage of different fields of science is also highly inconsistent. Because of this, special emphasis has been put on word embeddings and the review of the state-of-the-art in embedding training is also a significant part of this work.

Due to the lack of reliable and abundant training data in the realm of scientific texts, this work will look at unsupervised document similarity models that mainly rely on the underlying word embeddings for knowledge mining. However, we do consider that there is enough data to create a good evaluation dataset.

After a thorough search, it was concluded that there's no standard training or testing data in the scientific domain that fulfills the requirements described above. Therefore, this chapter discusses the collection of data and creation of all the necessary training and testing datasets. Special care was taken in creating a quality evaluation framework in the scientific domain, both for this review and future work by the community. It is intended to be one of the core contributions of this work.

3.1 Training corpus

The most prevalent word embedding algorithms learn the implicit meaning and context of words by scanning through large collections of free text. The key assumption made by the majority of these algorithms is that words that appear in similar contexts have similar meaning.

Even if this idea has been around since the 50s, Word2Vec (Mikolov, K. Chen, et al., 2013) was one of the first that used it efficiently to train word embeddings. Although, simple, the idea is surprisingly effective and it is the discovery that kick started the word embedding field. This simple co-occurrence technique not only yields vectors that encode similarity or relatedness, but, somewhat unexpectedly, also encodes more complex semantic relationships. All this will be further elaborated on with each individual algorithm.

As mentioned above, due to a lack of training datasets, document similarity models often rely on this knowledge mining quality that embeddings have. Word embeddings have, in fact, become one of the foundations of modern NLP.

Most academic work focuses on general purpose word embeddings and resources for that purpose are widely available. The most common sources of free text are the Google News corpus and the Wikipedia dump[1]. This sources are so standard that there's high quality pretrained word embeddings available[2] and they are often relied upon.

However, this work focuses on scientific content, which requires special treatment due to the abundance of highly specific terminology that cannot be found in other media. A quality specialized training corpus is key to ensure a reliable review. This section discusses the process of creating such a dataset.

3.1.1 Sources and rationale

ArXiv In the context of the overarching SCITODATE project intensive work has been done using the arXiv metadata stream[3]. The stream is updated daily and contains around 1M records as of date, 800K of which are physics articles, around 120K computer science and the rest is a mix of maths, chemistry, biology and economics. A vast majority of these records contain abstracts, which are the interesting part if we want to construct a corpus.

ArXiv is the biggest database of physics articles and the second biggest in computer science, behind DBLP. It is one of the best sources of scientific text available, and training on the abstracts has yielded reasonably high quality embeddings. However, using the whole database introduces too much noise due to the high variance in topics and even the 800K physics abstracts do not produce as big a corpus as would be desirable.

CORE Another good alternative is the CORE database[4]. It is a large meta aggregator of open access database and even contains a large number of preprocessed full-texts. However, again, the high variance in fields contained here makes it difficult to work with. A good subset is the one coming from PubMed, but it is not as big as the original one, so it's better to go to the source.

[1] Wikipedia dump: https://dumps.wikimedia.org/
[2] Word2Vec repository: https://code.google.com/archive/p/word2vec/
[3] ArXiv OAI-PMH: https://arxiv.org/help/oa/index
[4] CORE: https://core.ac.uk/

PubMed PubMed[5] probably the biggest database of open access articles in science. It specializes in the biomedical and pharmaceutical fields. The Open Access subset called PubMed Central contains 1.2M metadata records with more details than any other database. Most interestingly, the vast majority of those records contains high quality full-text bodies extracted from the original PDFs. PubMed has been identified as the best source for our current purposes.

3.1.2 Technical aspects

Data acquisition Fortunately there is a very well designed protocol exclusively used for sharing metadata of academic publications. It is, in fact, one of the main reasons why big aggregators such as Crossref, PubMed, DBLP or CORE can unify different databases with relatively low development costs.

The protocol is called OAI-PMH[6]. It keeps a sequential feed of metadata records and record updates. Each record has a timestamps expressing when it was added to the feed, which enables the user to easily retrieve date ranges or keep a local copy of the data updated. It is also a very convenient way of retrieving the whole database by not specifying dating information. The protocol takes care of keeping the server load at reasonable levels, which is why it is so widely adopted. But it is also usually reasonably fast to retrieve the whole repository.

Parsing The main obstacle is dealing with the schema. OAI-PMH usually deals with XML records, which is fairly standard. However, it can be problematic when your main data format is JSON or a similar dictionary-list nested structure, as conversion is non-trivial with complex schema.

There are some schema standards in the academic metadata environment. One of the most widely used is Dublin Core[7]. However, it is far too limited for many use cases. Because of that, aggregators tend to combine different standards, like CORE does[8], or define a very complex superset schema like the one from PubMed[9]. This second case is common, specially in cases where commercial sources are involved. Publishers usually deal with more complex metadata and each one has their own in-house schema.

Parsing the PubMed data has been time consuming. Extracting the title or the abstract is not too challenging. But converting more complex data such as authors, affiliations or references into a usable format is non-trivial. Such data is not too relevant for the corpus but a great deal of effort has been invested in this aspect in the context of the SCITODATE project. This metadata, specially the authors, is also used for training and evaluating document similarity models.

The full-text body is relevant for the purposes of this review though, and it is also hard to deal with. Fortunately, PubMed already takes the work of extracting the content from the original PDFs. However, the output XML has a custom schema that

[5] PubMed: https://www.ncbi.nlm.nih.gov/pubmed/
[6] OAI-PMH: https://www.openarchives.org/pmh/
[7] Dublin Core: http://dublincore.org/
[8] CORE schema: https://blog.core.ac.uk/files/data_schema_v0.2.png
[9] PubMed schema: https://jats.nlm.nih.gov/archiving/tag-library/1.1d3/element/article.html

is not documented. Work was done to analyze the schema empirically, to convert it into a much simpler nested section format and to render it into plain-text.

Cleaning Once the corpus has been created by converting all content bodies into plain-text and concatenating them, some further work needs to be done to clean and normalize the text. The cleaning function was designed empirically, by looking at large amounts of samples and adding rules until a reasonable level of cleanliness was observed.

After some cleaning, standard sentence and word tokenizers are applied, implemented in NLTK[10]. Some testing was done with lemmatization, but most implementations use WordNet[11] or similar KBs as reference data[12], which is not very effective in the scientific medium. After experimenting with a few alternatives, an English Porter stemmer was finally chosen for normalization, also implemented in NLTK.

Pruning It is common practice to do some simple frequency based word filtering when training word embeddings. Very frequent words, such as stop-words, do not hold much semantic meaning and end up adding noise to the system. Very infrequent words also cause issues, as there's not enough samples to train them correctly. Furthermore, in the scientific domain, we are most interested in established technical vocabulary, therefore, a word that is rarely mentioned is not very relevant in this context.

It is also important to create a central vocabulary so that the training and testing data, as well as the models, are correctly synchronized.

We solve both tasks by using the dictionary implementation from the Gensim library. It takes care of indexing the vocabulary and pruning.

The corpus creation is, therefore, done in 3 passes:

1. Extract the text fields, render to full text and clean in parallel.
2. Read the corpus sequentially to create the dictionary.
3. Re-create the corpus removing all the words that are not in the dictionary.

The first and second passes could have been done at the same time. However, the dictionary implementation is most likely not fork-safe, and using multi-processing accelerates the first pass considerably.

3.2 Word embedding evaluation

There doesn't seem to be a clear standard for word embedding evaluation.

[10] NLTK: http://www.nltk.org/
[11] WordNet: https://wordnet.princeton.edu/
[12] WordNet Lemmatizer: http://www.nltk.org/api/nltk.stem.html#module-nltk.stem.wordnet

Analogy The original Word2Vec article proposes an evaluation based on what they call analogy tests. They create a dataset were each records is composed by two pairs of words, such that both pairs have an equivalent internal relationship. Evaluation is done by checking the quality of prediction of the second pair by using vector arithmetics.

Such an approach ensures that the embeddings encode complex semantic meaning, instead of only similarity. However, the original dataset was relatively small. Other well known word embedding algorithms such as GloVe have also used this dataset, but no one seems to have invested time in extending it. Further research into such complex semantic relationships also suggests that the algorithm inherently may not learn the same intuitive relationships a human would expect (Fu et al., 2014). This may introduce further noise and decreases the usefulness of such an evaluation.

Models It is also a common approach to evaluate word embeddings as the input to a more complex model, and evaluate that model instead. For instance, GloVe proposes a Name Entity Recognition task for evaluation, and classification, sentiment analysis or clustering are also commonly used for evaluation. This may be easier to evaluate in some cases, but it gives more options to the authors, which leads to further noise and inconsistency.

Similarity datasets There is a collection of word similarity and relatedness available that has been commonly used for word embedding evaluation, the most complete one being WordSim[13]. WordSim or WS353 is very standard and is used in most evaluation procedures, but there's a collection of smaller datasets that are used more inconsistently, which makes different embedding algorithms hard to compare. GloVe from Standford, for instance, uses WS353, RG, SCWS and RW . FastText from Facebook, also uses WS and RW for English, but also focuses on other major languages: Gur65, Gur350 and ZG222 for German, RG65 for French and WS353 again for Spanish.

Unfortunately, these datasets are fairly small. WordSim itself only has 353 word pairs. These datasets are also very general,and do not cover specialized domain vocabulary. There is also the issue of similarity versus relatedness. Word embeddings do not distinguish between both and these datasets do. This can be a real issue as, for example, antonyms are considered to be highly related but completely dissimilar.

Effort for standardization is underway. The SemEval workshop[14] is a major effort to unify the semantic analysis field, and has a monolingual and cross-lingual similarity task. However, the scale of the datasets provided in the last iteration of the competition in 2017 are similar in scale to the ones mentioned above. Most importantly, they deal with very general vocabulary and the focus of this work is to evaluate all available techniques in the scientific domain.

A custom dataset Because of the lack of an appropriate standard for domain specific word embedding evaluation, the decision is made to create a custom one. A hand-made knowledge base is the effective equivalent of word embeddings when it comes to encoding semantic meaning. Knowledge bases are heavily curated and,

[13]WordSim: http://alfonseca.org/eng/research/wordsim353.html
[14]SemEval: http://alt.qcri.org/semeval2017/

even if they don't cover as broad as a vocabulary, they should be the quality ideal trained word embeddings should strive for.

A knowledge base could, of course, be used for analogy evaluation, as described above. However, this work focuses on document similarity, so word embeddings will be evaluated and optimized exclusively to encode a semantic similarity relationship.

3.2.1 Sources and rationale

The UMLS metaontology from PubMed has been selected as a source of reference data. As mentioned before, PubMed is probably the biggest and most detailed Open Access database in science. Equivalently, it is also safe to say that UMLS is the most complete knowledge base in science.

Like PubMed, UMLS is focused on the biomedical and pharmaceutical fields and is a perfect fit for our training corpus. UMLS is called a metaontology because it is designed to be a superset of all mayor knowledge bases in the field (LOINC, CPT, ICD-10 and SNOMED CT to name some). This makes it a bit more cumbersome to work with, but it also means that it has maximum coverage.

All data will be extracted from the Metathesaurus, which is the section of UMLS that contains the vocabulary.

3.2.2 Technical aspects

Data acquisition Access to UMLS is open but an explicit access application has to be sent and it takes a few days for it to be reviewed and accepted. After gaining access and downloading all the data, all the CONSO and REL files are extracted. The CONSO files contain the vocabulary itself divided into concepts. Concepts are sets of lexical representations and strings, equivalent to the synset concept in WordNet. The REL files store all the edges in the knowledge graph, the relationships between concepts. The data is stored in a relational format as CSV files, in the RRF schema. This data is enough to create the evaluation dataset.

Cleaning Unfortunately, the UMLS vocabulary is far from being clean. Many different sources are combined in UMLS.

The text cleaning for the creation of the training corpus was done empirically, and the same approach is taken to clean this vocabulary. Large amounts of samples have been observed and simple rules have been set that remove most of the noise.

The same tokenization and stemming from the corpus is applied here too, so that both vocabularies match as best as possible.

Pruning is then performed using the vocabulary extracted from the corpus.

Synonym extraction First, all equivalent string sets are extracted from the CONSO files and permutations are created to collect synonym pairs. The next step is to generate pairs using synonymy relationships.

A recent article that uses UMLS data to train a medical synonym extraction system has been identified (Wang, L. Cao, and Zhou, 2015). This article conveniently names the set of relationships that express synonymy based on their analysis. This same set of relationships is used to find synonym concepts in REL. More pairs are produced by taking this concept pairs, retrieving the set of strings they correspond to and by computing the product between the two sets.

The processing is performed with the aid of a SQLITE3 database. This database is easy to work with and offers a much higher performance than a naive solution.

Triplet creation The last step to have a usable test dataset is to create the triplets. The reason for this triplets is explained in section 3.4. We need to create triplets such that the first two elements are synonyms and the first and third aren't.

Due to the architecture of the knowledge base, finding synonym pairs is relatively easy, but identifying all synonym sets is a bit more complex. It involves finding all the connected components. It is not an expensive operation, it can be done by a simple BFS or a disjoin-set data structure, which is even more convenient in this case. However, the data is stored in a relational format. Doing this computation would require using an extra graph processing library and converting the data to a more graph-friendly format. Furthermore, we would need to do the whole computation in-memory and keep the sets stored while creating the triplets, which is non-trivial for the size of this knowledge base.

A naive stochastic approach is taken instead. We convert the synonym pairs into triplets by appending a random word from the vocabulary as the third element. The size of the vocabulary should ensure that the probability for an erroneous sample is trivially low.

3.3 Document similarity evaluation

Document similarity evaluation is, across the board, even more inconsistent than the evaluation of word embeddings. This is, in part, because document similarity falls under the field of document embeddings, which are not exclusively used for similarity. Similarity measures and document embeddings are the bases for common NLP tasks such as text classification or sentiment analysis. That's why similarity models are often evaluated by feeding them into more complex learning tasks.

Evaluation for classification or sentiment is far more established, and raw similarity datasets are sparse, so this makes evaluation easier. However, much like with word embeddings, this increases the options authors have for evaluation, which leads to further inconsistency. The case of Doc2Vec one of the most well known document similarity algorithms, showcases this issue.

Doc2Vec The original Doc2Vec article (Quoc V. Le and Mikolov, 2014) proposes three independent evaluation tasks: sentence sentiment analysis on the Standford

Sentiment Treebank Dataset, document sentiment analysis on the IMDB dataset and document similarity based on the queries and results of an unnamed search engine.

There's also a few third-party evaluations of Doc2Vec: (Lau and Baldwin, 2016) and (Dai, Olah, and Quoc V Le, 2015).

The first evaluation, (Dai, Olah, and Quoc V Le, 2015), does qualitative and quantitative analysis.

The qualitative analysis is anecdotal. They plot embeddings using t-SNE to look at patterns and do some simple Nearest Neighbor and analogy exercises with two short phrases: Machine Learning and Lady Gaga.

The quantitative analysis is somewhat more thorough. They create 172 triplets of ArXiv articles based on the author's domain knowledge. They also create 20K triplets with at least one ArXiv category in common. Finally, 19,876 more triplets are extracted from Wikipedia based, again on categories. A simple prediction task is then applied to these datasets for evaluation.

The second review article, (Lau and Baldwin, 2016), evaluates with a Forum Question Duplicate Detection task. They sort pairs by similarity score and use AUC and ROC as measures. They also use the SemEval dataset of human annotated document similarity.

SemEval Like with word embeddings, SemEval is one of the main drivers in standardizing semantic analysis work. The first task in SemEval 2017 is, precisely, a document similarity task. It includes a large quality dataset called STS of 8,628 sentence pairs extracted from news, media captions and forums.

It is exceptional work, and one of a kind in this field. However, it is, again, inappropriate for our use case. It focuses on sentences, while we attempt to establish similarity between abstracts or full-texts. And, like always, it is not domain specific, we need something exclusive to the scientific domain.

3.3.1 Sources and rationale

The focus of this work is to evaluate the document similarity state-of-the-art applied to academic publications. Unfortunately, as stated above, there's no standard dataset that could be used to evaluate such a task. We therefore, set out to create one of our own.

Data Raw data is the starting point. As we've already established when creating the training corpus, article metadata is widely available. Not all metadata is equally available though. Title, authors and date are common fare. Abstracts are not always available for subscription journals, but are otherwise common to come by. However, full-texts are hard to come by, and the same happens with references, which are only available in high quality Open Access databases such as PubMed.

Similarity link To evaluate document similarity, we need to identify an attribute that links articles to one another expressing some kind of semantic relationship. It is clear that collecting scored similarity pairs is nigh impossible with the available

data. Creating a human scored dataset is also beyond our means. However, we can find pairs which are likely far more similar than a random pair.

This type of reasoning is not uncommon. We have applied the same logic for evaluating word embeddings using a Knowledge Base. (Dai, Olah, and Quoc V Le, 2015) this Doc2Vec third party evaluation creates some datasets using categories from ArXiv and Wikipedia.

(Dann, Hauser, and Hanke, n.d.) cites (Carpenter and Narin, 1973) proposing the following assumptions for journal similarity: similar journals will have similar journal referencing patterns and similar journals will refer to each other. Inspired by this, they propose the following assumptions to evaluate document classification and clustering: articles from the same category are related and articles from the same journal are related.

Linking fields There's four metadata fields that could be used for this purpose.

The most obvious one would be to use categorization from the source database. Categorization is common, and there's some that are very thorough, like PubMed, which even has a custom ontology called MeSh describing the categorization. Categories are still broad though, some of the lowest level sub-fields in ArXiv, such as condensed matter, receive upwards of 200 new publications daily.

Another link could be journal information. However, journals are broad, even broader than categories, and this metadata is harder to index, is more error prone and is harder to come by.

References would be ideal. They are the closest equivalent to human scoring for article similarity. However, references are the single hardest piece of metadata to get hold of. When indexed, they are usually behind a pay-wall. It is possible to extract references from raw Open Access PDFs, but processing is expensive as machine learning models need to be used. Even when extracted and available, the greatest challenge is to cross-reference those citations to the abstracts or full-texts they correspond to.

Authors are also a good source for semantic linking. Authors most often than not stay in the same field of research throughout their career, so most of their articles should be considered similar. This is even truer if we combine author information with publications dates, as articles written by the same author in close succession should indeed be very similar.

3.3.2 Author link dataset

Acquiring author information comes with it's own challenges. Author names are available in almost all metadata databases, however, the names are all that is given. Authors often just state their initials and they aren't always consistent when writing their name throughout their career. Information about their institutions is provided some times, which solves some of the cases. However, this is still a grave problem with Asian names for example.

Author disambiguation is still an open problem and it has started to receive considerable attention both in academia and in industry.

PubMed comes to the rescue though, yet again.

Thanks to the ORCID[15] project there's now a unified ID systems for scholarly authors. ORCID requires authors to sign up and register their publications manually, which has been an obstacle for adoption. However, very recently big publishers have started to require this linking, and ORCID has quickly grown in the last two years because of it.

PubMed metadata includes these unique IDs when available. The amount of IDs is still very low. From analyzing the whole historical PMC dataset, we have observed that only about 1.5% of author mentions contain any kind of ID. We have only found 13K IDs which are mentioned more than once.

That amount should be enough for evaluation, but it would have been tight for training. This work focuses on unsupervised methods, so we are in no need for such training data. However, for future reference, there might be two more sources to draw from.

ORCID publishes a data dump since 2013. It includes article DOIs, so more semantic links could be established with a big enough article database. For this work we have also only looked at the PMC PubMed Open-Access subset, of around 1.2M articles. There's a bigger MEDLINE/PubMed database of 26M non-open-access records which may also contain author IDs.

3.4 Evaluation metric

The most common approach for evaluating word embeddings is to have a dataset of word pairs which are given similarity scores by humans. In this case, a simple Spearman correlation is computed to measure the difference between the human and embedding based similarity scoring.

However, as mentioned above, we have chosen not to rely on standard human datasets, as they don't extend to the scientific domain, where more technical vocabulary is required. Instead, we've used a well established Knowledge Base as a sample of human curated data and we've extracted synonym pairs from it.

The case with our document similarity dataset is also almost equivalent. We lack an explicit human scored dataset, but we extract human supervision from the author data.

In both cases, there's a semantic link. The assumption is that two linked elements are considerably more similar than most random pairs.

As referenced when creating the document similarity test data, this practice is not uncommon. Likewise, the community seems to have converged to a standard way to evaluate such a test dataset. Triplets are created, where the first two elements are linked and the first and third are not. This becomes a simple classification problem, where the error unit is the case where the first and second element are scored less similar than the first and third. The evaluation is a simple error rate measure.

[15] ORCID: https://orcid.org/

3.5 Computational benchmark

Computational performance benchmarks where performed while training all models and during evaluation when relevant. For all tests we record the maximum memory usage and the User CPU time. We elect to focus on User time because Kernel time is almost zero for most of the implementations and User time sums up the time spent on each CPU. User time also ignores waits. Therefore, this time measurement was deemed the most representative for the general performance of any model.

All benchmarks where performed at least three times to increase generality; the results where then averaged. The tests where performed on clean AWS `m4.2xlarge` Ubuntu instances. This instance type a general purpose computer with 32GB of RAM and 2.3 GHz Intel Xeon E5-2686 v4 (Broadwell) or 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) CPUs with 8 cores. The time measurements were done with the `time` command line tool.

In general, we have chosen to use the default hyperparameters proposed by the authors. This was considered the most practical option, as performing hyperparameter search for each algorithm would have considerably increased the amount of work and the size of this document, to an extent were its value would be questionable. The default hyperparameters are considered as part of the model design, so performing the experimental review with default values would still be a meaningful comparison. However, for fairness, we do lock the embedding size of word embeddings to 100 dimensions and the embedding size of documents to 800 dimensions.

Chapter 4

Review of word embedding algorithms

In this chapter we review the state-of-the-art in corpus word embedding algorithms. For each algorithm we give an intuition of the inner workings, a formal specification of the model, a complexity analysis, and an experimental benchmark.

4.1 Word2Vec

4.1.1 A bit of history

Word embeddings have a long history in academia. The Neural Network Language Model (NNML) (Bengio et al., 2003), for example, was a very influential work. It is a neural architecture that simultaneously learns word embeddings and a statistical language model, and it was made back in 2003. Over the last two decades, many iterations on this seminal model have been proposed, such as the replacement of a simple feed-forward network with an RNN.

However, word embeddings did not take off until (Mikolov, K. Chen, et al., 2013) proposed two very simple log-linear models that outperformed all previous complex architectures and, most importantly, drastically reduced the time complexity. The newly increased scalability made it possible to use much bigger corpora, which opened the door to more accurate embeddings, embeddings that could reliably be used as the basis for all kinds of NLP models.

Since the publication of the article in 2013, word embedding have become the foundations of modern deep-NLP. Of course, there has been many more proposals to improve on Word2Vec, but after 4 years it has proven to be a solid baseline and is still used regularly as the default source of embeddings.

4.1.2 Intuition

The main assumption that Word2Vec (Mikolov, K. Chen, et al., 2013) relies upon is the following one: words with similar contexts have similar meaning. This is not a new idea, it was proposed back in (Harris, 1954). However, training a model on this premise has proven to be surprisingly effective.

Word2Vec starts with a set of word vectors that are initialized randomly. It scans the corpus sequentially, always keeping a context window around the each word it looks at. At this point, there are a few differences between the BoW and Skip-gram models, but, in essence, the algorithm computes the dot product between the target word and the context words and tries to minimize this metric performing Stochastic Gradient Descent (SGD). Each time two words are encountered in in a similar context, their link, or spacial distance, is reinforced. The more evidence is found while scanning the corpus that two words are similar, the closer they will be.

There is a last challenge to address. The basic model we just described only provides positive reinforcement towards making the vectors closer. With an infinite corpus, the minimum state would be that all the vectors would be in the same position, which is obviously not the desired effect.

To address this Word2Vec initially proposed a Hierarchical Softmax regulator. Later on, they proposed an alternative method called Negative Sampling. This last one is simpler and has been shown to be more effective. The basic premise is that each time the distance between to vectors is minimized, a few random words are sampled and their distance to the target vector is maximized. This way, it is ensured that non-similar words stay far from each other.

4.1.3 The maths

Word2Vec is trained by scanning the corpus with a fixed sized window. The window has a central target word and a few neighboring words called the context. There is usually the same amount of context words at both sides of the target. The optimization is performed by SGD where each sample is a window and a loss function is defined between the target and the context vectors.

The original Word2Vec paper (Mikolov, K. Chen, et al., 2013) proposes two alternative loss functions CBoW and skip-gram. CBoW stands for Continuous Bag of Words and learns the word embeddings such that given the context the target word is predicted. Conversely, skip-gram predicts each context word given the target.

A following paper (Mikolov, Sutskever, et al., 2013) gives a more formal description. Let there be a corpus, a sequence of words $w_1, w_2, ..., w_T$. The window is defined by parameter c, where c words at the right and left of the target are taken.

In CBoW the context vectors are summed and used to predict the target. This is the objective function to be maximized.

$$\frac{1}{T}\sum_{t=1}^{T}\log p(w_t|\sum_{-c\leq j\leq c, j\neq 0} w_{t+j})$$

For skip-gram, in contrast, each context is predicted independently given the target.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

The probability is defined as a Softmax, where u_w is a target embedding vector for w and v_w is a context embedding vector. The u_w embeddings are the ones that are

kept, v_w is a side product. The following definition is used for skip-gram, for cBoW the target and context vectors would be swapped.

$$p(w_c|w_t) = \frac{\exp(v_{w_c}^T u_{w_t})}{\sum_{w=1}^{W} \exp(v_w^T u_{w_t})}$$

However, Softmax is too expensive to use as a loss function, as computing the gradient has a complexity proportional to the vocabulary size W. The second paper (Mikolov, Sutskever, et al., 2013) proposes two solutions. One is to use Hierarchical Softmax, which is a $O(\log_2 W)$ algorithm for estimating Softmax. The second alternative, and the most popular one, is negative sampling or, more formally, Noise Contrastive Estimation (NCE). The following is the objective function per window to be maximized in the case of negative sampling.

$$\log \sigma(v_{w_c}^T u_{w_t}) + \sum_{i=1}^{k} \mathbb{E}_{w_i} \equiv P_n(w)[\log \sigma(-v_{w_i}^T u_{w_t})]$$

k is a hyper-parameter that specifies the number of random negative samples to use in contrast to the positive pull between the target and the context. The negative samples are pulled from distribution $P_n(w)$. The authors found that using a transformed unigram distribution $U(w)^{3/4}/Z$ performs best.

One last detail worth mentioning is word subsampling. In practice, just scanning through the corpus does not achieve a good balance of evidence. Frequent words like stop-words (and, the, in...) have very generic meanings that do not contribute much to semantic content, they are essentially noise. However, those frequent words also have a big influence simply because they appear in more contexts. To avoid this issue Word2Vec simply ignores or deletes words randomly using a probability based on the frequency. Frequent words will be ignored more often so that they do not upset the balance with noise. Let $f(w_i)$ be the frequency of a word w_i in the corpus and let t be a hyperparameter.

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

4.1.4 Computational complexity

The first Word2Vec article (Mikolov, K. Chen, et al., 2013) gives a very detailed overview of the computational complexity of Word2Vec and its predecessors, Neural Language Models. We define the complexity of these algorithms with the following formula.

$$O = E \times T \times Q$$

Where E is the number of training epochs, the number of times the corpus is scanned, T is the size of the corpus and Q is specific for each model. This is intuitive, there will be $E \times T$ windows and, therefore, update steps to be performed. Q is the complexity of a single update step in the SGD.

For cBoW: $Q = C \times D + D \times \log_2(V)$ For Skip-gram: $Q = C \times (D + D \times \log_2(V))$

C is the window or context size, D is the dimensionality of the embeddings and V is the vocabulary size. Both models have very similar complexities. In cBoW we first average all the context embeddings and compute the dot product between the target and the context centroid, which translates to computing C dot products, each with D floating point operations. The $C \times D$ part also appears in Skip-gram. This is because we need to compute the dot product between the target and each context, which is the same number of operations. The $D \times \log_2(V)$ part corresponds to the Hierarchical Softmax. Skip-gram performs this operation C times because it technically has C independent loss functions per window.

Both components have a similar effect on complexity. The window size C is usually on the range of $[5, 20]$ and the corpus we have been working with has approximately 2M distinct tokens, which computes to $\log_2(2M) \equiv 21$. Here it can be seen that the introduction of the Hierarchical Softmax has a great effect on performance.

In the case of large vocabularies negative sampling can work even better. We would replace the $\log_2(V)$ with the K hyperparameter, which is usually set in the range $[3, 15]$.

4.1.5 Experimental review

In this section we present all the experimental results from applying Word2Vec to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We have individually tested cBoW and Skip-gram with embedding size of 100. We have also chosen to perform benchmarks with different sizes of corpora, so that we can inspect how the algorithm scales, both in terms of the computational resources needed and the accuracy of the evaluation. The original implementation provided by the authors[1] was used for all benchmarks.

4.1.5.1 Computational benchmark

The following diagrams (4.1 and 4.2) show the training time and maximum memory of cBoW and skip-gram.

Execution times clearly scale linearly, surprisingly clearly. Training was performed 3 times for each instance and results were consistent. This makes sense however. The algorithm needs a pass to build a dictionary and then a number of training passes, depending on the number of epochs chosen. The same update operation is performed on for each window and word in the corpus. Word2Vec is also implemented on clean C++ with minimum dependencies, so it is not surprising that there is no noisy overhead and that the scaling is so clear.

The memory requirements seem to scale at $O(\sqrt(n))$ in both cases, although it is not as clear. This also makes sense, as most of the memory is consumed by the vocabulary of embeddings. The vocabulary will increase as we explore bigger corpora, but there will be less and less new terms at large scales.

[1] Word2Vec repository: https://code.google.com/archive/p/word2vec/

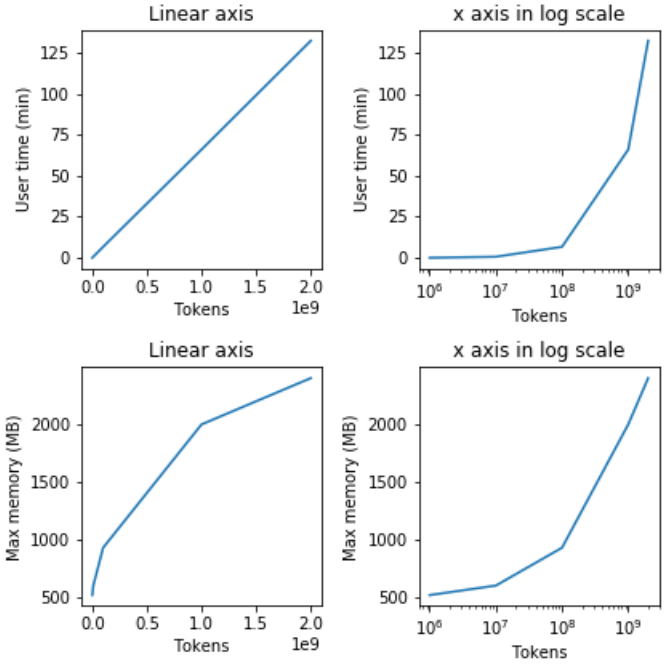


FIGURE 4.1: Computational benchmark of cBoW

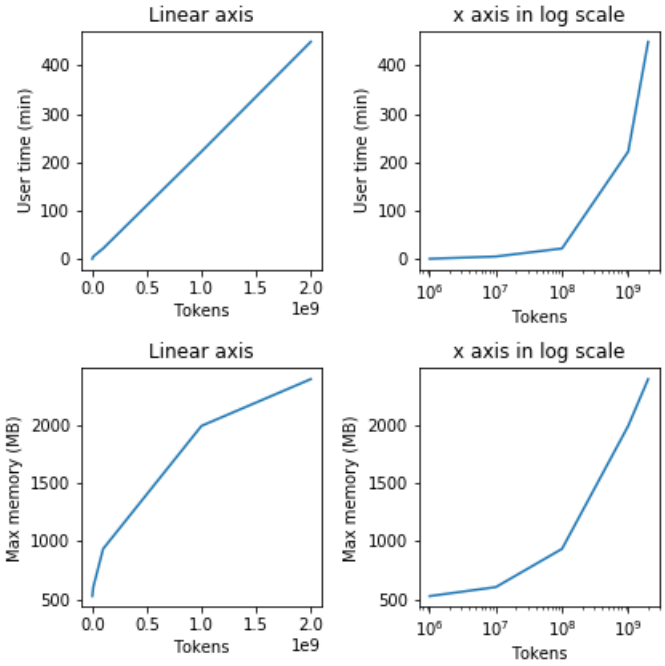


FIGURE 4.2: Computational benchmark of skip-gram

4.1.5.2 Evaluation

The figures (4.3 and 4.4) show how the accuracy of the word embedding scales with respect to the amount of tokens in the training corpus. The evaluation was performed by comparing the word embeddings with the UMLS triplets. It is important to mention that for smaller corpora there's a significant portion of the test words that are unknown to the model. We include both the total accuracy and the accuracy by

ignoring any triplet that has an unknown word.

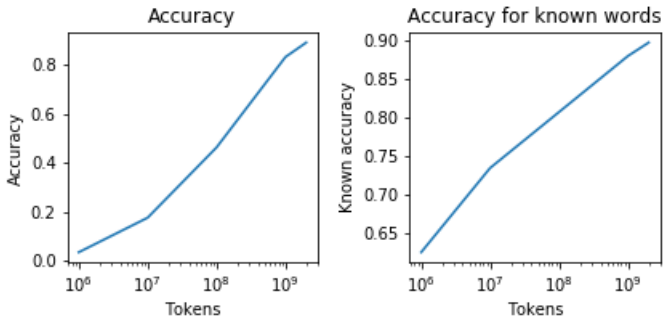


FIGURE 4.3: Evaluation of cBoW

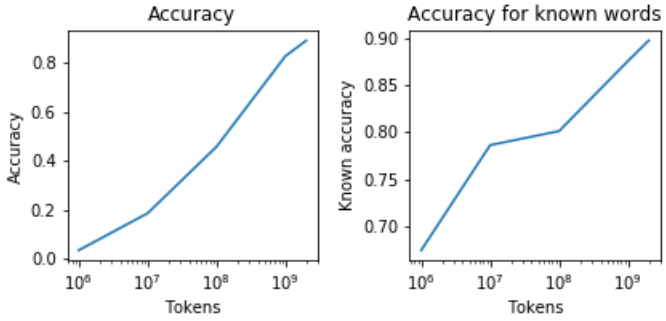


FIGURE 4.4: Evaluation of skip-gram

The results here are also very similar for both models. The results seem to improve logarithmically in both cases. In isolation, the performance of Word2Vec for scientific vocabulary seems to be excellent, even without hyperparameter tuning. Consider that the training and testing data where extracted from totally different sources and that there are 3,649 test triplets, which translates to 7,298 comparisons. A 90% accuracy in such conditions is worth noting. The exact accuracies can be seen in table 4.1.

Word2Vec evaluation	1M	10M	100M	1B	2B
cBoW - Total	0.03	0.17	0.46	0.83	0.89
Skip-gram - Total	0.04	0.18	0.46	0.83	0.89
cBoW - Known words	0.67	0.73	0.80	0.85	0.90
Skip-gram - Known words	0.67	0.79	0.80	0.88	0.90

TABLE 4.1: The Word2Vec evaluation accuracies by the number of tokens used for training, including total accuracy and known word accuracy.

4.2 GloVe

It is safe to say that GloVe (Pennington, Socher, and C. Manning, 2014) is the second most well known word embedding algorithm, after Word2Vec. Standford proposes a different take on the same underlying concept. The algorithm itself does not have to many similarities on the surface, but Word2Vec and GloVe are two sides of the

same coin. Because of this, both are currently considered equivalent for all intents and purposes. They both perform similarly on most tasks, although the popular perception seems to be that GloVe is marginally faster to train.

4.2.1 Intuition

In the original paper (Pennington, Socher, and C. Manning, 2014) the authors distinguish two model families for training word embeddings: global matrix factorization methods (such as LSA) and local context window methods (such as Word2Vec). They claim that both suffer significant drawbacks. While context window methods like Word2Vec perform well on the analogy task, they do not take advantage of global word co-occurrence statistics. The main motivation for GloVe is to find a middle ground: an algorithm that acts on global statistics but achieves the same vector space semantic structure as Word2Vec (analogies). GloVe also surfaces the reason why these kinds of semantic vector structures are created, by making the properties co-occurrence probabilities explicit.

Their main measure for similarity is co-occurrence probability. This can be better understood with an example, and the original paper gives an excellent one.

Consider two related words $i = ice$ and $j = steam$ (see table 4.2). We can examine the relationship between these words by looking at their co-occurrence with a set of k probe words, $\frac{P_{ik}}{P_{jk}}$. If we imagine that i and j define a semantic axis of physical state, we can observe that the word $k = solid$ we observe that for the specified ratio will be large, meaning that it is strongly positioned at the positive end of the scale. Likewise, we observe that for the word $k = gas$, the fraction will be small, indicating that it is semantically at the other end of the scale. Unrelated words such as $k = fashion$ will display a value close to one. The same happens with the word $k = water$ which is obviously highly related to both words, but it is redundant and irrelevant to express the relationship between both words, which is, again, degree of physical state.

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
$P(k\|ice)$	$1.9x10^{-4}$	$6.6x10^{-5}$	$3.0x10^{-3}$	$1.7x10^{-5}$
$P(k\|steam)$	$2.2x10^{-5}$	$7.8x10^{-4}$	$2.2x10^{-3}$	$1.8x10^{-5}$
$P(k\|ice)/P(k\|steam)$	8.9	$8.5x10^{-2}$	1.36	0.96

TABLE 4.2: From the original paper (Pennington, Socher, and C. Manning, 2014)

GloVe formalizes the phenomenon described above and trains the embeddings so that they simulate such a structure. This may seem a bit convoluted at first, it is hard to see directly where the similarity distance property comes from. But if you follow the maths, as we will do below, you can clearly see that the initial equation directly leads to optimizing the dot product between a word vector and its context vectors to be as close as their co-occurrence probability as possible.

We propose that the reason why GloVe and Word2Vec perform so similarly, is that they are essentially optimizing the same objective. They both act under the assumption that words with similar contexts have similar meaning.

The authors of GloVe initially claim that Word2Vec does not take advantage from global statistics, but sequentially scanning the corpus does implicitly capture those

statistics, as more frequent evidence of similarity further reinforces distances between embeddings. What GloVe does with explicit probabilities, Word2Vec does by actually encountering text structures with different frequencies.

Although the GloVe paper starts criticizing Word2Vec for not using the potential of global statistic, later on in the paper the authors do emphasize this close relationship between both models.

4.2.2 The maths

After defining an intuition for the algorithm, we will dive deeper into the maths of the optimization problem the algorithm is solving. We will follow the reasoning from the original GloVe paper, but we will try to summarize and simplify as much as possible.

Above we have described a property in co-occurrence probabilities that can be used to define semantic relationships. We have established a semantic axis with the words $i = ice$ and $j = steam$ and we put them against a set of probe words k to observe how they relate to this axis. This concept is formalized with the following equation.

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}}$$

This equation describes the target optimization problem. In short, a yet undefined F function should be applied to the word vectors and the output of F should approximate the probability fraction.

There is a vast number of possibilities for F, but the paper proposes a few properties that simplify the equation and removes the need to directly specify F.

First, F is reduced to taking a single scalar parameter for simplicity. Given the inherent linear nature of a vector space, the difference and dot product operations are chosen to combine the arguments.

$$F((w_i - w_j)^T \tilde{w_k}) = \frac{P_{ik}}{P_{jk}}$$

If we require F to be a homomorphism between the groups $(\mathbb{R}, +)$ and $(\mathbb{R}, X)$ we see that F can only be $F = exp$.

$$F((w_i - w_j)^T \tilde{w_k}) = \frac{F(w_i^T \tilde{w_k})}{F(w_j^T \tilde{w_k})}$$

$$F(w_i^T \tilde{w_k}) = P_{ik} = \frac{X_{ik}}{X_i}$$

$$w_i^T \tilde{w_k} = \log P_{ik} = \log X_{ik} - \log X_i$$

Where X_i is the total frequency of word i and X_{ik} is the count of the instances where i is in the context of k.

This equation can be further simplified by observing that $\log X_i$ is not dependent on k, so it is replaced by a bias b_i. For symmetry, another $\tilde{b_k}$ bias is added.

$$w_i^T \tilde{w_k} + b_i + \tilde{b_k} = \log X_{ik}$$

If we stop to look at this transformation, we will see that the optimization problem is actually quite simple. $F(w_i^T \tilde{w_k}) = P_{ik}$ implies that the dot product between two vectors is optimized to correlate to the probability of their co-occurrence. This is what we where referring to when we discussed the intuition and what links this algorithm to Word2Vec.

This equation is already simple enough, but there are two more factors that could be improved.

First, $\log X_{ik}$ diverges when there are no co-occurrences between i and k. To fix this the standard $\log x \rightarrow \log 1 + x$ is made. Second, very frequent co-occurrences are not too relevant and mostly contribute to increased noise. A weighting function $f(X_{ij})$ is added to compensate for this. There are many options for such an f but the authors converged on the following function.

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

The authors fixed $x_{max} = 100$ and they empirically found the most appropriate $\alpha = 3/4$. Curiously the same scaling was found to be the best for Word2Vec.

The last step is to just convert the equation into a least squares optimization problem. This is the final GloVe model.

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w_k} + b_i + \tilde{b_k} - \log(1 + X_{ik}))^2$$

4.2.3 Computational complexity

While typical window based methods like Word2Vec scale on corpus size, GloVe scales on vocabulary size. This is because GloVe keeps a co-occurrence matrix for all word pairs and uses this matrix for training. Therefore, a simple upper bound to complexity would be $O(|V|^2)$, V being the vocabulary.

Having a word embedding algorithm that scales on vocabulary size is very handy. After all, the vocabulary size stops growing on bigger corpora. Thanks to this, we could use arbitrarily large corpora for arbitrarily accurate co-occurrence statistics. However, it is common for real-world vocabularies to have hundreds of thousands of words, which would scale up to hundreds of billions in complexity, which is bigger than almost all current corpora.

It is possible, however, to establish a tighter upper bound in complexity. A typical co-occurrence matrix is bound to be sparse if the co-occurrence window is small enough. Therefore, the complexity will depend on the nonzero elements of the matrix, not all.

The authors take the two following assumptions to approximate the number of nonzero elements in the matrix:

1. The number of words in the corpus is proportional to the sum over all elements in the co-occurrence matrix. It is trivial to see that the corpus size should be double the sum.
2. The co-occurrence of two words can be approximated by a power-law function of the frequency rank of that word pair. Power-law patterns in language statistics are a well observed phenomena and this kind of modeling is standard in NLP.

Parting from those assumptions the authors arrive to a model of nonzero elements.

The authors mention that their training corpora was best modeled by $\alpha = 1.25$. Because of this, they conclude that GloVe's complexity upper bound is $O(|C|^{0.8})$, C being the corpus. This explains the popular perception that GloVe is usually slightly faster than Word2Vec, because the complexity of window based methods is approximately $O(|C|)$.

4.2.4 Experimental review

In this section we present all the experimental results from applying GloVe to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We use the standard embedding size of 100. We have also chosen to perform benchmarks with different sizes of corpora, so that we can inspect how the algorithm scales, both in terms of the computational resources needed and the accuracy of the evaluation. The original implementation given by the authors[2] was used in all the benchmarks.

4.2.4.1 Computational benchmark

Figure 4.5 shows the execution time and maximum memory usage of the training procedure of GloVe.

Much like with Word2Vec we see that clearly GloVe scales linearly on the corpus size. Both in time and memory, the scaling pattern of Word2Vec and GloVe is almost identical. However, GloVe needs around double the memory and almost six times the processing time. This directly contradicts the popular believe in the academia that GloVe is equivalent or slightly faster than Word2Vec.

4.2.4.2 Evaluation

Figure 4.6 shows how the accuracy of GloVe scales with corpus size. Again, we both show the total accuracy and the accuracy when ignoring test samples that contain unknown words.

Both the scaling pattern and the total accuracies are remarkably similar to Word2Vec's. This is consistent with the popular perception that Word2Vec and GloVe produce word embeddings with almost the same quality. The close correlation makes sense, as both algorithms essentially perform the same operation through different means. Table 4.3 shows the exact accuracies from evaluation.

[2]GloVe repository: https://github.com/stanfordnlp/GloVe

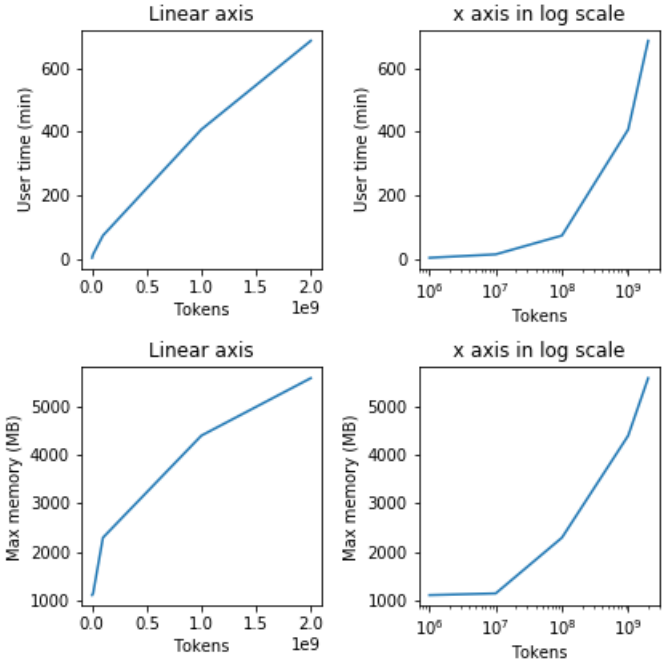


FIGURE 4.5: Computational benchmark of GloVe

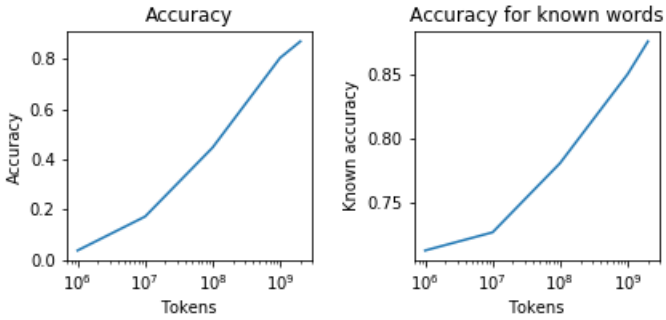


FIGURE 4.6: Evaluation accuracy of GloVe

GloVe evaluation	1M	10M	100M	1B	2B
Total accuracy	0.04	0.17	0.45	0.80	0.87
Known words	0.71	0.73	0.78	0.85	0.88

TABLE 4.3: The GloVe evaluation accuracies by the number of tokens used for training, including total accuracy and known word accuracy.

4.3 FastText

FastText (Bojanowski et al., 2016) is one of most recent mayor advances in word embedding algorithms. It was published this year (2017), again, by a group supervised by Tomas Mikolov, like Word2Vec, but this time at Facebook AI Research.

4.3.1 Intuition

The main contribution of FastText is to introduce the idea of modular embeddings. They are not the first ones to implement modular embeddings, there has been extensive work on morphologically sensitive embeddings during the last few years (Lazaridou et al., 2013; Luong and C. D. Manning, 2016; Botha and Blunsom, 2014; Qiu et al., 2014). However, they recently used this approach to beat the state-of-the-art in most mayor word multilingual similarity datasets while also reducing computational cost. Even if they did not invent the concept, they have showed its potential to the community.

The basic idea behind modular embeddings is that instead of computing an embedded vector per word, a vector is computed for subword components, usually n-grams, which are later combined by a simple composition function to compute the final word embeddings. This approach has multiple advantages.

One advantage is that the vocabulary tends to be considerably smaller when working with large corpora, which makes the algorithm more computationally efficient compared to the alternatives, although embedding retrieval is usually slower because of the need to compose. It also means that each component in the vocabulary will be repeated more, so less training data is needed.

Another big advantage is that subword information, such as morphological variations, are captured correctly. Other word embedding algorithms just take standard tokens as words, and may create different embeddings for morphological variations, which increases the noise. In the case of FastText, morphological variations keep most of their common components and have slight alterations applied to their embeddings based on the differences, such as different prefix or suffix. Using n-grams to capture morphological features may seem crude, as there are other models that do explicit morphological segmentation. This is intentional, though, as the simplicity of the method also increases generality. It is thanks to this generality that FastText performs well in vastly different languages. Thanks to dealing with subword components, it is also possible to reliably predict the embedding of previously unseen variations.

FastText is also a big step towards solving the problem of word embedding compositionality. If subword embeddings can be composed into word embedding, why not compose embeddings to create phrase embeddings? Or document embeddings? Indeed, even if it is not emphasized in the original paper, the official implementation includes a document embedding functionality, which will be reviewed in the next section. Phrase embeddings are also specially interesting in the scientific domain, as many technical concepts are identified by multi-word terms.

4.3.2 The maths

FastText uses the skip-gram model with negative sampling proposed for Word2Vec, which has, by now, become a standard base algorithm for many embedding techniques. The skip-gram model was explained in detail in the Word2Vec section 4.1.3.

The following is the specific skip-gram loss function used by FastText.

$$\sum_{t=1}^{T}[\sum_{c\in C_t} l(s(w_t, w_c)) + \sum_{n\in N_{t,c}} l(-s(w_t, n))]$$

Given a word vocabulary of size W where each word is represented with an index $w \in \{1, ..., W\}$. The model is trained on a corpus of size T with a sequence of words $w_1, ..., w_T$. A sliding window technique is used, where C_t is the set of word indexes surrounding a w_t target word from the corpus. To perform the negative sampling, a random set of $N_{t,c}$ words is selected from the vocabulary for each window. $l(x) = \log(1 + e^{-x})$ represents the logistic loss function. s is a similarity measure between two words, in the case of vanilla skip-gram, $s(w_t, w_c) = u_{w_t}^T v_{w_c}$, where u and v are the sets of embedding vectors for each target word and context respectively.

The main difference between Word2Vec and FastText is the subword model. Each word is subdivided into a series of n-grams. Given a vocabulary of G n-grams, $G_w \subset 1, ..., G$ is the set of n-grams contained in word w. In practice, for FastText, special boundary symbols $<$ and $>$ are added to all words at the beginning and the end respectively, to distinguish n-grams that are part of the prefix and suffix. Let us denote a word with boundary symbols as an augmented word. G_w will contain the full augmented word and all n-grams for $3 \leq n \leq 6$.

The n-grams are stored in a hash map of size $K = 2 \cdot 10^6$ to bound the memory requirements. The Fowler-Noll-Vo variant 1a hash function is used.

To complete the model the s similarity function is replaced to include the subword embedding composition. The composition is done by simply summing up all the pairwise dot product similarities between the target n-grams and the context vector. Let z_g be the embedding vector corresponding to the n-gram g.

$$s(w, c) = \sum_{g\in G_w} z_g^T v_c$$

4.3.3 Computational complexity

The computational complexity of FastText is effectively the same as the complexity of the skip-gram variant of Word2Vec as the learning procedure is effectively the same (section 4.1.4).

The main difference, again, is the added cost of splitting each word into its components, fetching their corresponding embedding vectors and compose them into the final word embedding. Let $|G_w|_{avg}$ be the average number of n-grams per word. This is only a linear increase in cost, as $|G_w|_{avg}$ would barely change in different corpora if they are big enough, so the complexity class stays the same. When a word embedding would be fetched in the hash map ($O(1)$) in vanilla skip-gram, in FastText $|G_w|_{avg}$ vectors would have to be fetched and composed ($O(2|G_w|_{avg})$).

4.3.4 Experimental review

In this section we present all the experimental results from applying FastText to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We

use the standard embedding size of 100. We have also chosen to perform benchmarks with different sizes of corpora, so that we can inspect how the algorithm scales, both in terms of the computational resources needed and the accuracy of the evaluation. The original implementation given by the authors[3] was used in all the benchmarks.

4.3.4.1 Computational benchmark

Figure 4.7 shows the execution time and maximum memory usage of the training procedure of FastText.

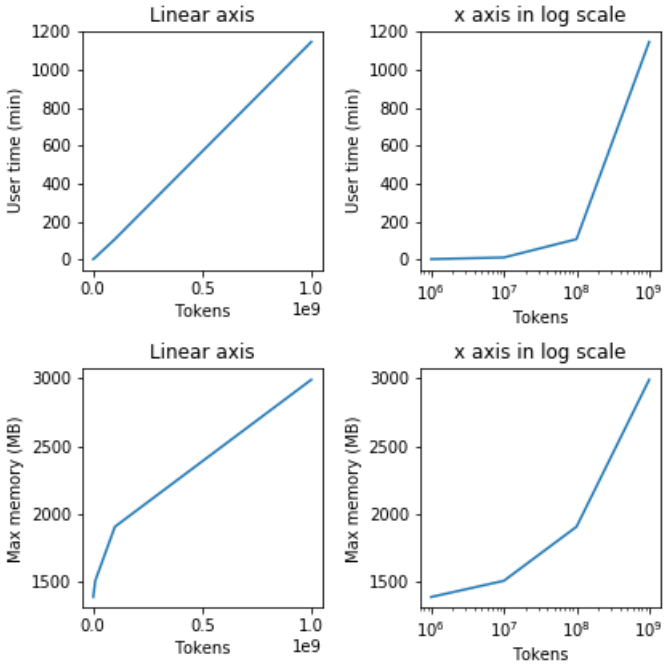


FIGURE 4.7: Computational benchmark of FastText

The scaling pattern is the same as with Word2Vec and GloVe, which is consistent with what we have seen in the theoretical analysis. However, it is twice as slow as GloVe and three times as slow as Word2Vec. Memory consumption is not too high though. This is surprising as the embedding vocabulary of FastText should be much bigger, as it stores both full word embeddings and n-gram embeddings. However, it sits squarely between GloVe and Word2Vec in memory requirement.

4.3.4.2 Evaluation

Figure 4.8 shows how the accuracy of FastText scales with corpus size. In the case of FastText, it does not make sense to distinguish the total accuracy and the known word accuracy, as FastText can infer the embedding of new words using n-gram embeddings.

FastText seems to be considerably superior to Word2Vec and GloVe. The accuracy at 1B tokens is notably higher than the accuracy of the other two with 2B. It also

[3]FastText repository: https://github.com/facebookresearch/fastText

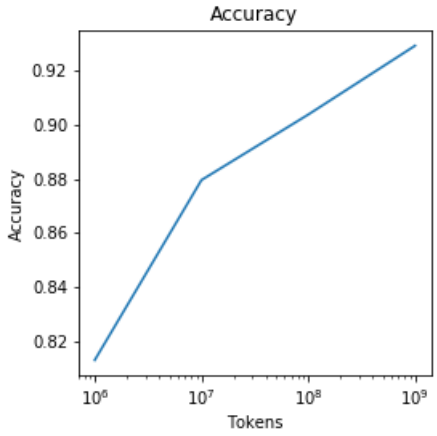


FIGURE 4.8: Evaluation accuracy of FastText

considerably outperforms the other two models in smaller corpora. From previous test we know that when training on a 1M token corpus only 5% of the test triplets are known. FastText achieves a 81% accuracy in this case while having to infer the rest of the 95% words only using n-grams.

It is possible that FastText may be specially effective on scientific text, as most technical words are derivative and share many of the same subword components. It is worth noting that all the work we have done has been with stemmed words. It is very likely that FastText would compared much better if we were working with non normalized words.

Table 4.4 shows the exact evaluation accuracies.

FastText evaluation	1M	10M	100M	1B
Accuracy	0.81	0.88	0.90	0.93

TABLE 4.4: The exact accuracies of FastText.

4.4 WordRank

WordRank (Ji et al., 2015) is a state-of-the-art word embedding algorithm. It is in many ways similar to the algorithms described until now, particularly it uses a context window to scan through corpora and optimize its word representations. However, the function WordRank optimizes for is different and novel. Inspired by standard evaluation methods, WordRank trains embeddings such that for each target word, all its context words are ranked by relevance. It is designed to be optimal for retrieving the most similar words to any target word. It also optimizes for precise distinction between the highest ranked similar words. Thanks to these features, WordRank achieves very similar performance to the rest of the state-of-the-art embedding algorithms while using much smaller noisy corpora.

4.4.1 Intuition

Almost all algorithms in the current wave of word embedding algorithms have a common framework. In essence, all these algorithms perform a matrix factorization operation on a matrix that relates words to each other. This matrix can be a co-occurrence matrix, like with GloVe, or a point-wise mutual information matrix, like with Word2Vec. GloVe does this factorization explicitly, but the rest usually do it implicitly by scanning corpora with a fixed size window, where the central word is distinguished as the target and the rest are the context. In the end, they train word embeddings such that their pairwise dot products approximate this matrix.

WordRank keeps the window based training framework, but optimizes for a different kind of similarity measure. Instead of approximating a pairwise measure between target and context words, it approximates a ranking of contexts per target word. In other words, all context words are ranked by relevance to each target word and WordRank enforces this ordering. This decision was inspired by how word similarity is usually evaluated using human similarity datasets. This priority for ranking is also a desirable feature for many higher level tasks.

Focusing on ranking makes WordRank much more resilient to noise. Thanks to this, it is able to perform as well as other state-of-the-art algorithms with much smaller and diverse corpora. The authors mention an example where a WordRank model trained on 17M tokens perform almost as well as a Word2Vec model trained on 7.2B. Granted, WordRank does not perform much better than the alternatives with big corpora. In any case, this robustness to noise can be very desirable for many use-cases.

WordRank goes further and tunes the balance of resolution in the ranking. The authors observe that it is more desirable to have a well distinguished ranking between the most similar words, while more different words do not need as much polishing. This is consistent with human intuition. We can clearly distinguish and rank highly related words but the difference in similarity between two very distant concepts is highly ambiguous. This is achieved by selecting a concave loss function, where loss is most sensitive when rank is small (at the top of the list).

4.4.2 The maths

Like most other word embedding algorithms, WordRank scans the corpus with a fixed sized window. The central word in the window is called a target and is denoted as w. The rest of the words are the context c. ω is the set of all target-context pairs in the corpus. ω_w will contain all the context words paired with w. Similarly ω_c contains all the words paired with a context c.

Also like most other word embedding algorithms, WordRank approximates the matrix factors U and V. U contains all the embeddings for targets denoted u_w and V contains all the embeddings for contexts v_c.

The dot product between two embeddings $< u_w, v_c >$ is interpreted as a relevance score of the pair. Therefore, we compute the rank by counting the number of context words that are more relevant than the chosen context.

$$rank(w,c) = \sum_{c' \in C\{c\}} I(< u_w, v_c > - < u_w, v_{c'} \leq 0) = \sum_{c' \in C\{c\}} I(< u_w, v_c - v_{c'} > \leq 0)$$

Given a function I that returns 1 or 0 depending on the condition being true or false. I is not differentiable, so, like in neural binary classification tasks, a differentiable upper bound is selected. In this case, the logistic loss $l(x) = \log_2(1 + 2^{-x})$ is used.

$$rank(w,c) \leq \bar{rank}(w,c) = \sum_{c' \in C\{c\}} l(< u_w, v_c - v_{c'} >)$$

Finally, the following objective function is chosen as a ranking loss.

$$J(U,V) = \sum_{w \in W} \sum_{c \in \omega_w} r_{w,c} \cdot \rho(\frac{\bar{rank}(w,c) + \beta}{\alpha})$$

$r_{w,c}$ is a association measure between a (w,c) pair. This same weighting has been mentioned before for some of the other word embedding models.

$$r_{w,c} = \begin{cases} (X_{w,c}/x_{max})^{\epsilon} & \text{if} X_{w,c}/x_{max} \\ 1 & \text{otherwise} \end{cases}$$

Where $X_{x,c}$ is the co-occurrence count matrix. For WordRank $x_{max} = 100$ and $\epsilon = 0.75$ are chosen.

ρ is a monotone concave ranking loss function. It is concave so that the loss is most sensitive at the top of the ranking. Any loss function with these properties is valid, several options are explored in the original paper. In practice, the summation over the context in the objective function is hard to compute when ρ is non-linear, so a first-order Taylor expansion is used as an approximation.

Finally, α and β are hyperparameters that tune the balance in accuracy resolution. As mentioned before, WordRank focuses on distinguishing lower rank elements (at the top of the list) by using a concave loss. α and β tune the scale of this contrast and how fast the focus decays as it goes down the list. α defines scale and β defines the offset of decay. The authors report that $\alpha = 100$ and $\beta = 99$ is the best setting for their experiments.

4.4.3 Experimental review

In this section we present all the experimental results from applying WordRank to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We use the standard embedding size of 100. We have also chosen to perform benchmarks with different sizes of corpora, so that we can inspect how the algorithm scales, both in terms of the computational resources needed and the accuracy of the evaluation. The original implementation given by the authors[4] was used in all the benchmarks.

[4]WordRank repository: https://github.com/shihaoji/wordrank

4.4.3.1 Computational benchmark

Figure 4.9 shows the resource consumption of the training process of WordRank. It is worth noting that WordRank reuses part of GloVe's implementation to produce a co-occurrence matrix.

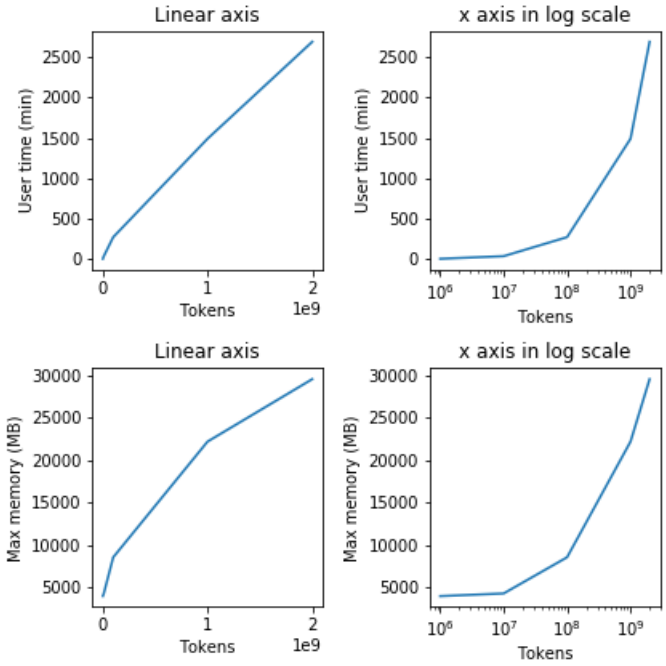


FIGURE 4.9: Computational benchmark of WordRank

There is nothing much remarkable about this analysis compared to the rest of word embedding algorithms. The scaling seems mot similar to GloVe's, most likely because of the reuse of the code. However, it looks like it has needed 4 times the resources for training.

4.4.3.2 Evaluation

Figure 4.10 shows how the accuracy of WordRank scales with corpus size. Again, we both show the total accuracy and the accuracy when ignoring test samples that contain unknown words.

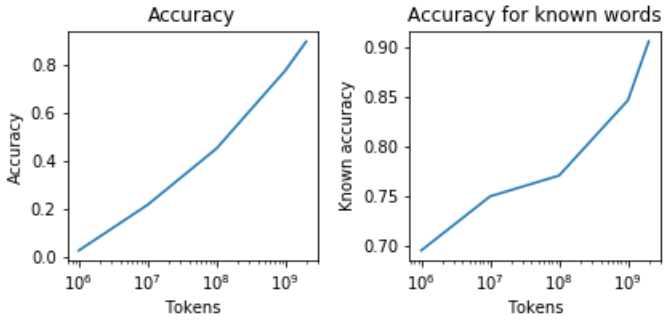


FIGURE 4.10: Evaluation accuracy of WordRank

For all intents and purposes, the results of WordRank are equivalent to Word2Vec and Glove. It looks like the advantage of WordRank on smaller corpora is not apparent for our dataset. Table 4.5 shows the exact accuracies from evaluation.

WordRank evaluation	1M	10M	100M	1B	2B
Total accuracy	0.02	0.21	0.45	0.78	0.89
Known words	0.69	0.75	0.77	0.84	0.90

TABLE 4.5: The WordRank evaluation accuracies by the number of tokens used for training, including total accuracy and known word accuracy.

4.5 Summary and conclusions

In this chapter we have analyze four pivotal state-of-the-art word embedding algorithms.

We first explored **Word2Vec**. You could say that Word2Vec is one of the main factors for the recent transformation of NLP towards Machine Learning. It was not the first word embedding algorithm by any means, there were well known neural models since the early 2000s. However, Word2Vec proved that a shallow model that can go through more training data is considerably superior to more complex deep models. It was also one of the first algorithms to produce embeddings of high enough quality to use as the foundation of many other tasks.

Word2Vec establishes a formula that is replicated by many algorithms after it. It computes word embeddings by scanning a fixed sized window over the corpus. A window has a central target word and a set of context words. Each window is a sample for SGD, where the loss of a window brings the target and context vectors closer.

GloVe was one of the next important word embedding algorithms. The authors realized that Word2Vec was factoring a PMI word-word matrix implicitly. They argued that window based methods do not capture global language statistics well enough. Therefore, they proposed factoring a co-occurrence matrix explicitly.

FastText is another very well known word embedding algorithm. It followed Word2Vec's window based framework, but it popularized the idea of modular embeddings by introducing subword components. Apart from having a single vector per word, FastText keeps embeddings of n-grams and it combines them together for each word. This introduces a minimal added overhead, but it considerably improves the model's practicality. In fact, FastText effectively outperformed all the other algorithms in our tests, likely because scientific terminology is even more regular than general purpose text.

The last word embedding algorithm we have explored is **WordRank**. It takes a somewhat different approach from the rest by optimizing nearest neighbor ranks instead of co-occurrence statistics. They also tune the loss function such that there is more resolution between words that are close to each other. This is supposed to make WordRank perform much better than the other algorithms in smaller corpora,

but the our benchmarks have not reflected that. In fact, even if the mindset is different, it turns out that WordRank performs a very similar operation to the rest, and the test show exactly that.

Overall, we see that most algorithms for word embedding in the current state-of-the-art closely follow the Word2Vec model. There are many alternatives with different tweaks, but most algorithms perform equivalently in most cases. The four years since the publication of Word2Vec have produced a few breakthroughs as well though, as exemplified by FastText. All the evaluation results have been unified in 4.6 for reference and to make comparisons easier.

Word embeddings	1M	10M	100M	1B	2B
W2V cBoW - Accuracy	0.03	0.17	0.46	0.83	0.89
W2V cBoW - Known	0.67	0.73	0.80	0.85	**0.90**
W2V Skip-gram - Accuracy	0.04	0.18	0.46	0.83	0.89
W2V Skip-gram - Known	0.67	0.79	0.80	0.88	**0.90**
GloVe - Accuracy	0.04	0.17	0.45	0.80	0.87
Glove - Known	0.71	0.73	0.78	0.85	0.88
FastText - Accuracy	**0.81**	**0.88**	**0.90**	**0.93**	-
WordRank - Accuracy	0.02	0.21	0.45	0.78	0.89
WordRank - Known	0.69	0.75	0.77	0.84	**0.90**

TABLE 4.6: All results from word embedding evaluation unified.

Chapter 5

Review of document similarity measures

In this chapter we review the state-of-the-art in Semantic Textual Similarity (STS). We focus on unsupervised algorithms that only use the information extraction done by word embedding to compute semantic similarities between sentences, paragraphs or documents.

Like with word embedding algorithms, we provide an intuition, a formal specification, a complexity analysis, and a benchmark for each model.

5.1 Baseline: VSM and embedding centroids

It is still early days in solving the problem of Semantic Textual Similarity (STS). A few novel solutions have been published during the last few years with the raise of word embeddings and deep learning. These new algorithms often report beating the state-of-the-art in certain select tasks, but the community has not yet seen groundbreaking results, specially with longer text at the paragraph and document levels. More classical document similarity methods still stand relevant and competitive, so we would be remiss to ignore them in a comprehensive review.

We establish a baseline with a few simple but effective methods and check how they compare to more modern solutions at different text sizes.

5.1.1 Baseline algorithms

VSM We start by resorting to the standard Vector Space Model (VSM). It is a simple model that represents each piece of text as a weighted distribution over the vocabulary. Each document is represented as a vector with the size of the vocabulary, where each element gives the frequency of a word in the document or some weight derived from the frequency. Apart from a numerical representation, the VSM also gives a few alternative similarity metrics, being cosine similarity or dot product the most popular ones. This type of text encoding is widely used as input for classical machine learning algorithms and is the de facto representation and similarity metric in information retrieval. The VSM is described in further detail in section 2.4. We experiment with the three mayor weighting schemes: BoW, Tf-idf and BM25.

Embedding centroids We also experiment with a naive approach of integrating word embeddings with the aforementioned classical similarity algorithms. Word embeddings are powerful representations and contain a great deal of contextual information. The intuition is that word embeddings will help in shorter technical text such as titles or abstracts, where exact word overlap may not often be enough. One simple way to compose word embeddings is to directly compute the centroid between all the word vectors. It follows that we may compute a weighted centroid using the VSM vectors as coefficients.

5.1.2 Experimental review

Table 5.1 shows the evaluation results of our baseline methods. Not surprisingly, VSM methods perform poorly on titles. This is simply because titles are short and word overlap is minimal even for related articles, specially in technical domains. However, simple VSM similarity performs very well on longer text, even at the abstract level, where word overlap is considerably less abundant than in article bodies. The advantage of more advanced weighting schemes is apparent, but not remarkable. The improvement also seems to reduce as the document is longer and when using embeddings.

VSM similarity	BoW	Tf-Idf	BM25
Titles	0.59	0.60	0.60
Abstracts	0.90	0.92	0.93
Bodies	0.95	0.95	0.96

TABLE 5.1: Document similarity accuracy for VSM cosine similarity.

For the embedding centroids, we used the Word2Vec Skip-gram embeddings after training on 2B tokens from our corpus. As table 5.2 shows, word embeddings clearly help a lot in cases where exact word overlap is rare, their performance on titles is much better. However, at abstract and body level, the performance of embedding centroids is equivalent or even worse than VSM similarity. It is likely that using embeddings adds some degree of noise and computing a centroid may lead to some information loss. It is also worth noting that embeddings, although harder to compute, are a dense representation. This is a practical advantage over sparse VSM vectors. It also means that, for longer text, VSM representations contain significantly more information than the compressed 100 dimensional embeddings. Embedding centroids perform competitively considering that they are much smaller than article body VSM representations.

Embedding centroids	BoW	Tf-Idf	BM25
Titles	0.91	0.91	0.60
Abstracts	0.91	0.92	0.93
Bodies	0.94	0.94	0.95

TABLE 5.2: Document similarity accuracy by using weighted embedding centroids.

5.2 Doc2Vec

Paragraph Vector (Quoc V. Le and Mikolov, 2014), as it is better known, Doc2Vec is a proposal for paragraph level embeddings from the research team responsible for Word2Vec. Doc2Vec, although well known, has not been as influential as Word2Vec. This is mostly because its performance for cosine similarity comparisons is somewhat underwhelming compared to more classical methods. Doc2Vec embeddings have been more useful as feature vectors of deeper machine learning algorithms instead.

In any case, it is relevant to review it, as it was one of the first steps in the new field of document embeddings. Word embedding compositionality is still an open problem, and Doc2Vec has merit in proposing one of the few unsupervised document embedding algorithms.

5.2.1 Intuition

Like with Word2Vec the authors propose two different models for training document embeddings in an unsupervised manner. In fact, the authors state that skip-gram was the inspiration for PV-DBOW while PV-DM has great resemblance to cBoW.

The Distributed Bag of Words (PV-DBOW) model proposes training the paragraph vectors having them as the input to a simple classification task that tries to predict other words in the paragraph. In practice, this implies minimizing the cosine distance between the document embedding and the word embeddings it contains. This is very similar to Skip-gram, which proposes using the target word to predict the words in a fixed size window around it.

The Distributed Memory model (PV-DM) looks at the problem from another angle. Like with cBoW a context window is used as input to predict a target word. In this case, the context is a fixed size window and the target is the word that comes next, so that some sequential information is preserved. In reality, the context words are either averaged or concatenated and the cosine distance to the next word is optimized. The difference between PV-DM and cBoW is that, along with the context words, the paragraph embedding is also introduced to the mix. Therefore, we can imagine the paragraph embedding as the vector that would correspond to a word that appears in all the contexts of a given paragraph.

In PV-DM the word embeddings are created together with the document embeddings. Intuitively, these word embeddings turn out to be almost equivalent to word2vec embeddings. In any case, the model always gives the option to use precomputed embeddings.

PV-DM usually performs better than PV-DBOW and it is the one that is usually used in practice. Even the original paper notes this, and states that PV-DM is superior on its own, but concatenating the embeddings of both models gives the best results.

5.2.2 The maths

The internals of Doc2Vec are extremely similar to Word2Vec (section 4.1.3). The word embedding training is done in the exact same way, with the same loss function, negative sampling regularization and word subsampling. As mentioned before, PV-DBOW is equivalent to Skip-gram and PV-DM is equivalent to cBoW. The only practical difference is that a paragraph vector is added to the context of each window.

5.2.3 Computational complexity

Doc2Vec has effectively the same complexity as cBoW (section 4.1.4). It is effectively the same algorithm, the only difference is that an extra global paragraph context vector is added to the window.

However, there are two differences that are worth noting. One important note is that in Doc2Vec the memory requirements scale linearly to the number of documents in the training corpus, which is not ideal. Inference of new document embeddings is also costly and potentially inaccurate. New document embeddings are inferred by running a few epochs on the document while locking the word embeddings.

5.2.4 Experimental review

In this section we present all the experimental results from applying Doc2Vec to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We use the standard document embedding size of 800. We have performed three separate scalability benchmarks on titles, abstracts and bodies to see how the algorithm performs with different types of text.

There is no original implementation given by the authors. The only public release of code was a forum post[1] by Tomas Mikolov where he provides an adapted version of the original Word2Vec implementation to train the Doc2Vec model. The accepted standard implementation for Doc2Vec is the one in the Gensim library[2]. This is the implementation we have used for benchmarking.

5.2.4.1 Computational benchmark

Figures 5.1, 5.2 and 5.3 show how the training of Doc2Vec scales for titles, abstracts and bodies respectively.

The most immediate realization is how fast the training is. The algorithm scales linearly on corpus size, like the rest, but the absolute time needed is much lower than alternatives such as Doc2VecC or Sent2Vec. This is somewhat surprising, as both Doc2VecC and Sent2Vec are designed to be shallow and fast and they are both implemented in clean C++ with almost no dependencies. In contrast, the implementation of Doc2Vec that was used was implemented using Python and Cython for the performance critical segments.

[1] Doc2Vec by Mikolov: https://groups.google.com/forum/#!msg/word2vec-toolkit/Q49FIrNOQRo/J6KG8mUj45sJ

[2] Doc2Vec in Gensim: https://radimrehurek.com/gensim/models/doc2vec.html

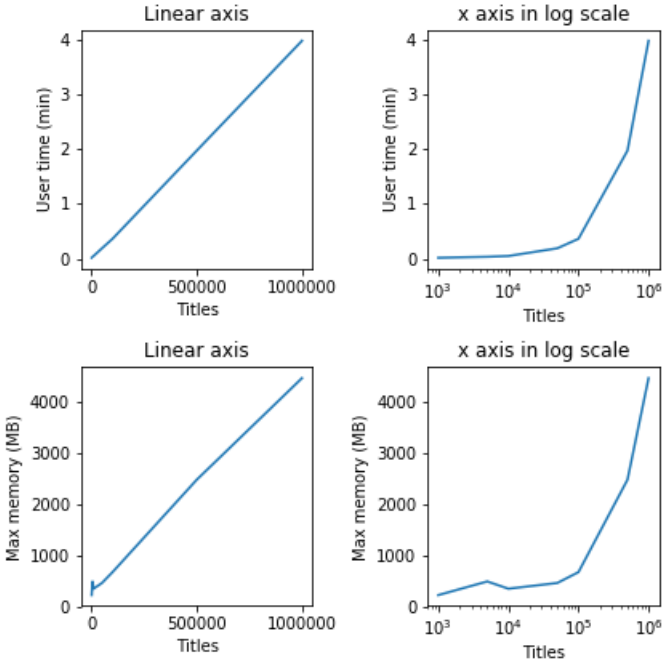


FIGURE 5.1: Computational benchmark of Doc2Vec applied to titles.

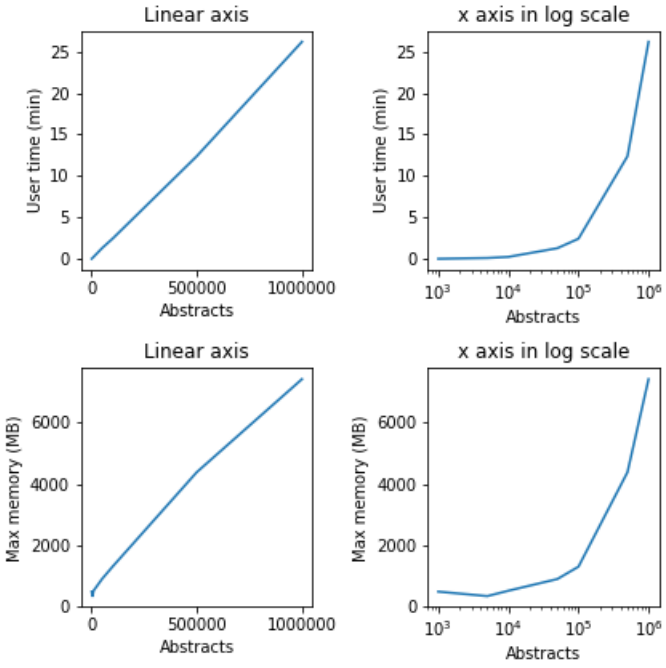


FIGURE 5.2: Computational benchmark of Doc2Vec applied to abstracts.

The memory scaling is very similar to Word2Vec, GloVe and the rest of the window based embedding algorithms. However, it looks more linear, specially with smaller sized documents. This makes sense as the document embedding matrix grows linearly, but the vocabulary most likely is the dominating factor in memory consumption with larger texts. In theory, memory consumption should grow linearly once enough text has been read to have a complete vocabulary. It looks like these tests are

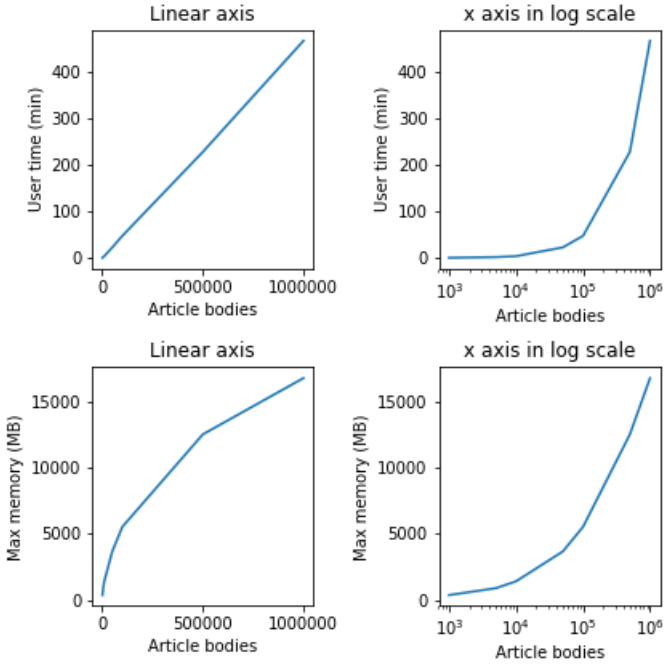


FIGURE 5.3: Computational benchmark of Doc2Vec applied to bodies.

before that point.

5.2.4.2 Evaluation

The following figures 5.4, 5.5 and 5.6 show the accuracies of Doc2Vec embeddings for the document similarity tasks between titles, abstracts and article bodies, respectively. It is worth noting that embeddings of the documents in the testing dataset were not trained during the training phase. They were inferred after the fact instead, which was considered a more realistic setup.

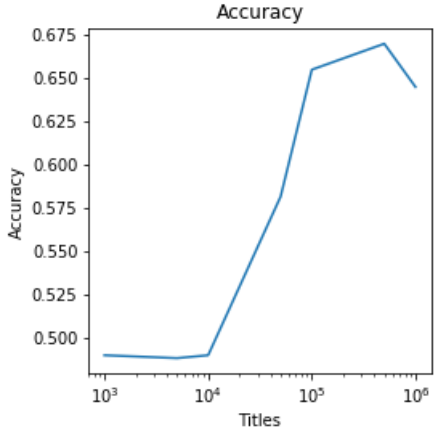


FIGURE 5.4: Accuracy of Doc2Vec applied to titles.

The results are rather poor for titles, which is not surprising. The algorithm is designed to work at the paragraph level and titles are considerably shorter than the

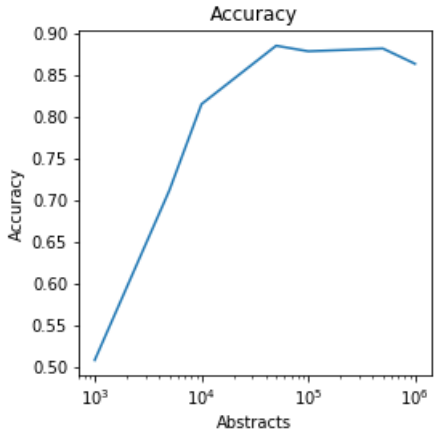


FIGURE 5.5: Accuracy of Doc2Vec applied to abstracts.

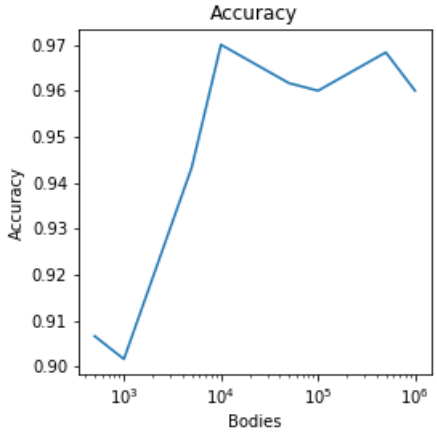


FIGURE 5.6: Accuracy of Doc2Vec applied to bodies.

other types of text, so there were less tokens to learn word embeddings from. The accuracies at the abstract level are better, but even after training on the full dataset (1.1M abstracts) the evaluation is still under the naive baseline. The performance on bodies is surprisingly good, however. Only 500 full article bodies where enough to outperform the accuracy of the other two models. It is worth noting that article bodies are several tens of times bigger than abstracts, but it is still impressive. The last model stays almost at perfect accuracy since the beginning. Table 5.3 shows the exact accuracies.

Doc2Vec evaluation	500	1K	5K	10K	50K	100K	500K	1M
Titles	-	0.49	0.49	0.49	0.58	0.66	0.67	0.65
Abstracts	-	0.50	0.71	0.82	0.89	0.88	0.89	0.86
Bodies	0.91	0.90	0.94	0.97	0.96	0.96	0.97	0.96

TABLE 5.3: Exact accuracies of Doc2Vec.

5.3 Doc2VecC

Doc2VecC (M. Chen, 2017), not be confused with Doc2Vec, stands for Document Vectors through Corruption. It is a recently published unsupervised document embedding algorithm. The Doc2VecC embedding of a document is simply computed by averaging the embeddings of its component words. However, the key aspect of Doc2VecC is that it trains its word embeddings precisely for this purpose. Doc2VecC enforces that a meaningful document representation is formed when averaging. This is done in an unsupervised manner by using corruption, an extension of Word2Vec word subsampling that has been applied on a few document and sentence embedding algorithms lately to aid unsupervised learning.

5.3.1 Intuition

Even if the main purpose of Doc2VecC is to generate document embeddings, it is, in fact, a word embedding algorithm like Word2Vec or FastText. The difference is that the word embeddings are specially trained such that averaging a document yields a meaningful representation with minimal noise.

Doc2VecC's architecture looks strongly like Doc2Vec's. They are both based on the C-BoW Word2Vec model, where a fixed size window is used to scan a corpus. The model learns to predict the central word in the window, the target, given the words around it in the window, the context. In Doc2Vec, document embeddings are trained by attaching a shared vector as a word embedding to every context in a document. After training, this document vector will contain the global information of the document that fills the missing information in local contexts and helps predict the target word. However, Doc2Vec has a serious downside: inferring the embedding for a new document is expensive as the optimization problem needs to be solved again for this new document while keeping the embeddings locked.

Doc2VecC addresses this with a simple modification. Instead of learning these global document vectors from scratch, they are created by averaging all the word embeddings in the document. For the same reason Doc2Vec generates good global representations, Doc2VecC will train word embeddings that are optimal for averaging into a global semantic sense of the document. This makes the model easier to train, because more information is shared. It also trivializes the task of inferring new document embeddings.

Another key aspect of Doc2VecC is the use of corruption. To decrease computational cost Doc2VecC randomly ignores significant parts of the text and deliberately zeros out dimensions from the document embedding while training. Introducing such noise may seem counterintuitive, but it is common practice in many areas of machine learning. It avoids over-fitting, improves generality and creates random variations of the training data to effectively increase the number of data samples and make the most of the dataset. It is equivalent to adding Gaussian noise to linear regressors or using drop-out in deep neural models. The authors also prove that the corruption applies an implicit regularization with desirable properties, such as reducing the influence of very frequent words.

5.3.2 The maths

The implementation of Doc2VecC is very similar to CBoW and Doc2Vec. The main difference is that a distinct global document embedding is generated by averaging and combined with all context windows in that document. It also applies mask-out corruption to that document embedding before doing the combination.

Like with all shallow bilinear embedding algorithms, a matrix factorization is performed, usually implicitly. The goal is to train the factor matrices V and U, where V contains the desired embeddings of target words and U is a side product that contains the embeddings of context words.

Training is done by passing a fixed size window through the corpus. The central word in the window is the target w^t and has the corresponding embedding vector v_w. The rest of the words around it are averaged into the context embedding c^t. For each document, a document embedding x_d is also created by averaging all the words in it.

Corruption is then applied on x_d through an unbiased mask-out. The mask-out is performed by zeroing out some dimensions of x_d with probability q and normalizing the rest to remove any bias:

$$\bar{x}_d = \begin{cases} 0 & \text{with probability q} \\ \frac{x_d}{1-q} & \text{else} \end{cases}$$

5.3.3 Experimental review

In this section we present all the experimental results from applying Doc2VecC to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We use the standard document embedding size of 800. We have performed three separate scalability benchmarks on titles, abstracts and bodies to see how the algorithm performs with different types of text.

We use the original implementation given by the authors[3]. The implementation is based on the first available implementation of Doc2Vec by Tomas Mikolov[4], which, in turn, is an adaptation of the original Word2Vec code.

5.3.3.1 Computational benchmark

Figures 5.7, 5.8 and 5.9 show how the training of Doc2VecC scales for titles, abstracts and bodies respectively. Unsurprisingly, the scaling pattern is very similar to both Doc2Vec and Word2Vec, as the implementation is just an adaptation of both. The main difference is in scale, Doc2VecC has needed between on average 12 times longer to train approximately. Although this may be simply because they were implemented using different technologies.

[3] Doc2VecC repository: https://github.com/mchen24/iclr2017

[4] Doc2Vec by Mikolov: https://groups.google.com/forum/#!msg/word2vec-toolkit/Q49FIrNOQRo/J6KG8mUj45sJ

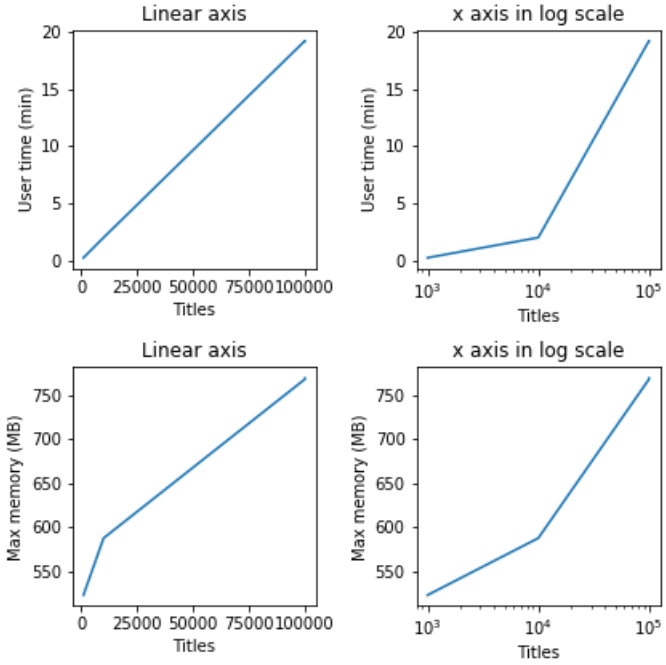


FIGURE 5.7: Computational benchmark of Doc2VecC applied to titles.

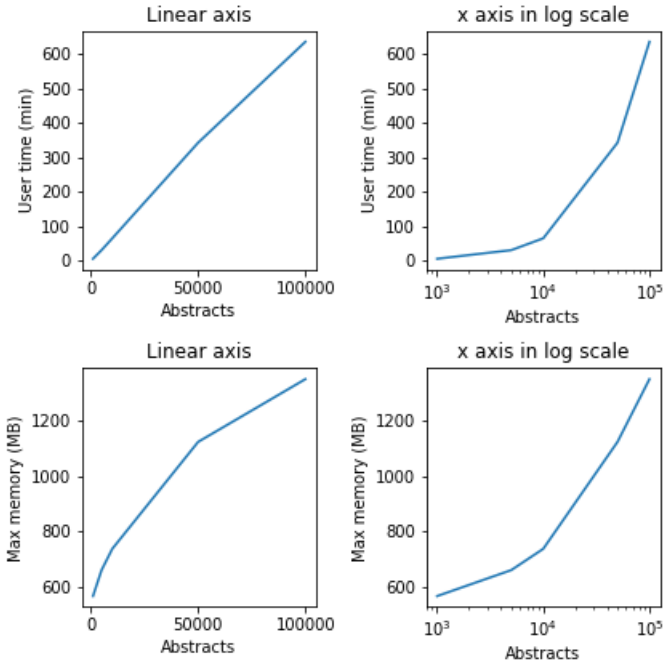


FIGURE 5.8: Computational benchmark of Doc2VecC applied to abstracts.

5.3.3.2 Evaluation

Figures 5.7, 5.8 and 5.9 show the accuracy of Doc2VecC on titles, abstracts and bodies respectively. Although the embeddings of the test documents were inferred in the same run as the training, their contents were not used as training data and the inference was done in isolation. The internals of the implementation were checked

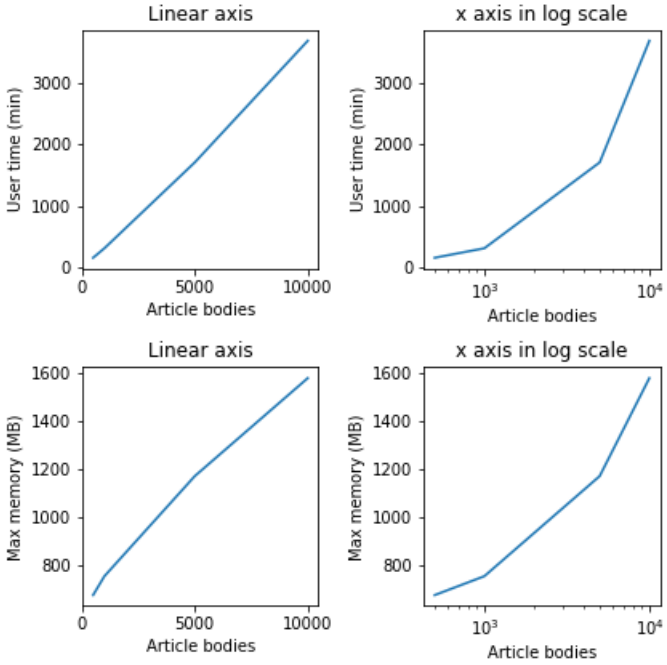


FIGURE 5.9: Computational benchmark of Doc2VecC applied to bodies.

to make sure of it.

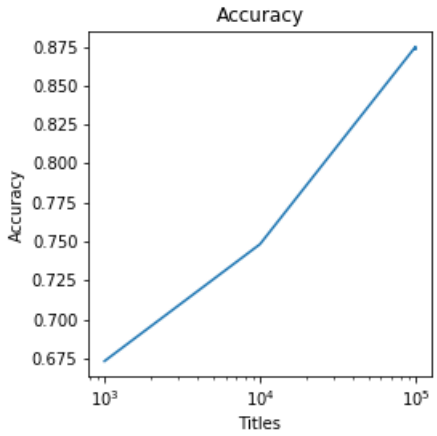


FIGURE 5.10: Accuracy of Doc2VecC applied to titles.

Compared to Doc2Vec, Doc2VecC seems to be more consistent across the board. We see the same trend of improved results with bigger documents, most likely because the algorithm was designed for paragraphs and full documents, and because there are more tokens to train word embeddings in longer text. However, the contrast between types of text is not as big in this case. The evaluations with titles are still below the baseline, but they are not bad accuracies. Abstract accuracies manage to achieve the same results as the baseline. Finally, body evaluations are very good overall, even with low training data, but not as good as with Doc2Vec.

Table 5.4 shows the exact accuracies on evaluation. The gaps in the table are due to the high cost of training Doc2VecC. Training abstracts and bodies with the whole

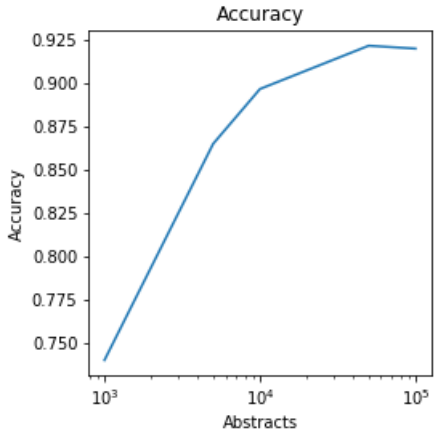


FIGURE 5.11: Accuracy of Doc2VecC applied to abstracts.

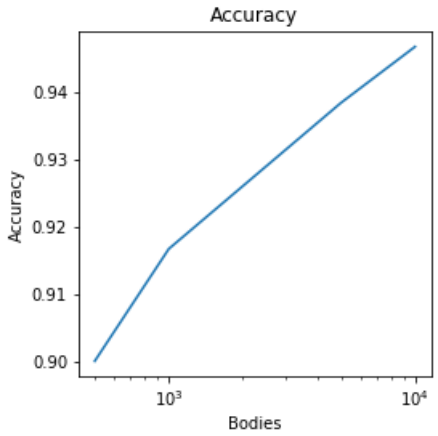


FIGURE 5.12: Accuracy of Doc2VecC applied to bodies.

training dataset was too time consuming. In both cases, however, the model seems to achieve convergence anyways.

Doc2VecC evaluation	500	1K	5K	10K	50K	100K	500K	1M
Titles	-	0.67	-	0.75	-	0.88	-	0.87
Abstracts	-	0.74	0.87	0.90	0.92	-	-	-
Bodies	0.90	0.92	0.94	0.94	-	-	-	-

TABLE 5.4: The exact accuracies of Doc2VecC.

5.4 Word Mover's Distance

While most document similarity methods focus on word embedding compositionality, Word Mover's Distance (WMD) (Kusner et al., 2015) proposes a physically motivated distance function that directly acts on two sets of word embeddings, without computing intermediate representations.

Because of this, it is no longer possible to store a simple vector representation for each document and compare them with a constant-time operation. The similarity

function is expensive to compute, and the need to compute it for each pair separately makes it inconvenient for some use cases, such as clustering.

However, it turns out that WMD is actually very effective and stays competitive between more modern deep learning alternatives. Although slow, it stays to this day the preferred method for content based recommendation at SCITODATE.

WMD's simplicity is also an added advantage. It is hyperparameter free and performs well at sentence, paragraph and document level. The physical motivation behind the similarity function makes it easy to interpret, debug and even visualize in great detail. It also makes it easy to extend and improve.

5.4.1 Intuition

The WMD similarity function is just a special case of the well known Earth Mover's Distance transportation optimization problem, also known as the Wasserstein metric. The Wasserstein metric is a common mathematical construct that can be used to compare two probability distributions. It is often used in computer science to compare discrete distributions such as images (Rubner, Tomasi, and Guibas, 1998) or, in this case, sets of embeddings. However, as the name suggests, the Earth Mover's problem has a simple physical intuition.

Let there be a given mass of earth. The earth is distributed in discrete piles, each with a different portion of the total mass. A goal is set to move all that earth to a different region and be organized there into a different distribution of piles. The distance between each pile of the first region and each pile of the second region is known. Moving earth takes effort or cost, based on the amount of mass moved and the distance traveled. The Earth Mover's Distance will be the minimum cost of redistributing all the earth to the new region in the specified arrangement.

Making a parallel with word embeddings and documents is not too complicated at this point.

WMD does not take word order into account, so it encodes both documents with the Vector Space Model. The vector representation could be Bag of Words, Tf-Idf or any other similar weighting scheme such as BM25. A document is therefore, a set of word embeddings, each with a distinct weight.

With this setup, the embeddings can be interpreted piles of earth. The embedded vector represents the position of the pile and distances between piles are defined as the standard euclidean distance between the embeddings. The mass of each pile will of course be the weight assigned to each embedding by the VSM representation of the document.

The goal is to optimally transform (move) the words of one document so that they become the second document. If two documents are semantically different, their embeddings will be far from each other, so the cost will be high. The reverse happens if the documents are similar. This is rather intuitive.

WMD will work best when the structure of the text does not carry most of the semantic information, as with all VSM models. This is the case with technical documents with very distinctive vocabulary, such as scientific text. The mere presence of certain technical words already give a strong signal towards predicting the topic of the document.

It turns out that for short fragments of such text with specialized vocabulary, words may not necessarily repeat between two similar texts. For example, a few sentences on web development will hardly share much vocabulary with some text on machine code compilation, yet, they are both computer science texts and are strongly correlated compared to other scientific texts. Using word embeddings gives us the necessary resolution to make such distinctions. This makes WMD specially well suited for academic abstracts compared to more traditional VSM methods, and yet, it stays simple enough to be highly interpretable.

5.4.2 The maths

To follow, we will give the formal specification of the optimization problem explained above to complement the intuition.

Let there be two documents D and D'. c_i denotes the number of times the word i appears in document D. Similarly, $c_i\prime$ denotes the number of times word i appears in document D'. Let us say that both documents are encoded in the VSM in the normalized bag-of-words (nBOW) form. This will be represented by d_i and $d_i\prime$ respectively, where $d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}$.

Let $X \in \mathbb{R}^{d \times n}$ be a word embedding matrix, where each column x_i is a word embedding of d dimensions. We define the pairwise distance between two embeddings as $c(i,j) = ||x_i - x_j||_2$.

Let T be a flow matrix which encodes the solution to the optimization problem. $T \in \mathbb{R}^{n \times n}$ is a sparse matrix where $T_{ij} \geq 0$ denotes the amount of word i that will be transferred to word j.

The goal is to minimize the transport cost, while keeping the total earth mass before and after the transport constant.

$$\begin{aligned} &\min_{T \geq 0} && \sum_{i,j=1}^{n} T_{ij} c(i,j) \\ &\text{subject to:} && \sum_{j=1}^{n} T_{ij} = d_i \quad \forall i \in 1,...,n \\ &&& \sum_{i=1}^{n} T_{ij} = d_j\prime \quad \forall j \in 1,...,n \end{aligned}$$

5.4.3 Computational complexity

Fortunately, this is a well studied transportation problem for which specialized solvers have been developed. Going into the state-of-the-art solvers and their complexity analysis is beyond the scope of this work though.

There are many solvers to choose from. The most well known implementation of WMD resides it the Gensim library[5]. This implementation directly uses PyEMD which is the Python port of (**pele2009**), which has a complexity of $O(n^3 \log n)$, n being the size of the vocabulary.

The original paper also proposes two more relaxed distance functions. They are both proven to be a lower bound of the complete WMD and are considerably faster to compute. The authors propose a method they call "prefetch and prune" which

[5]Gensim: https://radimrehurek.com/gensim/

involves ranking the documents with the cheaper distance functions to prune the document set and reduce the number of expensive WMD comparisons.

The first alternative distance function is centroid distance. It involves computing the embedding centroid for each document weighted by its VSM encoding. Document similarity is then measured by simply computing the euclidean distance between centroids. Building the centroids has linear complexity on the size of the document, but it can be done very efficiently with linear algebra libraries. The actual distance computation is done in constant time. This means that document ranking can be done very efficiently using a Nearest Neighbor data structure like the ones offered by the recently released faiss library (Johnson, Douze, and Jégou, 2017).

The second alternative distance function proposes removing one of the constraints of the WMD optimization problem. This reduces the complexity to $O(n^2)$ and it is regarded as a good approximation to the slower WMD for most datasets.

5.4.4 Experimental review

In this section we present all the experimental results from applying WMD to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We use the Word2Vec Skip-gram vectors trained on 2B tokens. It was considered that using Skip-gram would give the most comparable results, as it is still one of the most popular word embedding algorithms.

We use the WMD implementation from Gensim[6], it is considered the standard implementation. Gensim in turn uses the PyEMD[7] Earth Mover's Distance efficient solver.

A performance benchmark wasn't considered relevant in this case. There's no training phase with WMD, all the heavy lifting is done during word embedding training and document comparison. Text sizes are fairly consistent in our dataset, so a performance benchmark would not show much variation. Almost all the computing effort is also done by a well tested standard solver, testing it yet another time would not bring much value.

We have only evaluated WMD for titles and abstracts. This is mainly because of the lack of the necessary computational resources. While it took only 3 seconds to perform 600 title comparisons and 10 mins to perform the same amount of abstract comparisons, a single article body pair comparison took 8 mins to compute. This is a clear example of how madly WMD scales on text size. It is a useful comparison method for sentences and paragraphs, but it is not practical to go any bigger.

WMD evaluation	Accuracy
Titles	0.90
Abstracts	0.92

TABLE 5.5: Exact accuracies of WMD evaluation.

Table 5.5 shows the exact evaluation accuracies. They are good results, but they simply match the naive baseline.

[6] WMD in Gensim: `https://radimrehurek.com/gensim/models/keyedvectors.html`
[7] PyEMD repository: `https://github.com/wmayner/pyemd`

5.5 Skip-thoughts

Skip-thoughts (Kiros et al., 2015) is one of a few state-of-the-art algorithms for sentence embedding. Consistent with current trends, Skip-thoughts uses deep learning to train sentence embeddings in an unsupervised manner. In the case of embedding algorithms, unsupervised refers to the fact that there is no explicit gold standard dataset with scored pairs to train from. However, there is always an implicit supervision signal. In the case of Skip-thoughts, like the name suggests, a window method inspired by Skip-gram from Word2Vec provides this signal. The sentence embeddings are learned by exploiting the relation between consecutive sentences in free text.

5.5.1 Intuition

Skip-thoughts learns sentence embeddings very much like Word2Vec learns word embeddings. The embedding vectors are trained through a stochastic gradient descent (SGD) optimization procedure. A window is passed sequentially through the whole corpora. This window contains three consecutive sentences, where, much like in skip-gram, the model attempts to generate the two outer sentences by taking the central one as input. For each window, a prediction error is computed by the loss function, which, in turn, drives the SGD process.

Even if the learning procedure is very much like the one proposed for the Skip-gram model, the main difference between both algorithms lies on the model architecture itself. Word2Vec stores all the embedding vectors and retrieves each one whenever a cosine similarity operation between two words needs to be performed. Skip-thought also has sentence embeddings that are compared by cosine similarity, but it is not feasible to store the embedding vector for all possible sentences. Sentence embeddings always have to be inferred from their word embedding components, which is where most of the extra complexity of Skip-thoughts is.

Skip-thoughts relies on the now standard encoder-decoder deep neural architecture. The basic idea behind the design is that there is a deep neural network (DNN) that takes an input sequence and generates a compact vectorial representation. This vector embedding is then fed into the decoder which generates another sequence based on this input. The whole system is evaluated on the accuracy of the generated sequence and trained by gradient descent. There are a few alternatives, but the most common option is that both the encoder and decoder are recurrent neural networks (RNN). Often, both the encoder and decoder contain several layers of RNN. It is common practice for example to have two consecutive RNNs in the encoder that read the input in opposite directions. It is also common to have an attention mechanism in the decoder to decide what parts of the encoding to focus on in each generation time step.

This architecture was originally designed to perform machine translation and it has been the main driver of recent breakthroughs in the field. For example, the core of Google Translate (Wu et al., 2016) is a 8 level bidirectional RNN encoder-decoder architecture. When trained for long enough with enough data, it yields state-of-the-art results which are a significant improvement over the previous version based on classical statistical machine translation. It even makes the whole system modular, just attach the encoder of the source language and the decoder of the target language.

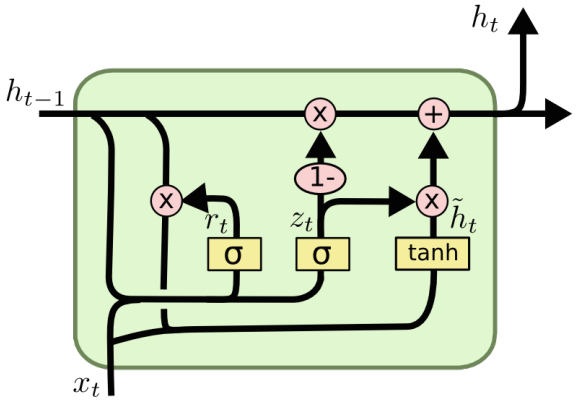
Skip-thought uses a similar architecture to train the model to infer sentence embeddings. The encoder is an unidirectional or bidirectional RNN. The output of the encoder is the sentence embedding, which is then used by two single layered RNN decoders to predict and generate the previous and next sentences in the text. In both cases, Gated Recurrent Units (GRU) are used as RNN cells. GRU cells are similar to the standard LSTM cells but are simpler and faster to train.

As described, sentence embeddings are the intermediate representations of the encoder-decoder architecture. They are the side product of an alternative supervised optimization problem. This is the key of Skip-thoughts' success, as the training data for this alternative problem is widely abundant, it is just free text.

Word embeddings are also produced during training. They are randomly initialized and optimized as part of the training process, although there is also the option of starting with pre-trained word embeddings. A key aspect that is addressed in the original paper, is that it is possible to encounter many words during inference that were not found during training. The authors propose a simple solution where they train a projection matrix using unregularized L2 linear regression to transform a large set of pre-trained embeddings to the Skip-thoughts more limited embedding space. This is specially relevant in the case of Skip-thoughts, as it is very expensive to train. The authors report training it for a week. This flexibility to work with arbitrary embeddings makes the model specially flexible and useful.

5.5.2 Model equations and training details

The Skip-thoughts deep neural model can be described in two distinct parts, the encoder and the decoder. The following equations describe their exact inner workings. The structure of both parts is fairly simple. The encoder is a sequence of GRU cells and there are two decoders that are sequences of conditional GRU cells. Most of the complexity in these equations is just the specification of the inner workings of a GRU cell.



While the LSTM cell has three control gates, the GRU cell has only two, which makes it cheaper to train while keeping a similar functionality. The two gates are named update gate z and reset date r. The intuition is that the reset gate decides how much of the hidden state h should be combined with the input x to generate the intermediate result $\bar{h}$. Then, the update gate decides the proportions with which h and $\bar{h}$ should be combined to generate the next hidden state and output.

The weights are divided into two, W is applied to the input x and U is applied to the hidden state h. There are three different weight matrices for each W and U, for r, z and h. Let $w^1, ..., w^N$ be a sequence of words. The GRU RNN will iterate over the sequence generating outputs h^t which should be interpreted as the representation of sequence $w^1, ..., w^t$. Let x^t be the word embedding of word w_t. The following equations define the encoder, which is a plain GRU RNN.

$$
\begin{aligned}
r^t &= \sigma(W_r x^t + U_r h^{t-1}) \\
z^t &= \sigma(W_z x^t + U_z h^{t-1}) \\
\bar{h}^t &= \tanh(W_h x^t + U_h(r^t \circ h^{t-1})) \\
h_i^t &= (1 - z^t) \circ h^{t-1} + z^t \circ \bar{h}^t
\end{aligned}
$$

The decoders are almost the same, but they use conditional GRU cells. Conditional GRU cells introduce three new weight matrices C_z, C_r and C_h. C is used in combination with h_i, the final output of the encoder, to add a bias to each gate. The other weight matrices are not shared between the two decoders, so they are distinguished with their respective decoder ID d.

Both decoders start with the last hidden state of the encoder. The first input of the decoders is a special $< eos >$ symbol and the following inputs will be the outputs of the previous time-step.

The following equations define the decoders.

$$
\begin{aligned}
r^t &= \sigma(W_r^d x^t + U_r^d h^{t-1} + C_r h_i) \\
z^t &= \sigma(W_z^d x^t + U_z^d h^{t-1} + C_z h_i) \\
\bar{h}^t &= \tanh(W_h^d x^t + U_h^d(r^t \circ h^{t-1}) + C_h h_i) \\
h_{i\pm1}^t &= (1 - z^t) \circ h^{t-1} + z^t \circ \bar{h}^t
\end{aligned}
$$

Finally, we define the probability of a word $w_{i\pm1}^t$ following the previous $t-1$ words.

$$
P(w_{i\pm1}^t | w_{i\pm1}^{<t}, h_i) \propto \exp(x_{i\pm1}^t, h_{i\pm1}^t)
$$

Thus, given a window of consecutive sentences (s_{i-1}, s_i, s_{i+1}) the following objective will be maximized using the training dataset as the source of truth.

$$
\sum_t \log(P(w_{i-1}^t | w_{i-1}^{<t}, h_i) + \sum_t \log(P(w_{i+1}^t | w_{i+1}^{<t}, h_i)
$$

The Adam optimization algorithm is used to perform the gradient descent. Adam is a modification of the base SGD algorithm. It is designed to accelerate convergence by adding dynamic learning rates, gradient normalization and momentum. It is inspired by RMSProp. Mini-batches of 128 are used and the gradients are clipped if their norm exceeds 10.

In the case where unidirectional RNNs are used, the uni-skip model is trained with 2,400 dimensions. For bidirectional RNNs, bi-skip, each will have 1,200 dimensions and the skip-thought vector will be the concatenations of both encodings. The third

alternative is to train both and concatenate the encodings into a 4,800 dimensional representation. This third model is called combine-skip and is reported to give better performance than the other two.

The recurrent matrices are initialized with orthogonal initialization and the non-recurrent ones with a uniform distribution $[-0.1, 0.1]$. The model is initially trained on a limited vocabulary of 20,000 words. The vocabulary is then expanded by learning a projection matrix to transform a set of pre-trained vectors into the Skip-thoughts vector space. This projection is learned by simple L2 linear regression.

5.6 Sent2Vec

As the name suggests, Sent2Vec (Pagliardini, Gupta, and Jaggi, 2017) is another state-of-the-art sentence embedding algorithm. However, even if the original paper focuses on sentence embeddings, the model is also general enough to generate embeddings of any other semantic unit, such as paragraphs or full documents. It is, in fact, in many ways like Doc2VecC. The core idea is to simply create document embeddings by averaging their component word embeddings, but to train those word embeddings such that they generate meaningful average document representations. Sent2Vec differs from Doc2VecC in that it optimizes for more local compositionality, like sentences or paragraphs, while Doc2VecC optimizes for a global view more appropriate for full document embeddings.

5.6.1 Intuition

The main observation in Sent2Vec is that there are two different trends in the world of embedding algorithms. Some authors insist in applying deep neural architectures, like Skip-thoughts. These models have more capacity to learn complex patterns but are very expensive to train. In contrast, shallow models are not as powerful but training them is so cheap that they can take advantage of much larger training datasets. This exact contrast was exemplified a few years ago with the discovery of Word2Vec. There were many iterations of neural models for word embeddings before (Bengio et al., 2003), but the idea of using a much simpler model and vast amounts of corpora made Word2Vec the breakthrough that it is now.

Sent2Vec is the proposal to do this same thing for sentence or document embeddings. Models like Skip-thoughts employ deep neural architectures with thousands of internal weights to train. However, recent publications (Arora, Liang, and Ma, 2016) have shown that, in many cases, simple weighted embedding centroids outperform these more powerful models. Much like Doc2VecC, Sent2Vec polishes the concept of embedding centroids and proposes a simple but efficient model like Word2Vec that can ingest much bigger corpora and beat the state-of-the-art.

Sent2Vec is, in practice, almost identical to the cBoW model from Word2Vec. Like in any other window based embedding algorithm, a context window is scanned sequentially through the corpus. The central word in the window is the target word and the rest are the context. In cBoW, the target word is predicted given the context. In practice, this means that the context embeddings are averaged and the dot product between the target embedding and the context centroid is minimized.

The authors of Sent2Vec point out that such a model trains optimal embeddings for additive composition. In other words, the embeddings are optimized such that when computing the centroid of a set of embeddings, the centroid contains the relevant information to express the meaning of the set. However, cBoW optimizes such composition for small arbitrary windows of consecutive words, which may not be semantically significant. Sent2Vec proposes to do the same thing with windows that are clear semantic units, such as sentences, paragraphs or full documents.

Another main difference between cBoW and Sent2Vec is that the contexts in Sent2Vec also contain word n-grams not optimized further for compositionality. This is specially interesting for scientific and technical text, as many concepts are often expressed by multi-word phrases. It is also worth mentioning that Word2Vec does random word sub-sampling as a regularization to improve generality. Random words are also deleted in Sent2Vec, but the sub-sampling is done on the context once all the n-grams have been extracted. Thanks to this, the n-grams are kept clean and potentially relevant syntactic structures are not broken.

5.6.2 The maths

Sent2Vec is, for all intents and purposes, a mere modification of the cBoW algorithm.

In general, Sent2Vec, much like Word2Vec, GloVe and FastText, is a matrix factorization problem. In the case of GloVe, the factorization is explicit, but the other algorithms do it implicitly by scanning the corpus with a window. We decompose U and V, where U contains the embeddings for target words and V contains the embeddings that are in the context window.

$$\min_{U,V} \sum_{S \in C} f_S(UVi_S)$$

The composition of the context words is kept unchanged, it is a simple average. The only differences are how the window is chosen and the inclusion of n-grams. In practice, only unigrams and bigrams are extracted, as including bigger n-grams would expand the vocabulary considerably. The sentence embedding v_S for a sentence window S is defined like so, where $R(S)$ is the set of all n-grams in the sentence.

$$v_S = \frac{1}{|R(S)|} \sum_{w \in R(S)} v_w$$

The loss function is also the same, including the negative sampling. Given the binary logistic loss function $l : x \to \log(1 + e^{-x})$.

$$\min_{U,V} \sum_{S \in C} \sum_{w_t \in S} (l(u_{w_t}^T v_{S\{w_t\}}) + \sum_{w' \in N_{w_t}} l(-u_{w'}^T v_{S\{w_t\}}))$$

The negative sampling distribution is also kept: $q_n(w) = \frac{\sqrt{f_w}}{\sum_{w_i} f_{w_i}}$. So is the probability for not deleting a word: $q_p(w) = \min\{1, \sqrt{t/f_w} + t/f_w\}$, where f_w denotes word frequency and t is a hyper-parameter.

One notable difference between cBoW and Sent2Vec, is that the V context embeddings are kept for Sent2Vec, instead of the U target embeddings. This is because the V embeddings are the ones that have been optimized for being composed into a centroid. This structure also enables to compute word-document similarities, by fetching the word embedding from the U matrix and generating the document centroid embedding from the V matrix.

5.6.3 Computational complexity

The computational complexity of Sent2Vec is the same complexity as cBoW. A notable difference, however, is that the inclusion of n-gram increases the vocabulary size considerably.

In the case of sentence and document embeddings, it is obviously not viable to keep the vectors pre-trained in a hash table. Sentence embeddings need to be created from their component word embeddings. However, in Sent2Vec this just means computing a vector centroid which is a constant operation and can be performed very efficiently with standard linear algebra tools. This is, in fact, the main advantage of Sent2Vec over deep neural models, where the sentence encoding is much more involved.

5.6.4 Experimental review

In this section we present all the experimental results from applying Sent2Vec to our dataset. All tests were done in clean isolated AWS `m4.2xlarge` with Ubuntu. We use the standard document embedding size of 800. We use the original implementation given by the authors[8] throughout the benchmark.

Sent2Vec is presented as a sentence embedding algorithm. However, it is in many ways similar to Doc2VecC and is general enough to apply to any body of text. Because of that, we decide to evaluate it like the other document embedding algorithms, by performing tree different experiments on titles, abstracts and article bodies.

5.6.4.1 Computational benchmark

Figures 5.13, 5.14 and 5.15 show how the training of Sent2Vec scales for titles, abstracts and bodies respectively. The scaling pattern of Sent2Vec is very similar to most of the algorithms we have analyzed, including word embedding and document similarity algorithms. This makes sense, as Sent2Vec is effectively a word embedding algorithm, optimized for document similarity, like Doc2VecC and Doc2Vec to some extent.

Its resource requirements are similar to Doc2VecC's, but a bit lower. It would be between Doc2Vec and Doc2VecC in terms of efficiency.

[8] Sent2Vec repository: https://github.com/epfml/sent2vec

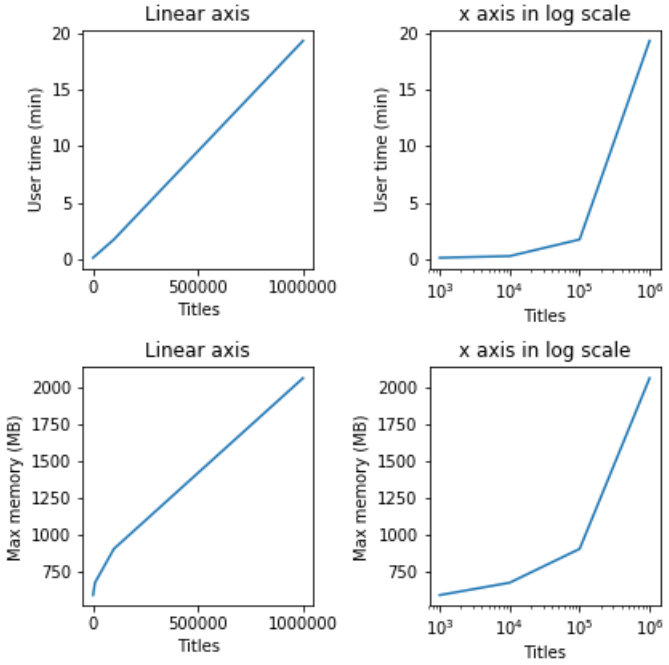


FIGURE 5.13: Computational benchmark of Sent2Vec applied to titles.

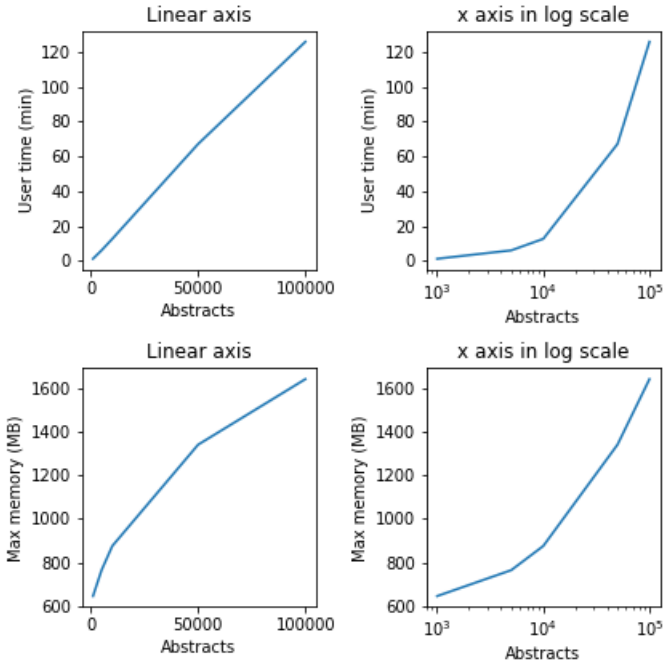


FIGURE 5.14: Computational benchmark of Sent2Vec applied to abstracts.

5.6.4.2 Evaluation

Figures 5.13, 5.14 and 5.15 show the accuracy of Sent2Vec on titles, abstracts and bodies respectively.

Curiously, Sent2Vec's learning pattern looks a lot like Doc2Vec and Doc2VecC's, but

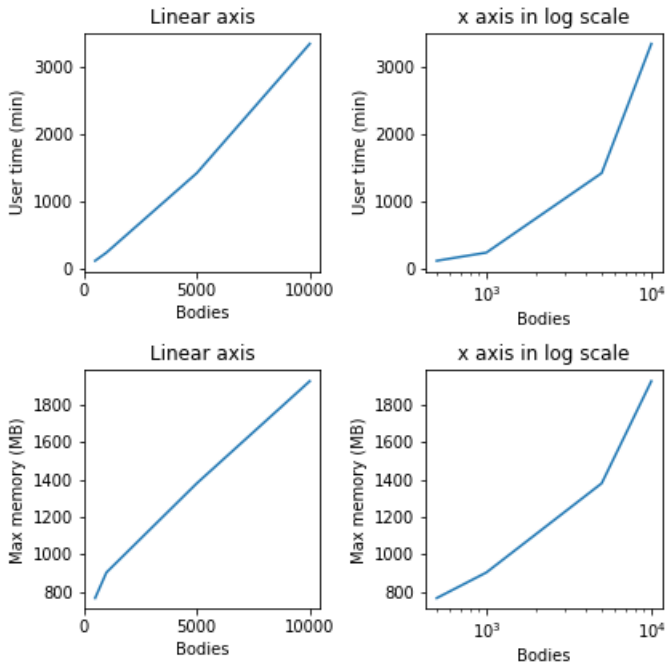


FIGURE 5.15: Computational benchmark of Sent2Vec applied to bodies.

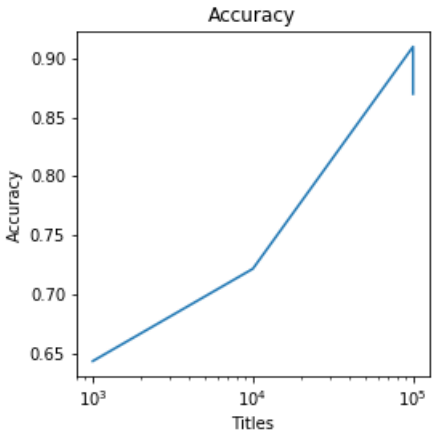


FIGURE 5.16: Accuracy of Sent2Vec applied to titles.

reversed. Sent2Vec performs considerably better than Doc2Vec and Doc2VecC on titles and its performance lowers with longer text. Sent2Vec doesn't match the baseline with abstracts and bodies, but they are still good results. This makes sense, Sent2Vec is specially optimized for sentences, but it is general enough to perform reasonably on larger text too, even if the authors do not mention it.

Table 5.6 shows the exact accuracies on evaluation. The gaps in the table are due to the high cost of training Sent2Vec. Much like with Doc2VecC, training abstracts and bodies with the whole training dataset was too time consuming. In both cases, however, the model seems to achieve convergence anyways.

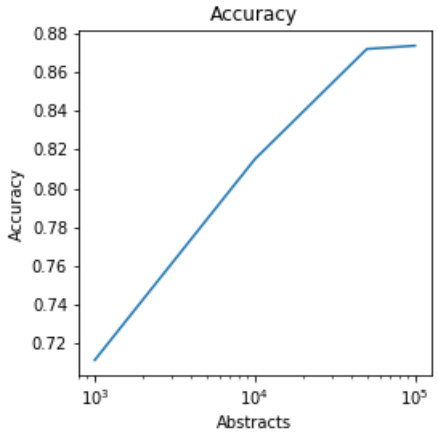


FIGURE 5.17: Accuracy of Sent2Vec applied to abstracts.

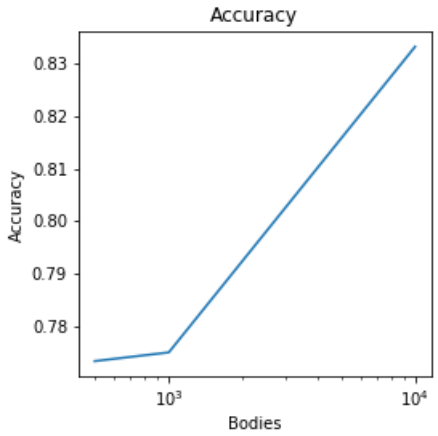


FIGURE 5.18: Accuracy of Sent2Vec applied to bodies.

Sent2Vec evaluation	500	1K	5K	10K	50K	100K	500K	1M
Titles	-	0.64	-	0.72	-	0.87	-	0.91
Abstracts	-	0.71	-	0.81	0.87	0.87	-	-
Bodies	0.77	0.76	-	0.83	-	-	-	-

TABLE 5.6: Exact accuracies of evaluation of Sent2Vec

5.7 Summary and conclusions

In this section we have performed a literature review of the state of the art in unsupervised document similarity based on word embeddings. It was clear while performing the literature review that it is still early days for Semantic Textual Similarity (STS). After looking at and evaluating some of the pivotal algorithms in the state-of-the-art, this assumption is further confirmed.

We have looked at two document embedding algorithms, the first one was **Doc2Vec**. Doc2Vec is almost a direct translation of Word2Vec into the context of documents. The only difference between Word2Vec and Doc2Vec is that Doc2Vec introduces an extra global vector to the context that represents the document the window is in. This way, Doc2Vec forces these global vectors to learn the information missing from

the local context and that is necessary to perform predictions in the global document context. Doc2Vec is well known in the community, but mostly because it follows Word2Vecs fame. In truth, its results are somewhat underwhelming, both in third party benchmarks and in our own tests. Doc2Vec barely outperforms the naive benchmark in the best of cases, yet, it is the best performing algorithm we have analyzed, which says a lot about the state of the field. It was also, by far, the fastest implementation; although it is not clear if it is the fastest algorithm, as it does not have a clear advantage on theoretical complexity.

The second document embedding algorithm we looked at was **Doc2VecC**. As the name suggests, Doc2Vec and Doc2VecC share the same general structure. However, a key difference is that Doc2VecC does not train its document embeddings from scratch, but it optimizes the word embeddings such that their average provides meaningful document representations. It also adds a corruption component, which heavily deletes random words from the training data to improve generality. Doc2VecC performed poorly on titles and abstracts, but matched the baseline on bodies even with very few sample documents.

The third algorithm we looked at was **Word Mover's Distance** (WMD), and it was somewhat of a standout. WMD defines a distance computation between documents or, more concretely, sets of word embeddings with weights or frequencies. The authors find that the Earth Mover's Distance optimization problem performs well for this task, although they do not give a good explanation why it is so effective when applied to language. WMD performed reasonably well, matching the baseline with abstracts. However, the scaling issue became immediately apparent on our tests, as evaluating on bodies became impossibly slow.

The last two algorithms where sentence embedding algorithms. The first of the two, **Skip-thoughts**, is the only deep neural model we have analyzed. Using deep models for tasks like this is popular in academia at the moment, but most of them are iterative improvements so they were not considered relevant for this work. Skip-thoughts uses the now standard encoder-decoder RNN architecture used in translation. It exploits the sequential nature of text to relate sentences to each other.

The last algorithm we analyzed was **Sent2Vec**. It is advertised as a sentence embedding algorithm but, in fact, it has more in common with Doc2VecC than Skip-thought. Sent2Vec also optimizes word embeddings such that their centroids provide meaningful sentence representations. The model is general enough to be applied to any document type though, it is not locked into the sentence structure like Skip-thoughts is. Sent2Vec is simply more optimized for small text than Doc2VecC, and the tests we have done reflect it clearly.

We have analyzed a wide variety of algorithms to solve the STS problem. Overall, though, the algorithms do not perform too differently to each other and only the best models with the largest training sets match the naive baseline proposed at the beginning of the chapter. The work that it is being done is certainly valuable, it will probably be the foundation to the next breakthrough, but it is still not good enough.

Table 5.7 unifies all the best evaluation results for comparison and reference.

STS evaluation	Baseline	Doc2Vec	Doc2VecC	WMD	Sent2Vec
Titles	0.91	0.65 (1M)	0.87 (1M)	0.90	**0.91 (1M)**
Abstracts	0.93	0.86 (1M)	**0.92 (50K)**	**0.92**	0.87 (100K)
Bodies	0.96	**0.97 (500K)**	0.94 (10K)	-	0.83 (10K)

TABLE 5.7: The best results of all the analyzed algorithms. It includes the amount of training samples between parenthesis when relevant.

Chapter 6

Conclusions

This work has been performed as a Bachelor's thesis. As such, the main goal of this project was to reflect and exercise the competences acquired during my Computer Science degree. By making this project happen, I show that I am capable of mapping out, reading, understanding and interpreting academic literature. I have also proven that I have the technical capabilities to collect large amounts of training data from various complex sources and the ability to find, compile, understand and adapt widely different program implementations. Finally, I show that I am able to perform a proper scientific study, with all the implied rigors, and that I can express my findings thoroughly and formally.

This work was also developed under the bigger SCITODATE umbrella project. SCITODATE is a startup I founded with the goal of extracting deep insight and trends from large amounts of academic text. This thesis has been written in parallel to SCITODATE work and it is supposed to be another small step in the direction of our long-term goal.

I have performed a thorough literature review of word embedding algorithms and solutions to the Semantic Textual Similarity task. I chose some of the pivotal state-of-the-art algorithm and I performed an in-depth analysis of all of them. I hope this document serves as reference to future researchers as an aid for understanding these algorithms, for getting a clearer view of the state-of-the-art, for getting ideas for future work or for choosing the best algorithms to use for a higher level application.

Another core contribution of this work has been collecting an extensive training and test dataset in the scientific domain. Most work on semantic language understanding focuses on general purpose language such as news or social media activity. Technical and scientific language has considerable differences, however, and, considering the importance of such texts, it was deemed relevant to perform a study applied to those domains.

Overall, we extract two main conclusions from this study. The first one is about word embedding algorithms. This field is currently very active in academia, new approaches to word embedding appear on a weekly basis. However, we see that most algorithms follow the initial Word2Vec framework very closely and do not offer any notable improvements over the original. It is true though, that there are a few standouts. FastText has considerably outperformed the rest and has more practical advantages.

The second main conclusion is about unsupervised semantic textual similarity. It is clearly early days for this task. It was clear while doing the literature review and it was further confirmed by the evaluation results. It is an active field, SemEval is a

fairly important annual workshop for example, but not as active as other NLP tasks. The state-of-the-art is narrow and most algorithms barely match the most simple baseline methods. However, there are many different approaches being explored. It is likely that we will see a breakthrough coming from this experimentation in the coming years.

6.1 Future work

The availability of text datasets in the scientific domain is the main barrier for progress in the quest for deeper automatic understanding of academic publications. We have build an extensive corpus of biomedical articles that includes separate titles, abstracts and article bodies. We have also built a set of evaluation triplets both for words and for documents, where the first two elements in the triplet are related and the third is not.

This is a good approach of working with semi-curated reference data. Knowledge bases such as UMLS are relatively well maintained, but it is not possible to avoid a certain amount of inconsistency and incompleteness. The author linking of articles is also a weak similarity linkage but it seems to have performed consistently on evaluations. However, distinguishing related pairs from random noise is hardly a challenge. High accuracies may be misleading in this case. The ideal would be to build properly curated human rated similarity pair datasets.

It is particularly interesting to extend the work done on article linkage. Using metadata like dates or references could yield more and better evaluation data. There is also room to extend the author linkage dataset to other centralized author identifier resources. The key advancement would be to collect enough data to be able to test supervised semantic textual similarity methods, which were left out of scope for this work.

In the context of SCITODATE this is a small starting step. We have a road-map of research projects ahead to deepen our automatic understanding of academic text. We continue by looking into Named Entity Recognition for technical terminologies. We intend to link text to scientific knowledge bases. After that, we also intend to automatically extend the knowledge bases and mine relationships and facts. It is particularly interesting for us to relate research topics with the equipments and materials needed to perform the research.

There is plenty of work to be done in general in semantic textual modeling, similarity and word embedding compositionality. They are active fields and they have been so for a few years now, but the overall conclusion has been that most models barely match simple baselines. I hope to see a breakthrough for these tasks in the next few years, a breakthrough like Word2Vec was.

Bibliography

Arora, Sanjeev, Yingyu Liang, and Tengyu Ma (2016). "A simple but tough-to-beat baseline for sentence embeddings". In:

Bengio, Yoshua et al. (2003). "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.

Bojanowski, Piotr et al. (2016). "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606*.

Botha, Jan and Phil Blunsom (2014). "Compositional morphology for word representations and language modelling". In: *International Conference on Machine Learning*, pp. 1899–1907.

Carpenter, Mark P and Francis Narin (1973). "Clustering of scientific journals". In: *Journal of the Association for Information Science and Technology* 24.6, pp. 425–436.

Chen, Minmin (2017). "Efficient vector representation for documents through corruption". In: *arXiv preprint arXiv:1707.02377*.

Dai, Andrew M, Christopher Olah, and Quoc V Le (2015). "Document embedding with paragraph vectors". In: *arXiv preprint arXiv:1507.07998*.

Dann, David, Matthias Hauser, and Jannis Hanke. "Reconstructing the Giant: Automating the Categorization of Scientific Articles with Deep Learning Techniques". In:

Fu, Ruiji et al. (2014). "Learning semantic hierarchies via word embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1, pp. 1199–1209.

Harris, Zellig S (1954). "Distributional structure". In: *Word* 10.2-3, pp. 146–162.

Ji, Shihao et al. (2015). "Wordrank: Learning word embeddings via robust ranking". In: *arXiv preprint arXiv:1506.02761*.

Johnson, Jeff, Matthijs Douze, and Hervé Jégou (2017). "Billion-scale similarity search with GPUs". In: *arXiv preprint arXiv:1702.08734*.

Kiros, Ryan et al. (2015). "Skip-thought vectors". In: *Advances in neural information processing systems*, pp. 3294–3302.

Kusner, Matt et al. (2015). "From word embeddings to document distances". In: *International Conference on Machine Learning*, pp. 957–966.

Lau, Jey Han and Timothy Baldwin (2016). "An empirical evaluation of doc2vec with practical insights into document embedding generation". In: *arXiv preprint arXiv:1607.05368*.

Lazaridou, Angeliki et al. (2013). "Compositional-ly Derived Representations of Morphologically Complex Words in Distributional Semantics." In: *ACL (1)*, pp. 1517–1526.

Le, Quoc V. and Tomas Mikolov (2014). "Distributed Representations of Sentences and Documents". In: *CoRR* abs/1405.4053. URL: http://arxiv.org/abs/1405.4053.

Luong, Minh-Thang and Christopher D Manning (2016). "Achieving open vocabulary neural machine translation with hybrid word-character models". In: *arXiv preprint arXiv:1604.00788*.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Pagliardini, Matteo, Prakhar Gupta, and Martin Jaggi (2017). "Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features". In: *arXiv preprint arXiv:1703.02507*.

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

Qiu, Siyu et al. (2014). "Co-learning of Word Representations and Morpheme Representations." In: *COLING*, pp. 141–150.

Rubner, Yossi, Carlo Tomasi, and Leonidas J Guibas (1998). "A metric for distributions with applications to image databases". In: *Computer Vision, 1998. Sixth International Conference on*. IEEE, pp. 59–66.

Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams, et al. (1988). "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3, p. 1.

Wang, Chang, Liangliang Cao, and Bowen Zhou (2015). "Medical synonym extraction with concept space models". In: *arXiv preprint arXiv:1506.00528*.

Ware, Mark and Michael Mabe (2015). "The STM report: An overview of scientific and scholarly journal publishing". In:

Wu, Yonghui et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144*.