Hierarchical Paragraph Vectors

Master Thesis

Author(s):
Elmer, Lukas

Publication date:
2015

Permanent link:
https://doi.org/10.3929/ethz-a-010541231

Rights / license:
In Copyright - Non-Commercial Use Permitted

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Hierarchical Paragraph Vectors

Master Thesis

Lukas Elmer

September 8, 2015

Advisors: Prof. Dr. Thomas Hofmann, Dr. Carsten Eickhoff

Department of Computer Science, ETH Zürich

Abstract

Most standard machine learning algorithms require fixed-length, low-dimensional vectors to perform well. However, when working with text, such representations are difficult to obtain. In 2014, Le and Mikolov presented a novel method for generating so-called word embeddings [16]. Their work represents the foundation for this master thesis.

The main goal of this thesis is to extend and generalize the word embedding model to a hierarchical paragraph vector model. This means that different parts of the vector represent different contexts which are shared among sibling structures originating from the same parent text block. For example, the first part of the vector can be used to describe the document, the second part to describe the chapter, the third part to describe the paragraph and the last part to describe the individual sentence.

In this thesis, we propose *Hierarchical Paragraph Vectors*, which exploit hierarchical document structures. When applying this novel method to sentiment analysis tasks, empirical results show that it can increase the quality of the word embeddings at the cost of greater execution overhead.

Acknowledgements

I would like to thank Dr. Carsten Eickhoff for his continuous support, insightful and straightforward advice, positive criticism, proofreading, and the supervision of my master thesis. It has been a pleasure working with him.

Additionally, I would like to thank Prof. Dr. Thomas Hofmann for giving me the opportunity to investigating the highly interesting topics of NLP and word embeddings, and for his valuable feedback.

Furthermore, I would like to thank everyone who supported me during my studies, especially my family, my business partners and colleagues[1], for their support, understanding, and positive attitude.

Finally, I would like to thank Marion for her continuous positive attitude, proofreading, and for being by my side.

[1]https://www.renuo.ch

Contents

Chapter 1

Introduction

Words are the building blocks of natural language. Words are categorized into word classes (for example verbs, nouns, adjectives) and any given grammar only allows certain combinations and word orders to form valid sentences. The meaning of these words is manifold, and naturally some words are more similar to each other than they are to others. Consider for example the three words "interesting", "fascinating" and "bottle". In this case, the first two words are more similar to each other. Hence, the idea to represent words in a way that captures these similarities, is natural. One way of achieving this is the use of semantically informed vector representations, so-called word embeddings.

Finding suitable word embeddings is a challenging and interesting task. The objective is to find a low-dimensional vector for each word and/or phrase in the vocabulary. When done correctly, these word embeddings have been shown to be useful to help solving difficult Natural Language Processing (NLP) problems, including machine translation [18, 28], sentence completion [17], sentiment analysis [9], and topic modeling [10].

Recent advancements [17, 16] allow finding high-quality low-dimensional word embeddings, while the processing speed is increased substantially due to many optimizations and/or approximations. Furthermore, reliable and fast implementations have been developed and released, one of them being Gensim [30]. Therefore, training a word embedding model is now feasible within minutes or hours, while it took days or months before [16].

For many applications, the data is formatted in a hierarchical structure, as can be observed in Figures 1.1, 1.2 and 1.3. Therefore, it is compelling to try exploiting these hierarchies to improve the quality of the word embeddings and/or to reduce the learning time while maintaining the word vector quality. To our best knowledge, this concept has not been investigated profoundly yet, using the recent advancements in [16], although there have been

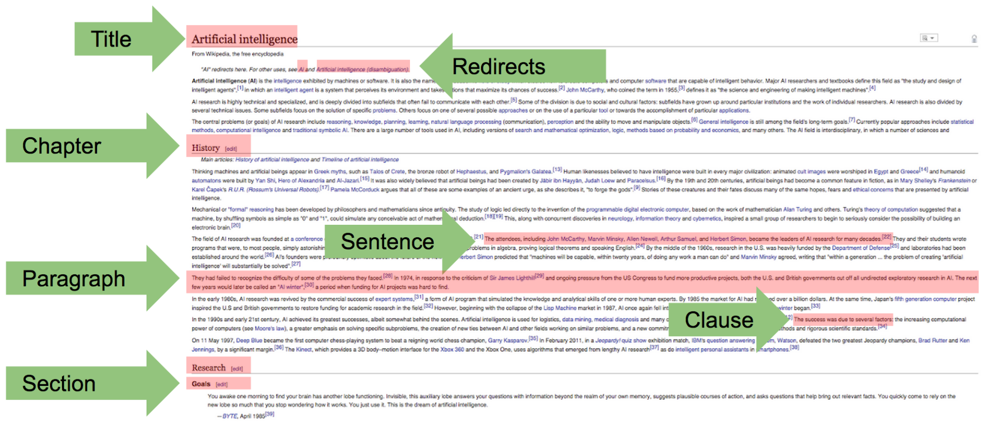
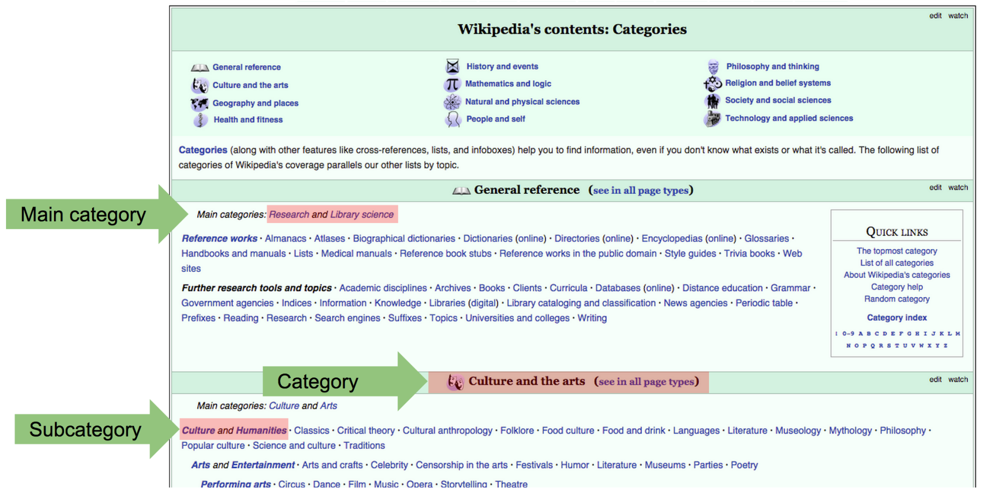


Figure 1.1: The Wikipedia article about artificial intelligence is segmented into semantic and syntactic hierarchies (for example title, chapter, section, paragraph, sentence, clause).[1]



Wikipedia's contents: Categories

General reference · History and events · Philosophy and thinking · Culture and the arts · Mathematics and logic · Religion and belief systems · Geography and places · Natural and physical sciences · Society and social sciences · Health and fitness · People and self · Technology and applied sciences

Categories (along with other features like cross-references, lists, and infoboxes) help you to find information, even if you don't know what exists or what it's called. The following list of categories of Wikipedia's coverage parallels our other lists by topic.

General reference (see in all page types)

Main categories: Research and Library science

Reference works · Almanacs · Atlases · Biographical dictionaries · Dictionaries (online) · Directories (online) · Encyclopedias (online) · Glossaries · Handbooks and manuals · Lists · Medical manuals · Reference book stubs · Reference works in the public domain · Style guides · Trivia books · Web sites

Further research tools and topics · Academic disciplines · Archives · Books · Clients · Curricula · Databases (online) · Distance education · Grammar · Government agencies · Indices · Information · Knowledge · Libraries (digital) · Library cataloging and classification · News agencies · Periodic table · Prefixes · Reading · Research · Search engines · Suffixes · Topics · Universities and colleges · Writing

Quick links
The topmost category
List of all categories
About Wikipedia's categories
Category help
Random category
Category index
0–9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Culture and the arts (see in all page types)

Main categories: Culture and Arts

Culture and Humanities · Classics · Critical theory · Cultural anthropology · Folklore · Food culture · Food and drink · Languages · Literature · Museology · Mythology · Philosophy · Popular culture · Science and culture · Traditions

Arts and Entertainment · Arts and crafts · Celebrity · Censorship in the arts · Festivals · Humor · Literature · Museums · Parties · Poetry

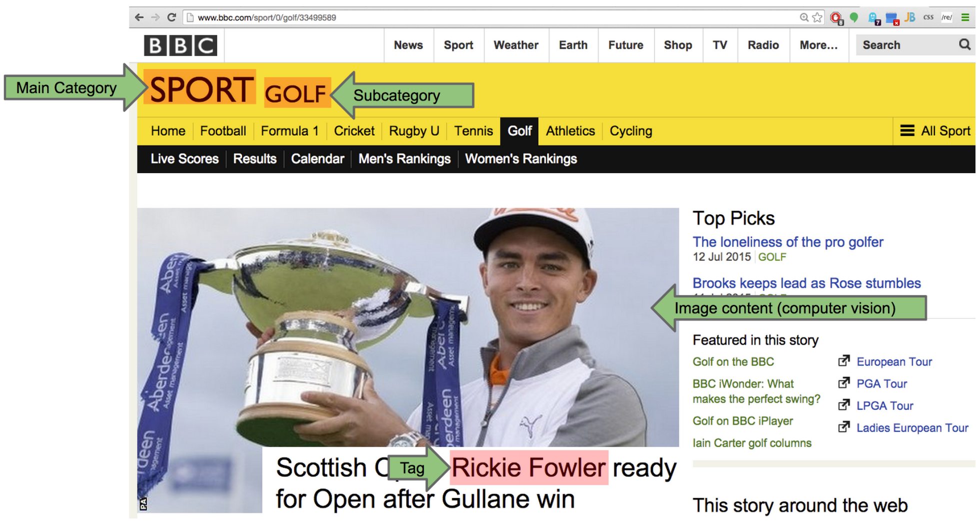
Performing arts · Circus · Dance · Film · Music · Opera · Storytelling · Theatre

Figure 1.2: Wikipedia articles are structured in categories. For example, the article "Bayes' theorem" can be found in the hierarchy "Bayesian statistics" < "Statistics" < "Mathematics and logic".[2]

motions towards this direction already [6].

This thesis is organized as follows: in Chapter 2, related work underlying this thesis is discussed. Chapter 3 introduces theory needed to understand this thesis. Chapter 4 investigates the formal core of this thesis by describing Hierarchical Paragraph Vectors. Chapter 5 describes the conducted experiments, and Chapter 6 concludes the results and previews future work.

[1]Source: https://en.wikipedia.org/wiki/Artificial_intelligence
[2]Source: https://en.wikipedia.org/wiki/Portal:Contents/Categories



Top Picks

The loneliness of the pro golfer
12 Jul 2015 GOLF

Brooks keeps lead as Rose stumbles

Featured in this story

Golf on the BBC
BBC iWonder: What makes the perfect swing?
Golf on BBC iPlayer
Iain Carter golf columns
European Tour
PGA Tour
LPGA Tour
Ladies European Tour

Scottish C Rickie Fowler ready for Open after Gullane win

This story around the web

Figure 1.3: This sports article is categorized into the category "sport" and subcategory "golf". Additional tags (for example the name of the person who is on the picture) may also be available.[3]

[3]Source: http://www.bbc.com/sport/0/golf/33499589

Chapter 2

Related Work

In this Chapter, the related research will be described. It will start with traditional and common representations, and describe their advantages and disadvantages. Next, more modern representations will be surveyed. Finally, publications describing hierarchical data will be described.

2.1 Common Representations and Methods

In NLP, probably the most widespread vector representation of text is the bag of words (BOW) representation [8]. For every word in the vocabulary, there is exactly one position in this vector, and if a word occurs n times in a text, this component will be n in the vector. Since the dimensionality of this vector is the size of the vocabulary, the BOW representation appeals by its simplicity. However, it suffers from high dimensionality, since the dimensionality is equal to the vocabulary size. Also sparsity is a major drawback, because a text block only contains a small fraction of all words in the vocabulary. Furthermore, the word order is lost, and therefore, different sentences with different meanings can have exactly the same representation.

Another commonly used representation is the bag-of-n-grams. While it mitigates some shortcomings of the BOW model, it also suffers from high dimensionality and sparsity.

Term frequency–inverse document frequency (TF–IDF) is a standard model in information retrieval. The term frequency is calculated by the number of times a word occurs in a document, while the inverse document frequency is the inverse of how often a word occurs in the whole corpus. The product of these two values is a weighted value, where rare words have a higher value when occurring in documents compared to frequently appearing words, for example stop words. Similar to the BOW model, a vector per document can be constructed by assigning a component of the vector per word in the corpus. Thus, it can be applied to any text. However, it also suffers from high dimensionality for the same reason as the BOW model does. To avoid this problem, the words can be ordered by term frequency, and only the top *n* words are considered in the TF–IDF vector. Other words in this text are simply ignored. More information about TF–IDF can be found in A.2 and in [29].

For example, let us assume that we would like to produce word embeddings for the following documents.

Document	Text
d_1	The cat sleeps on the sofa.
d_2	The mouse eats cheese.
d_3	The cat tries to catch the mouse.

BOW produces the following word embeddings.

	the	cat	sleeps	on	sofa	mouse	eats	cheese	tries	to	catch
d_1	2	1	1	1	1	0	0	0	0	0	0
d_2	1	0	0	0	0	1	1	1	0	0	0
d_3	2	1	0	0	0	1	0	0	1	1	1

3-gram of words produces the following word embeddings.

	the cat sleeps	cat sleeps on	sleeps on the	on the sofa	the mouse eats	mouse eats cheese	the cat tries	cat tries to	tries to catch	to catch the	catch the mouse
d_1	1	1	1	1	0	0	0	0	0	0	0
d_2	0	0	0	0	1	1	0	0	0	0	0
d_3	0	0	0	0	0	0	1	1	1	1	1

TF-IDF produces the following word embeddings.

	cat	catch	cheese	eats	mouse	on	sleeps	sofa	the	to	tries
d_1	0.34	0.0	0.0	0.0	0.0	0.45	0.45	0.45	0.53	0.0	0.0
d_2	0.0	0.0	0.58	0.58	0.44	0.0	0.0	0.0	0.35	0.0	0.0
d_3	0.32	0.42	0.0	0.0	0.32	0.0	0.0	0.0	0.5	0.42	0.42

Finally, all the representations discussed above (BOW, bag-of-n-grams, TF–IDF) do not capture the semantics of words. For example, the words “fast” and “quick” are as far apart from each other as they are from the word “blue”, even though the words “fast” and “quick” are semantically more similar to each other. It is even more surprising that such naïve models can perform relatively well.

2.2 Neural Networks for Language Models

The theory underlying artificial neural networks has been existing for a long time [1, 13, 25]. The two fundamental concepts for these networks are gradient descent and backpropagation. Feedforward neural network language models (NNLM) build on this technique and have deeply studied and tuned in recent publications [3, 4, 20, 27]. These NNLMs have been shown to outperform the n-gram models significantly. One major drawback of these techniques is that they are computationally expensive, and thus slow to train and test.

In 2013, Mikolov et al. published [17], where they describe the recurrent neural network language model (RNNLM). This network helps to overcome some shortcomings of the NNLM. Specifically, it has a short-term memory, and thus can model more complex behavior than shallow neural networks [11] [2]. A disadvantage of RNNs is their greater need for resources and thus prohibitively large computational cost. However, Mikolov et al. managed, with the help of approximate algorithms and an optimized implementation, to massively reduce computational complexity. The result of this are two new models, summarized under the name Word2vec: continuous bag-of-words models (CBOW) and continuous skip-gram models (skip-gram). More information about these models can be found in Chapter 3.

In their paper [12] in 2014, Le and Mikolov extended the CBOW and Skip-gram model by a paragraph vector. This new vector enables arbitrary blocks of text (for example articles, paragraphs, sentences) to learn and share a common vector, which acts as a memory to learn the word vectors. The CBOW model that is extended by the paragraph vector (PV) is called PV-DBOW, while the extended Skip-gram model is called Distributed Memory Model of Paragraph Vectors (PV-DM). The name “distributed memory” originates from the fact that the paragraph vector acts as a memory for the current context. These two models are also known as Doc2vec.

2.3 Alternative Word Embedding Models

In [7], the authors analyze the skip-gram model with negative sampling. They suggest that the embedding method by Mikolov et al. is implicitly factorizing a word-content matrix, and they attempt to implement this factorization directly. They conclude that their factorization method is better suited to optimize the objective by Mikolov et al. However, their model does not perform well on the word anology task.

In [22], the authors present the Global Vectors model (GloVe). Their model is a combination of matrix factorization methods and local context window methods. Though their model works differently compared to the model by Mikolov et al., the two models perform similarly well. In [26], the authors link the model by Mikolov et al. with GloVe by explaining out the similarities and the differences.

2.4 Exploiting Hierarchies

There have been many approaches to using hierarchies for NLP, specifically with NNLMs. However, to the best knowledge of the author, hierarchies have not yet been combined with the paragraph vectors of [12].

In [14], the authors build a hierarchy of single words by modeling the morphemic compositionality, which can be seen as a specialization of HPV.

In [21], the authors use clustered words from WordNet [5] by hierarchically decomposing the conditional probabilities as an alternative for importance sampling to speed up the training of the word embeddings. Furthermore, [20] extended the ideas in [21].

In [6], the authors use word embeddings, obtained by skip-gram, to learn semantic hierarchies. They claim that hierarchical information is encoded in word embeddings. This encourages the use of existing hierarchies to improve the quality of word embeddings.

Chapter 3

Preliminaries

In this Chapter, the models, algorithms and objectives of the two basic word embedding algorithms Word2vec and Doc2vec are described.

3.1 Word2vec

The term Word2vec summarizes two different models: the distributed bag of words model (CBOW) and the continuous skip-gram model (skip-gram). Both use neural networks, and they work very similarly. However, they have very different objectives and thus are able to capture different syntactic and semantic patterns. Furthermore, they both have different parameters. Let us have a closer look.

3.1.1 Objectives

In the paper, two different objectives for the CBOW model are proposed. Mathematically, this can be expressed in two formulas, where w_j is the j-th word, and c is the window size. The first objective is to predict the next word-based on c previously encountered words.

$$\underset{W}{\text{maximize}}\; p(w_t | w_{t-c},\, w_{t-(c-1)},\, \ldots,\, w_{t-2},\, w_{t-1})$$

The second objective is to predict the word in the middle of c previously encountered words and c following words.

$$\underset{W}{\text{maximize}}\; p(w_t | w_{t-c},\, w_{t-(c-1)},\, \ldots,\, w_{t-1},\, w_{t+1},\, \ldots,\, w_{t+(c-1)},\, w_{t+c})$$

In this thesis, CBOW will always use the second objective, which is predicting the middle word from surrounding words. This leads to the objective of maximizing the average log probability.

$$\underset{W}{\text{maximize}}\; \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neg 0} \log p(w_t | w_{t+j})$$

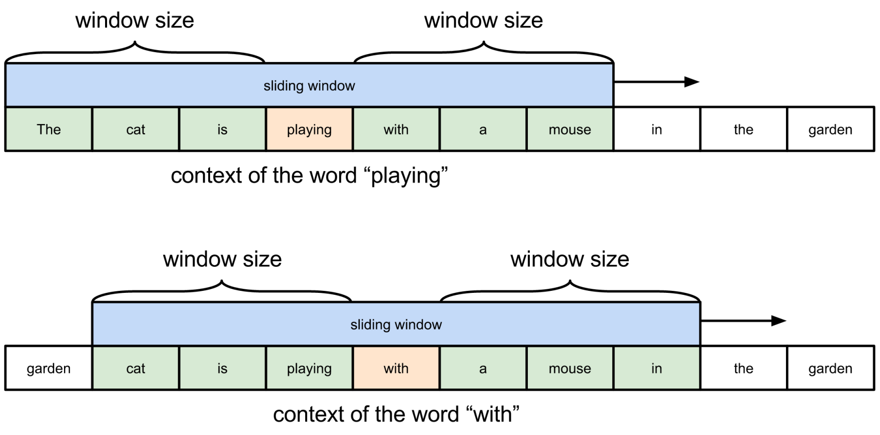


Figure 3.1: The window size of the sliding window defines which words are in the context of the middle word.

While the CBOW model tries to predict the middle word given a context, the skip-gram model tries to do the opposite, which is to predict the surrounding words given a single word.

$$\underset{W}{\text{maximize}}\ p(w_{t-c}, w_{t-(c-1)}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+(c-1)}, w_{t+c}|w_t)$$

Again, this leads to the objective of maximizing the average log probability.

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

3.1.2 Neural Network Language Model

To achieve the objectives described above, one recurrent neural network per objective is used. In the next few paragraphs, the neural network for the CBOW model is described. The neural network for the skip-gram model works analogous. The differences to the CBOW model are described at the end of this subsection.

The neural network consists of four layers: one input layer I, one projection layer P, one hidden layer H, and one output layer V. To model the weights of the connections between each connected layer, we use matrices. Since each two connected layers are fully connected to each other, the dimensionality of these matrices is the dimensionality of the upper layer times the dimensionality of the lower layer.

In the input layer, all words in the context of a word are listed, while the context is defined by the window size c. For example, if $c = 3$, then the context for the word "playing" of the sentence "The cat is *playing* with a mouse in the garden" is [the, cat, is, with, a, mouse], and the context for the word "with" is [cat, is, playing, a, mouse, in], see Figure 3.1. Therefore, the dimensionality of I is two times the window size c.

$$\dim(I) = 2c$$

Next, every word in the context is mapped to a d dimensional vector, where d is the word vector dimensionality. The weights of this projection define the resulting word vector. The dimensionality of the projection layer is the dimensionality of the input layer times the word vector dimensionality.

$$\dim(P) = \dim(I) \times d = 2c \times d$$

An equivalent model of the input and projection layer is to use the whole vocabulary U as input layer, while the words in the context are encoded with a 1 and the ones not in the context with a 0. Then the projection layer consists of all word vectors, and thus does not have to be changed when the context words change. Since the words that are not in the context are encoded as 0, the corresponding word vectors are multiplied by 0 and thus do not contribute to the prediction.

The hidden layer is fully connected to the projection layer, and has arbitrary dimensionality, usually between 500 and 2000.

$$\dim(H) = x \in [500,\ 2000]$$

Finally, the output layer has the same dimensionality as U, where the j-th value v_j is a score to calculate the conditional probability that the j-th word w_j occurs given the context $w_{x-c}, \ \ldots \ , w_{x+c}$. In the next subsection, we describe how to calculate these conditional probabilities.

$$\dim(V) = \dim(U)$$

Now let us consider the skip-gram model. The input layer consists of only one word, and the projection and the hidden layer have the same architecture and meaning as in the CBOW model. However, instead of having only one multinomial distribution, we have $2c$ multinomial distributions, one for each word in the context. Thus, the dimensionality of the neural network for CBOW is different.

$$\begin{aligned}
\dim(I) &= 1 \\
\dim(P) &= d \\
\dim(H) &= x \in [500,\ 2000] \\
\dim(V) &= 2c \times \dim(U)
\end{aligned}$$

3.1.3 Algorithm

As in the common neural network, the weights are learned via gradient descent and backpropagation. Let us again consider the algorithm for the CBOW. First, each word vector (one per word) is initialized randomly. Next, the documents are processed using a sliding window of size c (usually between 5 and 20). Then, the word vectors of the context are set as the projection layer, and the network tries to predict the middle word. The error gradient is then calculated and backpropagated by using soft-max. v_j is the value in the output layer per word in the vocabulary U.

$$p(w_t | w_{t-c}, \ldots, w_{t+c}) = \frac{e^{v_j}}{\sum_{j'=1}^{\dim(U)} e^{v_{j'}}}$$

This conditional probability is then compared to the actual output (the middle word). Based on the result (overestimation or underestimation), the error is backpropagated through the neural network, and the weights for the word vectors in this context are updated accordingly.

For example, consider again the sentence "The cat is playing with a *mouse* in the garden" with a window size $c = 3$ and the word to predict being "mouse". If the network predicts that the word "cat" is probable to appear (however, "cat" does not appear in the context of "mouse"), then this error is backpropagated and the word vectors for [playing, with, a, in, the, garden] are updated. In contrast, if the word "mouse" is predicted to be appearing in this context, the error is small and thus the backpropagation will not change the vectors [playing, with, a, in, the, garden] significantly.

The skip-gram algorithm works analogously to the CBOW model. It tries to predict the context of w_j, and backpropagates the mean error to change the word vector of w_j accordingly.

The two architectures are sketched in Figure 3.2. For a more extensive description on how exactly the Word2vec parameters are learned, see [17, 16, 19, 24, 7].

3.1.4 Computational Optimizations

While the two models CBOW and skip-gram discussed above are useful for understanding the basic models, these algorithms are computationally very inefficient. This is especially true when the vocabulary is large. Since there are nearly infinite amounts of written text available (for example Wikipedia, books, news, blogs) to train word embeddings on, it is imperative to minimize computational complexity in order to maximize accuracy by processing more text in less time. In this subsection, we will briefly describe the

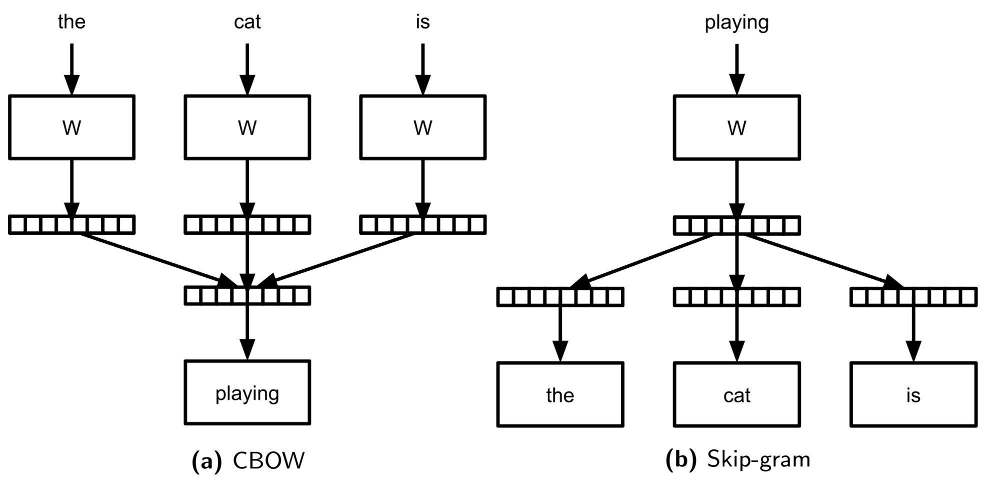


Figure 3.2: The architecture of the neural network for CBOW and skip-gram. On the top is the input layer, in the middle the hidden layer, and in the bottom the output layer.

theoretical optimizations. As we will later see in Chapter 5, additional optimizations regarding the implementation are also key to good results.

Since these optimizations are not necessary for understanding this thesis, we will only very briefly describe each concept. Interested readers can refer to [16, 24, 7] to understand these optimizations in greater depth.

Hierarchical soft-max [21, 20] calculates an approximated soft-max more efficiently.

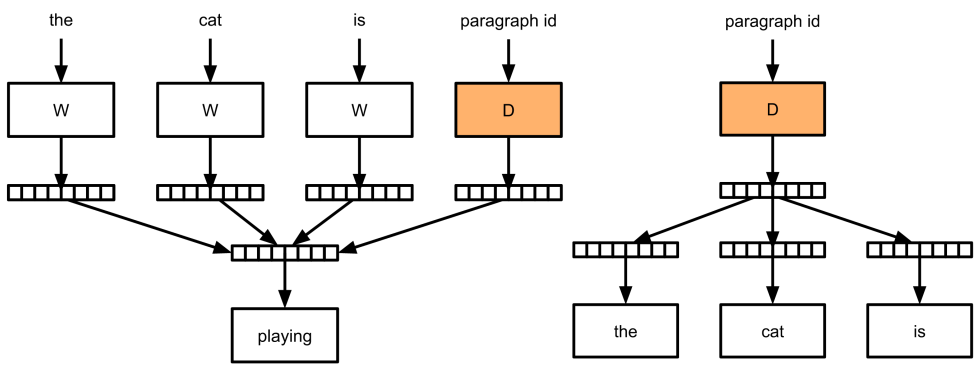
Negative sampling [16, 7] is a replacement for the soft-max and samples for every word w_j n words (usually between 5 and 20) from the vocabulary which are assumed not to appear with w_j.

Finally, *sub-sampling of frequent words* [16] favors rare words compared to frequent words, as rare words are more probable to contribute important information compared to frequent words like stop words. This is similar to the IDF part of TF-IDF.

3.2 Doc2vec

As described in the previous section, Word2vec works with a sliding window of a fixed size. Again, let us consider the sentence "The cat is playing with a mouse in the garden.". If $c = 3$, the words "cat" and "garden" will never occur together, and thus their respective word vectors will not be close to each other.

Doc2vec changes exactly that by adding an additional paragraph vector to each "paragraph". But let us first describe what is meant by a paragraph:



(a) PV-DM **(b)** PV-DBOW

Figure 3.3: The architecture of the neural network for PV-DM and PV-DBOW. On top is the input layer, in the middle the hidden layer, and in the bottom the output layer.

a "paragraph" is a text block with arbitrary length, for example a chapter, a section, a text paragraph or a sentence. In this section, the word "paragraph" should describe such a text block of arbitrary length, and a paragraph vector (PV) is a vector for such a paragraph, like a word vector is to a word.

Let us continue with the paragraph vector characterization. This paragraph vector serves as a common context, which is shared within the paragraph. In the previous example "The cat is playing with a mouse in the garden.", the paragraph vector allows to share a context between "cat" and "garden". This vector can also be thought of as a word which appears in the context of a larger entity than the sliding window size allows.

One key advantage of Doc2vec is that it can easily be built into Word2vec. Essentially, each paragraph vector is a word vector which appears in every context of the current paragraph. It can be trained the same way as the word vectors through gradient descent and backpropagation, using the same neural network.

The PV enhanced CBOW model is called paragraph vector distributed memory (PV-DM) to stress the role of the PV to act as distributed memory. For the skip-gram model, the PV enhanced model is called PV-DBOW and is illustrated in Figure 3.3.

Chapter 4

Hierarchical Paragraph Vectors

4.1 Hierarchies and NLP

Today, there is a vast amount of text available to be used in NLP. Many texts (for example articles, websites, books, comments, reviews and messages), are naturally divided into sub-texts (for example chapters, sections, paragraphs and sentences). Furthermore, they are usually also categorized or tagged (for example by authors, publishers, domain names, references, topics and genres).

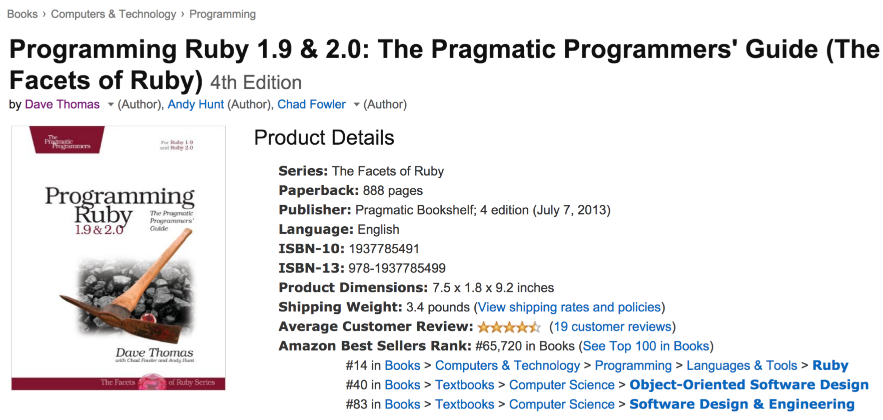
Consider for example a book about how to learn programming Ruby. This book is divided into multiple chapters, which, in turn, are divided into multiple sections and sentences. Furthermore, this book is found on Amazon in the category "Programming", which, in turn, can be found in "Computers & Technology", as can be observed in Figure 4.1.

Now what do these hierarchies represent? The structure represents concepts and abstractions, which represent potentially useful information. For example, if a person reads multiple articles about the French revolution, they can better predict which words will appear the next article about the French revolution. Thus, if this information can be used to learn the word embeddings, it lends itself for exploitation in the training of word embeddings.

4.2 Obtaining Hierarchies

From the perspective of a document, there are potentially two directions in which hierarchies are found. One is "upwards", and the other is "downwards". Let us define what we mean by that, and how these hierarchies can be obtained.

"Upwards" refers to hierarchies across multiple documents. As stated above, natural hierarchies are for example author names, publishers, domain names,



Books › Computers & Technology › Programming

Programming Ruby 1.9 & 2.0: The Pragmatic Programmers' Guide (The Facets of Ruby) 4th Edition

by Dave Thomas (Author), Andy Hunt (Author), Chad Fowler (Author)

Product Details

Series: The Facets of Ruby
Paperback: 888 pages
Publisher: Pragmatic Bookshelf; 4 edition (July 7, 2013)
Language: English
ISBN-10: 1937785491
ISBN-13: 978-1937785499
Product Dimensions: 7.5 x 1.8 x 9.2 inches
Shipping Weight: 3.4 pounds (View shipping rates and policies)
Average Customer Review: (19 customer reviews)
Amazon Best Sellers Rank: #65,720 in Books (See Top 100 in Books)
#14 in Books > Computers & Technology > Programming > Languages & Tools > **Ruby**
#40 in Books > Textbooks > Computer Science > **Object-Oriented Software Design**
#83 in Books > Textbooks > Computer Science > **Software Design & Engineering**

Figure 4.1: This book is categorized into "Programming" (on the very top of the image), "Ruby", "Object-Oriented Software Design" and "Software Design & Engineering", which, in turn, can be found in their respective parent category. There are also additional attributes, for example the language of the book and the authors.[1]

references, topics or genres. Where available, these can be obtained directly from the data. If this data is not available, it could be generated artificially by standard machine learning methods, for example by Latent Dirichlet Allocation (LDA).

"Downwards" describes hierarchies to be found within a single document. These could for example be chapters, paragraphs or sentences. These hierarchies are also often available directly through the data, or can be obtained by parsing the text or markup. For example, when an HTML document is given, the HTML tags (for example h1, h2, h3, p, br) can be used to split the text into multiple blocks, and the resulting text blocks can be parsed and split up by sentence.

4.3 Algorithms

4.3.1 HPV-DM

HPV-DM builds upon PV-DM. Instead of only allowing one paragraph vector per context, it allows multiple hierarchical paragraph vectors for the same context, as can be seen in Figure 4.2. Additionally, some hierarchical paragraph vectors appear multiple times in different contexts. Similar to the paragraph vectors, the hierarchical paragraph vectors can be thought of as additional memory for the current context, or as additional words which are always present in some context.

[1]Source: http://www.amazon.com/gp/product/1937785491

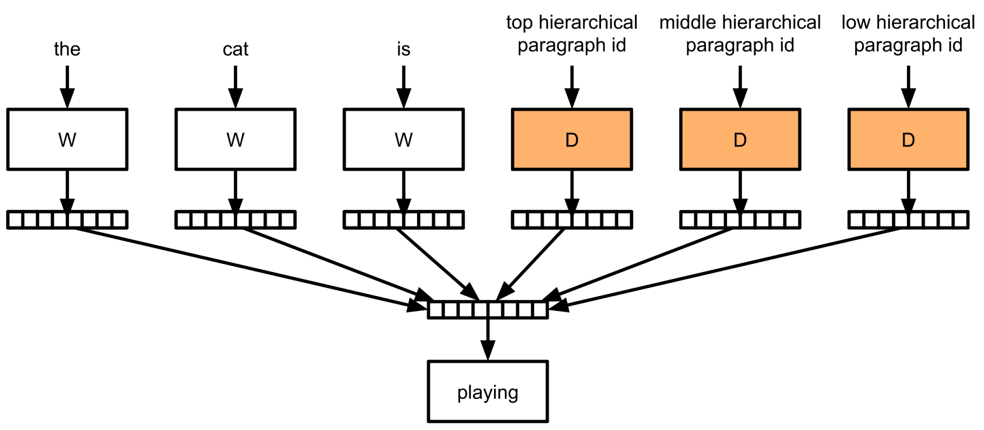

the
cat
is
top hierarchical paragraph id
middle hierarchical paragraph id
low hierarchical paragraph id
W
W
W
D
D
D
playing

Figure 4.2: The architecture of the neural network of HPV-DM. Multiple HPVs contribute to the prediction of the word in the context.

The algorithm works as follows. We split the text into multiple text blocks, according to the hierarchies we want to use. Every item in the hierarchy receives a *unique HPV identifier across the whole dataset*. Next, we add each resulting text block with the HPV annotations as paragraph vectors for training. Then, we train each sentence the same way we would train it with PV-DM, but now each text block has multiple HPVs (paragraph vectors), and each HPV is shared among text blocks which have the same HPV annotation. Therefore, each HPV annotation is expected to contribute to the prediction task. As a bonus, the algorithm generates an embedding for each HPV annotation.

For example, when we want to use category (upwards), document (the element for which we would like to have an embedding) and sentence (downwards), we generate, for example, "CAT-10" for category #10, "DOC-2" for document #2, and "DOC-2-SEN-3" for sentence #3 in document #2. Next, when sentence #3 in document #2 belongs to the categories #10 and #5, we generate "CAT-10", "CAT-5", "DOC-2" and "DOC-2-SEN-3" for this sentence. When we now train the sentences in document #2, the vector for annotation "DOC-2" is shared among the sentences.

4.3.2 HPV-DBOW

HPV-DBOW works analogously to the HPV-DM, but it enhances the PV-DBOW algorithm instead of the PV-DM algorithm. For the implementation, the only difference is that we use PV-DBOW instead of PV-DM for training. The neural network architecture is described in Figure 4.3. We notice that HPVs in higher hierarchies must support more abstract and general concepts.

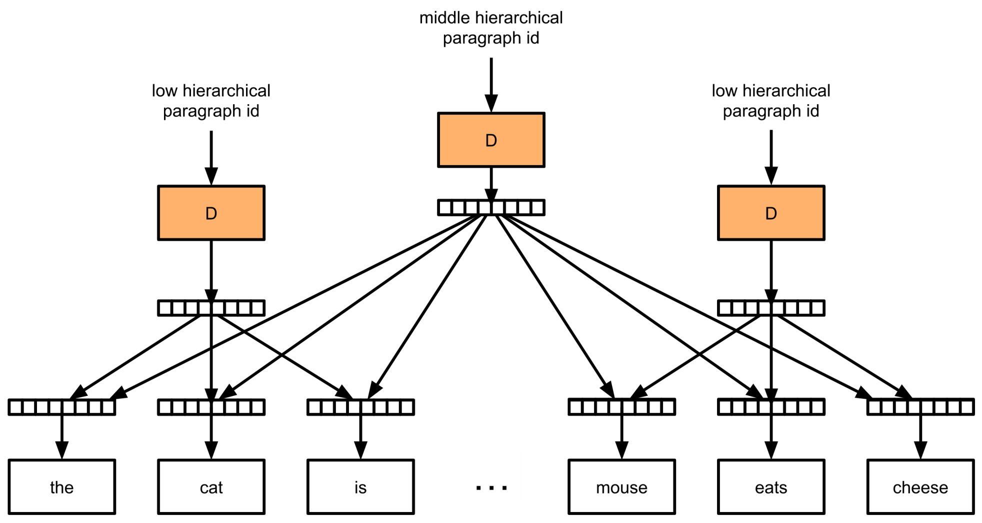

middle hierarchical paragraph id
low hierarchical paragraph id
low hierarchical paragraph id
D
D
D
the
cat
is
...
mouse
eats
cheese

Figure 4.3: The architecture of the neural network of HPV-DBOW. Every HPV predicts the context around it. HPVs in higher hierarchies represent more abstract concepts. For example, the lower HPV on the left side captures the behavior of cats, and the lower HPV on the right side captures the behavior of mice. The middle HPV must capture both concepts, which is the behavior of mammals.

4.3.3 Parameters

Because HPV-DM and HPV-DBOW directly extend HP-DM and HP-DBOW respectively, they directly inherit the parameters from these models. However, they introduce a new parameter that dictates which hierarchies should be exploited for learning. While this parameter is limited by the data available, it is important to consider only using the ones which improve the end result, as will be illustrated later.

4.4 Resources

When using HPV, we expect to extract additional information at the cost of greater overhead. To estimate how much overhead is introduced, we will discuss how the resources required change when running HPV compared to running PV. Therefore, we must first understand how PV influences the resources needed.

Let us start with some definitions. D is the (hierarchical) paragraph vector dimensionality, E is the word vector dimensionality, G is the number of unique different hierarchy elements used, K is the number of paragraph vectors per document, N stands for the number of documents used and U is the vocabulary.

For the Word2vec model, we need to store one word vector per word, which is $E \times \dim(U)$. The paragraph vector models need to additionally store the paragraph vectors, which is $D \times K \times N$ additional memory. Finally, HPV needs to store $D \times G$ additional hierarchical paragraph vectors. Let us write down the formula for the approximate additional memory consumption.

$$\frac{D \times G}{E \times \dim(U) + D \times K \times N}$$

Now, let us briefly discuss what these formulas mean by three typical examples. Let us assume $N = 100'000$ documents, $Z = 1'000$ average words per document, $D = E = 100$ dimensions, $dim(U) = 25'000$ vocabulary size, $W = 20$ window size, $K = 1$ paragraph vectors per document. For the first example, let us assume that we go up in the hierarchy, having 10 topics and 100 subtopics in total. This means $G = 110$. In this case, only about 0.009% additional memory is used. For the second example, assuming that we go downwards, we have 5 sentences per paragraph. This results in $G = 50'000$, which means that 143% additional memory is required. For the third example, let us assume that in addition to the sentences of the second example, there are 4 sub-sentences (in total 200'000) per sentence. Thus, $G = 250'000$, which results in 714% additionally required memory.

Because of the non-trivial optimizations of Word2vec, the runtime implications are more difficult to estimate. Thus, they are not estimated theoretically, but evaluated empirically in Chapter 5 instead.

4.5 Research Questions

As we have seen, HPV introduces additional model complexity. Depending on the number of implemented hierarchy layers, the model gains additional freedom. Can the neural network use this additional complexity to predict better values in the output layer? How many hierarchy layers improve the quality of the word embeddings, and which layers should be chosen? How much overhead is introduced in term of computational complexity and memory usage? Do some layers improve the learning speed of the neural network through the context sharing? We will address these questions in the Chapter 5.

Chapter 5

Experiments

The goal of the experiments is to evaluate the word vector quality. First, we briefly describe the data, preprocessing and evaluation metrics. Next, we describe the testing environments (hardware and software). Moreover, the baseline methods are outlined. Furthermore, the implemented HPV hierarchies are described, and the possible hyperparameters will be outlined. Finally, the results will be shown.

5.1 Experimental Setup and Data

5.1.1 IMDB Dataset

The Internet Movie Database (IMDB) dataset is a collection of movie reviews which have been written and rated by people. Each movie review consists of a text, which, in turn, consists of some sentences and has one or multiple paragraphs.

Additionally, half of the movie ratings have a rating $r \in [1,\ 10] \subseteq \mathbb{N}$, where a higher rating indicates that the reviewer likes the movie better. Furthermore, the movie ratings are divided into *negative ratings* (numeric rating of 1, 2, 3 or 4) and *positive ratings* (numeric rating of 7, 8, 9 or 10). *Neutral ratings* with a numeric rating of 5 or 6 have been excluded from the dataset. The task is to perform sentiment analysis on the movie reviews: for a given review text, predict if the sentiment is positive or negative.

Besides the rated reviews, the other half of the reviews do not contain a rating and thus can only be used to learn the word embeddings, as we will see later. The distribution of the 100'000 movie rating can be seen in Figure 5.1. More detailed information about the dataset can be found in [15].

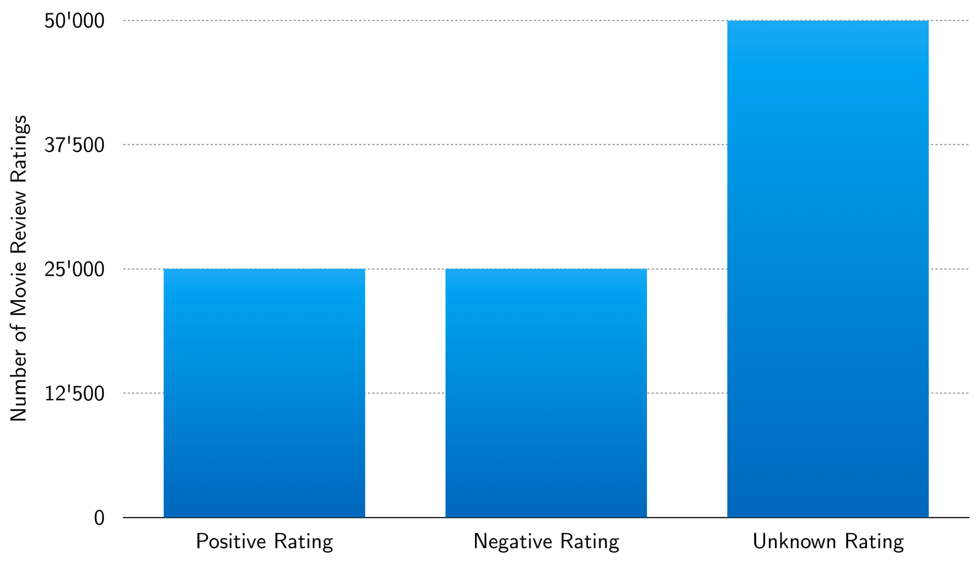


Figure 5.1: The movie review ratings consist of an equal amount of positive and negative ratings. For half of the dataset, the rating is unknown.

5.1.2 Preprocessing

Preprocessing of text can lead to significant differences in the outcome of the results. Additionally, it can also help to reduce computational complexity, for example by removing invalid characters, which would be used as words, or by converting some similar characters to one general character. However, preprocessing is not the focus of this master thesis, and is therefore only applied limitedly.

First, different single quote symbols (for example ′, ″, ‘) are consolidated, and so are different double quote symbols. Next, frequently used HTML tags are converted to text symbols. Furthermore, common smilies are replaced by text symbols. Finally, punctuation, brackets, and commonly used special characters (for example dollar, hashtag) are replaced by text symbols. Other invalid symbols and uncommonly used characters are removed from the text. Finally, all text is converted to lowercase.

5.1.3 Software

The original implementation by Mikolov only works for Word2vec, and would have to be extended to work with Doc2vec. Furthermore, this implementation is written in pure C, and is therefore difficult to modify, extend, and interact with.

There exist many implementations for Doc2vec. One of the fastest, if not the fastest implementation, is Gensim[1]. Furthermore, the implementation has a well-defined and simple interface to Python[2], and is well documented. For these reasons, Gensim was chosen as a library for the paragraph vector calculation. The exact versions are 3.4.3 for Python, and 0.12.1 for Gensim. The source code produced in the course of this thesis project can be found here[3].

5.1.4 Distributed Implementation

To run a large amount of experiments and to record the results, the experiments needed to be run on a cluster. Thus, an implementation, which can distribute jobs and record results, was required. One job that consists of the parameters described below is executed on exactly one node, and is run through the whole dataset. Thus, the computation is not distributed in the traditional map-reduce way.

Running one batch of experiments works as follows: on one central server, the experiments are scheduled. Then, a client requests the parameters for one experiment to run, runs the experiment with these parameters, and submits the result to the central server. This process is executed as long as there are scheduled experiments. An overview of the architecture can be found in Figure 5.2.

The central server is implemented in Ruby on Rails[4]. In addition to the scheduling and result collection, the server also implements an interface to analyze the results and to generate charts. The client is implemented in Python, see 5.1.3. Multiple clients request, run and submit experiments concurrently.

5.1.5 Hardware

The software was developed on an "Apple MacBook Pro, Core i7 2.8 (I7-4980HQ), 15-Inch, Mid-2014, Dual Graphics", model id "MacBookPro11,3"[5]. Also, the performance tests in 5.6.2 have been conducted on this laptop. The operating system of the laptop was OS X Yosemite 10.10.5.

The restricted and unrestricted best configuration experiments in 5.6.1 have been executed on 8 nodes of the DCO cluster at ETH. Each machine has 16 physical cores and 128 GB of physical RAM. The operating system of the servers was Fedora 21[6].

[1] https://radimrehurek.com/gensim/

[2] https://www.python.org/

[3] https://github.com/lukaselmer/hierarchical-paragraph-vectors

[4] https://github.com/lukaselmer/simple-job-runner

[5] http://www.everymac.com/systems/apple/macbook_pro/specs/macbook-pro-core-i7-2.8-15-dual-graphics-mid-2014-retina-display-specs.html

[6] https://getfedora.org/

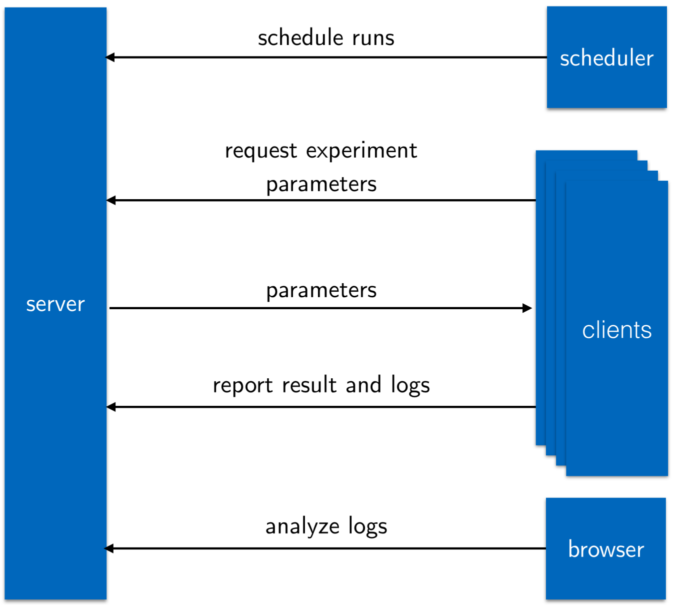


Figure 5.2: Architectural overview of the distributed implementation. The tiers communicate via HTTPS and JSON.

5.2 Evaluation Metrics

Recall that the dataset is balanced between positive and negative ratings. Therefore, a random estimator would guess the sentiment of 50% of the ratings correctly. The balanced dataset allows us to use accuracy as the first evaluation metric. We use the following definition to calculate the mean accuracy from the predicted reviews R.

$$\text{accuracy} = \frac{1}{\dim(R)} \sum_{r \in R} \begin{cases} 1 & \text{when } r_{\text{predicted sentiment}} = r_{\text{actual sentiment}} \\ 0 & \text{otherwise} \end{cases}$$

Note that this is a binary classification task.

The second and third evaluation metrics are training speed (computational complexity) and memory footprint which are measured empirically. The training speed is calculated by measuring the time from starting one training epoch until the training epoch has ended. Here, it is important to note that during the test, only one active program is being run on the machine, and no other computationally intensive programs are being run in the background. The memory footprint is measured by the size of a file to which the model is serialized.

Table 5.1: Overview of the implemented hierarchies.

Abbreviation	*Hierarchies*
NO-HPV	None
HPV-TOP	Topics
HPV-TOP-PAR	Topics, Paragraphs
HPV-TOP-PAR-SENT	Topics, Paragraphs, Sentences
HPV-PAR	Paragraphs
HPV-PAR-SENT	Paragraphs, Sentences
HPV-PAR-SENT-SUB	Paragraphs, Sentences, Sub-sentences
HPV-PAR-SENT-SUBNV	Paragraphs, Sentences, Sub-sentences (but without training the sub-sentence vectors)

5.3 Baseline Methods

There are two methods which draw a baseline to compare HPV to. The first baseline is to generate the word embeddings by using the commonly used term frequency - inverse document frequency (TF–IDF) method [23]. For the TF–IDF vectors, the features are ordered by term frequency across the corpus, and only the top *n* features are used to get a low-dimensional vector. More details about the implementation can be found in A.2.

The second baseline method is the standard implementation of Doc2vec provided by Gensim. As we will see, the same hyperparameters are used for the baseline and the HPV enhanced implementation.

Both baseline methods use the same classifier as the HPV implementations, which is a support vector classifier (SVC), see Table 5.3.

5.4 Implemented HPV Hierarchies

As we have seen before, there are two directions from which hierarchies can be obtained (upwards and downwards). Let us first investigate the upwards direction and thereafter continue with the downwards direction.

The IMDB movie rating dataset has natural hierarchies (for example the movie category, actors, year), but unfortunately, these hierarchies are not annotated in the dataset. Therefore, we use LDA to extract a synthetic hierarchy. We extract 20 topics from the data by running LDA for 20 iterations. Then we assign the two most corresponding topics, which have a probability greater 25% to occur per movie review. This topic is then stored as a special word. Runs using these topics are labeled as “TOP”.

Now let us turn our attention to the “downwards” hierarchies. First, we use paragraphs as a hierarchy, which we denote as “PAR”. Next, we parse

Table 5.2: The number of extracted elements from the data after preprocessing and splitting considering the HPV Hierarchies.

Variable	*Value*
Topics	20
Documents	100’000
Paragraph Vectors per Document	1
Vocabulary Size	185’957
Total Words	27’527’432
Average Words per Document	275
Paragraphs	303’448
Sentences	1’613’409
Sub-sentences	3’061’360

the sentences, denoted as “SENT”. Finally, “SUB” stands for sub-sentences, and “SUBNV” stands for sub-sentence splitting, but not learning a vector per sub-sentence. Finally, the hierarchies are combined. An overview can be found in Table 5.1. When we preprocess and split up the dataset, we extract the following number of elements displayed in Table 5.2. The exact implementation of the splitting is described in A.3.

5.5 Hyperparameters

Hyperparameter search is known to be a key challenge in applied machine learning. There are known procedures, such as grid search, but in practice, this problem is more difficult than it is in theory. The climax of this challenge is the explosive count of combinations of hyperparameters, where each combination could potentially outperform a very distant other combination, making it a non-convex problem. What makes it even more difficult is that it takes much time for one combination to evaluate, which we will see later.

In Table 5.3, the values of the most important hyperparameters are stated. Furthermore, the ranges, in which these values have been searched in, are described.

5.5.1 Epochs

The number of iterations the dataset is processed. For each iteration, the order of the texts is shuffled.

5.5.2 HPV Hierarchies

The hierarchies which should be exploited, see 5.4.

5.5.3 Word Vector Dimensionality

The dimensionality of the word vectors and paragraph vectors for HPV-DM and HPV-DBOW *each*.

5.5.4 Window Size

The window size in each direction (left and right), see Figure 3.1.

5.5.5 Negative Sampling

The number of negative samples per positive sample.

5.5.6 Frequent Word Downsampling HPV-DM

Threshold to downsample frequent words for HPV-DM.

5.5.7 Frequent Word Downsampling HPV-DBOW

Threshold to downsample frequent words for HPV-DBOW.

5.5.8 Learning Rate Type

The function that shows how to decrease the learning rate. Exponential (exp) or linear (lin). More details can be found in the Appendix A.1.

5.5.9 Classifier

The classifier used to learn (from the word embeddings and given ratings) to predict the rating (for new word embeddings / the test set). Additionally, different classifiers have different hyperparameters. Different classifiers have been tried.

RBF Radial basis function approximation using the Nystroem method[7].

LOG1 Logistic regression using scikit-learn[8].

LOG2 Logistic regression using statsmodels[9].

SVC Support vector classifier using scikit-learn[10].

[7] http://scikit-learn.org/stable/auto_examples/plot_kernel_approximation.html

[8] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

[9] http://statsmodels.sourceforge.net/0.6.0/generated/statsmodels.discrete.discrete_model.Logit.html

[10] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

Table 5.3: The value ranges and the values chosen from the most important hyperparameters. The step sizes for the value ranges were exponential at first, and after a reasonable hyperparameter combination was found, they were linear for fine-tuning.

Name	*Range / Options*	*Best / Chosen*
Epochs	[5, 50]	20
HPV Hierarchies	TOP, PAR, SENT, SUB, SUBNV	TOP
Word Vector Dimensionality	[16, 2000]	200, 48
Window Size	[5, 25]	10
Negative Sampling	[5, 30]	25
Frequent Word Downsampling HPV-DM	$[10^{-10}, 0.1]$	10^{-5}
Frequent Word Downsampling HPV-DBOW	$[10^{-10}, 0.1]$	10^{-3}
Learning Rate Type	exp, lin	exp
Classifier	RBF, LOG1/2, SVC	SVC

NN A neural network classifier using scikit-neuralnetwork[11].

5.6 Results

For this master thesis, more than 23’000 runs have been recorded systematically. Before that, the algorithm and framework have been developed, and the hyperparameters which should vary have been defined. Then, the first 10’000 runs have been used to find the best hyperparameters for NO-HPV and HPV-PAR-SENT-SUB using grid search. For the next 2’000 runs, HPV-PAR-SENT-SUBNV and HPV-PAR-SENT have been compared against NO-HPV and HPV-PAR-SENT-SUB. During the next 3’000 runs, the TF–IDF baseline has been evaluated, also in combination with NO-HPV, HPV-PAR-SENT-SUB, HPV-SUBNV and HPV-SENT-SUB. Furthermore, HPV-TOP and HPV-TOP-PAR-SEN were implemented, and 2’000 runs were conducted to compare them against NO-HPV. Moreover, HPV-TOP-PAR and HPV-PAR were implemented and compared against NO-HPV for the next 2’000 runs. Finally, the best performing configurations have been chosen to perform significance tests by executing multiple runs using the same configuration for the next 3’000 runs, before running the TF–IDF baseline for another 1’000 runs.

In this section, we will first investigate the accuracy. After this, we will turn our attention to the runtime and memory usage.

[11]https://github.com/aigamedev/scikit-neuralnetwork

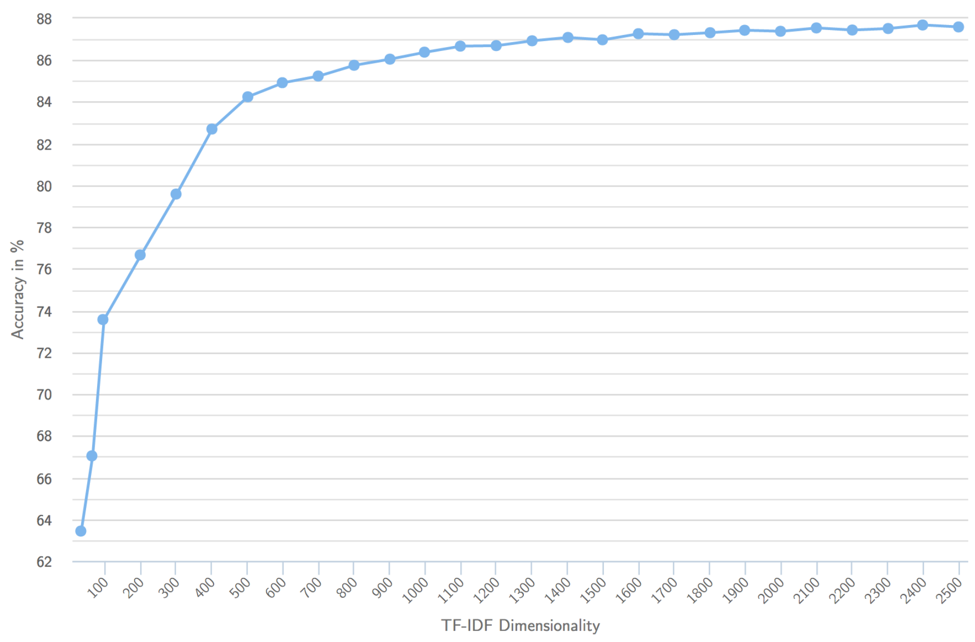


Figure 5.3: The accuracy increases when the TF–IDF dimensionality is increased.

5.6.1 Accuracy

In this subsection, we will compare the accuracy of the HPV implementations to each other and to the baselines Doc2vec and TF–IDF. First, TF–IDF is compared to Doc2vec. Next, the different HPV implementations are compared to each other. Finally, the best configurations are compared to the Doc2vec baseline NO-HPV.

Term Frequency - Inverse Document Frequency (TF–IDF)

As we can see in Figure 5.3, the accuracy increases when we increase the TF–IDF dimensionality. We also notice a linear increase between the 100 and 600 dimensions. When increasing the dimensionality further, the accuracy flattens out. The best accuracy of 87.9% for TF–IDF was measured with a dimensionality of 2700. When increasing the dimensionality even further up to 4000 dimensions, the accuracy stayed between 87.5% and 87.9%.

When we compare the result to the values after 10 epochs of Doc2vec with 48 dimensions, see Figure 5.6, we notice that the accuracy is approximately 88.5% in the Doc2vec case (using a total of 96 dimensions), compared to only 73.5% accuracy in the TF–IDF case (using 96 dimensions). This means that Doc2vec outperforms TF–IDF by a substantial margin when using low-dimensional vectors.

Let us now compare TF–IDF to Doc2vec with 200 dimensions, see Figure 5.5.

The accuracy for Doc2vec is approximately 90% after multiple epochs. When we compare this to the TF–IDF score using 400 dimensions, we find an accuracy of 82.7%. Again, Doc2vec beats the TF–IDF baseline by a considerable margin.

Different HPV Implementations

To compare the different HPV implementations against each other, the hyperparameters described in Table 5.3 have been used. Because some HPV implementations clearly outperformed other implementations in multiple different configurations, instead of using multiple runs and taking the average accuracy, only one run per implementation has been conducted with this configuration. Furthermore, some HPV implementations have a slow execution speed, as we will see in 5.6.2. Therefore, the additional computational cost to run multiple experiments for HPV implementations, which performed poorly, was not justified.

Let us now compare the different HPV implementations in Figure 5.4. First, we notice that after one epoch, all HPV implementations using the topic (TOP) as a hierarchy significantly outperform NO-HPV by about 2%. Also, HPV-TOP outperforms NO-HPV here. However, this gap is not significant for all epochs, which we will see later on. Next, we notice that the more we move down in the hierarchies, the worse the accuracy develops. For example, when using paragraphs, sentences and sub-sentences, the model performs significantly worse than NO-HPV. Finally, when combining the topic with the lower hierarchies, the topic seems to boost the accuracy throughout all epochs, and the additional hierarchies prevent a strong increase in accuracy compared to NO-HPV.

The best HPV implementations from this comparison (NO-HPV, HPV-TOP, HPV-PAR) are compared in more detail in the next two subsections. HPV-TOP-PAR is left out for this comparison, because it is a combination of HPV-TOP and HPV-PAR, which both are compared against each other in the next subsections.

Unrestricted Best Configuration

To compare the accuracy of the best HPV implementations to NO-HPV, the best two configurations have been chosen. The first configuration is unrestricted and thus can use an arbitrary word vector dimensionality. It uses a word vector dimensionality of 200 and the hyperparameters described in Table 5.3. The experiment has been repeated for 30 times with random initializations. Because these results represent the most important results of the empirical evaluation, significance tests have been conducted. We are using a wilcoxon signed rank test, and our hypothesis is that HPV-TOP outperforms

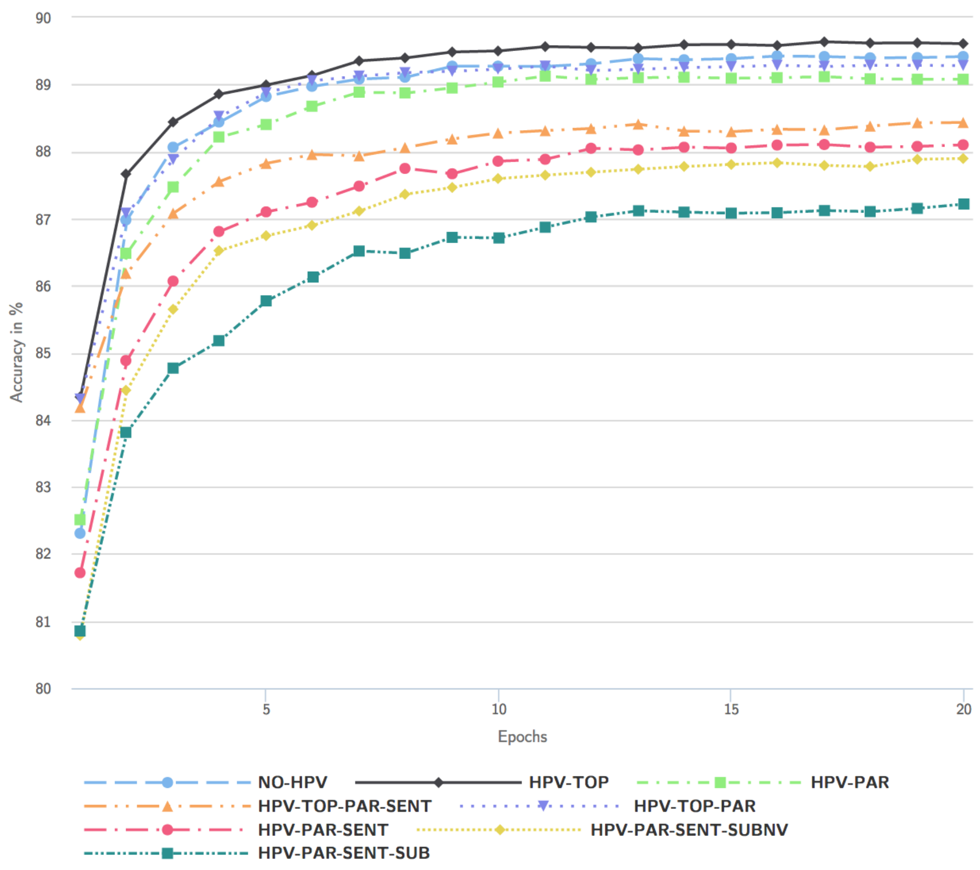


Figure 5.4: The accuracy increases when increasing the epochs for every HPV implementation, using a word vector dimensionality of 48. TOP boosts the accuracy significantly in the beginning. Low hierarchies do not increase the accuracy.

NO-HPV, which makes it a directional test. We use a level of significance of 0.025, which corresponds to a minimum z-score of 1.960[12].

As we can see in Figure 5.5 and Table 5.4, HPV-TOP clearly outperforms NO-HPV when only one epoch is run. When increasing the epochs, the accuracy of the prediction increases significantly for both of these implementations. HPV-TOP outperforms NO-HPV slightly by only about 0.1%. We also notice that we reach a plateau after about 9 epochs, after which the accuracy only fluctuates marginally.

When we compare HPV-PAR to NO-HPV, we notice that after only one epoch of training, they perform very similarly. However, as the epochs progress, HPV-PAR loses steam and cannot increase the accuracy as much as NO-HPV can. Thus, NO-HPV and HPV-TOP clearly outperform HPV-PAR.

[12] http://vassarstats.net/textbook/ch12a.html

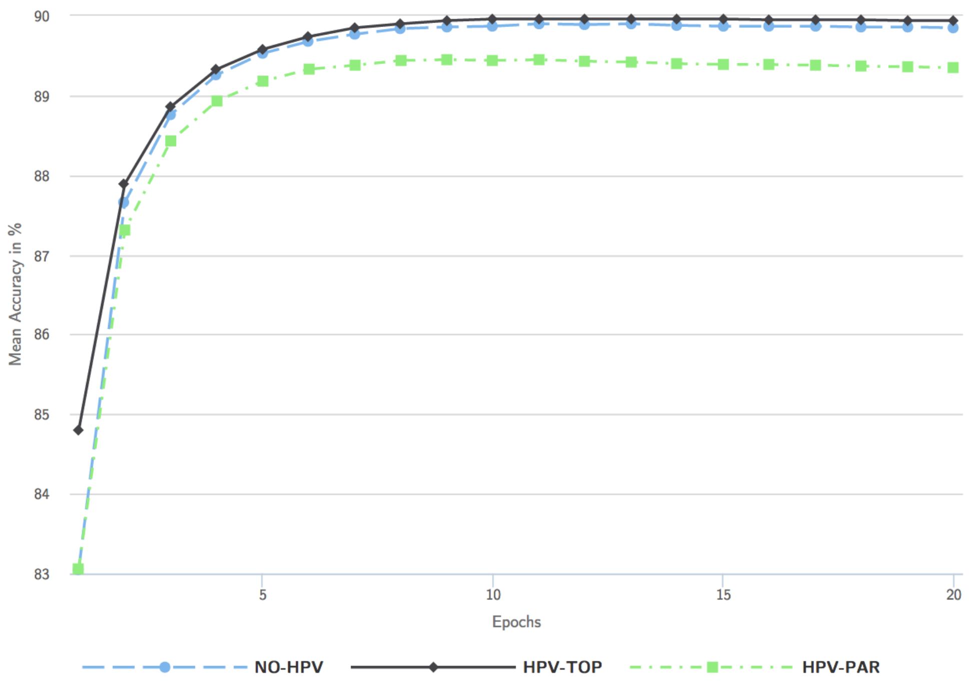


Figure 5.5: The mean accuracy of 30 experiments when increasing the epochs for the best configuration, using a word vector dimensionality of 200.

Restricted Best Configuration

The second best configuration is restricted to use a low word vector dimensionality of 48 dimensions for HPV-DM and HPV-DBOW *each*. For the significance tests, we apply the same strategy as in the best unrestricted configuration above, using 30 experiments with random initializations, a 0.025 level of significance, and the hypothesis that HPV-TOP outperforms NO-HPV.

In Figure 5.6 and Table 5.5, we again see an initial boost when using HPV-TOP compared to NO-HPV. However, after 8 epochs, the accuracy of HPV-TOP and NO-HPV converge. Also, in contrast to the 200 dimensional experiment, the accuracy increases slightly with further training.

For HPV-PAR, we notice similar behavior as before. After one epoch, the accuracy of HPV-PAR is the same as NO-HPV, and after that, HPV-PAR stays below NO-HPV and HPV-TOP.

5.6.2 Execution Speed and Memory Usage

In this subsection, we compare the execution speed and the memory usage of the different HPV implementations. For the memory usage, we try to validate our theoretical calculation from 4.4.

Table 5.4: The data displayed in Figure 5.5, which is the unrestricted best configuration, using word vector dimensionality of 200. The values, where HPV-TOP outperforms NO-HPV with a level of significance of 0.025, are marked with a *.

	Accuracy [%]		
Epoch	*HPV-NO*	*HPV-TOP*	*HPV-PAR*
1	83.05	*84.8	83.06
2	87.66	*87.89	87.31
3	88.76	*88.86	88.43
4	89.26	*89.33	88.93
5	89.53	89.58	89.18
6	89.68	*89.74	89.33
7	89.77	*89.85	89.38
8	89.84	*89.9	89.44
9	89.86	*89.94	89.45
10	89.87	*89.96	89.44
11	89.9	*89.96	89.45
12	89.89	*89.96	89.43
13	89.9	89.96	89.42
14	89.88	*89.96	89.4
15	89.87	*89.96	89.39
16	89.87	*89.95	89.39
17	89.87	*89.95	89.38
18	89.86	*89.95	89.37
19	89.86	89.94	89.36
20	89.85	*89.94	89.35

Execution Speed

To evaluate the execution speed, we ran 20 experiments with a word vector dimensionality of 48. The HPV implementation and the model influences the runtime substantially, which can be seen in Table 5.6. Furthermore, as we can see in Table 5.7, in total, all HPV implementations are slower than the Doc2vec baseline. However, we notice that some [HPV, model] combinations, for example [NO-HPV, HPV-DM] or [HPV-TOP, HPV-DM], have comparable execution speed.

Overall, the advantage HPV gains by training "faster" compared to PV when using a fixed amount of epochs, it is overshadowed by the slower training speed of HPV per epoch. This effect is displayed in Figure 5.7, where the epochs are scaled according to the execution overhead.

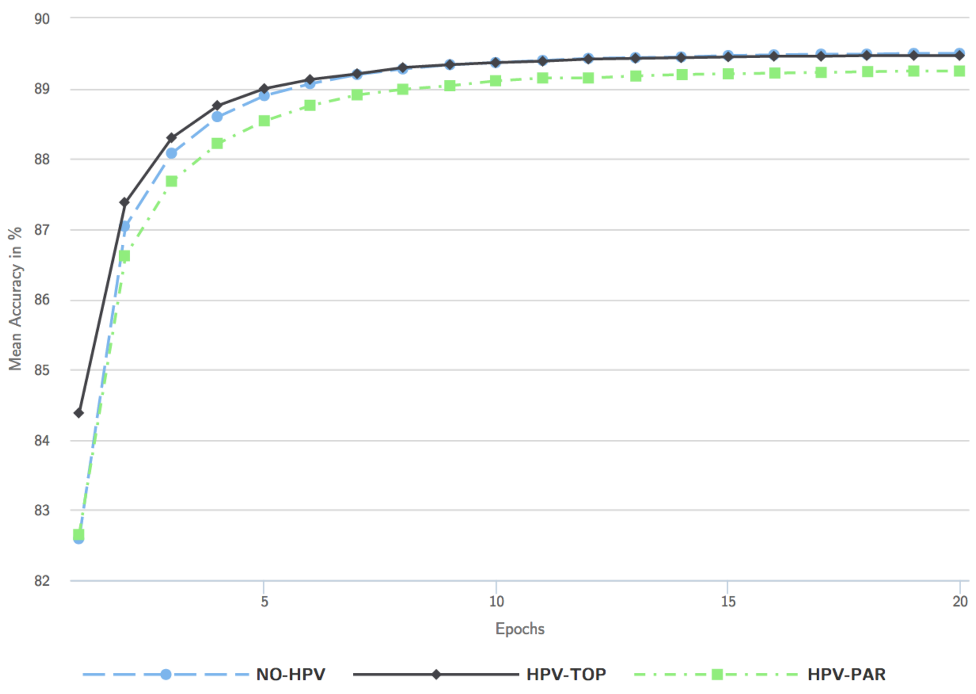


Figure 5.6: The mean accuracy of 30 experiments when increasing the epochs for the best configuration, using a word vector dimensionality of 48.

Memory Usage

Recall the formula for the additionally needed memory from 4.4.

$$\frac{D \times G}{E \times \dim(U) + D \times K \times N}$$

We ran the memory usage test with a window size $W = 10$ and $D = E = 48$ dimensions. From Table 5.2 we know $N = 100'000$ documents, $Z = 275$ average words per document, $dim(U) = 185'957$ vocabulary size, and $K = 1$ paragraph vectors per document. G depends on the HPV implementation. From the results displayed in Table 5.8 we notice that the formula for the memory usage overhead only fits approximately, and fails to predict the difference between HPV-PAR-SENT and HPV-PAR-SENT-SUBNV. At the time of writing, the source of this discrepancy is unknown, and may be caused by the implementation. Thus, it is not recommended to rely on the formula for the memory usage prediction.

Table 5.5: The data displayed in Figure 5.6, which is the restricted best configuration, using word vector dimensionality of 48. The values, where HPV-TOP outperforms NO-HPV with a level of significance of 0.025, are marked with a *. NO-HPV never outperforms HPV-TOP with a level of significance of 0.05 or lower.

	Accuracy [%]		
Epoch	*HPV-NO*	*HPV-TOP*	*HPV-PAR*
1	82.59	*84.38	82.65
2	87.04	*87.38	86.62
3	88.08	*88.3	87.68
4	88.6	*88.76	88.22
5	88.9	*89	88.54
6	89.07	*89.13	88.76
7	89.2	89.21	88.91
8	89.28	89.3	88.99
9	89.34	89.34	89.04
10	89.37	89.37	89.11
11	89.4	89.39	89.15
12	89.43	89.42	89.15
13	89.44	89.43	89.18
14	89.45	89.44	89.2
15	89.47	89.45	89.21
16	89.48	89.46	89.22
17	89.49	89.46	89.23
18	89.49	89.47	89.24
19	89.5	89.47	89.25
20	89.5	89.47	89.25

Table 5.6: Execution speed for training the word embeddings with a word vector dimensionality of 48 per epoch. 20 experiments per HPV implementation and model (HPV-DBOW, HPV-DM) have been conducted to calculate the average duration in seconds.

HPV	*Model*	*Average Duration [s]*	*Standard Deviation*
NO-HPV	HPV-DBOW	23.71	0.36
	HPV-DM	18.01	0.65
HPV-TOP	HPV-DBOW	65.61	0.61
	HPV-DM	18.7	0.42
HPV-TOP-PAR	HPV-DBOW	88.85	0.72
	HPV-DM	27.2	0.55
HPV-TOP-PAR-SENT	HPV-DBOW	118.62	9.19
	HPV-DM	89.98	4.3
HPV-PAR	HPV-DBOW	45.91	0.83
	HPV-DM	26.98	0.23
HPV-PAR-SENT	HPV-DBOW	76.63	3.56
	HPV-DM	78.85	0.22
HPV-PAR-SENT-SUBNV	HPV-DBOW	111.79	5.7
	HPV-DM	135.13	1.17
HPV-PAR-SENT-SUB	HPV-DBOW	120.08	2.84
	HPV-DM	140.72	1.6

Table 5.7: Execution speed for training the word embeddings with a word vector dimensionality of 48 per epoch. Here, the results from Table 5.6 are summarized.

	Total Average Duration	
HPV	*[s]*	*[%]*
NO-HPV	41.72	100
HPV-TOP	84.31	202
HPV-TOP-PAR	116.05	278
HPV-TOP-PAR-SENT	208.60	500
HPV-PAR	72.89	175
HPV-PAR-SENT	155.48	373
HPV-PAR-SENT-SUBNV	246.92	592
HPV-PAR-SENT-SUB	260.80	625

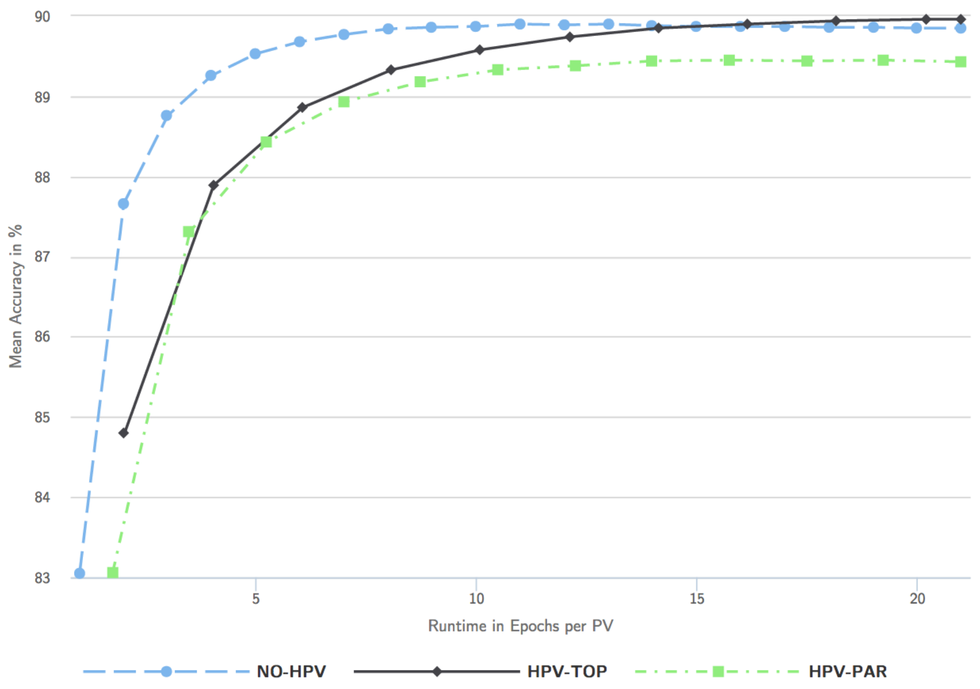


Figure 5.7: The mean accuracy of 30 experiments when increasing the epochs for the best configuration, using a word vector dimensionality of 200, where the epochs are scaled in relation to the execution overhead compared to NO-HPV. When considering this overhead, HPV-TOP trains considerably slower than NO-HPV, despite the initial boost in the first training epoch of HPV-TOP.

Table 5.8: Predicted memory usage vs. measured memory usage, using a word vector dimensionality of 48. RAM is the measured "real memory" as reported by task_info. It was measured during the training of the word embeddings model by using "ps aux" and represents a single observation.

		Memory Overhead [%]		
		Serialized Model		*RAM*
HPV	*G*	*Predicted*	*Measured*	*Measured*
NO-HPV	0	0	0	0
HPV-TOP	20	0	0	2
HPV-TOP-PAR	303468	161	121	15
HPV-TOP-PAR-SENT	1916877	1019	780	110
HPV-PAR	303448	161	121	18
HPV-PAR-SENT	1916877	1019	780	93
HPV-PAR-SENT-SUBNV	1916857	1019	1657	120
HPV-PAR-SENT-SUB	4978237	2647	2039	217

Chapter 6

Conclusion and Future Work

To summarize, we introduced a novel method for generating high-quality low-dimensional word embeddings by extending the state-of-the-art methods PV-DM and PV-DBOW, using hierarchical data. We compared the novel algorithms HPV-DBOW and HPV-DM to the baselines PV-DM, PV-DBOW and TF-IDF.

We evaluated the word embeddings by applying HPV-DBOW and HPV-DM to the IMDB movie review dataset and measured the accuracy of the predicted sentiment. To retain comparable results, we conducted a broad search to find suitable hyperparameter combinations. We reached an accuracy of 90%, using only elementary preprocessing.

Moreover, we implemented 7 different hierarchical paragraph vector variants, and conclude that using high-level hierarchies, such as topics and categories, we can increase the quality of the word embeddings at a cost of greater execution overhead. Low-level hierarchies, such as paragraphs and sentences, do not contribute to improving the quality of the word vectors. Furthermore, when using few training epochs, we have shown that hierarchical data can boost the quality of the word embeddings, but when scaled according to the execution speed, HPV trains slower than PV.

Finally, we analyzed the memory and runtime overhead when using HPV with different layers of hierarchies, and we compared it to PV-DM and PV-DBOW. While the overhead is significant, it is still feasible to process large amounts of text.

For future work, we suggest that HPV is evaluated on more datasets, where other hierarchies are available for training, and where HPV be evaluated on other tasks besides sentiment analysis. Furthermore, the embeddings should be trained and evaluated on larger corpora, so that multiple passes through the data may not be necessary.

Regarding the runtime and memory usage, we suggest that the implementation be analyzed in more detail. This can lead to optimizations which improve training speed and memory efficiency. Also, we recommend implementing different dimensionalities for word vectors and hierarchical paragraph vectors, ideally so that a different dimensionality can be defined per hierarchy layer.

Finally, more combinations of models and parameters should be evaluated, and HPV-DBOW and HPV-DM should be tested independently and in combination with other models.

Appendix A

Appendix

A.1 Learning Rate Type

For all final experiments, a maximum learning rate of $l_{max} = 0.025$ and a minimum learning rate of $l_{min} = 0.001 = 10^{-3}$ have been used. In the beginning, we used $l_{min} = 10^{-4}$, but after the learning rate decreased to below 10^{-3}, the accuracy did not change anymore. Thus, the minimum learning rate was set to 10^{-3}).

When only one epoch is scheduled, we use l_{max}. Otherwise, the learning rate is decreased after each epoch, using either a linear or an exponential function, see below.

A.1.1 Linear

By linear learning rate we mean a function which decreases linearly from the maximum learning rate l_{max} to the minimum learning rate l_{min}. To calculate the learning rate for each step, we first set the learning rate of the first step to l_{max}, and the learning rate of the last step n to l_{min}. The learning rate for epoch i is calculated as follows.

$$l(i) = l_{max} - (i-1) \times \frac{l_{max} - l_{min}}{n-1}$$

A.1.2 Exponential

The exponential case works analogously to the linear case. The learning rate for epoch i is calculated as follows.

$$l(i) = l_{max} \times \left(\sqrt[l_{max}-1]{\frac{l_{min}}{l_{max}}} \right)^{i-1}$$

A.2 TF-IDF

We use scikit-learn version 0.16.1 for the TF-IDF calculation[1] with all default options, except the option "max_features", which represents the maximum vector dimensionality.

From the documentation: "The actual formula used for tf-idf is $tf \times (idf + 1) = tf + tf \times idf$, instead of $tf \times idf$. The effect of this is that terms with zero idf, i.e. that occur in all documents of a training set, will not be entirely ignored. [. . .] Parameter max_features: If not None, build a vocabulary that only consider the top max_features features ordered by term frequency across the corpus."

A.3 Technical Implementation of Hierarchy Splitting

Hierarchy splitting is implemented in a modest fashion, which essentially works using regular expressions and is tuned towards the IMDB dataset. We use the following expressions to split each hierarchy. The implementation can be found here[2].

Paragraphs "<br /><br />", "<br />", "<hr>".

Sentences "!", "?", ".", and consecutive combinations of these symbols, for example "!!" and "!?!".

Sub-sentences ",", ";", ":", "(", ")".

[1] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[2] https://github.com/lukaselmer/hierarchical-paragraph-vectors

Bibliography

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann Machines. *Cognitive Science. Special Issue: Connectionist models and their applications*, 9(1):147–169 ST – A learning algorithm for Boltzmann M, 1985.

[2] Yoshua Bengio and Yann LeCun. Scaling Learning Algorithms towards AI. In Léon Bottou, Olivier Chapelle, D DeCoste, and J Weston, editors, *Large Scale Kernel Machines*, number 1, chapter 14, pages 321–360. MIT Press, 2007.

[3] Yoshua Bengio, Holger Schwenk, Jean Sébastien Senécal, Fréderic Morin, and Jean Luc Gauvain. Neural probabilistic language models. *Studies in Fuzziness and Soft Computing*, 194:137–186, 2006.

[4] Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference*, ICML '08, pages 160–167. ACM, Acm, 2008.

[5] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*, volume 71 of *Language, Speech, and Communication*. MIT Press, 1998.

[6] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning Semantic Hierarchies via Word Embeddings. *ACL*, Proceeding:1199–1209, 2014.

[7] Yoav Goldberg and Omer Levy. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. *CoRR*, abs/1402.3, 2014.

[8] Zellig S. Harris. Distributional structure. *Word*, 10(23):146–162, 1954.

[9] James Hong and Michael Fang. Sentiment Analysis with Deeply Learned Distributed Representations of Variable Length Texts. *cs224d.stanford.edu*, 2015.

[10] Shoaib Jameel and Wai Lam. An unsupervised topic segmentation model incorporating word order. *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '13*, page 203, 2013.

[11] Stefan Kombrink, Tomas Mikolov, Martin Karafiát, and Lukas Burget. Recurrent Neural Network Based Language Modeling in Meeting Recognition. *INTERSPEECH*, 2011.

[12] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. *CoRR*, abs/1405.4, May 2014.

[13] Yann LeCun. Une procedure d'apprentissage pour reseau a seuil assymetrique (A learning procedure for assymetric threshold networks). In *Proceedings of Cognitiva 85*, pages 599–604 ST – Une procedure d'apprentissage pour r, 1985.

[14] Minh-thang Luong and Christopher D Manning. Better Word Representations with Recursive Neural Networks for Morphology. In *Conference on Computational Natural Language Learning (CoNNL 2013)*, 2013.

[15] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, 2011.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, pages 1–9, 2013.

[17] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pages 1–12, 2013.

[18] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. *CoRR*, abs/1309.4, 2013.

[19] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[20] Aandriy Mnih and Geoffrey E. Hinton. A Scalable Hierarchical Distributed Language Model. *Advances in Neural Information Processing Systems 21*, pages 1081–1088, 2009.

[21] Frederic Morin and Y Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252, 2005.

[22] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2014.

[23] Anand Rajaraman and Jeffrey D. Ullman. Mining of Massive Datasets. *Lecture Notes for Stanford CS345A Web Mining*, 67:328, 2011.

[24] Xin Rong. word2vec Parameter Learning Explained. *CoRR*, abs/1411.2, 2014.

[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors, 1986.

[26] Tianze Shi and Zhiyuan Liu. Linking GloVe with word2vec. *CoRR*, abs/1411.5:5, November 2014.

[27] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word Representations: A Simple and General Method for Semi-supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 384–394. Association for Computational Linguistics, 2010.

[28] Lior Wolf, Yair Hanani, Kfir Bar, and Nachum Dershowitz. Joint word2vec Networks for Bilingual Semantic Representations. *International Journal of Computational Linguistics and Applications*, 5.1:27–44, 2014.

[29] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting TF-IDF term weights as making relevance decisions. *ACM Transactions on Information Systems*, 26(3):1–37, June 2008.

[30] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Hierarchical Paragraph Vectors

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Elmer	Lukas

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, September 8, 2015	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.