



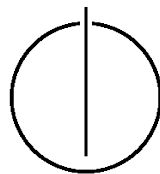
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master thesis in informatics

**Applications of Deep Learning in Natural  
Language Processing for Information  
Extraction on German Language Documents.**

Miguel Fernando Cabrera Granados







FAKULTÄT FÜR INFORMATIK

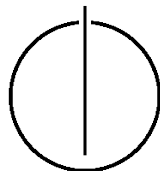
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master thesis in informatics

Applications of Deep Learning in Natural Language  
Processing for Information Extraction on German  
Language Documents.

Anwendungen von Deep Learning in der automatischen  
Sprachverarbeitung, der Dokumentenklassifikation auf  
deutschsprachigen Dokumenten

Author: Miguel Fernando Cabrera Granados  
Supervisor: Prof. Dr. Patrick van der Smagt  
Advisor: Dipl.Inf. Christian Osendorfer  
Date: June 25, 2014





Ich versichere, dass ich diese Master arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this master's thesis is my own work and I have documented all sources and material used.

München, den 25. Juni 2014

Miguel Fernando Cabrera G.



---

## Acknowledgments

There is not enough space on this page to name all the individuals that have positively influenced this work. First, I would like to thank all the support from my family in Colombia and Spain. Their support and enormous understanding have been extremely important. This culminating work is possible because of them.

I would like to thank the founders of Gini GmbH for giving me the opportunity of working on such interesting topic. I would like to thank Dr. Christoph Ringlstetter who served as my main advisor for this work, and with whom I had the chance to have interesting conversation full of relevant feedback and important criticism. Special gratitude to my colleagues Daniel and Agnes for investing a lot of their time and energy proofreading and offering insightful comments and valuable feedback.

In the local side, special thanks to Dipl.Inf. Christian Osendorfer for his great support and steering that helped me to set the right scope for this work.

Finally, I would like to thank all those that in one or another way supported me through the duration of the studies. Fellow students and friends I learned a lot from. All of them coming from different countries, but all sharing the same desire of becoming better persons everyday. Thank you guys, you know who you are.

Miguel Fernando Cabrera G.  
Munich, June 25, 2014





---

## Abstract

The success of machine learning algorithms depends on the representation of the data used. Specific domain knowledge can be used to design good representations. However, these representations are limited to a specific problem or task, and to the amount of available labeled data. Another approach is to automatically learn generic priors that can be used in different tasks and context. In the field of natural language processing, recent work has been done in obtaining such priors by learning useful vector representation of words from unlabeled data. The representations can then be used to improve existing natural language processing systems. These word vectors are obtained using special neural network architectures trained on billions of tokens. However, most of these models are learned and evaluated on English language corpora. In this work, *Word2vec*, a recent neural network based toolkit for learning word representations is used on German language data. The goal is to evaluate the learned representations of words in different language processing and information retrieval tasks. In particular, a semantic-syntactic evaluation set is constructed for the German language. In addition to that, the learned word vector representations are used as features for a classifier of German language business documents. The learned features outperformed existing handcrafted features and performed similar to other state-of-the-art approaches.



# Contents

<b>Acknowledgements</b>	<b>vii</b>
<b>Abstract</b>	<b>ix</b>
<b>I. Introduction and Theory</b>	<b>1</b>
<b>1. Introduction</b>	<b>3</b>
1.1. Motivation and Objectives . . . . .	3
1.2. Structure of the Thesis . . . . .	4
<b>2. Related Work</b>	<b>5</b>
2.1. Deep Learning and Representation Learning . . . . .	5
2.2. Natural Language Processing . . . . .	6
2.3. Representations of Text . . . . .	6
2.4. Local Representations . . . . .	6
2.4.1. $N$ -grams . . . . .	7
2.4.2. Bag-of-words (BOW) . . . . .	7
2.4.3. 1-of- $N$ coding . . . . .	7
2.5. Continuous Representations of Words . . . . .	7
2.5.1. Distributional Word Representations . . . . .	8
2.5.2. Distributed Representations . . . . .	8
2.6. Statistical Language Models . . . . .	9
2.7. $N$ -Grams based language models . . . . .	9
2.8. Maximum Entropy Language Models . . . . .	9
2.9. Neural Network Language Models . . . . .	10
2.9.1. Feedforward Neural Network Language Model . . . . .	11
2.9.2. Recurrent Neural Net Language Model . . . . .	13
2.9.3. Neural Network Language Model Proposed by Mikolov . . . . .	14
<b>3. Description of the Word2Vec Model</b>	<b>17</b>
3.1. Introduction . . . . .	17
3.2. Word2vec Generalities . . . . .	17
3.3. Word2vec Architectures . . . . .	18
3.3.1. Continuous Bag of Word (CBOW) Architecture . . . . .	18
3.3.2. Skip-Gram Architecture . . . . .	20
3.4. Word2vec Training Algorithms . . . . .	21
3.4.1. Hierarchical Softmax . . . . .	21
3.4.2. Negative Sampling . . . . .	22

3.5. Other Parameters . . . . .	23
3.5.1. Dimension . . . . .	23
3.5.2. Window . . . . .	23
3.5.3. Subsampling of Frequent Words . . . . .	23
3.6. Evaluation of the Word Representations . . . . .	24
3.7. Conclusion . . . . .	24
 <b>II. Applications</b>	 <b>27</b>
 <b>4. Empirical Study of Word Vectors for the German Language</b>	 <b>29</b>
4.1. Task Description . . . . .	29
4.2. Adapting the tasks to the German Language . . . . .	30
4.2.1. Common Capital and City and All Capital . . . . .	30
4.2.2. Currency . . . . .	30
4.2.3. City in State . . . . .	31
4.2.4. Man-Woman . . . . .	31
4.2.5. Adjective to Adverb . . . . .	31
4.2.6. Opposite . . . . .	31
4.2.7. Comparative and Superlative . . . . .	32
4.2.8. Present Participle . . . . .	32
4.2.9. Nationality Adjective and Plural nouns . . . . .	33
4.2.10. Past Tense and Plural Verbs . . . . .	33
4.3. Summary of Changes . . . . .	33
4.4. Description of Experiments . . . . .	33
4.4.1. Dataset . . . . .	34
4.4.2. Model Training . . . . .	36
4.4.3. Evaluation . . . . .	36
4.5. Empirical Results . . . . .	36
4.5.1. Comparing German and English Word Vectors . . . . .	39
4.5.2. Effect of Preprocessing on the learned word vectors . . . . .	40
4.6. Conclusion . . . . .	42
 <b>5. Word Vector Features for Document Classification of German language Business Documents</b>	 <b>45</b>
5.1. Introduction . . . . .	45
5.2. Text Categorization . . . . .	46
5.2.1. Performance Measures . . . . .	47
5.3. Gini document platform . . . . .	48
5.4. The Gini Document Classifier . . . . .	49
5.5. Experiments . . . . .	49
5.5.1. Gini Document Database and Dataset Description . . . . .	50
5.5.2. Handcrafted Features . . . . .	51
5.5.3. Bag of Words (BOW) Features . . . . .	51
5.5.4. Word Vector based Features . . . . .	51
5.6. Training and Evaluation . . . . .	52

5.6.1. Handcrafted Features . . . . .	52
5.6.2. BOW Features . . . . .	53
5.6.3. Word Vector Features . . . . .	53
5.7. Results . . . . .	55
5.7.1. Visualization of Features . . . . .	57
5.7.2. Generalization Power . . . . .	57
5.8. Conclusion & Future Work . . . . .	59
<b>6. Conclusion and Future Work</b>	<b>63</b>
 <b>Appendix</b>	 <b>67</b>
<b>A. Types and Features of the Gini Document Classifier</b>	<b>67</b>
A.1. List of Document Types Available in the Gini Database . . . . .	67
 <b>Acronyms</b>	 <b>70</b>
 <b>List of Figures and Tables</b>	 <b>73</b>
 <b>Bibliography</b>	 <b>75</b>



## **Part I.**

# **Introduction and Theory**





# 1. Introduction

From the early days of computers, people have dreamed about machines with intelligence. Computers able to exhibit complex behavior and able to solve tasks assigned by users. If we assume that this problem is too difficult to solve immediately, we can think of several steps that will lead towards the development of intelligent machines. One of the first tasks to solve is the one of language understanding, i.e. giving the machine to process and understand human language. This by definition is a hard task even for human beings, as natural language is inherently ambiguous.

Current computational language modeling techniques are based on statistical measurements and hardcoded rules. Although they perform fairly well in several tasks, they are far from achieving human like results in many natural language processing problems. One way to try to achieve human like behavior in a machine is to emulate the way of human brain works. This is the main idea behind the initial use of neural networks in the early days of artificial intelligence in the 60s and 70s. The last decade has seen a revival of this trend via *Deep Learning* approaches. *Deep Learning* tries to learn useful high-level representations of data using raw data via hierarchical architectures. This is similar in principle to the theories of brain development proposed by the cognitive neuroscientists in the early 1990s.

In the field of natural language processing, most of the research has focused on trying to obtain words representation that allow to extract meaning from unlabeled text, mostly in English language. However, English lack of the morphological complexity exhibited by other Germanic and non-Germanic languages such as German and Spanish. Therefore, it is of interest to evaluate what information can extract these models from morphologically richer languages. Besides that, one the performance of these representation of words is assessed, it is important to design or adapt machine learning system that can take advantage of these representations.

## 1.1. Motivation and Objectives

The goal of this thesis is to describe deep learning approaches to learn word features, performing an empirical study of the performance of such representation in the German language and evaluating the applicability of such methods in existing problems, in particular the field of automated text categorization.

There are two main motivations behind this objective. First, it is interesting to evaluate word representation of languages other than English. This allows measuring to some degree the generalization power and the ability to extract useful features of these models. Second, most of the existing data existing is unlabeled. Thus, developing pipelines that allow to take advantage of large amount of available data is necessary to improve existing systems that process natural language.

For that purpose an empirical evaluation of the *Word2vec* model on German language corpora has been chosen. *Word2vec* is a tool that by means of a neural network learns vector representation of words. *Word2vec* is used because it has shown to produce representations that contain rich relationships among words. In the application area, the field of automatic document classification of German language document has been selected. The reason for this is two-fold. On one hand, the amount work has been done in applying such representation in the general area of automated document classification is limited. On the other hand, the nature of the data set chosen, from which limited amount of document is available; offer the perfect chance to evaluate the performance obtained by using features learned from unlabeled data.

### 1.2. Structure of the Thesis

The theoretical foundation required for the understanding of this thesis is described in chapter 2. This chapter also gives an overview about related work in the of language modeling and text representations. In particular, it describes the model that led to the development of *Word2vec*. Chapter 3 describes in detail the architecture *Word2vec*. Afterwards in chapter 4 an empirical evaluation of the *Word2vec* model on a German language corpus is performed. Chapter 5 explores the application of these representations in the field automated text categorization. Finally, in chapter 6 the thesis is concluded and possible future work is outlined.

## 2. Related Work

This chapter introduces the basic concepts necessary to understand the models and tools used in this work. Section 2.1 introduces basic concepts of deep learning and representation learning. After that, in sections 2.2 to 2.5 the basics of natural language processing and text representation are discussed. Sections 2.6 and 2.7 introduce statistical language modeling and the classical  $N$ -gram based approach. Finally, Section 2.9 explores in detail some of the existing language models based on neural networks. The reason for this, is that continuous word vector representation model used in this work, namely *Word2vec*, relies heavily on concept developed by these architectures. Thus, understanding previous models will help to understand the rationale and motivation behind *Word2vec*.

### 2.1. Deep Learning and Representation Learning

Performance of machine learning methods relies heavily on the choices of the data representation or features used to train the algorithm. Therefore, choosing the right feature set is crucial when developing machine learning systems. This step, generally part of a more complex processing pipeline, is called feature engineering. Although an important step, feature engineering is expensive in term of labor needed to perform it successfully. Feature engineering generally requires expert knowledge of the application domain. In order to expand the applicability and speeding up the development of machine learning based systems, it would be desirable to depend less on feature engineering by employing algorithms that could extract meaningful information from raw input, turning this information into features to be used in different machine learning settings.

Feature learning or representation learning is a set of techniques in machine learning that learn a transformation from these raw inputs to a representation that can be effectively exploited in a supervised learning task such as classification [2]. These representations are learned by using several layers of non-linearities, where each layer learns increasingly complex level of representation. However, high depth is not a requirement to learn useful representation. Depending on the tasks relatively shallow architectures produce useful features.

One important concept in representation learning is the concept of *distributed representations*. A distributed representation is a representation where  $k$  out of  $N$  representation elements or feature values cannot be independently varied, e.g., they are not mutually exclusive.  $K$  features being turned on or active represent each concept while each feature is involved in representing many concepts [2]. There are many reasons for which having automatically learning distributed representations are desirable to have. First, distributed representations enable to fight the *curse of dimensionality* by capturing a huge number of possible input configurations in reasonably sized learned representations. These representations often offer similarity models that can be useful to improve existing algorithms.

Second, handcrafting features is time consuming, and it often generates over-specified and incomplete representation, requiring repeating work when switching tasks or domains. Finally, many supervised machine learning techniques require labeled data while in domains the majority of data is unlabeled. By learning representations based on unlabeled data, this information can be harnessed to learn powerful classifiers.

## 2.2. Natural Language Processing

Natural language processing Natural Language Processing (NLP) is a field of computer science, artificial intelligence, and linguistics that studies efficient mechanisms to provide natural language understanding and generation machines. Statistical NLP comprises all quantitative approaches to automated language processing, including probabilistic modeling, information theory, information theory and linear algebra [19]. Modern NLP techniques make use of machine learning, data mining and others data driven techniques.

Major tasks of NLP include Part-of-Speech tagging (POS) (detecting part of speech in a sentence), Named Entity Recognition (NER) (Determining which items of a sentence map to proper names, e.g. person, location, etc.), Sentiment Analysis (Extracting subjective information usually from a set of documents), among others. In some cases, sets of related tasks are grouped into subfields that are often considered separately from NLP. Information Retrieval (IR) is concerned with storing, searching, and retrieving information, in particular information stored as text. Although considered a separate field, many tasks of IR such a Text Categorization (TC) rely on NLP methods. Thus, there is an overlap between the two fields in several applications.

## 2.3. Representations of Text

Many of the tasks of NLP and IR deal with textual data. Therefore the way of representing this information is of key importance and affects directly the performance of the methods. For Statistical and machine learning techniques, these representations become the input features of such algorithms. Thus, using appropriate representation becomes really important as the suitability of the representation depends on the specific task.

Text can be represented at several abstraction level. The most of the common work at the document level, representing the complete document as a mathematical object (e.g. a vector). Other representation work based on sequence of words extracted from a text, i.e.  $n$ -grams or at word level (i.e. word vector embedding). Based on the characteristics of the representations they can be classified as local or continuous.

## 2.4. Local Representations

A local representation is a vector of features that characterize the meaning of the symbol. These features are mutually exclusive. That is, each feature represents a unique characteristic.

### 2.4.1. *N*-grams

An *n*-gram is a sequence of *n* items from given text corpus. The items can be phonemes, syllables, letters or words. In this work, the term *n*-gram refers only to word *n*-grams. An *n*-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram"; size 3 is a "trigram". Larger sizes are sometimes referred to by the value of *n*. As the frequencies of the *n*-grams are of interest for statistical modeling, the number of occurrences usually accompanies these representations. This representation have been used successful used in language modeling (see section 2.6). *N*-gram based language models have been the dominant approach for language modeling since the 1980s [1].

### 2.4.2. Bag-of-words (BOW)

Bag of Word is a document level representation in which each document of a corpus is assigned a vector of dimensionality *V*, where *V* is the vocabulary of that corpus. Each component of this vector has an associated word of the vocabulary. The values of each of the component can be binary, i.e. representing the word occurrence or absence in the document, or a more sophisticated metric such as term frequency-inverse document frequency (tf-idf) [33].

Term frequency  $tf_j$  is the number of times that a word *j* appears in a document *d*. The document frequency  $df_j$  is the number of documents the term *j* appears in the entire corpus. With these two values, tf-idf for a word *j* is calculated as follows:

$$\text{tf-idf}_j = tf_j(d) \cdot \log\left(\frac{N}{df_j}\right), \quad \forall j \in V$$

With *N* the total number of documents in the document set. This representation of text does not take into account the actual order of the words inside the document. Despite of that, it has performed well in many IR tasks (e.g. search, TC, etc.) [34]. Variations and extension of this technique are still used in modern IR systems.

### 2.4.3. 1-of-*N* coding

This is a word level representation. It is also called *one hot representation*. The feature vector has the same length as the size of the vocabulary of the corpus. All the features or dimensions are zero but one. Representing a unique word of the vocabulary. This representation is commonly used as an input for language models based on neural networks (see section 2.9.)

## 2.5. Continuous Representations of Words

Continuous representation of words can be divided between distributional and distributed representations.

### 2.5.1. Distributional Word Representations

Distributional representations are based on the idea that a lot of information can be obtained by representing a word by means of its neighbors. Distributional representations are generally used to obtain document level representations. Distributional word representations are based upon a cooccurrence matrix  $F$  of size  $V \times C$ . Each row of  $F$  is the initial representation of a word  $w$ , and each column is some context [37]. The context is generally windows of words surrounding  $w$ . Each value of the context is represented using a type of frequency count such as tf-idf. Commonly used techniques for obtaining distributional representations are Latent Semantic Analysis (LSA) [11] and Latent Dirichlet Allocation (LDA) [6].

#### Latent Semantic Analysis

LSA is the application of Singular Value Decomposition (SVD) to a matrix where each row is a document represented using the BOW model. In this way LSA induces distributional representations over  $F$  in which each column is a document context [37]. One drawback of this technique is that, for large databases the SVD calculation becomes intractable, since the computation is extremely expensive.

#### Latent Dirichlet Allocation

LDA is a generative probabilistic model of a corpus and the idea behind it is that the documents are represented as weighted relevancy vectors over latent topics, where a topic is characterized by a distribution over words [6]. Learning the various distributions (the set of topics, their associated word probabilities, the topic of each word, and the particular topic mixture of each document) is the objective of applying this topic modeling.

Similar to LSA the result of LDA is a lower dimensional representation of the document. In particular, LDA reduces the document to a fixed set of values (i.e. a vector), based on the probability distribution of the words in the documents.

### 2.5.2. Distributed Representations

The advantages of learning and using distributed representations were already discussed in section 2.1. At the word level, distributed word representations assign a real-valued vector to each word. As with other distributed representations, the difference with local representations lies in the representation mechanism, i.e. a single feature does not represent a single concept or relationship. In the context of NLP, Hinton [14] introduced distributed representations for textual data. Bengio et al. [1] combined distributed representations with neural network probability prediction to apply them to statistical language models by using his neural network based language model. Collobert et al. [8], Mnih et al. [28] and Mikolov et al. [23, 24] have further developed similar architectures focusing on improving the performance and estimation accuracy. Section 2.9 discusses some of these models with more detail.

## 2.6. Statistical Language Models

A statistical language model assigns a probability to a sequence of  $m$  words  $P(w_1, \dots, w_m)$ . In most statistical language model it is assumed that the probability of  $i$ -th word  $w_i$  only depends on the sequence of the  $n-1$  previous word. Using this assumption, the probability  $P(w_1, \dots, w_m)$  can be calculated by [1]:

$$P(w_1, w_2, \dots, w_{t-1}, w_t) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_t|w_1, w_2, \dots, w_{t-1}). \quad (2.1)$$

Most probabilistic language models approximate  $P(w_t|w_1, w_2, \dots, w_{t-1})$  using a fixed context of size  $n-1$ , i.e. using  $P(w_t|w_{t-n+1}, \dots, w_{t-1})$ .

Language model has applications in fields such speech recognition and data compression, where the model tries to predict the next word in a speech sequence. In the field of IR language model are used generally to rank documents based on the probability that a language model would generate terms from a specific query [19].

Although not in an extensively, this section covers existing statistical language models. In particular, the dominant  $n$ -gram based model and other models that serve as basis for the architecture of *Word2vec* are discussed.

## 2.7. $N$ -Grams based language models

The most frequently used language models are based on the  $n$ -gram statistics, which are basically word co-occurrence frequencies [19]. The maximum likelihood estimate of probability of word  $A$  in context  $H$  is then estimated by:

$$P(A|H) = \frac{C(HA)}{C(H)} \quad (2.2)$$

where  $C(HA)$  is the number of times that the  $HA$  sequence of words has occurred in the training data. The context  $H$  can consist of several words, for the usual trigram models  $|H| = 2$ . For  $H = 0$ , the model is called unigram. The unigram model only models the word probability on the corpus, and does not take into account historical information [19, 21].

As many of these probability estimates are going to be zero (for all words that were not seen in the training data in a particular context  $H$ ), smoothing needs to be applied. This works by redistributing probabilities between seen and unseen (zero-frequency) events, by exploiting the fact that some estimates, mostly those based on single observations, are greatly over-estimated.

## 2.8. Maximum Entropy Language Models

Maximum Entropy (ME) model is an exponential model with the following form :

$$P(w|h) = \frac{e^{a_w}}{Z(h)} \quad (2.3)$$

with  $a_w$  defined as:

$$a_w = \sum_i \lambda_i f_i(w, h) \quad (2.4)$$

where  $w$  is a word in a context  $h$  and  $Z(h)$  is the normalization factor:

$$Z(h) = \sum_{w_i \in V} e^{\sum_j \lambda_j f_j(w, h)} \quad (2.5)$$

Where  $f_i(w, h)$  are the feature functions based on the word and its context. Therefore, the problem of training a ME consists of finding the weights  $\lambda_i$  and selecting the features  $f_i$ , which are not learned from the data. Usually, the feature functions  $f(w, h)$  for these models are  $n$ -grams probabilities. ME models can be seen as simple neural networks without a hidden layer and can be trained using stochastic gradient descent [5, 21].

## 2.9. Neural Network Language Models

As its name describes, Neural Network Language Models (NNLM) are language models based on Neural Networks, exploiting their ability to learn distributed representations to reduce the impact of the *curse of dimensionality* [1]. In other words, a Neural Network (NN) is used to estimate the probability described by equation (2.1) based on a sequence of previous words.

The advantage of using distributed representations is that they allow the model to generalize well to sequences that are not in the set of training word sequences, but that are similar in terms of their features. Because neural networks tend to map nearby inputs to nearby outputs, the predictions corresponding to word sequences with similar features are mapped to similar predictions [1, 3].

However, due to the nature of the multilayer architecture common to neural networks, NNLMs suffer from performance issues, where the hidden layers and the output layer are generally the bottleneck. In order to be able to compare the different models in terms of their computational complexity, it is necessary then to define it in terms of their architecture. For all the models discussed here, the complexity is proportional to [22]:

$$O = E \times T \times Q, \quad (2.6)$$

$E$  is the number of training epochs (i.e. number of passes through the entire training set),  $T$  is the number of words in the training set and  $Q$  depends on each architecture. Common choices are  $E = 3-50$  and  $T$  up to one billion. All models are trained using stochastic gradient descent and backpropagation [3, 22].

The following section describes each of the most representative models and their associated complexity  $Q$ .



### 2.9.1. Feedforward Neural Network Language Model

In the Feedforward Neural Network Language Model (FNNLM) introduced by Bengio [3], the probabilistic prediction  $P(w_t|w_{t-n+1}, \dots, w_{t-1})$  in equation (2.1) is obtained as follows. First, each word  $w_t$  is associated to a one-hot vector of dimension  $V$  with an entry at position  $t$ , and zeros otherwise. Then each vector is mapped to a  $d$ -dim feature vector  $C_{w_t}$ . The feature vectors  $C_w$  are interpreted as the columns of a parameter matrix  $C$ , which represents the linear transformation between the  $V$ -dimensional one-hot vectors and the  $d$ -dim feature vectors.

Vector  $C_{w_k}$  contains the learned features for word  $k$ . Let the vector  $x$  denote the concatenation of these  $n - 1$  feature vectors:

$$x = (C_{w_{t-n+1},1}, \dots, C_{w_{t-n+1},d}, C_{w_{t-n+2},1}, \dots, C_{w_{t-2},d}, C_{w_{t-1},1}, \dots, C_{w_{t-1},d}). \quad (2.7)$$

The probabilistic prediction of the next word, starting from  $x$  is then obtained using a standard artificial neural network architecture for probabilistic classification, using the softmax activation function at the output units [5]:

$$P(w_t = k|w_{t-n+1}, \dots, w_{t-1}) = \frac{e^{a_k}}{\sum_{l=1}^N e^{a_l}} \quad (2.8)$$

where

$$a_k = b_k + \sum_{i=1}^h W_{ki} \tanh(c_i + \sum_{j=1}^{(n-1)d} V_{ij} x_j) \quad (2.9)$$

where the vectors  $b, c$  and matrices  $W, V$  are also parameters of the NN (in addition to matrix  $C$ ). Let us denote  $\theta$  for the concatenation of all the parameters. The number of hidden units  $h$  and the number of learned word features  $d$  control the complexity of the model.

The neural network is trained using a gradient-based optimization algorithm to maximize the training set *log-likelihood*

$$L(\theta) = \sum_t \log P(w_t|w_{t-n+1}, \dots, w_{t-1}). \quad (2.10)$$

The gradient  $\frac{\partial L(\theta)}{\partial \theta}$  can be computed using the backpropagation algorithm [5], extended to provide the gradient with respect to  $C$  as well as with respect to the other parameters.

The gradient  $\frac{\partial L(\theta)}{\partial \theta}$  is given by :

$$\begin{aligned} \frac{\partial}{\partial \theta} \log P_{\theta}^n(w_t = k) &= \frac{\partial}{\partial \theta} e^{a_k} - \frac{\partial}{\partial \theta} \log \left( \sum_{l=1}^N e^{a_l} \right) \\ \frac{\partial}{\partial \theta} \log P_{\theta}^n(w_t = k) &= \frac{\partial}{\partial \theta} e^{a_k} - \frac{\partial}{\partial \theta} \sum_{l=1}^N P(w_t = l) \frac{\partial}{\partial \theta} e^{a_l} \end{aligned} \quad (2.11)$$

The calculation  $\frac{\partial L(\theta)}{\partial \theta}$  is expensive to calculate because it implies a sum over all the scores  $e^{a_k}$  over all possible words on the vocabulary.

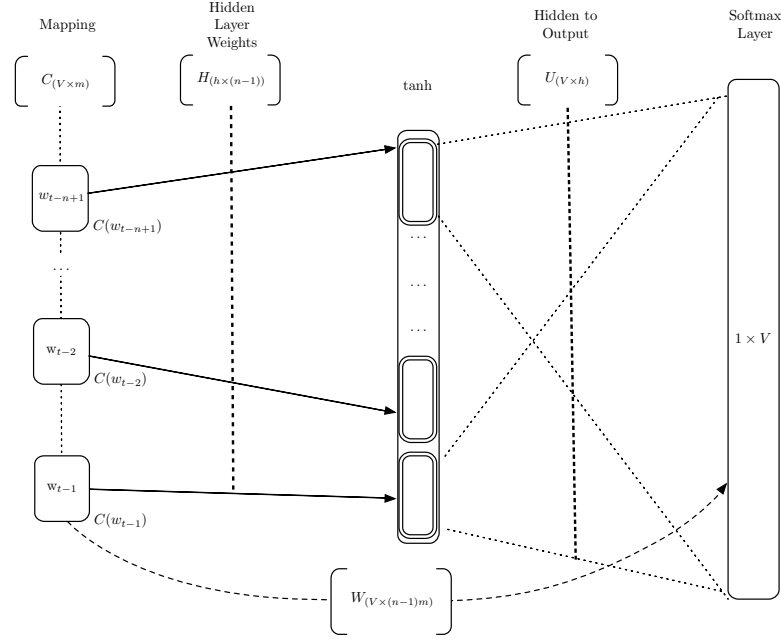


Figure 2.1.: Feed Forward Neural Network Model Language Model Architecture.  $C(i)$  is the  $i$ -th word feature vector. [3].

### Model Complexity

Figure 2.1 is a graphical depiction of the model. It consists of three layers, input, projection and output layers. The  $N$  previous words are encoded using 1-of- $V$  encoding, where  $V$  is the size of the vocabulary.  $N$  previous word representations to  $w_t$  are concatenated using a shared projection matrix to the projection layer  $P$  (dimensionality  $N \times D$ ). This layer is then connected to a hidden layer which is in turn used to compute the probability distribution over all the words in the vocabulary resulting in an output layer with dimensionality  $V$ .

### Model Complexity

Based on the description above, the computational complexity of each training example is:

$$Q = N \times D + N \times D \times H + H \times V, \quad (2.12)$$

The dominant term is  $H \times V$ , because of the gradient calculation described in the previous section [22]. That is, the bottleneck of this architecture reside on the last layer because in order to obtain normalized probability via the *softmax* layer is necessary to evaluate model for each word in the vocabulary. Many solutions are proposed to avoid the complexity in the last layer. One possibility is to use hierarchical versions of the softmax [30, 20]. The other alternative is to avoid normalized models by finding a approximation to find the likelihood gradient [29]. If binary tree representations are used, the number of

output units can decrease down to  $\log_2(V)$ . Most of the complexity will be then caused by the term  $N \times D \times H$ .

### 2.9.2. Recurrent Neural Net Language Model

One of the major issues of the FNNLM approach is that it uses a fixed context that needs to be specified at training time. A language model based on recurrent neural networks does not use a context with a size limit. This is achieved by using recurrent connections inside the network for arbitrarily long time. The Recurrent Neural Network Language Model (RNNLM) proposed by Mikolov uses the so-called simple recurrent neural network model. The network has an input layer  $x$ , hidden layer  $s$  (also called context layer or state) and output layer  $y$ . Input to the network in time  $t$  is  $x(t)$ , output is denoted as  $y(t)$ , and  $s(t)$  is state of the network (hidden layer). Input vector  $x(t)$  is formed by concatenating vector  $w$  representing current word, and output from neurons in context layer  $s$  at time  $t - 1$ . Input, hidden and output layers are then computed as follows [23]:

$$x(t) = (w(t), s(t - 1)) \quad (2.13)$$

$$s_j(t) = f \left( \sum_i x_i(t) u_{ij} \right) \quad (2.14)$$

$$y_v(t) = g \left( \sum_j s_j(t) h_{kj} \right) \quad (2.15)$$

Where  $f(z)$  is *sigmoid* activation function and  $g(z)$  is the traditional *softmax*.  $y_v(t)$  represents the probability of that the next word is  $W_v$ .

Figure 2.2 depicts the architecture of the simple recurrent neural network. Similar to the FNNLM,  $w(t)$  represents the word in time  $t$  encoded using 1-of- $V$  coding.  $s(t - 1)$  represents the output values in the hidden layer from the previous time step. Therefore  $x(t)$ , the input, has the size of the vocabulary  $V$  and of vector  $s(t - 1)$ . After the network is trained, the output represents the probability in equation (2.1) [21].

### Model Complexity

Assuming that the hidden layer and the word representation share the same dimensionality with the hidden layer, the complexity per training example of the RNNLM model is given by:

$$Q = H \times H + H \times V, \quad (2.16)$$

In a similar fashion to the FNNLM, the term  $H \times V$  can be efficiently reduced to  $H \times \log_2(V)$  by using the hierarchical softmax formulation. Thus, most of the complexity comes  $H \times H$ .

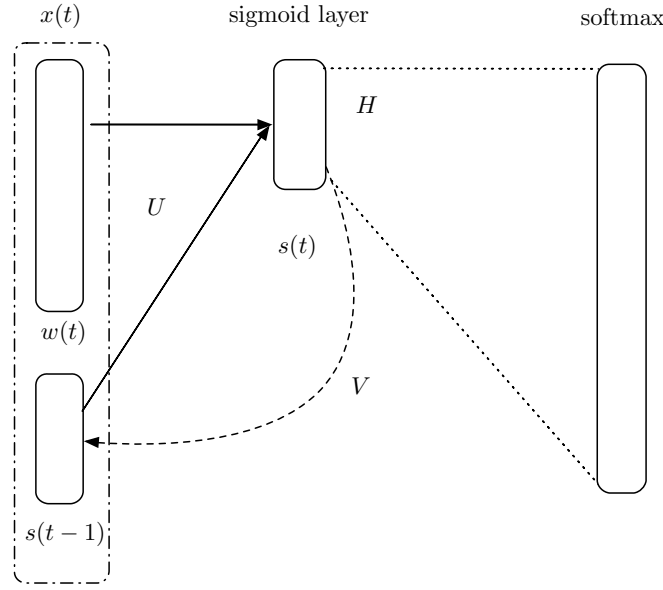


Figure 2.2.: Recurrent Neural Network Language Model Architecture

### 2.9.3. Neural Network Language Model Proposed by Mikolov

The NNLM proposed by Mikolov use a similar approach to the NNLM described in section 2.9.1. However, in Mikolov's model the neural network architecture is split in two steps. In the first section of the architecture a bigram neural network is trained. Given a word  $w$  from a vocabulary  $V$ , the neural network tries to estimate the the probability distribution of the next word in the text. Both the input and output are of size  $|V|$ , were the words a usually is encoded with 1-of- $K$  encoding, with  $K = |V|$ . There is is one hidden layer of size  $|D|$ . This layer is of arbitrary size (reported by the authors to range from 20-60). As with other models, the output layer is a softmax layer. The network is trained using the standard *backpropagation* algorithm [24].

The second step consists of a  $N$ -gram network. That is a network that tries to predict  $w_t$  based on previous  $N - 1$  words. The network is trained in a similar way, however the input vector does not encode previous words using 1-of- $K$  encoding, but is formed using  $N - 1$  projections from the bigram network. These projections are the learned weights of dimension  $|D|$  learned by the big-gram network. Figure 2.3 shows a schematic of the two-network architecture.

#### Model Complexity

As this model is composed of two parts, we then split the complexity definition in two . One for the calculation of the bigram representation and one for the final probability estimation:

$$Q_1 = V \times D + D \times V \quad (2.17)$$

$$Q_2 = N \times D \times H + H \times V \quad (2.18)$$

$$Q = Q_1 + Q_2 \quad (2.19)$$

With  $Q_1$  and  $Q_2$  the complexity of the bigram network and the  $N$ -gram network respectively. Although not mentioned by the authors, efficiency improvement can be achieved by using hierarchical formulations of the softmax. If used, most of the complexity lies in the  $N \times D \times H$  term of  $Q_2$ .

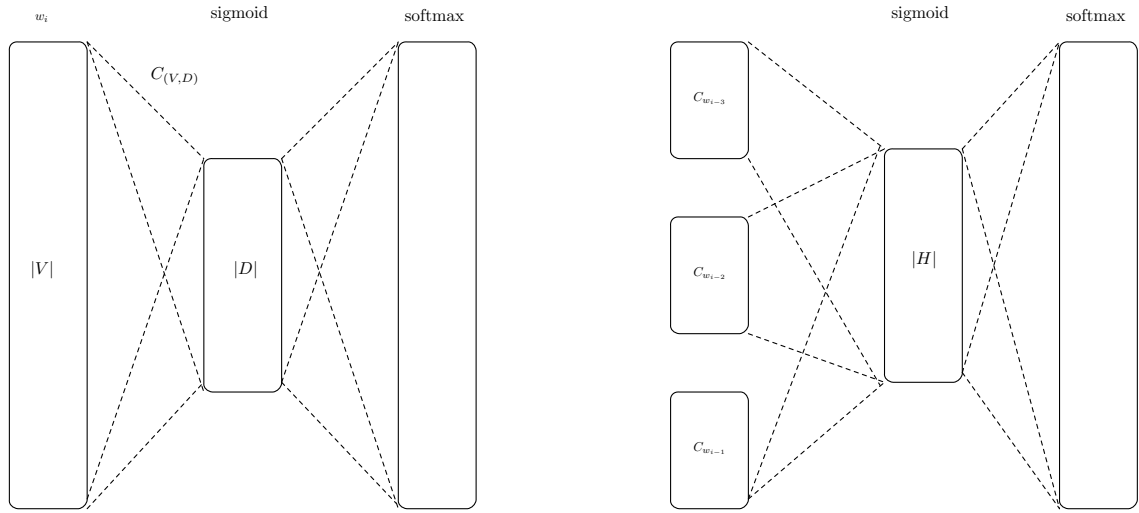


Figure 2.3.: Architecture of the NNLM proposed by Mikolov.



## 3. Description of the Word2Vec Model

### 3.1. Introduction

Continuous space language models learned by a neural network have demonstrated good results across a variety of tasks in the field of NLP. As pointed out by the authors, these continuous word vector representations achieve a level of generalization not possible with traditional  $N$ -gram based models [24]. In addition to the language model itself, the word vector representations are learned in the initial layers of the NN. These induced representations have been used successfully to improve the performance of existing NLP tasks [8] [37].

As discussed in section 2.9.3, one way to learn a NNLM is to learn the word vectors using a NN with a single hidden layer. Then, these word vectors are used to train another NN, in this way the final NNLM is learned [24]. What is particular about this architecture is that the word vectors are learned without even constructing the complete language model. This means that in practice, the two steps can be performed separately [24].

This chapter describes a recent approach to efficiently learn high-dimensional word representation [22]. It has been found that the word vectors learned in this manner, not only capture linguistic regularities of words but also semantic relationships. These regularities can be represented as linear translations. For example, the result of a vector calculation  $vector("Madrid") - vector("Spain") + vector("France")$  is closer to  $vector("Paris")$  than to any other word vector. [22, 26, 27]

As the word vector generated from this architecture is used through this work, it is necessary to describe this particular model with more detail.

The rest of the chapter is structured as follows: section 3.2 gives a general description of *Word2vec*. Section 3.3 and 3.4 describe in more detail the available architecture and training algorithms. Finally, section 3.5 describes additional parameters that directly affect the generated learned representations.

### 3.2. Word2vec Generalities

*Word2vec* is an open source project released in 2013 by the Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. [22] Its overall architecture has been described in several paper published at the time of its release [22, 26, 27] and some previous work [21]. The source code, sample corpus and the dataset used to evaluate the quality of the word vectors can be downloaded freely from the project's web page<sup>1</sup>. *Word2vec*'s architecture has its origin in the architecture described back in section 2.9.3 where a NNLM is learned separately. First learning the word representation using a NN and then learning to estimate the probability described in equation (2.1) using a  $N$ -gram neural network. *Word2vec* is a simpler model,

---

<sup>1</sup><https://code.google.com/p/word2vec/>

as it does not try to learn the full probability distribution as a full NNLM does. Instead, it focuses on learning good representations of words by only performing the first step of the aforementioned architecture.

As described in section 2.9, the complexity of all these models arises from the non-linear layers that allow the NN to obtain representations. *Word2vec* removes the additional hidden layer allowing it to be trained efficiently on much more data [22].

*Word2vec* is not a single architecture or model, it is formed by a group of related architectures, learning algorithms, optimization approaches and some tricks. The combination of all of these characteristics allows the model to learn word representation in an efficient way. It has been shown that these representations capture semantic and syntactic similarities as mean of vector offsets, allowing the model to answer complex semantic and syntactic questions by means of simple vector operations [26]. This property has also been used to perform word vector based translation [25].

In the *Word2vec* project the following parameters can be chosen:

- **Architecture:** Skip-gram or Continuous Bag of Words (CBOW).
- **Training Algorithm:** Hierarchical Softmax (HS) or Negative Sampling (NEG).
- **Additional parameters:** Discussed in section 3.5

Table 3.1 summarizes the effect of the parameter choices on the resulting word vector representations. Then comment column is taken from the description on the project web page.

Table 3.1.: Summary model training parameters

Parameter	Possible Values	Comment
Architecture	skip-gram	Slower, better for frequent words.
	CBOW	Faster than skip-gram.
Training Algorithm	HS	Better for infrequent words.
	NEG	Better for frequent words
Sub-sampling	1e-3 to 1e-10	It can improve both accuracy and speed for large data sets Useful values are in range
Dimensionality	10-640	Higher dimensions generally provide better results. it depends on data availability.
Context (Window size)	5 - 15	The number of word used to predict. Larger windows generate more topical relationships.

## 3.3. Word2vec Architectures

### 3.3.1. Continuous Bag of Word (CBOW) Architecture

This architecture is similar in spirit to the FNNLM described in section 2.9.1 where a context of  $n$  words is used, with two main differences. First the non-linear hidden layer is removed



and second the projection layer is shared for all words (not just the projection matrix of word vectors). This means that all the words are projected onto the same position on the hidden layer, and their vectors averaged. The training criterion is to correctly classify the middle word of a window of  $N$  that is used per train cycle. That is, words from the past and the future are used to predict the word in the center of a sentence.

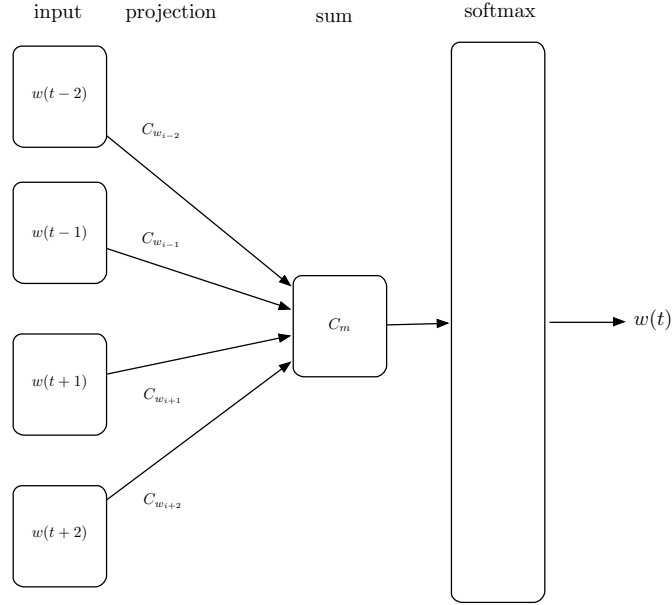


Figure 3.1.: CBOW Architecture

More formally, given a sequence of words with 1-of- $K$  representation  $w_1, w_2, w_3, \dots, w_T$  and a window of size  $n$ . Defining  $C_m(t)$  as:

$$C_m(t) = \frac{\sum_{-n \leq j \leq n, j \neq 0} C_{w_{t+j}}}{2n} \quad (3.1)$$

The CBOW architecture tries to maximize the average log probability:

$$\frac{1}{T} \sum_{t=1}^T \log p(w_t | C_m(t)), \quad (3.2)$$

With the probability  $p(w_O | v)$  defined as:

$$p(w_O | v) = \frac{e^{(C'_{w_O} \top v)}}{\sum_{w=1}^V e^{(C'_w \top v)}} \quad (3.3)$$

Where  $C_w$  and  $C'_w$  are the input representation (the weights from the input layer to the projection layer) and output representation (the weights from projection layer to the softmax layer) respectively. Figure 3.1 shows a graphical representation of CBOW architecture.

Similarly as other NNLMs, the complexity of this architecture is defined by its structure:

$$Q = N \times D + D \times \log_2(V) \quad (3.4)$$

Assuming a for of hierarchical softmax is used to efficiently calculate the probability distribution, as described for all the NNLMs in section 2.9.

### 3.3.2. Skip-Gram Architecture

The Skip-gram architecture is similar to CBOW, but it instead of predicting the current word based on the context. Each word is used as input to the NN that predicts words within certain range before and after it (i.e. it tries to predict the surrounding words or context). [22]. Figure 3.2 show depicts this approach graphically.

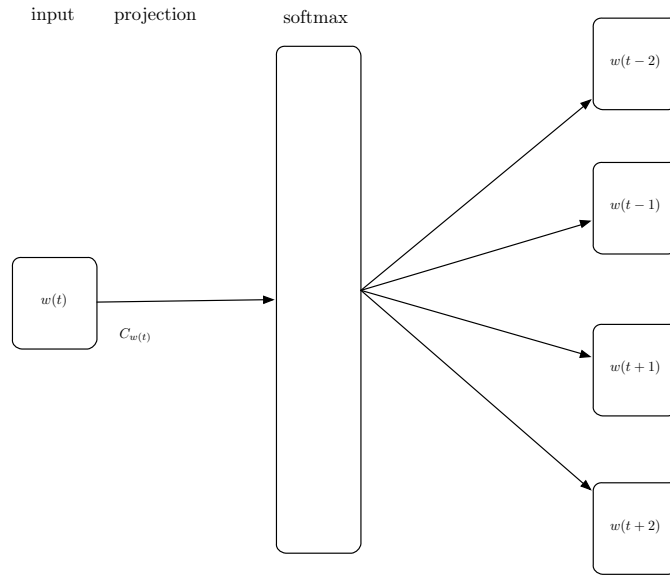


Figure 3.2.: The Skip-gram architecture. The current word  $w(t)$  is used to predict the surrounding words.

Similarly to CBOW, if a sequence of training words  $w_1, w_2, w_3, \dots, w_T$  is given, the probability to maximize is give by

$$\frac{1}{T} \sum_{t=1}^T \sum_{-r \leq j \leq r, j \neq 0} \log p(w_{t+j} | w_t) \quad (3.5)$$

with  $r$  is the size of the window, i.e. the training context is  $2r$ . Increasing this range improves quality of the resulting word vectors but it also increases the computational complexity.

Using the softmax function the probability  $p(w_O | w_I)$  is given by:

$$p(w_O | w_I) = \frac{e^{(C'_{w_O} \top C_{w_I})}}{\sum_{w=1}^V e^{(C'_w \top C_{w_I})}} \quad (3.6)$$

Where  $C_w$  and  $C'_w$  has the same meaning as in the CBOW formulation, and  $V$  is the number of words in the vocabulary.

The training complexity of this architecture is proportional to [22]:

$$Q = R \times (D + D \times \log_2(V)), \quad (3.7)$$

Where  $R$  is the maximum distance of words. A random number  $r$  in the range  $< 1$ ;  $R >$  is selected. Then,  $r$  words from the history and  $r$  words from future are used as the correct labels for the current word.

### 3.4. Word2vec Training Algorithms

The availability of different training algorithms is other the principal characteristics of *Word2vec*. In fact, the selection of the training algorithm has a large impact on the quality of the learned vector representation, and on the computational performance.

The formulations for calculating  $p(w_i|v)$  and  $p(w_O|w_I)$  ( equations (3.3) and (3.6) ) are impractical because the cost of computing them is proportional to  $V$ , which is often large. Therefore, computational efficient alternatives need to be devised in order to train this model in real world data sets. Here when the training algorithms come into play, as they are alternative to this computationally expensive calculation.

#### 3.4.1. Hierarchical Softmax

An efficient alternative to calculating the full softmax is the Hierarchical Softmax. Morin and Bengio [30] were the first to introduce this technique in the context of NNLMs. Hierarchical Softmax (HS) is more computationally efficient than a normal softmax layer because instead of evaluating output nodes  $V$  (i.e. each of the words on the output layer) to obtain the probability distribution, it only needs to evaluate about  $\log_2(V)$  nodes [26].

This is achieved by using a binary tree to represent the output layer with the  $V$  of the vocabulary as its leaves, for each node, explicitly represent the relative probabilities of its child nodes. Previous work reports several strategies to create the binary tree [28], all of them with different effects on the performance. The authors of *Word2vec* use a Huffman Tree to represent the hierarchical softmax.

To build the tree, the probabilities are calculated beforehand just counting each occurrence of the word in the text and dividing it by the total number of words (the unigram probability). With these probabilities, the Huffman codes are calculated [15], and thus the tree is built. The words with high probabilities get shorter codes ending up on top tree. As the probabilities of the child nodes can be obtained from its parent, it is not necessary to visit all then children but just take it from that node. Figure 3.3 shows an example of how this Huffman tree would look if created from an imaginary and highly unlikely corpus.

More precisely, the tree is used as follows: each word  $w$  is reached by an appropriate path from the root of the tree (i.e. the Huffman code). Let  $n(w, j)$  be the  $j$ -th node on the path from the root to  $w$ , and let  $L(w)$  be the length of this path. So  $n(w, 1) = \text{root}$  and  $n(w, L(w)) = w$ . In addition, for any inner node  $n$ , let  $ch(n)$  be an arbitrary fixed child of  $n$  and let  $\llbracket x \rrbracket$  be 1 if  $x$  is true and -1 otherwise. Then the hierarchical softmax defines the  $p(w_O|w_I)$  by [26] :

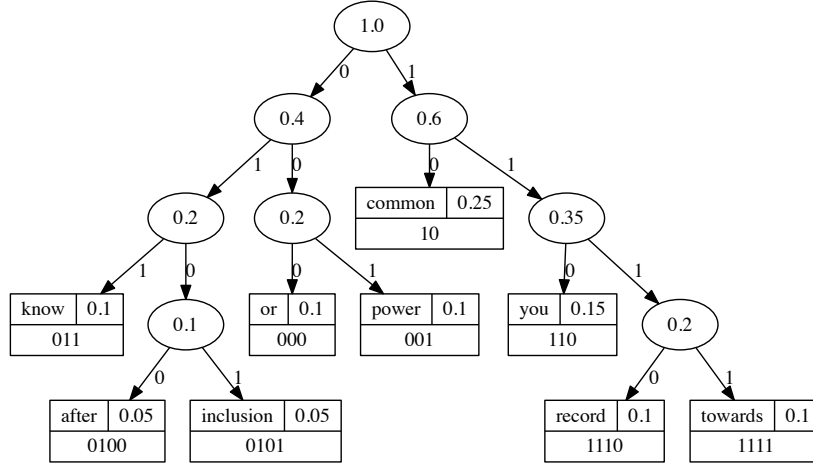


Figure 3.3.: An example of a Huffman binary tree used to calculate the hierarchical softmax. This is a small imaginary corpus of 8 words. The leaf nodes contain the words, the probability and the Huffman codes. Inner nodes contain just the probabilities of its children.

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma \left( \mathbb{I}[n(w, j+1) = ch(n(w, j))] \cdot C'_{n(w, j)}{}^\top C_{w_I} \right) \quad (3.8)$$

Where  $\sigma(x) = 1/(1 + \exp(-x))$ . In other words, when calculating the softmax only the nodes on the way to the leaf node representing the  $w$  are used. And the softmax objective is to predict the Huffman code of it (i.e. the size of the Huffman code is the length of the path  $L(w)$ ). Thus, the cost of computing the log-probability and the gradient is proportional to  $L(w)$  which for a binary tree is not greater than  $\log_2(V)$ .  $C_{w_I}$  is again the representation of word  $w_I$  but  $C'_{n(w, j)}$  is the projection of the node  $n$  on the tree. So the the number of output nodes is also approximately  $\log_2(V)$ . A equivalent model replacing  $C_{w_I}$  by  $v$  is easily constructed for the CBOW architecture. .

### 3.4.2. Negative Sampling

An alternative to the hierarchical softmax is Noise Contrastive Estimation (NCE), which was introduced in the context of language model by Mnih and Teh [35]. NCE is in turn based in the concept of importance sampling, which use a known distribution to try to approximate the gradient based on samples not from the original distribution  $p(w_t)$  but from a known distribution  $q(x)$  easy to sample from. In other words, instead of fitting the density model, it is learned to classify from samples of the data distribution and samples from a noise distribution  $q(x)$  that is already know. As the objective of *Word2vec* is not to learn a language model but useful representation, so authors simplified to the original NCE and defined an alternative formulation named NEG:

$$p(w_O|w_I) = \log \sigma \left( C'_{n(w,j)}{}^\top C_{w_I} \right) + \sum_k \mathbb{E}_{w_i \sim P_n(w)} \left[ \log \sigma \left( -C'_{w_i}{}^\top C_{w_I} \right) \right] \quad (3.9)$$

The main difference between this approach and NCE needs both samples and the numerical probabilities of the noise distribution, while NEG uses only samples [26]. Equivalently to hierarchical softmax, a formulation from the CBOW architecture can be constructed by replacing the appropriate variables. In the *Word2vec* implementation, the noise distribution  $P_n(w)$  is the unigram distribution  $U(w)$  raised to the  $3/4$ rd. The authors selected this empirically given that outperformed other alternative distributions such as the standard unigram and the uniform distribution.

### 3.5. Other Parameters

This section describes the additional parameters needed by the model and that affect the quality of the word vectors.

#### 3.5.1. Dimension

Perhaps the most important parameter after the architecture and the training algorithm is the dimension of the learned word vector. This parameter defines the projection layer size. Generally speaking the larger the better. However, in order to benefit from a larger dimension size, more data need to be used, as it reaches a upper bound based on the availability of the data. The author of *Word2vec* report dimension from 100 to 640 on the largest dataset used. This, as it was just mention is corpus dependent.

#### 3.5.2. Window

An important parameter that needs to be defined at training size is the *window* or context size. Generally speaking the context size is two times the window. More formally, the context  $R(w_i)$  comes from a window of size  $r$  around the word:

$$R(w_i) = w_{i-k}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+k},$$

In the CBOW architecture, the window defines the number of words used to calculate the continuous bag of word ( equation (3.1)). For the Skip-gram architecture, the window defines the number of words that the  $w_i$  needs to predict from a sentence. Generally speaking, larger context generally provide better word representation but at expense of computation time. The window is generally set empirically its will depend on the grammatical structure of the source language of the corpus used and the task at hand.

#### 3.5.3. Subsampling of Frequent Words

In large corpora the most frequent words occur many times (e.g. “in”, “at”, “the”, etc.). These words provide less information value than other rare words [26]. *Word2vec* discards a word  $w_i$  in the training set with a probability computed by the formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \quad (3.10)$$

where  $f(w_i)$  is the frequency of word  $w_i$  and  $t$  is a threshold that can be specified by the user at training time. The subsampling of words also affect indirectly the size of the window. In addition to the subsampling paramter, *Word2vec* offer a parameter called *min-count* that discards all the words that appear less than a number of times in the entire corpus. As the subsampling and the removal of *min-count* words occur *before* the actual context creation, the actual effective window size increase. These include words that are content-full and linearly far from the word in focus, therefore making the similarities more topical.

## 3.6. Evaluation of the Word Representations

In their original paper [22], the authors introduced a comprehensive test that contained five types of semantic questions. In the original dataset there were 8869 semantic and 10675 syntactic questions. To create this set the authors compiled a list of similar word pairs manually and then they generated the list by connecting the word pairs. Table 3.2 shows examples from each category of the original tasks.

To evaluate the quality of the word vectors, questions are answered in the following manner: given the word pairs  $(a, b)$  and  $(c, d)$  a question is considered to be answered correctly if for the resulting vector  $X$  the closest vector measured in cosine similarity is  $vector(d)$ .  $X$  is calculated by applying the following operation  $X = vector(b) - vector(a) + vector(c)$ . Given two vectors  $A$  and  $B$ , the cosine similarity is calculated in the following way:

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

Word representations capture different types of similarities among words. These similarities are expressed in linear relationship among the vectors. Therefore, with the operation described above, what is being attempted is to answer the question “*What is the word that is similar to  $c$  in the same sense as  $b$  is similar to  $a$ ?*”.

For example for the words (“*small*”, “*smaller*” ) and (“*big*” , “*bigger*”), the  $vector(“bigger”)$  should be the closest to the resulting vector  $X$ . If the representation of the word in vector is good, then it is possible to find the answer to this question using operation [22].

## 3.7. Conclusion

*Word2vec* allow training word vector representations from large corpus of text in a efficient way. The vector representations capture semantic and syntactic relationship, relationships that can be explored via simple vector operations. In this chapter we described the architecture, algorithm and techniques behind it. There are two main differences between *Word2vec* and a traditional NNLM, first it does not care to estimate the actual probability of a next word given a previous context, but only of obtaining good representation. In that

Table 3.2.: Examples of the original types of syntactic questions in the Semantic-Syntactic Word Relationship test as defined by [22].

Type of Relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Past verbs	work	works	speak	speaks

sense *Word2vec* is not a language model, but a feature learning algorithm. Second, it uses words not only from the past, as more traditional language models do, but also from the future, allowing the model learn complex relationships among words. Finally, it is not a deep architecture. In fact, the architecture only contains a layer. This, among other things allows training efficiently representations that are useful to improve NLP tasks.





**Part II.**

**Applications**



## 4. Empirical Study of Word Vectors for the German Language

The previous chapters introduced the concepts and related work in the fields of NLP, NNLM and vector representation of words. This chapter will focus on the application of a particular model, namely *Word2vec*, to the German language.

Although evidence that word vector representations might work well in other languages other than English has been presented [25], at the time of writing, there are not in depth studies on this subject. Also, there are not related works presenting syntactic or semantic relationship evaluation similar to the ones performed by [22] but using a corpus in a language other than English.

Previous work has shown that n-gram back-off language models, the state of the art technique, does not perform as well for inflectional languages as they work for English [24]. Inflectional languages tend to overlay many morphemes to denote grammatical, syntactic or semantic changes to a word. For example, in German, *gelb* means *yellow* and *(der) Baum* translates to *tree*. However, in order to say “*yellow tree*” in German, it is necessary to inflect the adjective: *gelber Baum*. This generates problems for statistical based methods, as many tokens represent the same concept.

It has been show that NNLM captures in a better way the complexity of such languages [24, 25]. Therefore, it is possible that word vectors generated by such model will improve and simplify many NLP applications in these languages as they have been shown to do in English [8, 37]. Consequently, making the evaluation of the quality of the word vectors in inflective languages very attractive.

In this chapter the applicability of of word vector representation to German language is studied. The main idea is to evaluate how the word vectors learned by *Word2vec* perform in German. The rest of the chapter is divided as follows: The first part describes the selected tasks, the rationale behind the adaption made to fit the German language and the dataset. The second part describes the training approach as well as the most important results. It includes a performance comparison of the word vectors in German with their counterpart in English as well as the evaluation of some corpus preprocessing approaches that might affect the performance of the word representation for German.

### 4.1. Task Description

Section 3.6 described that approach of the authors of *Word2vec* to evaluate the quality of the word vectors learned. In order to evaluate the performance of the word vectors in the same way, but using another language, a similar set of questions needs to be created. Two approaches might be used to accomplish it. The words can be translated, either automatically or by hand, or the dataset can be generated from scratch in a similar manner as the

original was. In this work the translation is performed by hand using a German - English online dictionary <sup>1</sup> as guide and verified by two German native speakers.

Regardless of the method chosen, some tasks of this set are biased towards the morphology of English and could not be easily mapped to another language, and in particular to German. Therefore, a simple translation would not be enough to obtain an equivalent task set for another language. For that reason each of the original 14 tasks was analyzed to either adapt it to German or discard it completely.

## 4.2. Adapting the tasks to the German Language

Many of word pairs from the original semantic-syntactic tasks are language dependent. In order to adapt them to German, each of type of question was evaluated. Some of them could be easily adapted to German just by translating the respective word. However, some of them either do not make sense from the morphology point of view, or they simply do not apply for German. From the original 14 types of questions, the German version contains created in this work only contains 10. An equivalent set was also constructed in parallel for English by removing the part that could not be adapted. This was done with the purpose of performing a comparison. Below each of the tasks are described in detail along with steps performed to adapt it to German when it was possible.

### 4.2.1. Common Capital and City and All Capital

This group of questions tries to solve analogies between country and capitals. The vector model is asked the semantic relationship *Capital Country*. For example: “*What is the word that is similar to Germany in the same sense as Paris is similar to France?*”. The model should produce a vector close to “*Berlin*”. The difference between the two subsets is that common capital contains the 22 most “*common*” countries - capital word pairs. However, the author did not describe the criteria used to select countries of the set. The other subset of word pairs contains country - capital pairs of less known countries such as *Ghana - Accra* and *Harare - Zimbabwe*.

This task was adapted by simply translating the names from the English ones to their German counterparts.

### 4.2.2. Currency

These questions model the relationship between currencies and countries. So the question might be phrased as el is asked the semantic relationship *Capital Country*. For example: “*What is the word that is similar to Argentina in the same sense as Peso is similar to Colombia?*” For the currency section all of the pairs were translated using the corresponding articles from the German Wikipedia<sup>2</sup> as source for the information.

---

<sup>1</sup><http://dict.cc>

<sup>2</sup><http://de.wikipedia.org>

### 4.2.3. City in State

This group of questions tries to evaluate the semantic relationship between a city and a state for cities in the United States of America. However, the objective of the experiment is to have a standard, almost language independent comparable test set for the word vectors generated by *Word2vec*. Tasks that are centered in information mostly found in a specific language corpus are not relevant and will not provide good information about the quality of the vectors. For this reason, the complete task was removed from the dataset. An alternative could have been to replace it with a similar task (e.g. using cities in Germany / Austria). In this case however, a comparison with another language would have been impossible as the information sources might not contain the information required or even the words themselves.

### 4.2.4. Man-Woman

These tasks try to evaluate the semantic relationship between words associated with men and women. i.e. *“What is the word that is similar to Man in the same sense as Sister is similar to Woman?”*. For this tasks the equivalent German version of the word were used. For validation of the translation the online German - English dictionary was used.

### 4.2.5. Adjective to Adverb

This task models the relationship between an adjective and its correspondent adverb. However in German this task cannot be applied, because unlike English, German does not make a distinction in form between a predicate adjective and an adverb [12]. For example in English:

- The man is quiet. (quiet = adjective)
- The man sings quietly. (quietly = adverb)

Whereas a German speaker would say:

- Der Mann ist leise. (leise = adjective)
- Der Mann singt leise. (leise = adverb)

For this reason the trained word vector will not be able to differentiate the roles. Thus, this tasks is removed from the set.

### 4.2.6. Opposite

This task tries to model the *opposite* semantic relationship between words. Although it may seem relatively easy to adapt this task to German there things that need to be taken into account. In German, the most common “antonym” is not necessarily used nor they have a regular morphological structure as in English. For example in English is usual to form the opposite by adding the prefix *in* , *un* , *im* or *dis*. In fact, all of the word pairs that appear in the English version of this set are formed in such way. This is not always

true in German. For example, the opposite for the word *überzeugend* (roughly translating to *convincing* in English) cannot be formed by adding a suffix. The only way to form the opposite is by adding the negation, e.g. *nicht überzeugend*. Another example would be the pair *geschmackvoll* - *geschmacklos*, which are as well opposites, but where the suffix is the difference. Yet, most of the word in these tasks could be also build in a similar fashion as in the English set. Therefore, this task is adapted by looking for the best translations that can be formed with maintaining the common structure, even if they are not the more close translation.

#### 4.2.7. Comparative and Superlative

The comparative task tries to evaluate the syntactic relationship between a word and its comparative form. For example, the word pairs *young* and *younger*. Respectively, the superlative task aims to evaluate the superlative relationship (e.g. *safe* and *safest*).

Many of the words belonging to this dataset can be easily translated to German directly. However, there are some caveats. First, English has many more words for representing the same semantic concept than German. English belongs to the family of Germanic languages, therefore shares the common structure and grammar. However, many words have been borrowed from Latin, which expands the vocabulary, in many cases replacing the words of Germanic origin. For example, the word *tall* comes from Old Germanic and the word *large* comes from Latin. Both words could be translated to German as *hoch*. However, the English word *high* of German origin can also be translated as *hoch*. That is, a unique concept in German can be represented by many words in English. Given this, in order to translate properly the tasks, only one word representing the same concept is maintained in both datasets, making the number of word pairs equivalent.

#### 4.2.8. Present Participle

These tasks try to model the relationship between verbs and its participle (e.g. *come* - *coming*). In English is used the participle in much more ways than German [10]:

- To form the progressive (continuous) aspect: "*Jim was sleeping*".
- As an adjective phrase modifying a noun phrase: "*The man sitting over there is my uncle*."
- adverbially, the subject being understood to be the same as that of the main clause: "*Looking at the plans, I gradually came to see where the problem lay*".
- similarly, but with a different subject, placed before the participle (the nominative absolute construction): "*He and I having reconciled our differences, the project then proceeded smoothly*".
- more generally as a clause or sentence modifier: Broadly speaking, the project was successful.

In German, the present participle is used almost exclusively as an adjective or adverb and as such it is inflected (e.g. *laufende*, *laufender*, *laufended*, all might be translated as

'running'). This implies that in a model such as *Word2vec* the same word after the declension will be represented with different words, possibly resulting in a low performance on the syntactic task.

#### 4.2.9. Nationality Adjective and Plural nouns

The nationality adjective tasks model the relationship between a country and the adjective representing the nationality. For example, for the word *Colombia*, the respective adjective in English is *Colombian*. In German, the nationality is also an adjective, and as such, it is inflected based on the case and genus of the substantive. For example *Kolumbianisch* may appear inflected as *Kolumbianischer*, *Kolumbianischem*, *Kolumbianische*, etc. Similar to the participle, many words will represent the same concept, thus low correct answer rate are expected in this task. The plural nouns task models the relationship of a word and its plural form (e.g. *tree* - *trees*). German has many more words that their plural form is the same as the singular. This is a factor that will affect the results of the evaluation, as the same word is never taken as a valid answer in this task.

#### 4.2.10. Past Tense and Plural Verbs

The tasks find syntactic relationship between a verb and its past tense and the conjugation respectively. In English the verbs are conjugated in the past the same way for all the person. This does not hold for German, as the conjugation of the simple past varies based on the person. Therefore, these questions are removed from the test.

### 4.3. Summary of Changes

Table 4.1 shows a summary of the changes performed in order to generate a set for the German language.

In parallel to these changes, the English version was also modified to match this new set of tasks so a comparison could be performed in a reasonable equivalent manner. Table 4.2 shows an example of the tasks in German. Only the tasks that were kept are shown.

### 4.4. Description of Experiments

This section describes the experimental set-up used to evaluate the German *Word2vec* model. The objective of these experiments is to answer several questions regarding the performance of *Word2vec* the applicability of *Word2vec* on languages different than English, in particular for German. In particular:

- What is the performance of the German language word vector model in comparison to that of the English language?
- Given the morphological differences and complexity of languages like a German, is *Word2vec* able to capture the information properly?

Table 4.1.: Summary of changes to the original task

Task	Action
Common capital city	Translated
All capital cities	Translated
Currency	Translated
City-in-state	Removed - Language dependent
Currency	Translated
Man-Woman	Translated
Adjective to adverb	Removed - does not apply in German
Comparative	Translated - Some words removed due to polisemy
Superlative	Translated - particle <i>am</i> not used
Opposite	Translated
Present participle	Translated
Plural Nouns	Translated
Past verbs	Removed
Nationality adjective	Translated

- Is there any preprocessing of the training data that might help to capture word vector (e.g. Stemming)?

In order to answer these questions the following tasks are performed:

- The training of several Word2vec models using the Wikipedia German database.
- Training of several several Word2vec models using Wikipedia English as training database.
- Comparison of the performance of the models in the modified task described in section 4.2.

##### 4.4.1. Dataset

In the original paper [22] the author trained the word vectors using Google News internal dataset containing about 6 billions of tokens. This dataset is not available publicly. The authors have made available the pre-trained 300-dimensional word vector from Google's internal dataset containing 3 million words and phrases.

The closest open dataset available with such variety in German is the Wikipedia. The English as well as the German Wikipedia are used for the experiments. It is important to notice, that as it happens with most of the information available in World Wide Web, The majority of is in English, and as discussed before, the amount of data available affects the quality of the trained word embeddings.

For the English and German word vector model the Wikipedia is used. For German the complete dump of the 24<sup>th</sup> of October of 2013 of the German Wikipedia is used. For



Table 4.2.: Examples of word pair of task used to evaluate the German word vector model. The umlaut marking diaeresis has been replaced by the equivalent diphthongs ae,oe and ue.

Type of Relationship	Word Pair 1		Word Pair 2	
Common capital city	Bern	Schweiz	Kairo	Aegypten
All capital cities	Zagreb	Kroatien	Aschgabat	Turkmenistan
Currency	Argentinien	peso	Lettland	lats
Family	Bruder	Schwester	Grossvater	Grossmutter
Man-Woman	Koenig	Koenigin	Prinz	Prinzessin
Opposite	zulaessig	unzulaessig	moeglich	unmoeglich
Comparative	billig	billiger	warm	waermer
Superlative	schlecht	schlechtesten	gut	besten
Present participle	fliegen	fliegend	laufen	laufend
Nationality adjective	Weissrussland	weissrussisch	Frankreich	franzoesisch
Plural nouns	Banane	Bananen	Loewe	Loewen

English, the dump of all the articles from the 4<sup>th</sup> of November of 2013 is used. The dates are reported for informative purpose. The dataset can be obtained from the Wikipedia servers.<sup>3</sup>

Since these dumps are in XML and contains characters and words that are not of interest, before building the word vector representation the following clean up procedure is applied:

- Remove all XML/HTML tags from the file preserving text and caption for URLs and images.
- Remove special characters such as &, [, }, etc.
- make all characters lowercase.
- spell out numbers.
- remove any other character that is not alphanumeric.

In addition to this, for the German language the additional modification were performed

- Transform *umlauts* to the ASCII spelling. e.g. ö is transformed to oe, ü is transformed to ue and so on.
- The *Eszett* or *scharfes S* (Symbol 'ß') is transformed to ss.

<sup>3</sup><http://dumps.wikimedia.org/>

To perform these changes, a Perl script originally designed to perform these tasks (for the English Wikipedia) was adapted.<sup>4</sup> This script is linked on Word2vec's Google code project site<sup>5</sup>.

Table shows 4.3 the datasets used to train the file the word vectors with their size and number of tokens after the preprocessing.

Table 4.3.: Summary of datasets

File Name	Size (GB)	Number of tokens
dewiki-20131024-pages-articles.txt	6.5	1051584822
enwiki-20131104-pages-articles.txt	18	3185913717

We can see that the English dataset contains almost 3 times as many tokens as the German one. This means that for the English word vector model there much more sentences to train form, and therefore can be expected a better performance from this model in the tasks described before.

#### 4.4.2. Model Training

There are many parameters that can be modified in order to adjust the model to perform better on both the semantic and syntactic tasks. The selection of these parameters affects not only the accuracy of the model but also the training speed. Table 3.1 shows an overview of the most important parameters with short description. For a complete description of each of them please refer to section 3.

For training, the public C implementation of *Word2vec* was used. This is a highly optimized C implementation. It was slightly modified to account for the different number to task categories and to provide more debugging information useful for the model training.

In a similar fashion as in the original paper [22], in order to quickly train and evaluate the models only the 30.000 most common words of the dataset are used to perform a first evaluation of the tasks. That is, from the vocabulary only these words are used, and questions are evaluated if all the words on the question are inn this subset of the vocabulary.

#### 4.4.3. Evaluation

As the original article describes, the overall accuracy for all question types is calculated as well as for each question type separately (semantic, syntactic). A question is then assumed to be answered right only if the closest word to the vector computed using is exactly the same as the correct word in the question.

### 4.5. Empirical Results

As mentioned before, with the purpose of obtaining a quick evaluation of the model, only a subset of the dataset was evaluated it with the 30.000 most common words on it. The

---

<sup>4</sup><http://mattmahoney.net/dc/textdata.html>

<sup>5</sup><https://code.google.com/p/word2vec/>

German syntactic and semantic task described in section 4.1 is used to assess the quality of the model.

The four possible combinations of architecture and models were evaluated quickly using this approach. For that, a dimensionality of 300 was chosen with a fixed window of 10. Table 4.4 shows the result of these tests. The best architecture and algorithm combination is Skip-gram and NEG for this training setup.

Table 4.4.: Comparison of the combination of the available architectures and algorithms. Only questions containing words form the most frequent 30k are used. Numbers are in percents.

Model Architecture	Algorithm	Semantic Accuracy	Syntactic Accuracy	Total Accuracy
CBOW	HS	76.65	31.73	56.59
CBOW	NEG	81.17	50.26	65.73
Skip-gram	HS	77.59	34.02	55.82
Skip-gram	NEG	<b>84.40</b>	48.58	66.54

After evaluating different architectures, the model was evaluated with different training data sizes and dimensionality. Table 4.5 shows the results of this training. The main difference in comparison with the original work is that we use directly NEG instead of HS for the training algorithm.

From table 4.5 can be also noticed that if neither the dimensionality of the word vectors or the number of training words alone can improve the performance of the word vectors. Only the combination of an increasing amount of both will improve the accuracy. In fact, as can be seen using high dimension with a small amount of training words might even hurt performance.

Table 4.5.: Accuracy on subset of the semantic task Word relationships test set using vector Skip-gram architecture with NEG. Only questions containing words form the most frequent 30K are used.

Dimensionality / Training Words	24M	49M	98M	196M	392M	780M
50	6.84	17.51	33.95	42.61	43.49	49.02
100	6.84	18.61	38.64	50.30	56.97	57.90
300	5.57	18.22	37.05	52.71	63.77	<b>77.33</b>
600	4.85	17.40	34.02	49.01	61.80	66.92

After performing the previous experiments, it was visible that the best architecture and training algorithm were Skip-gram and NEG respectively. With this two parameters we evaluated various combinations of other model with the full data, evaluated initially again on the 30.000 most common words and then in the full dataset of questions. The subsampling parameter was obtained empirically from trying distinct values of it. As it happens, it also matches the one obtained for the English language in the original work [22].

Table 4.6 shows overview of the best models obtained from the complete German Wikipedia dataset and their performance on the full set of questions. In contrast with the original work [26, 22], the best model has only 300 dimensions. This is caused by the fact the models were trained with considerable less training data than Google News dataset. Nonetheless, the dimension is higher than the usual ones found in other word vector embedding models trained using similar techniques and similar datasets.[37, 9].

Table 4.6.: Accuracy of the best models obtained using the Skip-gram architecture with Negative Sample on the German Wikipedia dataset. A subsampling of  $10^{-5}$  is used

Method	Win.	Dim	Syntactic [%]	Semantic [%]	Total accuracy [%]
NEG-10	5	300	59.4	25.3	42.3
NEG-10	5	400	57.5	25.6	41.5
NEG-10	5	500	54.3	24.3	39.2
NEG-15	10	300	63.0	24.9	<b>44.0</b>
NEG-15	10	400	58.9	26.2	42.6
NEG-15	10	500	60.1	24.3	42.3

The second column of table 4.7 shows the result of the best German model separated by category. The fourth column shows the performance of pre-trained word vectors on the Google News data set. Next section will discuss the relationship between these columns in more detail.

We can note that section *currency* from the semantic dataset; the *present participle* and *superlative* sections are the one with the worst performance. This result can be explained by the dataset. German Wikipedia although one of the largest German language corpus, it lacks variety on its contents, having only some text for unpopular topics, as in this case the currencies or small countries. However, by exploring the model can be seen that the concept of currency is maintained, although the exact semantic relationship is not built. For example, taking one of the tasks: ('Algerien', 'dinar', 'Angola', 'kwanza'). If we define  $x$  as the result of applying:  $x = \text{vector}(\text{dinar}) - \text{vector}(\text{Algerien}) + \text{vector}(\text{Angola})$ , then in order for this task to be answered correctly, the closest vector to should be  $\text{vector}(\text{kwanza})$ . However the top-5 closest vectors are:

Word	Cosine similarity to $x$
<i>peso</i>	0.6193921566009521
<i>centavos</i>	0.6159522533416748
<b>kwanza</b>	0.6066386699676514
<i>kwacha</i>	0.6008095741271973
<i>waehrung</i>	0.6000255942344666

Although the closest vector is not the expected one, all the words are related somehow with the concept of *money* or *currency*. For example the word *peso* is the currency of many Latin American countries, and centavo means *cents* in English. In fact, with some additional modification to the operations it is possible to obtain the desired answer.

For example if we define  $x$  as  $x = \text{vector}(\text{dinar}) - \text{vector}(\text{Algerien}) + \text{vector}(\text{Angola}) + \text{vector}(\text{Afrika})$ , that is, the same operation as before but adding the vector representation of the word *Afrika*, it produces a vector with the following top-5 closest vectors:

Word	Cosine similarity to $x$
<b>kwanza</b>	0.7224324941635132
<i>angolas</i>	0.6794506311416626
<i>angolanische</i>	0.6792401075363159
<i>kwacha</i>	0.6728493571281433
<i>metical</i>	0.6719954609870911

Which would return the correct answer for the task. This does not work all the cases, but it shows that the semantic relationship is built albeit not maintaining the exact relation between words as the tasks expect. In the other cases a similar phenomena occurs: the right answer is in the top-5 or top-10 of closest word representation of the resulting vector. This suggests that with more training data, it is possible to build better word representations that fulfill the relationships defined by the analogical reasoning task.

Table 4.7.: Accuracy of the best models separated by category using the German Wikipedia, the English Wikipedia and the pre-trained Google News dataset word vectors.

Category	Accuracy DE [%]	Accuracy EN [%]	Accuracy EN Pre [%]
capital common countries	89.33	96.05	83.20
capital world	72.33	88.68	79.13
currency	7.97	13.63	27.37
family	47.62	79.64	84.58
Total Semantic	63.12	78.40	72.88
opposite	14.67	33.62	42.73
comparative	33.06	75.83	90.84
superlative	3.69	28.25	87.34
present-participle	3.17	60.61	78.22
plural	36.79	70.20	86.04
nationality adjective	33.33	89.31	89.93
Total Syntactic	24.83	63.46	81.99
Total	44.02	70.47	77.73

#### 4.5.1. Comparing German and English Word Vectors

In this section models trained using the English language Wikipedia and the German Wikipedia are compared. It is clear that from the linguistic point of view it does not make sense to compare two language models for different languages. That is also truth for word vector representations. However, the purpose of this section is to assess to the possible

extend the behavior the word vector model on languages with different degrees of syntactic complexity. For that, the quality of the word vector obtained from a German language dataset is compared to ones trained on an English corpus, namely the Wikipedia.

Although in section 4.2 we described the creation of a equivalent analogical reasoning task for the German language, obtaining a unbiased comparison between vector models from different languages is a hard task. First there is not an equivalent dataset for German language. Although for both models the Wikipedia raw text is used to train the word vectors, the English Wikipedia contains much more text than the German Wikipedia, triplicating it both in size and number of words.

In addition to this, the original data, namely the Google News dataset, on which the original word vectors were trained, is not available [22]. However, some pre-trained word vectors were made available by the authors <sup>6</sup>, and are used reference to compare the two vector models trained from Wikipedia. These word vectors were trained with about billion of words and phrases.

Table 4.7 shows the best model for each language in each category. Both German and English behave fairly similar per category, giving the impression that the model works in a similar manner in German language. However for the syntactic task the performance is different when compared with other languages. In particular for the superlative and the present-participle the performance is low compared to other categories in German and with the equivalent categories for English. This may be caused by the complex and different German syntax and usage. The data size is also an important factor that affects the overall performance. We can see that when we compare the dataset of English trained on the Wikipedia with pre-trained vectors.

Although the pre-trained vectors are trained with much more data than the ones from Wikipedia, for some tasks the performance is better for the later. That suggest that is not only the size but the *content* of the text might have important effect on the performance. Content in these context means if the text contains the semantic or syntactic relationships that the test set expects. In other words, less data containing appropriate semantic and/or syntactic relationships might be better for training than a big corpus with no context. Another example of this will be shown in chapter 5, where word vectors generated from Wikipedia German corpus are used to perform document classification of unrelated documents.

##### 4.5.2. Effect of Preprocessing on the learned word vectors

One of the driving ideas behind of feature learning is to learn useful representations with no prior knowledge and with minimum human involvement. However, as the same authors of *Word2vec* state, it is likely that adding information to the model about morphological relationship of words, and in general any additional relevant prior knowledge, will produce better representations. One way of indirectly adding such information to the model without modifying its formulation is to perform preprocessing on the training dataset. This section explores how preprocessing, in particular stemming, affects the quality of the word vectors.

---

<sup>6</sup><https://code.google.com/p/word2vec/>

In many NLP applications, in particular, IR related tasks; text preprocessing is a common, if not necessary, step to obtain good results. Given highly inflective nature of German is natural to expect improvement in certain analogical reasoning tasks. One of the most common preprocessing steps in NLP is stemming (i.e. grouping words that share the morphological root). Stemming has proven successful in the field of TC [34]. For languages such as German where the nouns and adjectives are also inflected it is helpful to reduce the vocabulary size when performing similar tasks.

To evaluate the effect the stemming on word vector quality, different stemming approaches based on the Porter stemmer [32] were applied to the original German Wikipedia text corpus described back in section 4.4.1:

- Partial stemming algorithm using only the step\_1 suffixes of the stemmer algorithm.
- Application of the complete Porter stemmer algorithm.
- Remove the declensions of words ending in *sch* (i.e. *sche*, *scher*, *schem*, etc.) and replace it simply by *sch*.

Table 4.8.: Comparison of different stemming schemes. The model were trained with the best parameter after stemming the original file using the strategies described in section 4.5.2.

Category	No Stemm.[%]	Custom Stemm.[%]	Full Stemm.[%]	Replaced <i>sch</i> [%]
capital comm. count.	89.33	56.92	38.53	91.11
capital world	72.33	39.00	10.19	73.21
currency	7.97	5.77	0.11	7.62
family	47.62	6.06	5.19	48.92
Total Semantic	63.12	33.94	10.71	63.94
opposite	14.67	13.33	2.66	15.33
comparative	33.06	25.22	0.00	34.69
superlative	3.69	0.00	0.00	4.06
present-participle	3.17	1.48	0.00	2.56
plural	36.79	1.81	0.45	34.37
nationality adjective	33.33	77.19	1.51	<b>90.06</b>
Total Syntactic	24.83	25.66	0.71	39.38
Total	44.02	29.82	5.74	51.82
Questions seen	95.57	67.73	7.71	94.22

All the stemming process was performed using the popular Python library for NLP Natural Language Toolkit (NLTK) [4]. The description of the stemmer algorithm for the German language can be found online<sup>7</sup>.

The first modification aims to remove the plural (already affecting the outcome of the syntactic task) and the declension of adjectives and nouns. The second modification runs a

<sup>7</sup><http://snowball.tartarus.org/algorithms/german/stemmer.html>

full Potter stemmer algorithm. The last was an additional experiment aimed to verify that performing some modification to the training dataset would allow the model to perform better in certain tasks.

Note that all modifications, although probably helping in some of the tasks would undoubtedly incur in loss of performance in another. For example, if some adjectives are stemmed, the ability of recognizing comparatives or superlative are lost.

Table 4.8 shows the results of training the word vectors using the best parameters on this modified dataset. The rows show the performance per section of the set of tasks. The *Questions Seen* row shows the amount of question that after the pre-processing can be answered because the word still exists on the vocabulary. Both of the stemming based on the Porter stemmer hurt the performance of the classifier. In particular the full stemming causes the model to only achieve as little as 5.74% accuracy and only seeing 7.71% of the total questions. Surprisingly enough, the preprocessing that consisted of removing the declension of adjectives ending in *sch* not only improved the target category, *nationality adjective* that went from 33.33% to 90.06% accuracy but also slightly improved the accuracy of other categories by losing less of 2% of question seen. In other words, some well-chosen preprocessing step helped to improve the performance not only for specific tasks but in general over the whole task set.

To conclude, the results shown that although the traditional approach to text preprocessing for IR will not work to improve word vector quality in the context of these semantic and syntactic tasks, some hand picked preprocessing might indeed improve the model accuracy indirectly providing the algorithm with morphological information of words.

## 4.6. Conclusion

This chapter explored the performance of *Word2vec* model in the German language. To evaluate the quality of the learned word representations, similar logical reasoning tasks were constructed. These German language tasks were based on the English tasks used in the original work. The result shows that the model performs well in German, although the complexity of the language causes certain tasks to underperform when compared to the equivalent tasks in English. However, the comparison is not balanced, as there is more training data available in English than in German. The size of available corpus is a key factor on obtaining good representations.

Another important outcome of this empirical exploration is that the specifics of the text affect the quality of the word vectors. The relationship of the text and the objective task is important. For example, when comparing the English word vectors training from Wikipedia with the vectors learned from Google News dataset, some of the tasks performed better with ones from Wikipedia, although the amount of text used was much less. This means that for specific tasks, a large amount of text might not be necessary to learn good word representation.

Finally, preprocessing approaches were applied to the Wikipedia corpus to evaluate whether performing such operations helps to improve the quality of the learned representations. None of the approaches based on the Potter stemmer improved the performance of the model. Quite the opposite, all of them affected considerably the results. The heuristic preprocessing, based solely on stemming specific suffixes, not only improved 3x times



the performance on the specific target task, but it also produced a slight improvement on other unrelated tasks. This suggests that specific handpicked approaches to preprocessing, oriented towards a specific goal, might improve the quality of vectors.



## 5. Word Vector Features for Document Classification of German language Business Documents

### 5.1. Introduction

One of the motivations of feature learning is to be able to obtain useful representation that will improve the performance of existing tasks. In the case of IR/NLP related tasks word vector features have been shown to improve existing tasks such as NER, Chunking and Sentiment Analysis [37] [9]. Word vector features have also shown promising results in fields as machine translation and speech processing [8] [25].

In the field of TC there has been limited work on improving this task by using word vector representations. One of the possible reasons behind this, is that many of the current methods work well enough and they have been the industry standard in the last decade [34]. Another possible reason comes from the technical side. Namely, when using word representation, each word is represented as a  $n$ -dimensional vector, therefore a large document could be only represented as big matrix. However, most of the existing methods work only with vector representation of documents. This makes it hard to adapt existing techniques to these new features.

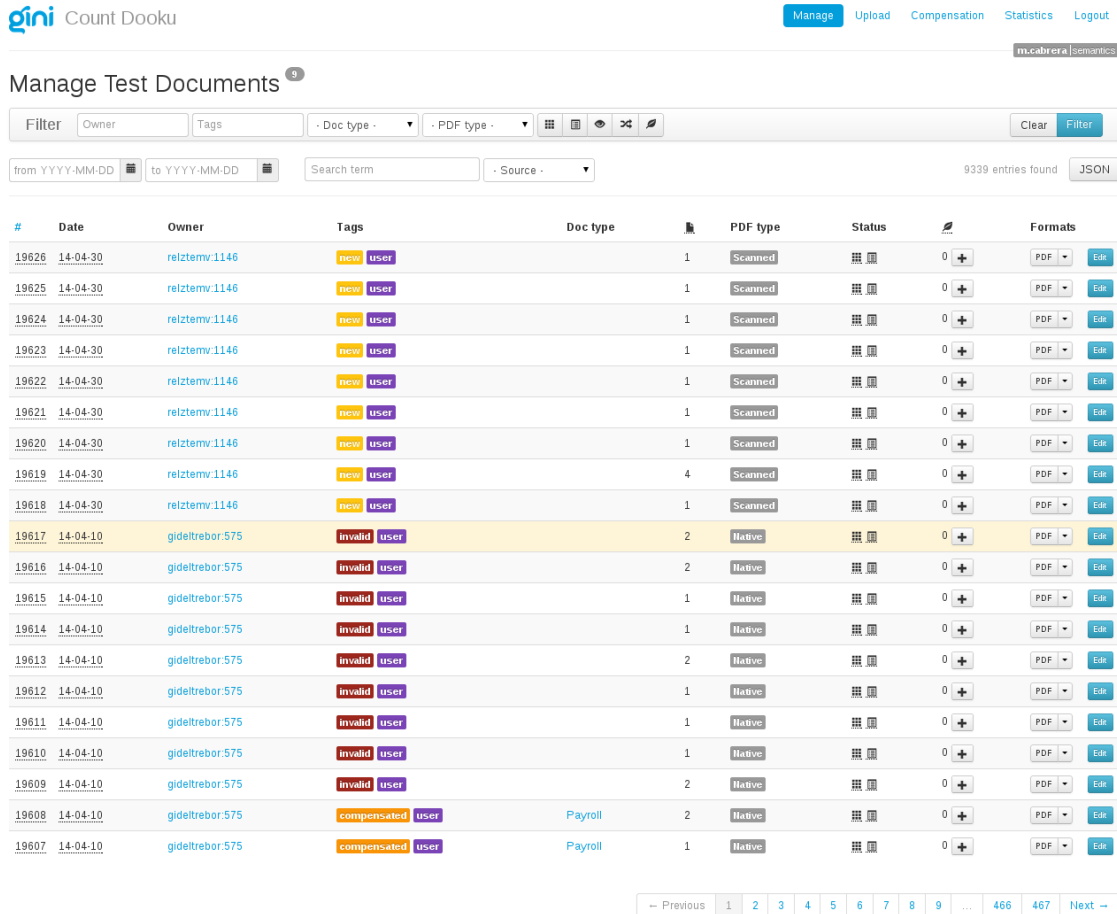
The task of Sentiment Analysis has been tackled by using learned word features from unlabeled data set [17]. To achieve this, different ways of representing a document from its word vectors were tried, all of them including some form of averaging. As Sentiment Analysis can be seen as a variation of traditional TC, this same approach is therefore used for performing document classification on this work. In fact, in the aforementioned article, the authors use same traditional approaches for TC combined with word vector features to train the sentiment predictor.

This chapter compares the performance of document representations obtained from word vector representation against other types of features in the context of classifying business-related documents in German. The objective is therefore to improve the existing classifier by using this new word representation. As an additional result from this work, other more traditional approaches are also evaluated, thus offering a benchmark among the current techniques used in the company, the state-of-the-art techniques and newer word vector based approaches for document classification. Given this, the objective of this chapter is two-fold: on one hand the objective is to evaluate the performance of word vector representations for document classification, and on the other, improve the existing classifier used by the company, even if this improvement is not based on the learned word representations.

The rest of the chapter is divided as follows: in section 5.2 the basics of text categorization are described. Section 5.3 describes Gini's document platform along with the require-

## 5. Word Vector Features for Document Classification of German language Business Documents

ments and challenges behind this application problem. Section 5.4 describes the approach currently used by the company. The dataset, the features and the classification algorithms used are described in section 5.5. Section 5.6 describes how the training of the algorithm was performed. Afterwards, section 5.7 discusses the most important results as well some particularities of the learned word vectors. Finally, section 5.8 the most important outcomes as well possible future research directions are discussed.



#	Date	Owner	Tags	Doc type	PDF type	Status	Formats
19626	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19625	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19624	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19623	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19622	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19621	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19620	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19619	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19618	14-04-30	relztemv.1146	new user		Scanned		PDF Edit
19617	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19616	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19615	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19614	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19613	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19612	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19611	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19610	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19609	14-04-10	gideltrebor.575	invalid user		Native		PDF Edit
19608	14-04-10	gideltrebor.575	compensated user	Payroll	Native		PDF Edit
19607	14-04-10	gideltrebor.575	compensated user	Payroll	Native		PDF Edit

Figure 5.1.: Dooku - Gini document storage platform.

## 5.2. Text Categorization

The goal of TC is to automatically assign, for each documents, categories selected among a predefined set. In opposition to the machine learning typical multi-class classification task which aims at attributing one class among the possible ones to each example, documents in a text categorization task may belong to several categories.

Formally, TC is the task of assigning a boolean value to each pair  $\langle d_j, c_j \rangle \in \mathcal{D} \times \mathcal{C}$  where  $\mathcal{D}$  is a domain of documents and  $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$  is a set of predefined categories. A true

value assigned to the pair  $\langle d_j, c_j \rangle$  indicates the decision of filling  $d_j$  under the category  $c_j$ , and a false value the decision of not filling it under that category [34].

More formally the task is to approximate a unknown target function  $\check{\Phi} : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$  using  $\Phi : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$  such that  $\Phi$  and  $\check{\Phi}$  “coincide as much as possible” [34].

So far only the binary case has been discussed, however in many realistic settings there are many possible (mutually exclusive or not) categories in which a document can be classified. In this situation there are two approaches to deal with the problem, namely *One-vs-All* and *One-vs-One*.

In the *One-vs-All* strategy, one classifier is trained per class. For each classifier, the class is fitted against other classes. Only  $n$  classifiers are needed  $n$  the number of classes. This makes the approach computationally efficient.

In the *One-vs-One* strategy one classifier is constructed per pair of classes and the class with more votes is selected. This approach needs  $n(n - 1)/2$  classifiers, meaning a  $O(n^2)$  training complexity. However, as each classifier only works with only two of the classes at a time, each classifier is trained with a subset of the available data, which might be useful for classifiers that do not scale well with the number of samples.

TC relies heavily on models that have been developed originally for IR. the reason of this is that TC is a content based document management task, and such it shares many characteristics with other IR tasks such as text search, specifically IR-style document representation using BOW model described in section 2.4.2.

### 5.2.1. Performance Measures

One question that arises after reading the definition of TC is how to measure the how well the classifier function  $\check{\Phi}$  (the approximation or hypothesis) matches the unknown target function  $\Phi$  (effectiveness). This is generally measured in terms of the classic IR metrics of precision  $P$  and recall  $R$  but adapted to the specific case of TC.  $P$  is defined as the probability that if a given a random document  $d_x$  is classified under the category  $c_i$  the classification is correct:

$$P(\check{\Phi}(d_x, c_i) = T \mid P(\Phi(d_x, c_i) = T))$$

Analogously  $R$  is defined as the probability that if a random document  $d_x$  is ought to be classified under  $c_i$ , this decision is taken by the classifier function:

$$P(\Phi(d_x, c_i) = T \mid P(\check{\Phi}(d_x, c_i) = T))$$

or the binary case of document categorization, *precision* and *recall* can be expressed mathematically using the numbers of true positive ( $TP$ ), false positive ( $FP$ ), true negative ( $TN$ ) and false negative ( $FN$ ) documents classified. The true or false value means that the classification was performed correctly or not, and the positive or negative define whether it belonged to the evaluated category or not.

With these definitions at hands precision  $P$  and recall  $R$  are defined as:

$$P = \frac{TP}{TP + FP} \quad , \quad R = \frac{TP}{TP + FN} \quad (5.1)$$

Another commonly used metric in the field is the  $F_1$  score can be interpreted as a weighted average of the precision and recall, where an  $F_1$  score reaches its best value at 1 and worst score at 0. Formally:

$$F_1 = \frac{2RP}{P + R} \quad (5.2)$$

For the case of multiple classes, the notion of *precision*, *recall* and  $F_1$ -score can be applied to each label independently. There are however ways to combine the results to obtain a global performance metric including *micro*-averaging, *macro*-averaging and *weighted* [34]. As the problem explored in this work is unbalanced, unless stated otherwise, *weighted* approach is used to reporting the results. As its name indicates, this approach takes into account the class unbalance by weighting the per-class result by the number of samples when obtaining the global performance measures. More formally  $P$ ,  $R$  and  $F_1$  are defined as:

$$P_{weighted} = \frac{1}{\sum_{t \in L} |y_t|} \sum_{l \in L} \frac{TP_l}{TP_l + FP_l}$$

$$R_{weighted} = \frac{1}{\sum_{t \in L} |y_t|} \sum_{l \in L} \frac{TP_l}{TP_l + FN_l}$$

$$F_{1weighted} = \sum_{l \in L} \frac{2R_l P_l}{P_l + R_l}$$

Where  $L$  is the number of classes and  $y_l$  is the set true sample-label pairs.

### 5.3. Gini document platform

Gini currently owns a database German language documents. These documents are personal and business related documents that have been bought from persons all over Germany. Document types are very broad and go from long insurance contracts to short remittance slips. The documents are stored in a central system database that can be accessed via an API (*Dooku*). Documents can be either *native* or *scanned*. *native* Document are those coming from original digital documents (e.g. a PDF file). *Scanned* documents on the other hand, are uploaded from an image obtained using a scanner or a camera (e.g. a mobile phone camera). Given the variability of the capture devices of the scanned documents, the quality of such images is not standard. This is an important factor to consider, given that after being uploaded all the documents are processed using a Optical Character Recognition (OCR) solution, so the quality of the images affect the accuracy of the text extraction. After the text has been extracted, it is stored into a database using a custom XML format. Other relevant information such the position of the words on the page; font size and style, etc. is also stored in the custom XML. Besides this information, a document is also assigned *tags* or labels help filter the results when querying the document database for particular characteristic of the documents (e.g. quality, source, etc.) Figure 5.1 shows a screen capture of the application with the most important columns.

After being stored and preprocessed, these documents are annotated by a human. The annotations include labels such as *sender name*, *bank account* (in the case of financial documents), *amount to pay* and *document type* (Document Type (DocType)). Using this annotation several machine learning algorithms are trained. These classifiers are then exposed to clients also via an API. Table A.1 shows the complete list of DocTypes available in the system and their description.

## 5.4. The Gini Document Classifier

One of the functionalities that Gini offers to their customers is document classification. As mentioned in the previous section, the classifier is trained from the annotations made by humans (labeled samples). In the particular case of the DocType classifier, it is implemented using Support Vector Machine (SVM) with handcrafted features such as the appearance of a particular word, the font of specific words, the number of pages of the documents, etc. These features once constructed are fed into the SVM implementation to generate a classifier. This classifier is then exposed through a Web service to client applications.

The particular implementation of the SVM is based on Java implementation of the popular LIBSVM [7] using a RBF Kernel. The training is done using grid search and the best model in 10-fold cross-validation with all the training data available is taken.

There are some external and internal challenges that this classifier is faced with:

- The quality of the data: As much of the data comes from varying degrees of capture devices, the quality of the OCR is not perfect, thus producing a lot of noise in the stored text.
- Feature generation and extension: Every time a new DocType is required to be supported, a manual analysis of this particular document type should be done with the purpose of identifying what features are important and then once selected, are then added to the document. No feature selection procedure is performed.
- No validation of the classifier is done on a validation set and quality is measured on documents belonging to the training set. This validation step, albeit common and required in a scientific setting, is many times overlooked in the industry.

Taking in consideration all these challenges, the objective is then to design a better classifier that allows solving the previously mentioned issues at least partially.

## 5.5. Experiments

As mentioned in the introduction, three different approaches are going to be compared, specifically:

- The current SVM implementation with the handcrafted features.
- A new SVM implementation using BOW features.

- A new SVM implementation using *Word2vec* word vector as features.

As can be seen, the underlying classification algorithm is kept while the real evaluation is performed on the features. The idea is to verify that in fact better features account for better classification. The following sections describe each of the features and the specifics of each algorithm.

### 5.5.1. Gini Document Database and Dataset Description

Section 5.3 described briefly the platform where document that are owned by Gini reside. This section describes the specific of the documents used for training the classifier.

Gini database of documents consists of approximately 9339 different documents. From these, only 2685 are used to train the document classifier. Table 5.1 show the distribution of the data set among the classes. As can be seen from the data set, not all the existing classes are used. Furthermore, the class called *Other* includes documents that do not fall into any of the categories listed. It is also a unbalanced data set, having a number of instances per class ranging from 69 to 650. As all the documents classes need not to be categorized, only those required by the business needs are used. However if all documents are put into the *Other* class, this will create large unbalanced class that will hurt the performance of the classifier. In Addition to this, many of the other documents are not annotated or annotated wrongly, that is, they are correct documents but have not been properly annotated with the correspondent DocType. Therefore, using these documents will probably create noise in the dataset.

Table 5.1.: Distribution of document type in Gini DocType classifier training set.

Document Type	# Instances
CreditCardStatement	158
Other	650
BankStatement	369
Contract	178
InsurancePolicy	105
Payroll	69
Invoice	570
Reminder	596
Total	2685

As mentioned previously, there was not an official test set used to evaluate the classifier performance. Therefore a new one was defined using a stratified random selection of 20% original set for the testing and the other 80% for training, ending up with 2156 documents for training 539 for testing. The same documents are used to evaluate all other classifier / feature sets.



### 5.5.2. Handcrafted Features

As mentioned back in section 5.4 the current Gini document classifier is based `LIBSVM` [7] using `RBF` Kernel. The features used to train this classifier are handcrafted features that count the number of words (actually the matches of specific regular expressions) in the text, the font size of some words, the digit character ratio, the average font size, etc. In total 697 features like these are used.

### 5.5.3. BOW Features

For the BOW features, the traditional tf-idf [33, 34] discussed in section 2.4.2 is used as features. For the actual feature extraction the `TfidfTransformer` from `Scikit-learn` [31] was used ending up with 55122-dimensional sparse vectors.

As for pre-processing for feature extraction, typical stop word substitution was performed with the help of `NLTK` [4]. Additional stop word clean up was detected by looking at the vocabulary, for example extremely large words (more than 30 characters) and word containing non German alphanumerical characters were verified that were not the result of wrong OCR readings and were removed accordingly. In addition to this, words that appeared in 90% of the documents were removed before generating the tf-idf features.

### 5.5.4. Word Vector based Features

As mentioned early in the document, using word vector features from TC poses some challenges from the algorithm point of view. Although in theory we could represent a document as the concatenation of all the vector representation of the words it contains, this in practice would be impossible as first each document has a different number of words which would imply high-dimensional variable size dense vector representing a document that could not be used to train with traditional learning algorithm and in particular the current classification algorithm used by Gini.

Another approach described in previous work [17] named *document mean representation* consists of average of the vector representation of the words present in the document. More formally, assuming that there is vocabulary of size  $|V|$  and with learned  $\beta$ -dimensional word vector representation of that vocabulary, then  $R \in \mathbf{R}^{(\beta \times |V|)}$  is the matrix of learned word vectors. The representation of a document  $d \in |V|$  can be obtained using the following method:

$$d = Rv$$

Vector  $v$  is a  $|V|$ -dimensional binary vector representation of the document. The resulting vector  $d$  is then normalized or the mean can be applied (hence the name mean).

Although simple in principle, this document representation has shown good results in tasks such as sentiment analysis and even an improvement when used in tandem with more traditional document representation [18], even though in the cited referenced, they used another word vector representation that did not have the linear compositionality characteristic of the *Word2vec* generated word representation [26].

### Training of the Word Vector Features

In order to use word vector representations for document classification it is necessary to obtain the word vectors. All the documents of the document classification training set are in German, therefore a German language word vectors are needed. One available option is to use corpus like the German Wikipedia. However, as previously mentioned in chapter 4, the nature of the text from which the word vectors are generated influences the quality of the word representation and their performance in specific tasks. For that reason, the whole unlabeled document database of Gini (described in section 5.5.1) is used to generate the word vector used for document classification.

The data from Gini document is stored as XML documents in a relational database. These document were exported and preprocessed. Besides the text, the format contains the coordinates of the position of each word existing in the document. This information is not necessary for the purpose and is therefore removed. In addition, the preprocessing described back in section 4.2 is also applied to this dataset.

Table 5.2 compares the two data sets used to generate the document classification. There is a big difference in vocabulary and number of token trained. However, as it will be shown in the result section, this does not necessarily mean that for this particular task the word vector generated from Wikipedia will generate features that perform better.

Table 5.2.: Comparisson of the datasets used to train the word vector for DocType classification

Dataset	Size (MB)	# Tokens	Vocabulary Size
Gini Dataset	27	3657159	33596
Wikipedia German	6656	1051584822	1871739

## 5.6. Training and Evaluation

This section describes the training and evaluation procedure used for each of the used features. As all of them are different, and in particular, the word vector based requires the previous generation of word vectors, additional steps are required to select the word vector model that provides a real generalization power. The evaluation measures used for the comparison are the standard ones used in document classification: *precision*, *recall*, *f-score*. These measures are weighted to account for unbalanced in the classes. These performance measures are described in detail in section 5.2.1.

### 5.6.1. Handcrafted Features

For the handcrafted features and BOW based classifier a standard 10-fold cross-validation on the training data was used to select the best model. As mentioned in section 5.4, this classifier is based on on LIBSVM [7]. Afterwards the classifier were evaluated on the testing set and based on those results are compared against each other.

### 5.6.2. BOW Features

For the BOW feature based classifier as well as for the word vector based one, the SVC implementation of `Scikit-learn` with a linear kernel and automatic class weights was used. This implementation is also based on `LIBSVM`. However, after evaluating other implementation, the faster `LinearSVC` based on `LIBLINEAR` [13] produced faster training times and slightly better results. This classifier was also tested on the handcrafted features but there was a significant loss in performance.

Table 5.3.: Top-5 of evaluation of document classification task using *document mean vector representation* generated from word vectors learned using Gini unlabeled dataset. *no validation set used*

Arch.	Algo.	Win.	Dim.	Prec.	Rec.	F-Score	Prec. train	Rec. train	F-score train
HS	CBOW	5	500	0.8822	0.8812	0.8813	0.9754	0.9749	0.9749
HS	CBOW	5	600	0.8933	0.8923	0.8923	0.9671	0.9661	0.9661
HS	CBOW	5	460	0.8787	0.8775	0.8777	0.9635	0.9624	0.9624
HS	CBOW	5	540	0.8990	0.8979	0.8979	0.9530	0.9508	0.9509
HS	CBOW	5	400	0.8842	0.8849	0.8841	0.9514	0.9499	0.9497

### 5.6.3. Word Vector Features

For the *document mean vector representation* two datasets were available to train the word vectors: The Gini dataset and the German Wikipedia. For the Wikipedia corpus we selected the model that performed the best as describe in section 4.4.3. As for the word vector generated from Gini dataset size of the dataset allowed a fast model generation. So around 120 *Word2vec* model generation combinations were the parameters summarized in table 3.1 were varied. By using word vector in this way, it is easy to overfit the model to the training data. In fact, as the results showed, the high the dimension of the word vectors the better the performance on the training set, however on the testing set it decreased. For that reason is important that when training using word vector based features to use a validation set to avoid overfitting on the training set.

Table 5.4.: Top-5 of evaluation of document classification task using *document mean vector representation* generated from word vectors learned using Gini unlabeled dataset. The model were trained using a subsampling of  $10^{-5}$ .

Algorithm	Architecture	Window	Dim.	Precision	Recall	F1-Score
HS	Skip-gram	15	160	0.9182	0.9165	0.9167
HS	Skip-gram	15	340	0.9156	0.9146	0.9146
HS	Skip-gram	15	200	0.9127	0.9110	0.9111
HS	Skip-gram	10	460	0.9106	0.9091	0.9092
HS	Skip-gram	15	240	0.9010	0.9091	0.9091

Table 5.3 shows an example of this behavior. The word vector model with the highest dimension, in this case using the CBOW architecture, obtains the best training performance. However, when tested against the testing set the result were not considerable better. Another way to explore this behavior is to plot a performance measure against the size of the word vector. Figure 5.2 shows clearly this phenomenon. After reaching high values for the word vector dimension, the model star overfitting the training data.

After training using a evaluation set to avoid overfitting, model trained with Skip-gram architecture performed better. Table 5.4 summarizes the results on the testing set after training with the validation set. The large size of the window is expected, as large window allow the model to capture semantic information in opposition to shorter ones that are generally used for capturing syntactic relationships.

When using the validation step to avoid overfitting, the performance on the testing set increase considerably while obtaining an acceptable performance on the training set. This additional complexity in the training could be avoided using different strategies. For example, a specific logical reasoning tasks targeted to the document classification task could be created. Another possibility is to adjust the word vectors at the same time that the document classifier is being training, i.e., *fine-tuning* the word representation for this particular task.

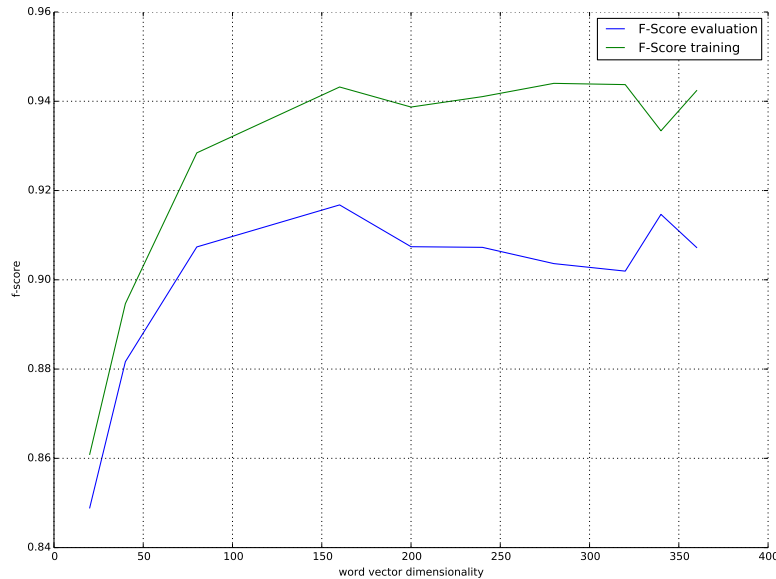


Figure 5.2.: Training set evaluation set f-score vs word vector dimension for model trained used HS as training algorithm with a *window* of 15.

## 5.7. Results

This section compares the different models and discussed the most important results. Table 5.5 summarizes the results. In general terms, BOW features work better for the document classification of this dataset. However, the mean document representation performs very similar in term of accuracy. It is also important to notice the amount of data used to train the word vector is very small in comparison to data used to evaluate the logical reasoning tasks in chapter II.

Table 5.5.: Comparison of performance of different features. The precision, and recall and F1-Score are weighted based on the number of samples per class.

Feature	Precision	Recall	F1-Score
BOW	<b>0.9242</b>	<b>0.9239</b>	<b>0.9238</b>
Word Vectors (Gini)	0.9170	0.9147	0.9130
Word Vectors (Wikipedia)	0.8645	0.8578	0.8514
Handcrafted	0.8636	0.8581	0.8587

The table also display the results of using word vectors trained on the Wikipedia German in the same way as the one trained with Gini data (Section 5.5.4). For this particular experiment the model with best performance in the logical reasoning tasks as described in section 4.5 was used. Even though the Wikipedia is a general corpus of text, containing no domain specific information, the word mean document representation seem to convey enough information to create an acceptable classifier, even surpassing in performance the hadcrafted features on the evaluation set.

To visualize better the performance of a particular classifier for each of different classes a confusion matrix is useful. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. Therefore, a *good* confusion matrix has most of the values on the diagonal, i.e. the predicted class matches the actual class.

Figure 5.3, 5.4 and 5.5 show the confusion matrix for the *mean document representation features* (i.e. word vector based classifier), the BOW features and the handcrafted features respectively. As the classes are unbalanced, the percentage of classification under that class is shown. For example, the cell on the first row and first column shows the percentage of all the `BankStatement` document class that were predicted (correctly) under `BankStatement`, the second column the same document but (incorrectly) predicted under the class `Contract`.

Both BOW and word vector based features behave pretty similar. The document class `Other` appears to be the harder to classify correctly, as all of the classifiers had problem with it. This can be explained for the nature of the class, which is composed of different type of documents, even from existing classes on the training dataset, thus creating a lot of noise into the dataset. Furthermore, in the BOW based and word vector based classifier this is the only class that less than 90% were correctly classified.

## 5. Word Vector Features for Document Classification of German language Business Documents

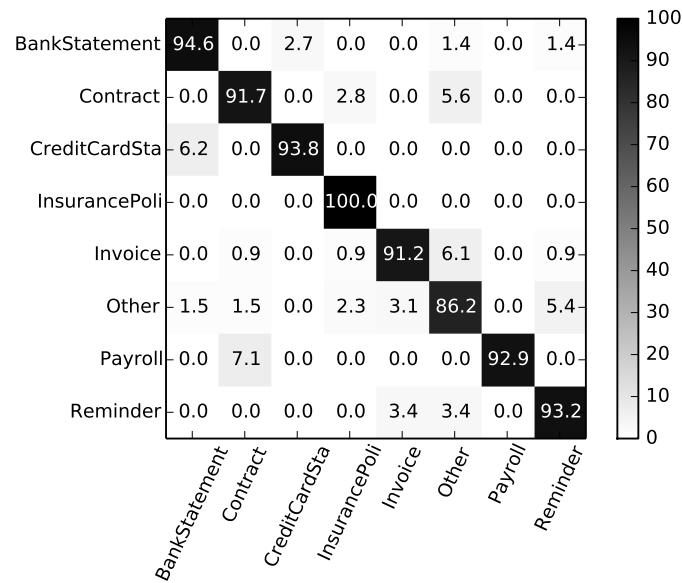


Figure 5.3.: Confusion matrix of the classification results using *document mean vector representations* features. The values are the percentages of samples classified under a specific document type.

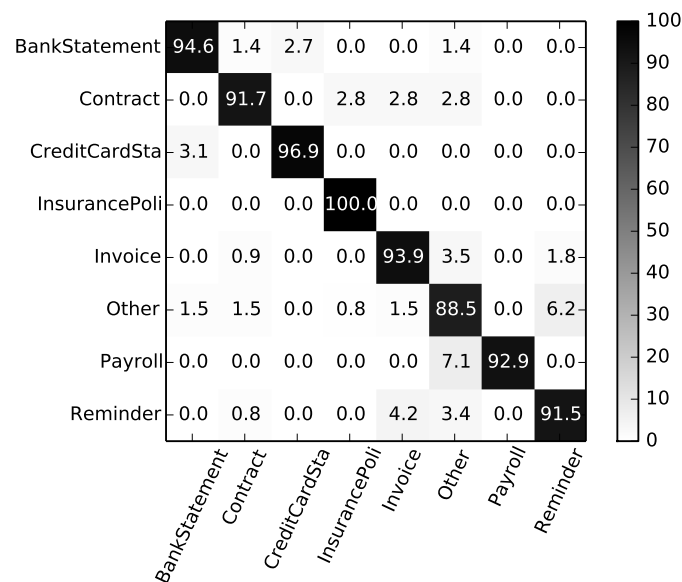


Figure 5.4.: Confusion matrix of the classification results using BOW features. The values are the percentages of samples classified under a specific document type.

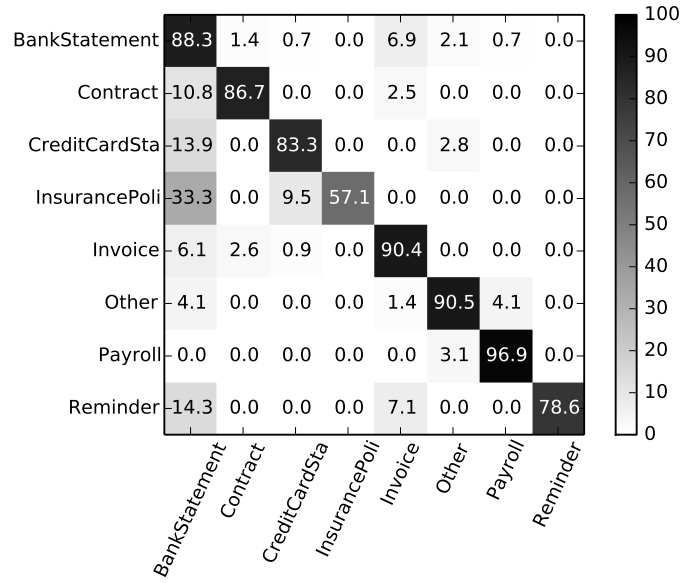


Figure 5.5.: Confusion matrix of the classification results using the handcrafted features . The values are the percentages of samples classified under a specific document type.

### 5.7.1. Visualization of Features

Another way to study the characteristics of the features is through the visualization in a 2D/3D projection. For that purpose the t-Distributed Stochastic Neighbor Embedding (t-SNE) [38] is used to generate a 2D embedding of the features. This technique besides generating good visualization it often reveal underlying structure of a dataset, otherwise hidden when using other techniques for visualizing high dimensional data.

Figure 5.6, 5.7 and 5.8 show the the embeddings of the *document mean vector representation*, the tf-idf BOW and handcrafted features of the training set respectively. The t-SNE visualization of the word vector based features reveals a structure of the documents represented using this features. Apart from some outliers, most of the document form well established clusters.

For the BOW features the embeddings show also a structure and some clusters, although it is less clear that then one of word vector features. The handcrafted features however, show almost no clear structure. It is arguable then that this structure is correlated to the classification performance or at least offer a good visual hint of the quality of the representation / features.

### 5.7.2. Generalization Power

So far it is clear that both BOW and word vector based features are better than the handcrafted ones, with BOW performing slightly better than the word vector features. The question now is to decide which feature to use based on the situation. An interesting factor to take into consideration in order to select a particular set of features it is the gen-

## 5. Word Vector Features for Document Classification of German language Business Documents

---

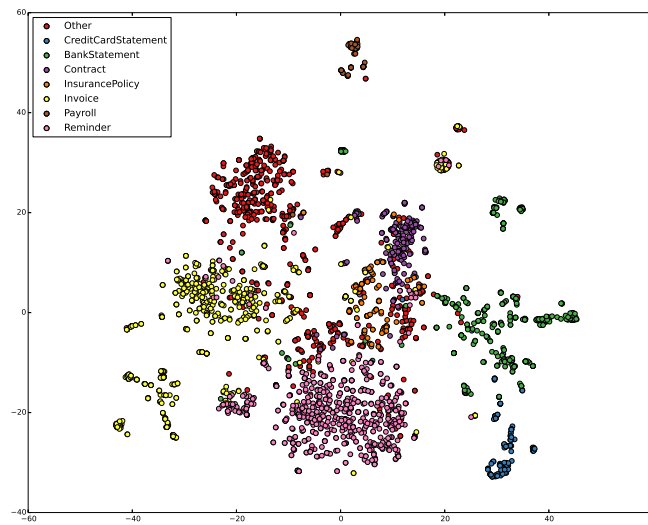


Figure 5.6.: t-SNE visualization of *document mean vector representation* of the training set

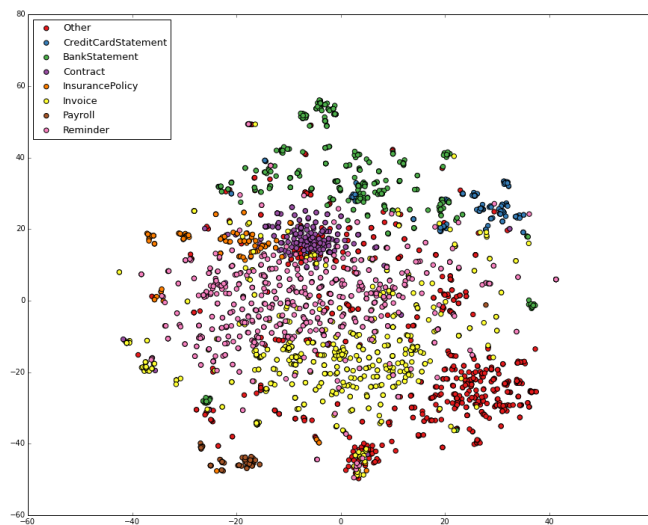


Figure 5.7.: t-SNE visuzalization of BOW features of the training set.

eralization power that they offer. To evaluate this it is useful to visualize a lerning curve *learning curve*. A learning curve evaluates the training and test scores for different training



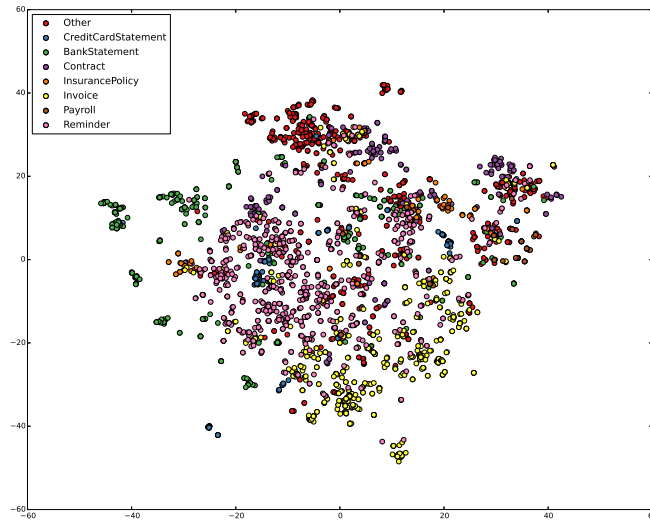


Figure 5.8.: t-SNE visualization of handcrafted features of the training set.

set sizes. Thus allowing visualizing how well the generalization is with smaller amount of training data, i.e. the bias of the estimator.

Figure 5.9 shows the learning curve of both the BOW based and the word vector classifier. The figure show that for small training set sizes the word vector features generalize better. The reason of this is that many of the semantic knowledge is obtained via the previous unsupervised training of the word vectors. For datasets with few training samples, this might be a good alternative to generate classifiers that generalize better with less labeled trained data.

## 5.8. Conclusion & Future Work

This chapter presented an approach of using word vectors features for document classification in the context of German language business document. In addition to this, a comparison with the more traditional tf-idf based features and an existing handcrafted feature set was performed. This chapter shows that *Word2vec* word vector features works similar to the state-of-the-art techniques for document classification, even when generated with a relatively small corpus.

Both the *Word2vec* based features and the BOW perform better than the handcrafted features for most document classes. Both approaches are semi-automated in the sense that the specific features do not need to be handpicked. However, for BOW features the larger the training corpus generally the larger and sparser the feature set is. Thus, feature selection and dictionary-based methods will become necessary. Word vector features do not suffer of this, as they are generated using an unsupervised algorithm. Yet there is

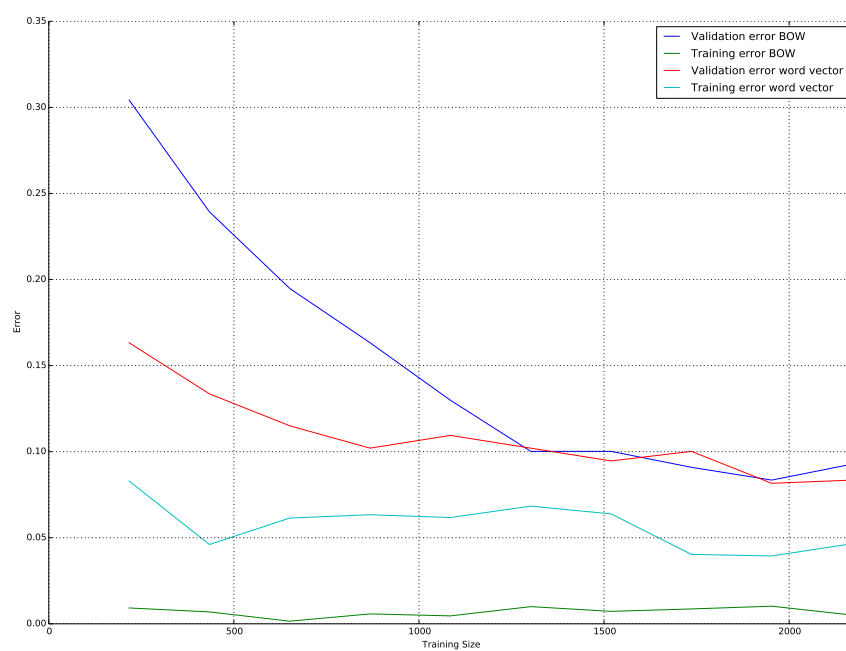


Figure 5.9.: Plot of training and validation set mean classification error for different training sizes

no way to directly ensure the quality of the word vectors for this particular task. Thus, when using word vector features it is necessary to either use an additional validation set to set the parameter of the word vector model or define an equivalent logical reasoning task targeted towards document classification. Another approach to solve this is to first generate the pre-trained vector and then when realizing the document class predictions, propagate error back to the word vector. This could be done by replacing the SVM with a full-fledge neural network and a Softmax layer or by using a NN in tandem with a SVM in a similar way to [36].

Another interesting characteristic of using word vector for document classification, and in general any other IR/NLP task, is the generalization power that word vector based features provides. When new unlabeled data is made available the word vector can be re-generated, thus obtaining better features by simply providing more data to the word vector model. Even with unrelated unlabeled data, the word vector generated with them provided even better performance than the handcrafted feature. It is then expected then, that with an equivalent, or at least larger amount of domain specific text the word vector features provide better generalization power. Further experimentation is then necessary of explore the extent of this generalization power.

Document classification is one of the IR/NLP tasks inside Gini. Therefore, it would be interesting to evaluate how the word vector features can be used to improve/replace the existing Conditional Random Fields (CRF) based NER tasks inside the company, maybe in a similar way to [37].

Finally, this chapter shows that word vector features, are useful even in absense of large amount of data to train them from. It would be interesting to evaluate the word vector once that more data becomes available.



## 6. Conclusion and Future Work

This master thesis studied the behavior of distributed representation learned by the *Word2vec* model on German language data. It also evaluated and compared the learned distributed representations against traditional handcrafted features for the classification of German language business document. The evaluation of the word vector features included not only the analytical evaluation of the obtained word representations, but also the construction of a set semantic and syntactic question in the German akin to the one existing already for the English language. This is useful because it will allow other word vector model to evaluate the performance in the German language. The evaluation also yielded interesting results regarding the different factors affecting the performance of the *Word2vec* model. For instance, the impact of the semantic information contained in the corpus on learning the model. In particular, when building word representations that are topical smaller corpus of text containing with richer semantic relationships are preferable to large amounts of text with no specific context. The study of the word vector representation showed that although the model captured semantic relationships correctly, for the syntactic tasks it was not as successful. This can be explained by the morphological complexity of the German language in comparison to English. So, in order to capture properly these relationships it is necessary also to give the model more information about the morphological structure of the language.

In the context of document classification, the learned word vector representations did not outperform the state of the art tf-idf features. However, they did perform better than the handcrafted features on the German language business document used for the evaluation. However, the difference between the performance measures is not large. This is relevant if the amount of data used to train the model is taken into account. By comparison, the trained model by Google on the Google News dataset used terabytes of Google news data. The model trained on Gini dataset contained less than 100MB of information and yet they performed fairly similar. This suggests that with more relevant data it is possible to learn better representations, thus improving the performance of the document classifier. It is important to make emphasis on the word *relevant*, as a model trained on the German Wikipedia was also used yielding good results as well but not outperforming either the tf-idf features or the model learned on business documents. However, this is a good example of transfer learning where an unrelated dataset is used to learn features that can be used to tackle a completely different problem.

The features used to classify document were just the averaged sum of word vector representation of the used documents. Although it is reported that for long documents this will not yield good results [26], for this particular problem they performed well. Yet, more sophisticated word representation for long documents are needed in order to account for the importance of words and the context. Recent work goes in this direction by obtaining fixed-length distributed representation of sentences and documents via an unsupervised learning algorithm similar to *Word2vec*, but in addition having a paragraph matrix which

represents the missing information from the current context and act as memory of the topic of the paragraph [16]. However, this only work so far for short sentences / documents. Thus, techniques for obtaining distributed representation of long documents are yet to be developed.

Finally, given the nature of distributed representation, once the unsupervised learning of the features is complete, further supervised fine-tuning can be achieved using the specific tasks objective as evaluation function. This particular problem has not been yet tackled for word vector representations in the context document classification, let alone for paragraph or long document representations.

# Appendix





## A. Types and Features of the Gini Document Classifier

### A.1. List of Document Types Available in the Gini Database

Table A.1.: List of document types available in the Gini Dataset

DocType code	Description
Administrative_offence	Administrative offence - Administrative Rechnungen, wie z.B. Strafzettel, Bußgeldbescheide
admission_ticket	Admission ticket - Eintrittskarte
airline_ticket	Airline ticket - Flugticket (Boardkarte, Buchung)
bank_statement	Bank statement - Kontoauszug
confirmation_of_termination	Confirmation of termination - Kündigungsbestätigung
contract	Contract - Vertrag
contract_confirmation	Contract confirmation - Bestätigung Vertrag (alle Domänen)
contract_energy	Contract energy - Energie (Vertrag)
contract_extension	Contract extension - Vertragsverlängerung
contract_insurance_automobile	Contract insurance automobile - Kfz-Versicherung (Vertrag)
contract_insurance_household	Contract insurance household - Hausratversicherung (Vertrag)
contract_insurance_legal_costs	Contract insurance legal costs - Rechtsschutz (Vertrag)
contract_insurance_third_party_risk	Contract insurance third party risk - Haftpflicht (Vertrag)
contract_telco	Contract telco - Telko (Vertrag)
credit_card_statement	Credit card statement - Kreditkartenabrechnung
credit_note	Credit note - Gutschrift
delivery_note	Delivery note - Lieferschein
insurance_policy	Insurance policy - Versicherungsunterlagen

invoice	Invoice - Rechnung
lease_contract	Lease contract - Mietvertrag Leasingvertrag
letter	Letter Brief - (interessante Docs für die es keinen Doc-Type gibt)
medical_finding	Medical finding - Medizinischer Befund
medical_insurance	Medical insurance - Krankenversicherungsdokument
notice_of_termination	Notice of termination - Kündigung
offer	Offer - Angebot
order_confirmation	Order confirmation - Bestellbestätigung, Auftragsbestätigung
payroll	Payroll - Gehaltsabrechnung
receipt	Receipt - Kassenzettel
reminder	Reminder Mahnung
remittance_slip	Remittance slip - Überweisungsträger
return_form	Return form - Rückgabeformblatt (z.B. ecommerce amazon)
social_security_statement	Social security statement - Sozialversicherung, Meldebescheinigung zur Sozialversicherung
statement_energy_consumption	Statement energy consumption - Verbrauchsabrechnung (Strom, Gas, Wasser, etc)
statement_pension	Statement pension - Rentenbescheid, Rentenversicherungsunterlagen
stock_document	Stock document - Wertpapierdepotauszüge, Dividendenabrechnungen, Orderbestätigungen
tax_assessment	Tax assessment Steuerbescheid
tax_document	Tax document - Lohnsteuerbescheinigung, Bestätigung für die Steuererklärung
tax_return	Tax return - Steuererklärung
taxi_receipt	Taxi receipt - Taxi-Quittung
terms_and_conditions	Terms and conditions - Geschäftsbedingungen
travel_expense_report	Travel expense report - Reisekostenabrechnung

# Acronyms

<b>SVM</b>	Support Vector Machine
<b>NNLM</b>	Neural Network Language Models
<b>FNNLM</b>	Feedforward Neural Network Language Model
<b>RNNLM</b>	Recurrent Neural Network Language Model
<b>NLP</b>	Natural Language Processing
<b>NN</b>	Neural Network
<b>TC</b>	Text Categorization
<b>IR</b>	Information Retrieval
<b>HS</b>	Hierarchical Softmax
<b>CBOW</b>	Continuous Bag of Words
<b>NLTK</b>	Natural Language Toolkit
<b>NER</b>	Named Entity Recognition
<b>VSM</b>	Vector Space Model
<b>OCR</b>	Optical Character Recognition
<b>DocType</b>	Document Type
<b>BOW</b>	Bag of Words
<b>tf-idf</b>	term frequency-inverse document frequency
<b>t-SNE</b>	t-Distributed Stochastic Neighbor Embedding
<b>CRF</b>	Conditional Random Fields
<b>NCE</b>	Noise Contrastive Estimation
<b>NEG</b>	Negative Sampling
<b>POS</b>	Part-of-Speech tagging
<b>LDA</b>	Latent Dirichlet Allocation
<b>LSA</b>	Latent Semantic Analysis

**SVD** Singular Value Decomposition

**ME** Maximum Entropy

## List of Figures

2.1. Feed Forward Neural Network Model Language Model Architecture. $C(i)$ is the $i$ -th word feature vector. [3]. . . . .	12
2.2. Recurrent Neural Network Language Model Architecture . . . . .	14
2.3. Architecture of the NNLM proposed by Mikolov. . . . .	15
3.1. CBOW Architecture . . . . .	19
3.2. The Skip-gram architecture. The current word $w(t)$ is used to predict the surrounding words. . . . .	20
3.3. An example of a Huffman binary tree used to calculate the hierarchical softmax. This is a small imaginary corpus of 8 words. The leaf nodes contain the words, the probability and the Huffman codes. Inner nodes contain just the probabilities of its children. . . . .	22
5.1. Dooku - Gini document storage platform. . . . .	46
5.2. Training set evaluation set f-score vs word vector dimension for model trained used HS as training algorithm with a <i>window</i> of 15. . . . .	54
5.3. Confusion matrix of the classification results using <i>document mean vector representations</i> features. The values are the percentages of samples classified under a specific document type. . . . .	56
5.4. Confusion matrix of the classification results using BOW features. The values are the percentages of samples classified under a specific document type. . . . .	56
5.5. Confusion matrix of the classification results using the handcrafted features . The values are the percentages of samples classified under a specific document type. . . . .	57
5.6. t-SNE visualization of <i>document mean vector representation</i> of the training set . . . . .	58
5.7. t-SNE visualization of BOW features of the training set. . . . .	58
5.8. t-SNE visualization of handcrafted features of the training set. . . . .	59
5.9. Plot of training and validation set mean classification error for different training sizes . . . . .	60



# List of Tables

3.1. Summary model training parameters . . . . .	18
3.2. Examples of the original types of syntactic questions in the Semantic-Syntactic Word Relationship test as defined by [22]. . . . .	25
4.1. Summary of changes to the original task . . . . .	34
4.2. Examples of word pair of task used to evaluate the German word vector model. The umlaut marking diaeresis has been replaced by the equivalent diphthongs ae,oe and ue. . . . .	35
4.3. Summary of datasets . . . . .	36
4.4. Comparison of the combination of the available architectures and algorithms. Only questions containing words form the most frequent 30k are used. Numbers are in percents. . . . .	37
4.5. Accuracy on subset of the semantic task Word relationships test set using vector Skip-gram architecture with NEG. Only questions containing words form the most frequent 30K are used. . . . .	37
4.6. Accuracy of the best models obtained using the Skip-gram architecture with Negative Sample on the German Wikipedia dataset. A subsampling of $10^{-5}$ is used . . . . .	38
4.7. Accuracy of the best models separated by category using the German Wikipedia, the English Wikipedia and the pre-trained Google News dataset word vectors. . . . .	39
4.8. Comparison of different stemming schemes. The model were trained with the best parameter after stemming the original file using the strategies described in section 4.5.2. . . . .	41
5.1. Distribution of document type in Gini DocType classifier training set. . . . .	50
5.2. Comparisson of the datasets used to train the word vector for DocType classification . . . . .	52
5.3. Top-5 of evaluation of document classification task using <i>document mean vector representation</i> generated from word vectors learned using Gini unlabeled dataset. <i>no validation set used</i> . . . . .	53
5.4. Top-5 of evaluation of document classification task using <i>document mean vector representation</i> generated from word vectors learned using Gini unlabeled dataset. The model were trained using a subsampling of $10^{-5}$ . . . . .	53
5.5. Comparison of performance of different features. The precision, and recall and F1-Score are weighted based on the number of samples per class. . . . .	55
A.1. List of document types available in the Gini Dataset . . . . .	67





# Bibliography

- [1] Y. Bengio. Neural net language models. 3(1):3881, 2008. revision #91566.
- [2] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.
- [3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.
- [4] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, Beijing, 2009.
- [5] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.
- [6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.
- [7] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [8] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML, 2008*.
- [9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011.
- [10] E.A.H. Dictionaries. *The American Heritage Book of English Usage: A Practical and Authoritative Guide to Contemporary English*. Houghton Mifflin Harcourt, 1996.
- [11] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '88*, pages 281–285, New York, NY, USA, 1988. ACM.
- [12] M. Durrell. *Hammer's German Grammar and Usage, Fifth Edition*. Hrg Series. Taylor & Francis, 2011.
- [13] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

- [14] Geoffrey E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12. Hillsdale, NJ: Erlbaum, 1986.
- [15] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.
- [16] Q. V. Le and T. Mikolov. Distributed Representations of Sentences and Documents. *ArXiv e-prints*, May 2014.
- [17] 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-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.
- [18] Andrew L Maas and Andrew Y Ng. A probabilistic model for semantic word vectors. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.
- [19] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.
- [20] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Cernocky. Strategies for training large scale neural network language models. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 196–201, Dec 2011.
- [21] Tomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno university of technology, 2012.
- [22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
- [23] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Kei-kichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1045–1048. ISCA, 2010.
- [24] Tomas Mikolov, Jiri Kopecky, Lukas Burget, Ondrej Glembek, and Jan Cernocky. Neural network based language models for highly inflective languages. In *ICASSP*, pages 4725–4728. IEEE, 2009.
- [25] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168, 2013.
- [26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.
- [27] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751. The Association for Computational Linguistics, 2013.

- [28] Andriy Mnih and Geoffrey Hinton. A scalable hierarchical distributed language model. In *In NIPS*, 2008.
- [29] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26*, pages 2265–2273. 2013.
- [30] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *AISTATS’05*, pages 246–252, 2005.
- [31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [32] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.
- [33] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523, 1988.
- [34] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.
- [35] S. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77–89, October 1997.
- [36] Yichuan Tang. Deep learning using support vector machines. *CoRR*, abs/1306.0239, 2013.
- [37] 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, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
- [38] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. 2008.