Word Embeddings for Natural Language Processing

THÈSE N^{O} 7148 (2016)

PRÉSENTÉE LE 26 SEPTEMBRE 2016

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

LABORATOIRE DE L'IDIAP

PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Rémi Philippe LEBRET

acceptée sur proposition du jury:

Prof. J.-Ph. Thiran, président du jury
Prof. H. Bourlard, Dr R. Collobert, directeurs de thèse
Dr Y. Grandvalet, rapporteur
Prof. G. Attardi, rapporteur
Dr J.-C. Chappelier, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2016

He who is not courageous
enough to take risks will
accomplish nothing in life.
— Muhammad Ali

To my grandfathers. . .

Acknowledgements

I would like to express my gratitude to people who contributed to this thesis and my life as a Ph.D. student.

First, I would like to thank Dr. Ronan Collobert for giving me the opportunity to join his team at Idiap for doing my doctoral studies. Thanks for being always available during the first two years to teach me all the tricks of the trade for training neural networks. I would also like to thank Dr. David Grangier and Dr. Michael Auli for the day-to-day supervision during my 6-month internship at Facebook, where I learned a lot. At that time of my PhD, it gave me enthusiasm and energy to move forward. Many thanks to Prof. Hervé Bourlard for making Idiap such a great place to work, to Nadine and Sylvie for their kindness and responsiveness. I am also particularly grateful to the Hasler Foundation for supporting me financially.

I really appreciate the time and dedication my thesis committee devoted to reviewing my work: Prof. Giuseppe Attardi, Dr. Yves Grandvalet, Dr. Jean-Cédric Chappelier, and Prof. Jean-Philippe Thiran. Thank you for the fruitful discussions and your comments.

A special dedication goes to my fellow colleagues from the applied machine learning team at Idiap: Dimitri, Joël, and Pedro. Thanks to Dimitri for teaching us all the specificities of Switzerland and Game of Thrones; to Joël for organizing great team dinners composed of Raclette and Fondue; and to Pedro for always being in a good mood and for his Brazilian accent. I have a special thought for César, the Fifth Musketeer, who joined the team for a year before boarding a plane to Montreal. Also thanks are in order to all my floorball teammates: Bastien, Laurent, Pierre-Edouard, David, Hugues, Christian, Michael, Manuel, Elie, Paul, Romain, Rémi.

This Ph.D. has given me the opportunity to discover Switzerland, specifically the Canton of Valais. I would like to thank the people of Le Trétien (the small mountain village where I lived) for the warm welcome. I particularly want to thank Monique and Christian for their love and presence; and my closest neighors Vicky and Thierry, Rita and Benoit, Marie-Noëlle, and Damien for their friendship and generosity.

The two last years of this journey have been stressful with a lot of questions and concerns. This period would have been very difficult without the unconditional support and love of my wife Claire. Thank you for all the encouragement, your *joie de vivre* and for the sacrifices that

you made for me. Now it is my turn to support you in your project.

Finally, I want to thank my family for their loving support through the years. Thanks to my parents, Christine and Philippe; to my sister, Julie, her husband, Eddie, my nephew, Matilin, and my niece, Margaud. Thanks for frequently making the long trip to Le Trétien, where we shared some good moments together. Last but not least, I thank my grandmother, Jeanne, for its unfaltering support.

Lausanne, 6 August 2016 R. L.

Abstract

Word embedding is a feature learning technique which aims at mapping words from a vocabulary into vectors of real numbers in a low-dimensional space. By leveraging large corpora of unlabeled text, such continuous space representations can be computed for capturing both syntactic and semantic information about words. Word embeddings, when used as the underlying input representation, have been shown to be a great asset for a large variety of natural language processing (NLP) tasks. Recent techniques to obtain such word embeddings are mostly based on neural network language models (NNLM). In such systems, the word vectors are randomly initialized and then trained to predict optimally the contexts in which the corresponding words tend to appear. Because words occurring in similar contexts have, in general, similar meanings, their resulting word embeddings are semantically close after training. However, such architectures might be challenging and time-consuming to train.

In this thesis, we are focusing on building simple models which are fast and efficient on large-scale datasets. As a result, we propose a model based on counts for computing word embeddings. A word co-occurrence probability matrix can easily be obtained by directly counting the context words surrounding the vocabulary words in a large corpus of texts. The computation can then be drastically simplified by performing a Hellinger PCA of this matrix. Besides being simple, fast and intuitive, this method has two other advantages over NNLM. It first provides a framework to infer unseen words or phrases. Secondly, all embedding dimensions can be obtained after a single Hellinger PCA, while a new training is required for each new size with NNLM. We evaluate our word embeddings on classical word tagging tasks and show that we reach similar performance than with neural network based word embeddings.

While many techniques exist for computing word embeddings, vector space models for phrases remain a challenge. Still based on the idea of proposing simple and practical tools for NLP, we introduce a novel model that jointly learns word embeddings and their summation. Sequences of words (i.e. phrases) with different sizes are thus embedded in the same semantic space by just averaging word embeddings. In contrast to previous methods which reported *a posteriori* some compositionality aspects by simple summation, we simultaneously train words to sum, while keeping the maximum information from the original vectors.

These word and phrase embeddings are then used in two different NLP tasks: document classification and sentence generation. Using such word embeddings as inputs, we show that good performance is achieved in sentiment classification of short and long text documents with a convolutional neural network. Finding good compact representations of text

documents is crucial in classification systems. Based on the summation of word embeddings, we introduce a method to represent documents in a low-dimensional semantic space. This simple operation, along with a clustering method, provides an efficient framework for adding semantic information to documents, which yields better results than classical approaches for classification. Simple models for sentence generation can also be designed by leveraging such phrase embeddings. We propose a phrase-based model for image captioning which achieves similar results than those obtained with more complex models. Not only word and phrase embeddings but also embeddings for non-textual elements can be helpful for sentence generation. We, therefore, explore to embed table elements for generating better sentences from structured data. We experiment this approach with a large-scale dataset of biographies, where biographical infoboxes were available. By parameterizing both words and fields as vectors (embeddings), we significantly outperform a classical model.

Key words: word embedding, natural language processing, PCA, artificial neural networks, language model, document classification, sentence generation

Résumé

Le *word embedding* est une méthode d'apprentissage automatique qui vise à représenter les mots d'un vocabulaire dans des vecteurs de réels dans un espace à faible dimension. En s'appuyant sur un grand corpus de textes non annoté, de telles représentations vectorielles peuvent être calculées pour capturer à la fois des informations syntaxiques et sémantiques sur mots. Ces word embeddings, lorsqu'ils sont ensuite utilisés comme données d'entrée, se sont révélés être un grand atout pour une grande variété de tâches en traitement automatique du langage naturel (TALN). Les techniques récentes pour obtenir ces représentations de mots sont principalement basées sur des modèles de langue neuronaux (MLN). Dans de tels systèmes, les vecteurs représentant les mots sont initialisés aléatoirement, puis entrainés à prédire de façon optimale les contextes dans lesquels ils apparaissent. Étant donné que les mots apparaissant dans des contextes similaires ont, en principe, des significations semblables, leurs représentations vectorielles sont sémantiquement proches après l'apprentissage. Cependant, de telles architectures sont généralement difficiles et longues à entrainer.

Dans cette thèse, nous nous concentrons sur la construction de modèles simples qui sont à la fois rapides et efficaces avec des ensembles de données à grande échelle. En conséquence, nous proposons un modèle basé sur le simple comptage de mots pour calculer les word embeddings. Une matrice de probabilité de cooccurrences peut être facilement obtenue en comptant directement, dans un grand corpus de textes, les mots de contexte entourant les mots du vocabulaire d'intérêt. L'obtention des word embeddings peut alors être considérablement simplifiée en effectuant une ACP de cette matrice, avec la distance de Hellinger. En plus d'être simple, rapide et intuitive, cette méthode présente deux autres avantages par rapport aux MLN. Tout d'abord, cette méthode permet l'inférence de nouveaux mots ou expressions (groupes de mots). Deuxièmement, toutes les dimensions de word embeddings peuvent être obtenues après une seule ACP, alors qu'un nouvel apprentissage est nécessaire pour chaque nouvelle taille d'embeddings avec les MLN. Nous évaluons ensuite nos représentations de mots sur des tâches classiques d'étiquetage de mots, et nous montrons que les performances sont similaires qu'avec des word embeddings obtenus par l'intermédiare de MLN.

Alors que de nombreuses techniques existent pour le calcul de word embeddings, la représentation vectorielle de groupe de mots reste encore un défi. Toujours dans l'idée de proposer des outils simples et pratiques pour le TALN, nous introduisons un modèle qui apprend conjointement les word embeddings et de leur sommation. Les séquences de mots de tailles différentes sont ainsi encodées dans le même espace sémantique, juste en moyennant les embeddings des mots. Contrairement aux méthodes précédentes qui ont observé *a posteriori*

des propriétés de compositionnalité par simple sommation, nous apprenons aux vecteurs de mots à s'aggréger, tout en gardant le maximum d'informations des vecteurs originaux.
Ces embeddings de mots et groupes de mots sont ensuite utilisés dans deux tâches de TALN différentes : la classification des documents et la génération automatique de phrases. En utilisant les word embeddings comme données d'entrée dans un réseau de neurones convolutifs, nous montrons que de bonnes performances sont obtenues pour la classification de sentiments dans des documents textuels, aussi bien longs que courts. Les systèmes classiques de classification doivent souvent faire face à des problème de dimensionalité, il est donc crucial de trouver des représentations de documents compactes. En se basant sur notre modèle de sommation des word embeddings, nous introduisons une méthode pour représenter les documents dans un espace sémantique de faible dimension. Cette opération de sommation, suivie d'une méthode de clustering, permet d'ajouter efficacement de l'information sémantique aux documents, et ainsi d'obtentir de meilleurs résultats que les approches classiques pour la classification. Des modèles simples pour la génération automatique de phrases peuvent également être conçus en tirant profit de ces embeddings de groupes de mots. Nous proposons ainsi un modèle pour le légendage automatique d'images qui obtient des résultats similaires que des modèles plus complexes à base de réseaux de neurones récurrents. En complément des embeddings de mots ou de groupes de mots, les éléments non textuels peuvent aussi être utiles pour la génération de texte. Nous proposons ainsi d'explorer l'encodage d'éléments issus de tableaux pour générer de meilleures phrases à partir de données structurées. Nous expérimentons cette approche sur une grande collection de biographies issues de Wikipedia, où des tableaux d'informations sont disponibles. En paramétrant les mots et les champs des tableaux comme vecteurs (embeddings), nous obtenons ainsi de meilleurs résultats qu'avec un modèle classique.

Mots clefs : word embedding, traitement automatique du langage naturel, ACP, réseau de neurones artificiels, modèle de langue, classification de documents, génération automatique de phrases

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BC	Bhattacharyya Coefficient
BLEU	Bilingual Evaluation Understudy
BOSC	Bag Of Semantic Concepts
BOW	Bag Of Words
BP	Brevity Penalty
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRF	Conditional Random Field
ED	Eigen Decomposition
GloVe	Global Vectors
GPU	Graphical Processing Unit
HLBL	Hierarchical Log-Bilinear
KN	Kneser-Ney
LDA	Latent Dirichlet Allocation
LM	Language Model
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
MAP	Maximum A Posteriori
MaxEnt	Maximum Entropy
ML	Maximum Likelihood
MLP	Multilayer Perceptron
NB	Naive Bayes
NCE	Noise-Contrastive Estimation
NER	Named Entity Recognition
NLP	Natural Language Processing
NNLM	Neural Network Language Model
NP	Noun Phrase
PCA	Principal Component Analysis
PMI	Pointwise Mutual Information
PP	Prepositional Phrase
RNN	Recurrent Neural Network
SG	Skip-Gram

SVD	Singular Value Decomposition
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
VP	Verbal Phrase

1 Introduction

1.1 Motivation

Natural language can be described as a system which "makes infinite use of finite means" (Humboldt, 1836), meaning that an infinite number of sentences can be generated from a finite set of words. One can understand a sentence never before heard or express a meaning never expressed before. This capacity to generate an infinite number of possible sentences results from some underlying mechanisms. Linguists have defined compositionality, hierarchy, and recursion as the universal features of human language. The *Principle of Compositionality* (Frege, 1892) emphasizes that the meaning of an expression is a function of the meanings of its parts and of the way they are syntactically combined. Sentences are *hierarchically structured*: words are grouped into phrases, which are grouped into higher-level phrases, and so on until they form a complete sentence. A logical form of a sentence can then be derived from its syntactic structure (Montague, 1974). *Recursion* theorizes that longer sentences can be constructed by inserting recursively, sentences within another sentence (Chomsky, 1957). Human language permits infinite recursion, at least in theory, which makes this feature a fundamental aspect of language for some linguists (Hauser et al., 2002). All those features are connected: recursion requires hierarchy and hierarchy requires compositionality. This big picture of linguistic theories tells us how it is crucial to capture the meaning of words along with their compositionality, the basis for understanding all sentences.

Natural language processing (NLP) aims at modeling the interactions between computers and human language, which involve many different tasks such as information extraction, document classification, natural language generation, machine translation, question answering, etc. To tackle this variety of tasks, one need to build "intelligent" systems, capable of understanding and analyzing human language. For such systems to work, it is, therefore, necessary to capture the meaning of words into vector spaces, along with modeling the compositionality, hierarchy and recursion of these vector spaces.

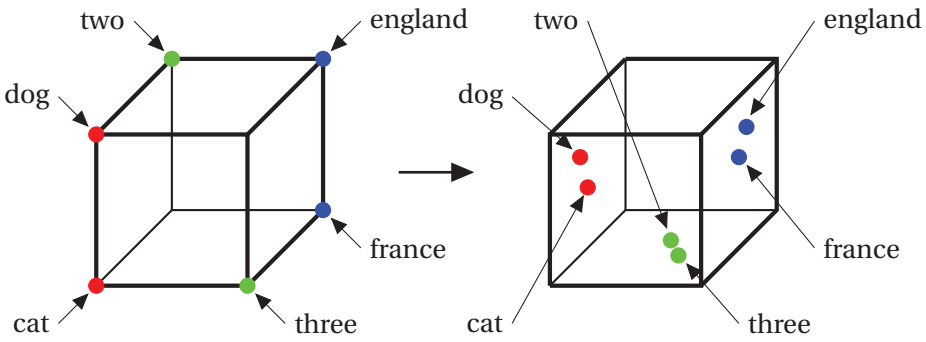


Figure 1.1 – Two different approaches for encoding words into vector spaces. One-hot encoding on the left-hand side. Word embedding on the right-hand side.

1.2 Objectives

Modeling natural language is a diverse and complex problem which involves many underlying mechanisms, as mentioned previously. This thesis focuses on bringing efficient and effective methods for computers to understand the foundations of human language: the meaning of words and their compositionality. The objectives are the following:

- **Capturing the meaning of words into dense vectors.** Compared with other fields of artificial intelligence where inputs are represented as vector spaces (images in computer vision, temporal signals in speech processing), machines are facing the challenge of getting words as inputs when dealing with natural language. A classical approach to address this issue is to define a $|\mathcal{W}|$-dimensional vector where all entries are set to 0 except for a single entry that identifies a word $w_t \in \mathcal{W}$ (a fixed vocabulary of words); this is called *one-hot encoding*. As illustrated in Figure 1.1, a one-hot vector is a poor way to encode the meaning of words, since a separate dimension represents each word. A better solution is rather to encode words into dense vectors where the dimensions capture syntactic and semantic properties about words, such that related words are close in this continuous vector space. Such representation in a much lower dimension space compared to the vocabulary $\mathcal{W}$ is called *word embedding*.
- **Composing words into phrases.** Following the *Principle of Compositionality*, we then want to define a function that combines word embeddings to get representations of phrases (i.e. sequences of words). Similar phrases are often composed of different number of words. As illustrated in Figure 1.2, such a composition function must combine word embeddings independently of their number to represent words and phrases into the same semantic space.

Having semantic representations of words and phrases, the objective is then to use them for solving NLP tasks. In this thesis, we leverage these embeddings for attacking two tasks in both *supervised* and *unsupervised* manners:

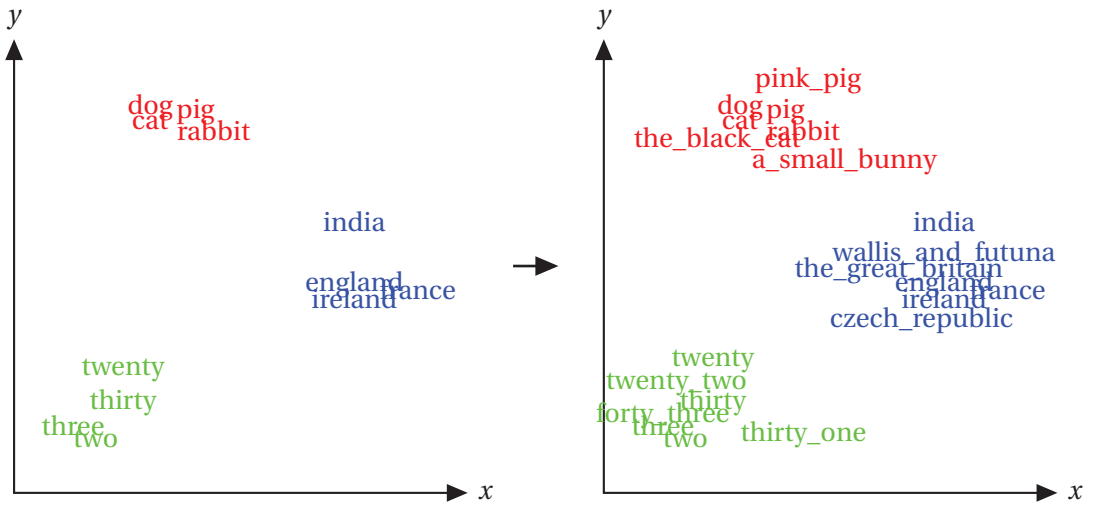


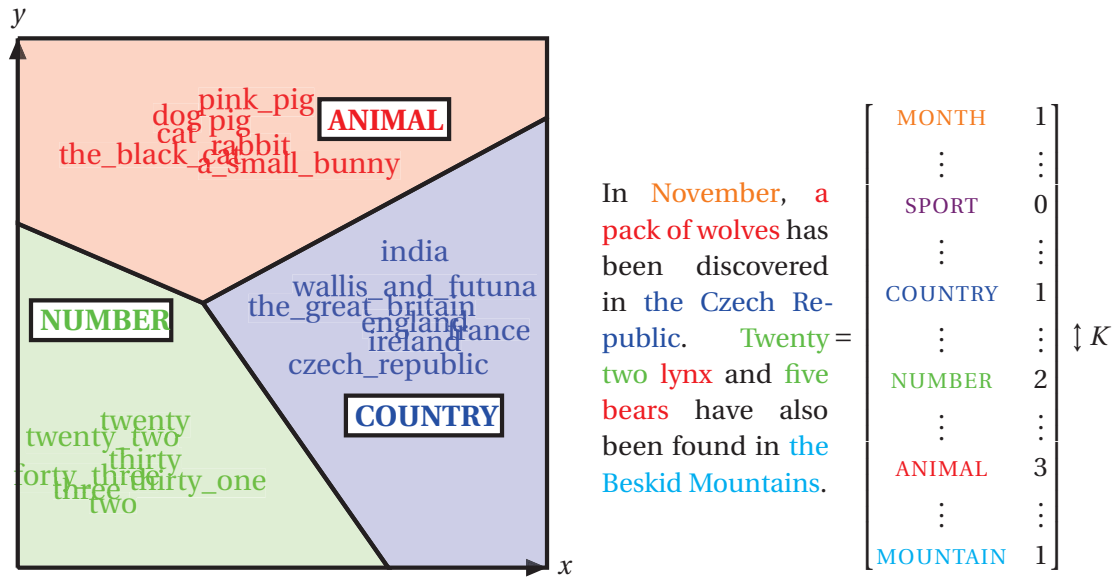
Figure 1.2 – From word to phrase embeddings.

- **Document classification.** Given a document, and assuming we are given a fixed size vocabulary $\mathcal{W}$ (possibly including million of words), a standard approach is to count word occurrences in the document. The $|\mathcal{W}|$-dimensional vector of counts is a classical representation (often called *bag-of-word* representation (BOW)), which is used as input for training standard machine learning classifiers. Consequently, such systems do not rely on the meaning of words for classifying documents, since the classifiers are trained to find the most discriminative *keywords*. In this thesis, we want to go beyond keyword-based models, investigating new ways to find *semantic document representations* in an unsupervised manner. By capturing the meaning of words and phrases in the same embedding space, expressions from the same semantic group are now close in this space. As illustrated in Figure 1.3, *semantic concepts* can be defined to represent a given document as a *bag of semantic concepts*. Such representation adds semantic information while reducing the number of features dramatically compared to a classical BOW representation.

  In supervised learning, *neural network* models are established as powerful models for classification problems, like object recognition in computer vision, or acoustic modeling in speech recognition. Despite their success, they have not yet been extensively applied for text classification. Document representations are traditionally high dimensional and sparse feature vectors, which do not fit well with neural networks. By encoding all words in a given document with embeddings, we obtain a real-value representation which enables the use of neural network models for text classification. Furthermore, word embeddings carry semantic information that can help to attack more complex text classification problems.

- **Sentence generation.** While document classification is about understanding natural language (i.e. reading) to provide the right prediction, sentence generation is about analyzing computer-based representation (e.g. images or tables) with the goal of con-

verting it into natural language (i.e. writing). Traditional methods are based on a series of processes that makes decisions for transforming concepts (e.g. objects in images, or values in tables) into words or expressions, which are then combined to form sentences, according to rules of syntax, morphology and orthography. Consequently, such approaches generate text using cut-and-paste actions without understanding the meaning of what they produce. In this thesis, we want to investigate whether vector-based representations of words and expressions can be used for generating sentences. We want to leverage our semantically rich representations to propose more efficient and effective models for natural language generation, by bypassing the need for hand-crafted rules.



(a) Semantic concepts from phrase embeddings.

(b) Bag of semantic concepts.

Figure 1.3 – Defining semantic concepts from phrase embeddings for representating text document as a bag of semantic concepts. Best viewed in color.

1.3 Thesis Contributions

The first objective of this thesis is to encode words into real-value vectors:

- **Word embeddings through Hellinger PCA.** There has been a lot of effort to capture the meaning of words into vector space models, often called *word embeddings*. Popular approaches such as Brown clustering algorithm (Brown et al., 1992) have been used with success in a wide variety of NLP tasks. Recently, approaches based on neural network language models (NNLM) have revived the field of learning word embeddings. However, a neural network architecture can be hard to train. Finding the right hyper-parameters to tune the model is a challenging task and the training phase is, in general, computationally expensive. This thesis aims to show that word embeddings can be obtained

using simple (linear) operations. Linguists assumed long ago that words occurring in similar contexts tend to have similar meanings (Harris, 1954; Firth, 1957). We, therefore, propose to compute word embeddings using the word co-occurrence statistics and a well-known dimensionality reduction operation such as Principal Component Analysis (PCA), along with an appropriate metric (i.e. the *Hellinger* distance). An evaluation on classical NLP tasks shows that such a simple spectral method can generate word embeddings as good as with neural network architectures. Besides being simple, this method is also very fast and provides a framework to infer new words or phrases.

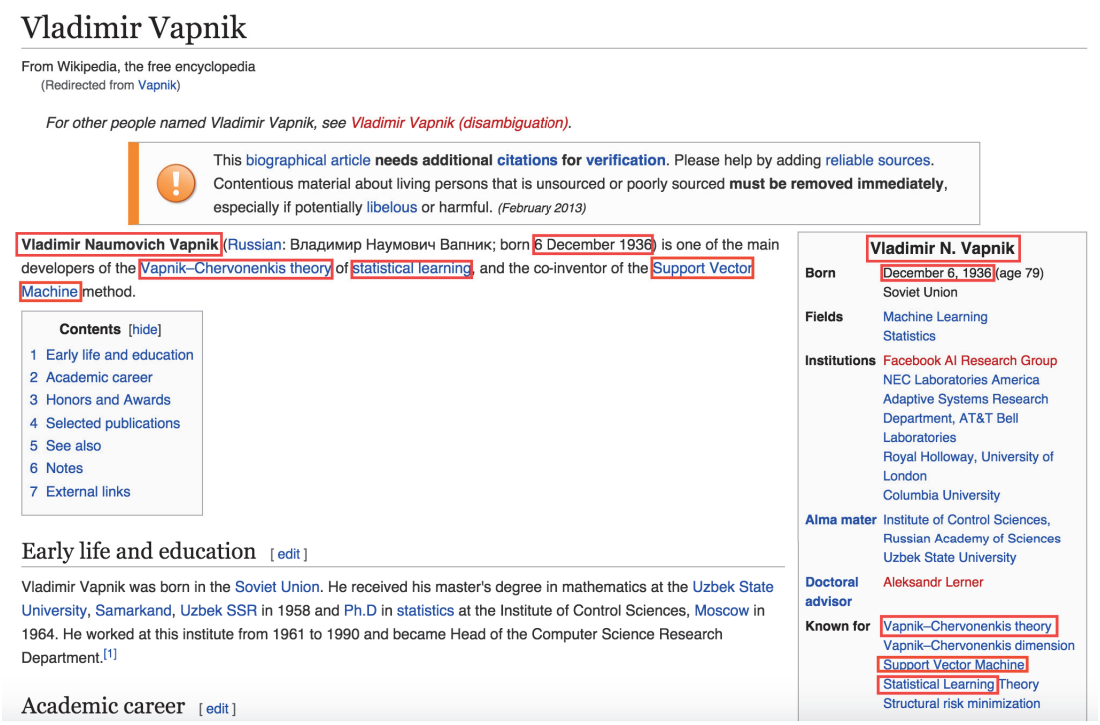
This work has been first presented at a deep learning workshop (Lebret and Collobert, 2013), before being published in an NLP conference (Lebret and Collobert, 2014).

This first contribution enables the encoding of words into continuous low-dimensional vectors. We leverage these word embeddings for solving document classification and sentence generation in the two following contributions:

- **Convolutional neural networks for sentiment classification.** Neural network architectures have shown their potential in several supervised NLP tasks, by using word embeddings. Good performance is achieved on tasks where the syntactic aspect is dominant such as part-of-speech tagging, chunking and named entity recognition (Turian et al., 2010; Collobert et al., 2011). Such system can also be a good alternative to classical approaches for supervised tasks where the semantic aspect predominates, such as in sentiment classification. Inspired by its success in computer vision, we propose a convolutional neural network (CNN) model for predicting sentiments in text documents. Each word in a given document is first mapped into a pre-trained embedding, thanks to an embedding layer. Then, a convolutional layer locally extracts sentiments from sequences of words. Finally, a max layer pools the best local sentiments into a global document representation which is used to train a linear classifier. As word embeddings are usually generated over large corpora of unlabeled data, words are represented in a generic manner. Because the system is trained in an end-to-end manner, embedding specialization is easily done which yields good performance in sentiment classification of short (tweets) and long (movie reviews) documents.

  This work has been published in Lebret and Collobert (2014) for long document classification. Experiments on short documents have been reported in Lebret et al. (2016b).

- **Generating text from structured data.** Unsupervised learning methods for NLP can also be developed by leveraging systems that rely on embeddings. In this thesis, we propose to explore the use of NNLM for the concept-to-text generation, which addresses the problem of rendering structured records into natural language (Reiter et al., 2000). In contrast to previous work, we scale to the large and very diverse problem of generating biographies for personalities based on Wikipedia infoboxes (i.e. a fact table describing a person). As illustrated in the example in Figure 1.4, there exists a large overlap of sequences of words between an article and its infobox. To tackle this problem we thus



Vladimir Vapnik

From Wikipedia, the free encyclopedia
(Redirected from Vapnik)

For other people named Vladimir Vapnik, see Vladimir Vapnik (disambiguation).

This biographical article **needs additional citations for verification**. Please help by adding reliable sources. Contentious material about living persons that is unsourced or poorly sourced **must be removed immediately**, especially if potentially libelous or harmful. *(February 2013)*

Vladimir Naumovich Vapnik (Russian: Владимир Наумович Вапник; born 6 December 1936) is one of the main developers of the Vapnik–Chervonenkis theory of statistical learning, and the co-inventor of the Support Vector Machine method.

Contents [hide]
1 Early life and education
2 Academic career
3 Honors and Awards
4 Selected publications
5 See also
6 Notes
7 External links

Early life and education [edit]

Vladimir Vapnik was born in the Soviet Union. He received his master's degree in mathematics at the Uzbek State University, Samarkand, Uzbek SSR in 1958 and Ph.D in statistics at the Institute of Control Sciences, Moscow in 1964. He worked at this institute from 1961 to 1990 and became Head of the Computer Science Research Department.[1]

Academic career [edit]

Vladimir N. Vapnik

Born	December 6, 1936 (age 79) Soviet Union
Fields	Machine Learning Statistics
Institutions	Facebook AI Research Group NEC Laboratories America Adaptive Systems Research Department, AT&T Bell Laboratories Royal Holloway, University of London Columbia University
Alma mater	Institute of Control Sciences, Russian Academy of Sciences Uzbek State University
Doctoral advisor	Aleksandr Lerner
Known for	Vapnik–Chervonenkis theory Vapnik–Chervonenkis dimension Support Vector Machine Statistical Learning Theory Structural risk minimization

Figure 1.4 – Screenshot of the English Wikipedia page about Vladimir Vapnik. Red rectangles highlight the overlap of sequences of words between the infobox and the introduction section.

introduce a statistical generation model conditioned on a Wikipedia infobox. We focus on the generation of the first sentence of a biography which requires the model to select among a large number of possible fields to generate an adequate output. Such diversity makes it difficult for classical count-based models to estimate probabilities of rare events due to data sparsity. In our case, we can address this issue by parameterizing words and fields as embeddings, along with a neural language model operating on these embeddings (Bengio et al., 2003). This factorization allows us to scale to a large number of words and fields compared to classical approaches where the number of parameters grows as the product of the number of words and fields. As with our CNN model for sentiment classification, our model exploits structured data both globally and locally. Global conditioning summarizes all information about a personality to understand high-level themes such as that the biography is about a scientist or an artist while local conditioning describes the previously generated tokens regarding their relationship to the infobox. Our evaluation on biography generation shows that our model can generate a wide variety of natural sentences, with high overlap (BLEU) compared to human reference sentences.

This work has been published in Lebret et al. (2016a).

Given that we built representations of words in a vector space, the *Principle of Compositionality* can be addressed by combining word embeddings to get representations of phrases or

sentences:

- **Composing word embeddings.** Given pre-computed word embeddings, some compositional models involve vector addition or multiplication (Mitchell and Lapata, 2010). Others use the syntactic relations between words to treat certain words as functions and other as arguments such as adjective-noun composition (Baroni and Zamparelli, 2010) or noun-verb composition (Grefenstette et al., 2013). Still based on the idea of proposing efficient and effective methods for NLP, we propose to learn word embeddings that can be averaged together to quickly compute representations of phrases in the same embedding space, with no addition of memory. To get representations in a low-dimensional space, we will reproduce the Hellinger PCA of the word co-occurrence matrix using an autoencoder network. In Mikolov et al. (2013b), the authors train word embeddings with linear models and show that some nice properties can be found using an additive compositionality, meaning that such representations exhibit linear structures. For combining word representations in a common semantic space, a compositional model based on vector addition is chosen due to its simplicity and its inherent structure. The autoencoder architecture enables this compositional function to be trained simultaneously with the word vector representations.

  This work has been published in Lebret and Collobert (2015b) which introduces the learning of word embeddings via autoencoders. The joint learning with summation has been published in Lebret and Collobert (2015c).

Defining a common semantic space for n-grams (i.e. sequences of n words) can be useful for many NLP tasks. We leverage these new embeddings for solving our two tasks of interest:

- **Bag of semantic concepts for document classification.** *Bag-of-words* (BOW) based models have long been state-of-the-art methods to classify documents. These techniques count words (1-grams) within documents. Classifiers are then trained to identify the most discriminative words for classifying documents. Documents sharing those discriminative words are considered similar. Such models work very well in practice as certain topic keywords are indicative enough for classification. However, having n-gram representations could help to increase classification performance for certain types of documents. For instance, it has been shown that classification accuracy for movie reviews can be improved by using 2-grams (Wang and Manning, 2012). This outcome is not really surprising, since a document that contains 2-grams such as *not good* or *not funny* would probably be classified as positive with a BOW model. By leveraging the ability of our word embeddings to compose, we propose instead a bag of semantic concepts model, where semantic concepts are defined with a K-means clustering of all n-gram representations. Using the same classifiers as with BOW-based models, we show that such model yields good results in the classification of movie reviews and news.

  This work has been published in Lebret and Collobert (2015a).

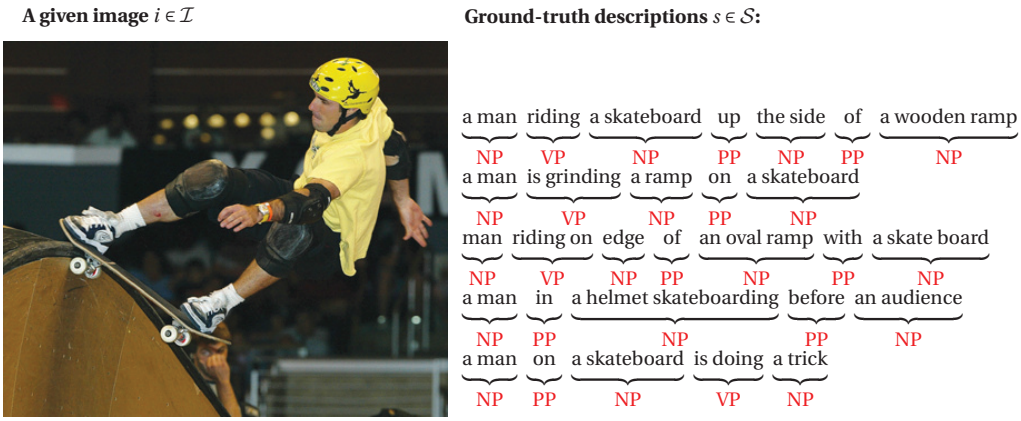


Figure 1.5 – Chunking analysis of the ground-truth descriptions of a training image from MS COCO dataset. NP, VP and PP mean, respectively, noun phrase, verbal phrase and prepositional phrase.

- **Phrase-based model for image captioning.** Composing word embeddings is also valuable in the context of sentence generation. As illustrated in Figure 1.5, an exploratory analysis of image captions reveals that their syntax is quite simple. The ground-truth descriptions can be represented as a collection of noun, verbal and prepositional phrases. The different entities (e.g. *a man*, *a skateboard*, *a wooden ramp*) in a given image are described by the noun phrases while the interactions or events between them are explained with verbal or prepositional phrases. For automatically generating image captions, we thus propose to train a model that predicts the set of phrases present in the sentences used to describe the images. By leveraging our embedding model, phrase vector representations are obtained by aggregating the word embeddings that compose the phrases. Image vector representations can also be easily obtained from some pre-trained convolutional neural networks, leveraging previous work in computer vision. We then introduce a model that learns a common embedding space between these phrase and image representations. By introducing a constrained language model based on our prior knowledge about the caption syntax, we generate syntactically correct sentences from the subset of phrases that best describe a given test image. Evaluation on two datasets using the BLEU metric shows that our generated sentences achieve similar performance as humans and other more complex models.

  This work has been published in Lebret et al. (2015b), and extended in Lebret et al. (2015a).

1.4 Thesis Outline

The rest of the thesis is structured as follows:

- **Chapter 2, Background.** We first introduce the standard feed-forward neural network, with the specific layers for NLP. We then describe the standard machine learning ap-

proaches for text document classification. Finally, we describe language modeling which is at the core of sentence generation.

- **Chapter 3, State-of-the-art Word Representations.** We provide information about other approaches for computing word embeddings. We start with a clustering method; we then describe the main methods based on neural network language models.

- **Chapter 4, Word Embeddings through Hellinger PCA.** This chapter introduces our proposed model for getting word embeddings. We conduct an exploratory analysis of the parameters needed to build the word co-occurrence matrix, relying on benchmark datasets of word similarities and word analogies. We then describe the different techniques for performing the Hellinger PCA. Finally, we compare the obtained embeddings with other state-of-the-art methods on two classical NLP tasks: chunking and named entity recognition.

- **Chapter 5, Towards Phrase Embeddings.** We introduce a model that jointly learns word embeddings and a function for combining them to compute phrase embeddings. We also describe a new dataset extracted from Wikipedia, which contains noun phrases and verbal phrases used to train and test the embeddings.

- **Chapter 6, Sentiment Classification with Convolutional Neural Network.** This chapter describes a convolutional neural network for text document classification. We then evaluate the proposed model in sentiment classification of short (tweets) and long (movie reviews) documents.

- **Chapter 7, N-gram-Based Model for Compact Document Representation.** This chapter gives an alternative to classical BOW models for document representations, leveraging our phrase embeddings. It introduces the *bag of semantic concepts* model which adds semantic information coming from n-grams, while offering compact document representations. We evaluate the proposed model on sentiment classification of a small and a large dataset of movie reviews, as well as text news classification.

- **Chapter 8, Phrase-based Image Captioning.** In this chapter, we propose a phrase-based model for image captioning. Based on the analysis of captions from two large datasets, we propose to decompose the sentences describing the images into noun, verbal and prepositional phrases. Thanks to our embedding model described in Chapter 5, we introduce a bilinear model which learns a common semantic space between phrases and images. We then use our prior knowledge about the caption syntax to define a constrained language model which produces sentences from a given set of top-ranked phrases.

- **Chapter 9, Generating Text from Structured Data.** This chapter presents a NNLM approach for generating text from Wikipedia infoboxes. We introduce a statistical language model with both local and global conditioning, as well as its neural network version. We also describe the dataset of biographies extracted from Wikipedia, which is used to train and test this model.

- **Chapter 10, Conclusion.** This last chapter concludes this thesis and proposes directions for further research in NLP using word and phrase embeddings.

2 Background

This thesis mainly focuses on machine learning techniques based on neural networks for tackling two tasks in NLP: document classification and sentence generation. In this chapter, we first introduce neural network models with a focus on how they deal with natural language. We then describe the classical approaches for text document classification. Sentence generation is often possible thanks to language modeling, which is detailed in the last section, going from statistical language models to neural network language models.

2.1 Neural Network Models for Natural Language Processing

Machine learning algorithms in NLP are traditionally based on linear models trained over very high dimensional yet very sparse feature vectors. Inspired by the success of neural networks in computer vision, nonlinear neural networks models have recently emerged for tackling NLP problems. It, therefore, requires a paradigm shift in the way the models are dealing with the features. Fully connected feed-forward neural networks learn over dense inputs, which need to switch from classical sparse feature vectors to continuous word vector features, aka *word embeddings*.

2.1.1 Mathematical Notations

We consider a neural network $\phi_\theta(\cdot)$, with parameters θ. Any feed-forward neural network with L layers can be seen as a composition of functions $\phi^l_\theta(\cdot)$, corresponding to each layer l:

$$\phi_\theta(\cdot) = \phi^L_\theta(\phi^{L-1}_\theta(\ldots\phi^1_\theta(\cdot)\ldots))\,. \tag{2.1}$$

For NLP, the first layer is the *embedding layer*, the final layer is the *output layer* and the in-between layer are called the *hidden layers*.

In the thesis, bold upper case letters represent matrices ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$), and bold lower-case letters represent vectors ($\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$). $\mathbf{A}_i$ represents the i^{th} row of matrix $\mathbf{A}$, $[\mathbf{A}]_{i,j}$ is the scalar at row

i and column j. For a vector $\mathbf{v}$, we denote $[\mathbf{v}]_i$ the scalar at index i in the vector. Unless otherwise stated, vectors are assumed to be column vectors. We use $[\mathbf{v}_1;\mathbf{v}_2]$ to denote vector concatenation.

2.1.2 Feed-forward Neural Networks

Single-layer perceptron

The simplest neural network is the single-layer perceptron, which is a linear function of its inputs $\mathbf{x} \in \mathbb{R}^{d^{\text{in}}}$:

$$\phi_\theta = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.2}$$

where $\theta = \{\mathbf{W}, \mathbf{b}\}$ are the trainable parameters with $\mathbf{W} \in \mathbb{R}^{d^{\text{out}} \times d^{\text{in}}}$ the weight matrix, and $\mathbf{b} \in \mathbb{R}^{d^{\text{out}}}$ a bias term.

Multi-layer perceptron

Nonlinear problems are addressed by introducing a nonlinear hidden layer l:

$$\phi^l_\theta = \mathbf{W}^l h(\phi^{l-1}_\theta) + \mathbf{b}^l \tag{2.3}$$

where $h(\cdot)$ is a nonlinear function that is applied element-wise to the previous layer output $\phi^{l-1}_\theta \in \mathbb{R}^{d^{l-1}}$ (also called a *nonlinearity* or an *activation function*). $\mathbf{W}^l \in \mathbb{R}^{d^l \times d^{l-1}}$ and $\mathbf{b}^l \in \mathbb{R}^{d^l}$ are the weight matrix and the bias term of the next linear transformation. Networks with more than one hidden layer are usually named *deep* neural networks.

Output layer

A neural network outputs a d^{out} dimensional vector. When $d^{\text{out}} = 1$, such a network outputs only a scalar and can be used for regression (the output corresponds to a score), or for binary classification (the class depends on the sign of the output). For a k-class classification problem, it outputs a $d^{\text{out}} = k > 1$ dimensional vector where each dimension is associated with a class (the maximal value corresponds to the predicted class). The output can also be interpreted as a distribution over class assignments (all entries are positive and sum to one), when using a softmax transformation of the output layer (see Section 2.3.3 for more details).

Common nonlinearities

Several nonlinear function $h(\cdot)$ exist. The choice is purely an empirical question.

Sigmoid The sigmoid activation function $\sigma(\cdot)$ transforms each value x into the range $[0,1]$.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.4}$$

Hyperbolic tangent (Tanh) The hyperbolic tangent transforms each value x into the range $[-1,1]$.

$$\text{Tanh}(x) = \frac{e^{2x-1}}{e^{2x+1}} \tag{2.5}$$

Hard-Tanh It is a "hard" version of the hyperbolic tangent. It has the advantage of being slightly cheaper to compute, while leaving the generalization performance unchanged (Collobert, 2004).

$$\text{HardTanh(x)} = \begin{cases} -1 & \text{if } x < 1 \\ x & \text{if } -1 \leq x < 1 \\ 1 & \text{if } x > 1 \end{cases} \tag{2.6}$$

2.1.3 Transforming Words into Feature Vectors

When dealing with natural language, the input **x** encodes features such as words, part-of-speech tags or other linguistic information. Conventionally, supervised lexicalized NLP approaches take a word and convert it to an index, which is then transformed into a feature vector **f** using a *one-hot* representation. We consider a fixed-sized word vocabulary $\mathcal{W}$. Given a word $w_t \in \mathcal{W}$, $\mathbf{f}_t$ is a $|\mathcal{W}|$-dimensional vector where all entries are zeros except at the t^{th} feature, in which the value is 1:

$$\mathbf{f}_t = \Big[\underbrace{0 \ , \ 0 \ , \ 0 \ , \ \dots \ , \ 1}_{\text{at index } t} \ , \ \dots \ , \ 0 \ , \ 0 \ , \ 0 \Big]^T \in \mathbb{R}^{|\mathcal{W}|}. \tag{2.7}$$

The core of neural network based models is to stop representing each feature as one-hot representations and representing them instead as dense vectors, the so-called *word embeddings*.

Embedding layer

A layer is thus defined to transform the one-hot representations into word embeddings. Given a sequence of T words $\{w_1, w_2, \dots, w_T\}$, each word $w_t \in \mathcal{W}$ is embedded into a d^{wrd}-dimensional vector space, by applying the following operation:

$$\phi_\theta(w_t) = \mathbf{E}\mathbf{f}_t \tag{2.8}$$

where the matrix $\mathbf{E} \in \mathbb{R}^{d^{\text{wrd}} \times |\mathcal{W}|}$ represents all the word embeddings to be learned in this layer, just as the other parameters of the network. In practice, we use a lookup table to replace this computation with a simpler array indexing operation where $\mathbf{E}_{w_t} \in \mathbb{R}^{d^{\text{wrd}}}$ corresponds to the embedding of the word w_t. This lookup table operation is then applied for each word in the sequence. A common approach is to concatenate all resulting word embeddings:

$$\phi_\theta(w_1, w_2, \ldots, w_T) = \left[\mathbf{E}_{w_1}; \mathbf{E}_{w_2}; \ldots; \mathbf{E}_{w_T}\right] \in \mathbb{R}^{(d^{\text{wrd}} \times T)}. \tag{2.9}$$

This vector can then be fed to further neural network layers.

2.2 Document Classification

In document classification, we are given a description $d \in \mathcal{D}$ of a document, where $\mathcal{D}$ is the document space; and a fixed set of classes $\mathcal{Y} = \{y_1, y_2, \ldots, y_m\}$. Typically, the document space $\mathcal{D}$ is represented as a high-dimensional space, the so-called *bag-of-words* model. Such representations are then used to learn classification models or topic models.

2.2.1 Bag-of-words Model

With a bag-of-words model, a document is represented as the bag of its words. Documents are first pre-processed for avoiding unnecessarily large feature vectors. Usual pre-processing methods involve the filtering of meaningless words (e.g. it, a, the, that), reducing the number of distinct words by only keeping word stems, and performing lowercase conversion. By defining this *word feature* list $\mathcal{W}$ to consider, a document $d \in \mathcal{D}$ is represented as a $|\mathcal{W}|$-entry vector $\mathbf{v}_d$ where each dimension corresponds to a separate word:

$$\mathbf{v}_d = \left[w_1, w_2, \ldots, w_{|\mathcal{W}|}\right]^T. \tag{2.10}$$

Each word $w_i \in \mathcal{W}$ can then transformed into a real value feature, in order to train classifiers such as MaxEnt or SVM. The simplest approach is to use binary values (appearing or not in the document):

$$w_i = \begin{cases} 0 & \text{if } w_i \notin d \\ 1 & \text{if } w_i \in d \end{cases}. \tag{2.11}$$

Another possibility is to use the frequency of the word w_i in d, called term frequency (tf_{w_i}). Some term weightings have also been defined to reflect how important a word is for a document. A popular method is the term frequency-inverse document frequency (TF-IDF) model:

$$w_i = \text{tf}_{w_i} \times \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} | w_i \in d\}|}, \tag{2.12}$$

where

- tf_{w_i} is term frequency of word w_i in document d (a local parameter),
- $\log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} | w_i \in d\}|}$ is inverse document frequency (a global parameter). $|\mathcal{D}|$ is the total number of documents in the document set; $\left|\{d \in \mathcal{D} | w_i \in d\}\right|$ is the number of documents containing the word w_i.

2.2.2 Topic Models

In topic modeling, the objective is to discover the latent *topics* that occur in a collection of documents. Using such techniques, documents are represented in much lower dimensional spaces than with bag-of-words models while modeling the hidden semantic structures. These representations are usually ideal for document indexing, but they can also be used as an alternative to bag-of-words models for document classification. Topic models are either based on probabilitic models (e.g. *Latent Dirichlet Allocation*), or can be derived from distributional semantic modeling (e.g. *Latent Semantic Analysis*). Such latent variable models are approaches to *unsupervised learning*.

Latent Dirichlet Allocation

The idea behind Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is to model documents as a mixture of various topics, where a topic is defined to be a distribution over a fixed vocabulary of words, with a Dirichlet prior. Assuming that the collection of documents contains K topics, each document exhibits these topics with different proportions. More formally, LDA defines a hidden variable model of documents. The observed data are the words of each document and the hidden variables represent the latent topical structure, i.e., the topics themselves and how each document exhibits them. The interaction between the observed documents and hidden topic structure emerges from the probabilistic generative process associated with LDA, the imaginary random process that is assumed to have produced the observed data.

Latent Semantic Analysis

Latent Semantic Analysis (LSA) (Landauer and Dumais, 1997) measures occurrence frequency of words in documents to discover a set of *hidden* topics (i.e. concepts). LSA write frequencies as a term-document matrix; it is a sparse matrix whose rows correspond to terms and whose columns correspond to documents. Each entry of the matrix is then transformed using a weighting scheme, such as TF-IDF. A mathematical technique called *singular value decomposition* (SVD) is then used to reduce the number of rows while preserving the similarity structure among columns. After selecting the K largest singular values, each document is then expressed as a K-dimensional vector, where each entry gives the degree of participation of the document in the corresponding topic.

2.2.3 Classification Models

Given a training set of m hand-labeled documents $(d_1, c_1), \ldots, (d_m, y_m)$, we wish to learn a classifier γ that maps documents to classes:

$$\gamma : \mathcal{D} \mapsto \mathcal{Y}. \tag{2.13}$$

This type of learning is called *supervised learning*. Successful classifiers for text classification are Naive Bayes (NB), Maximum Entropy (MaxEnt), and Support Vector Machine (SVM).

Naive Bayes Classifier

Naive Bayes is a probabilistic learning method which has shown to work well on text classification (Manning and Schütze, 1999). The probability of a document d being in class y is computed as:

$$P(y|d) \propto P(y) \prod_{w_i \in d} P(w_i|y) \tag{2.14}$$

where $P(w_i|y)$ is the conditional probability of word w_i occurring in a document of class y. $P(y)$ is the prior probability of a document occurring in class y. Bayesian classifiers attempt to build a probabilistic classifier based on modeling the underlying word features in different classes. The classification of text documents is then based on the posterior probability of the documents belonging to the different classes on the basis of the word presence in the documents. Given a new unlabeled document d, the objective is to find the best class for this document, which is the most likely or *maximum a posteriori* (MAP) class y_{MAP}:

$$y_{\text{MAP}} = \operatorname*{argmax}_{y \in \mathcal{Y}} \tilde{P}(y|d) = \operatorname*{argmax}_{y \in \mathcal{Y}} \tilde{P}(y) \prod_{w_i \in d} \tilde{P}(w_i|y) \tag{2.15}$$

where the probabilities $\tilde{P}$ are estimated from the training set.

Maximum Entropy Classifier

The idea behind Maximum Entropy (MaxEnt) models is that one should prefer the most uniform models that satisfy a given constraint (Nigam, 1999). Unlike the Naive Bayes classifier, the MaxEnt classifier does not assume that the features are conditionally independent of each other. Additional features such as bigrams and phrases can thus be included in the model without worrying about features overlapping.

We define a feature $f_i(d, y)$ which is any real-valued function between the document d and the class y. The expected value of feature f_i with respect to the empirical distribution $\tilde{P}(d, y)$

is:

$$\sum_{d\in\mathcal{D}}\sum_{y\in\mathcal{Y}}\tilde{P}(d,y)f_i(d,y). \tag{2.16}$$

The probability $\tilde{P}(d)$ of the document d to be chosen from the training data is:

$$\tilde{P}(d)=\frac{1}{|\mathcal{D}|}. \tag{2.17}$$

Because each document from the training data has been labeled, it is clear that:

$$\tilde{P}(y|d)=\begin{cases}0 & \text{if } y\neq y(d)\\ 1 & \text{if } y=y(d)\end{cases}, \tag{2.18}$$

where $y(d)\in\mathcal{Y}$ denotes the class which is assigned to d by the expert. The expected value of feature f_i in Equation 2.16 can be rewritten as follows:

$$\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}f_i\big(d,y(d)\big) \tag{2.19}$$

The expected value of feature f_i with respect to the model $P(y|d)$ is equal to:

$$\sum_{d\in\mathcal{D}}\sum_{y\in\mathcal{Y}}\tilde{P}(d)P(y|d)f_i(d,y). \tag{2.20}$$

Maximum entropy allows us to constrain the expected value to be the equal to the empirical value:

$$\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}f_i\big(d,y(d)\big)=\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}\sum_{y\in\mathcal{Y}}P(y|d)f_i(d,y). \tag{2.21}$$

Then, it can be shown that if we find the $\{\lambda_1,\ldots,\lambda_n\}$ parameters which maximize the dual problem, the probability given a document d to be classified as y is equal to:

$$P(y|d)=\frac{\exp\big(\sum_i\lambda_i f_i(d,y)\big)}{\sum_{y\in\mathcal{Y}}\exp\big(\sum_i\lambda_i f_i(d,y)\big)} \tag{2.22}$$

To classify a new document (given that the lambda parameters have been estimated), a MAP decision rule is used to select the category with the highest probability.

Support Vector Machines

Support Vector Machines (SVM) are a learning tool for solving two-class pattern recognition problem (Cortes and Vapnik, 1995). Given a set of training documents $(\mathbf{v}_1,y_1),\ldots,(\mathbf{v}_m,y_m)$ with $\mathbf{v}_i\in\mathbb{R}^{|\mathcal{W}|}$ and $y_i\in\{-1;1\}$, we search for a separating hyper-plane that divides positive and

negative samples from each other with maximal margin. A SVM is trained via the following optimization problem:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i\,, \tag{2.23}$$

with constraints

$$y_i(\mathbf{v}_i \cdot \mathbf{w} + b) \geq 1 - \xi_i \quad \xi_i \geq 0, \quad \forall i\,, \tag{2.24}$$

where weight vector $\mathbf{w}$ and constant b are learned, and C is a penalty parameter. Joachims (1998) sums up why SVM are appropriate for learning text classifiers:

- *High dimensional input space*: SVM use over-fitting protection.
- *Few irrelevant features*: SVM do not need an aggressive feature selection.
- *Document vectors are sparse*: each document vector $\mathbf{v}_d$ contains only few entries which are not zero.
- *Most text categorization problems are linearly separable.*

There are nonlinear extensions to the SVM, but the linear kernel usually outperforms nonlinear kernels in text classification (Yang and Liu, 1999).

In the multiclass classification framework, the standard approach is to reduce the single multiclass problem into multiple binary classification problems. Two strategies can be envisaged for building such binary classifiers:

1. *one-versus-all* which distinguishes between one of the classes and the rest,
2. *one-versus-one* which distinguishes between every pair of classes.

2.2.4 Evaluation Metrics

For classification, the terms *true positives* (TP), *true negatives* (TN), *false positives* (FP), and *false negatives* (FN) compare the results of the classifier under test with trusted external judgment. The terms positive and negative refer to the classifier's prediction, while true and false refer to the external judgment's prediction.

Accuracy

The accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{2.25}$$

Precision

Precision measures the exactness of a classifier. A higher precision means less false positives, while a lower precision means more false positives. This is often at odds with recall, as an easy way to improve precision is to decrease recall.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.26}$$

Recall

Recall measures the completeness, or *sensitivity*, of a classifier. Higher recall means less false negatives, while lower recall means more false negatives. Improving recall can often decrease precision because it gets increasingly harder to be precise as the sample space increases.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.27}$$

F-1 score

Precision and recall can be combined to produce a single metric known as the F-1 score. It can be interpreted as a weighted average of the precision and recall. The F-1 score is the harmonic mean of precision and recall:

$$\text{F}_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.28}$$

2.3 Language Models

Language modeling (LM) is a central task in NLP for problems involving natural language generation. Given a sentence $s = w_1, \ldots, w_T$ composed of T words from a vocabulary $\mathcal{W}$, a

language model estimates:

$$P(s) = \prod_{t=1}^{T} P(w_t | w_1, \ldots, w_{t-1}) . \tag{2.29}$$

2.3.1 N-gram Models

Let $c_t = w_{t-(n-1)}, \ldots, w_{t-1}$ be the sequence of $n-1$ context words preceding $w_t \in s$. In an n-gram language model, Equation 9.1 is approximated as

$$P(s) \approx \prod_{t=1}^{T} P(w_t | c_t) , \tag{2.30}$$

assuming an order n Markov property. Typically, n is taken to be two or three, corresponding to a bigram or trigram model, respectively. The probabilities $P(w_t|c_t)$ are estimated over a large corpus of text (called *training data*),

$$P_{ML}(w_t|c_t) = \frac{P(c_t w_t)}{P(c_t)} \tag{2.31}$$

$$= \frac{n(c_t w_t)}{\sum_{w_t} n(c_t w_t)} , \tag{2.32}$$

where $n(\alpha)$ denotes the number of times the string α occurs in the text. This is called the *maximum likelihood* (ML) estimates for $P(w_t|c_t)$. Although the maximum likelihood estimation is intuitive, it becomes poor with a small training data (Chen and Goodman, 1996).

2.3.2 Smoothing Models

When the amount of training data is insufficient, there are many events α such that $n(\alpha) = 0$ in Equation 2.31. The ML estimate is therefore $P_{ML}(w_t|c_t) = 0$. Even if an event has never been observed in training data, it does not mean it will never occur at inference time. Smoothing methods have thus been introduced to deal with data sparsity.

Add-one smoothing

Also called Laplace smoothing (Laplace, 1820), this simple smoothing technique just adds one to all the counts:

$$P_{+1}(w_t|c_t) = \frac{n(c_t w_t) + 1}{n(c_t) + |\mathcal{W}|} \tag{2.33}$$

It prevents zero probabilities, but tends to reassign too much mass to unseen events.

Good-Turing estimation

The Good-Turing estimate (Good, 1953) is central to many smoothing techniques. The idea is to re-estimate the amount of probability mass of n-grams with zero or low counts by looking at the number of n-grams with higher counts. Let n_r be the number of n-grams that occur r times, an adjusted count r^* is computed as follows:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}. \tag{2.34}$$

For an n-gram $c_t w_t$ with r counts, this count is then converted into a probability:

$$P_{GT}(c_t w_t) = \frac{r^*}{N}, \tag{2.35}$$

where $N = \sum_{r=1}^{\infty} r\, n_r$. In practice, the Good–Turing estimate is not used by itself for n-gram smoothing, because it does not include the combination of higher-order models with lower-order models necessary for good performance.

Interpolation methods

Higher and lower order n-gram models have different strengths and weaknesses. High-order n-grams are sensitive to more context, but have sparse counts. Low-order n-grams consider only very limited context, but have robust counts. The idea with interpolation methods is to combine them. For instance in a trigram model, the estimate is defined as follows:

$$P(w_t|w_{t-2}, w_{t-1}) = \lambda_1 \times P_{ML}(w_t|w_{t-2}, w_{t-1}) + \lambda_2 \times P_{ML}(w_t|w_{t-1}) + \lambda_3 \times P_{ML}(w_t), \tag{2.36}$$

where λ_1, λ_2 and λ_3 are three additional parameters with $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$. The λ_i can be estimated on some held-out data set using EM algorithms.

Kneser–Ney smoothing

Kneser-Ney (KN) smoothing is considered as the most effective method for both higher and lower order n-gram models. This method is optimized for giving high probability to lower-order model only when count is small or zero in higher-order model. Considering the bigram *San Francisco*, we want to give *Francisco* a low unigram probability because it mostly occurs after *San*. For each unigram w_t, we then count the number of different words that it follows, normalized by the number of words preceding all words:

$$P_{CONTINUATION}(w_t) = \frac{\left|\{w_{t-1} : n(w_{t-1} w_t) > 0\}\right|}{\sum_{w'_t} \left|\{w'_{t-1} : n(w'_{t-1} w'_t) > 0\}\right|}. \tag{2.37}$$

For bigrams, the estimate is defined as follows:

$$P_{KN}(w_t|w_{t-1}) = \frac{\max\big(n(w_{t-1}w_t)-\delta,0\big)}{n(w_{t-1})} + \lambda_{w_{t-1}} P_{CONTINUATION}(w_t)\,. \tag{2.38}$$

The parameter δ is a constant which denotes the discount value subtracted from the count of each n-gram, usually between 0 and 1. λ is a normalizing constant which defines the probability mass that have been discounted:

$$\lambda_{w_{t-1}} = \frac{\delta}{n(w_{t-1})}\Big|\{w_t : n(w_{t-1}w_t) > 0; \quad \forall w_t \in \mathcal{W}\}\Big|\,. \tag{2.39}$$

For n-grams, we have by recursion:

$$P_{KN}(w_t|c_t) = \frac{\max\big(n(c_t w_t)-\delta,0\big)}{n(c_t)} + \frac{\delta}{n(c_t)}\Big|\{w'_t : n(c_t w'_t) > 0\}\Big| P_{KN}(w_t|w_{t-(n-2)},\ldots,w_{t-1})\,. \tag{2.40}$$

2.3.3 Neural Network Language Models

Neural neural language models (NNLM) address the n-gram data sparsity issue through parameterization of words as vectors (word embeddings) and using them as inputs to a neural network (Bengio et al., 2003). By applying the softmax function (Bridle, 1990) to the output layer of the network with parameters θ, the conditional distribution corresponding to context c_t, $P(w_t|c_t)$, is defined as

$$P_\theta(w_t|c_t) = \frac{\exp\big(\phi_\theta(w_t,c_t)\big)}{\sum_{i=1}^{|\mathcal{W}|}\exp\big(\phi_\theta(w_i,c_t)\big)} = \frac{\exp\big(\phi_\theta(w_t,c_t)\big)}{Z_\theta(c_t)}, \quad \phi_\theta \in \mathbb{R}^{|\mathcal{W}|}, \tag{2.41}$$

where $\phi_\theta(w_t,c_t)$ is the output score that quantifies the compatibility of word w_t with context c_t, and $Z_\theta(c_t)$ is the partition function that normalizes this into a probability distribution. Word embeddings obtained through NNLM exhibit the property whereby semantically close words are likewise close in the induced vector space. This feature will be discussed in more details in Section 3.2.

Maximum likelihood learning

NNLM are trained by maximizing a likelihood over the training data, using stochastic gradient descent. This learning procedure leads to optimizing cross-entropy between the target probability distribution (e.g., the target word that should be predicting), and the model predictions. We can express the log-likelihood for a single word/context observation (w_t, c_t) as follows:

$$\mathcal{L}(w_t,c_t;\theta) = \phi_\theta(w_t,c_t) - \log Z_\theta(c_t) \tag{2.42}$$

Because it requires computing $\phi_\theta(w_t, c_t)$ for all words in the vocabulary $\mathcal{W}$, maximum likelihood training of neural language models is tractable but tends to be very slow and expensive.

2.3.4 Recurrent Neural Networks

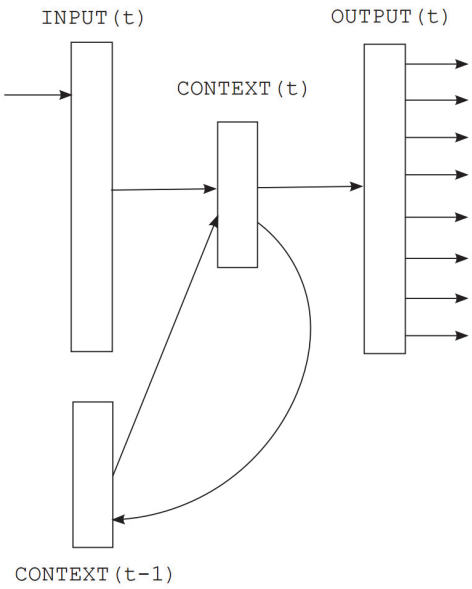


Figure 2.1 – Simple recurrent neural network (Mikolov et al., 2010).

Unlike standard feed-forward neural networks, recurrent networks retain a state that can represent information from an arbitrarily long context window. This feature can be useful in language generation for modeling long-term dependencies.

Figure 2.1 illustrate a simple recurrent neural network (RNN) (Mikolov et al., 2010). The input layer $\mathbf{x}_t$ represents input word at time t, $\mathbf{y}_t$ is the output layer, and the hidden layer $\mathbf{h}_t$ (also called *context* layer or *state*) maintains a representation of the sentence history. The layers are connected with weight matrices $\theta = \{\mathbf{U}, \mathbf{W}, \mathbf{V}\}$. The values in the hidden and output layers are computed as follows:

$$\mathbf{h}_t = \sigma\left(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}\right), \tag{2.43}$$

$$\mathbf{y}_t = g\left(\mathbf{V}\mathbf{h}_t\right). \tag{2.44}$$

where $\sigma(\cdot)$ is the sigmoid activation function (see Section 2.1.2), and $g(z)$ is the softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}. \tag{2.45}$$

As for NNLM, the model is trained using standard backpropagation to maximize the data conditional likelihood:

$$\mathcal{L}(\theta) = \prod_t P\big(\mathbf{y}_t | \mathbf{x}_1 \ldots \mathbf{x}_t\big) \tag{2.46}$$

Long Short-Term Memory

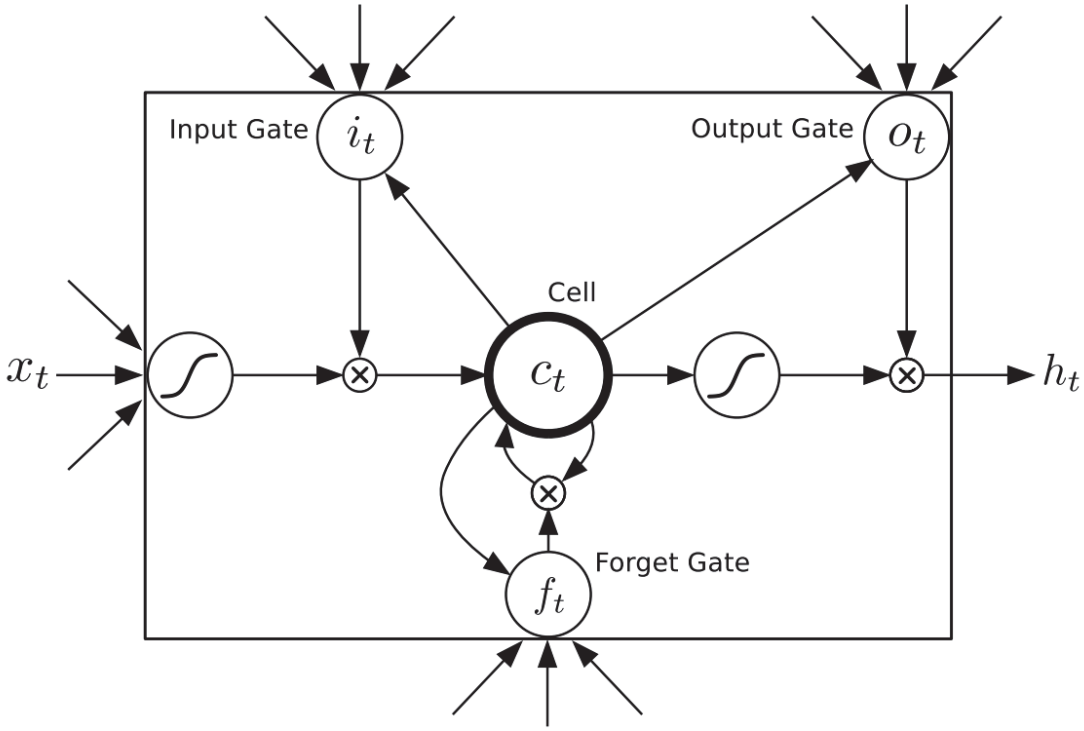


Figure 2.2 – Long Short-term Memory Cell (Graves, 2013).

In practice, standard RNN encounter difficulties for storing information about long-past inputs (Hochreiter et al., 2001). Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) have therefore been introduced to provide a solution to the exploding and vanishing gradient problem (Bengio et al., 1994).

Figure 2.2 illustrates a single LSTM *memory cell* (Graves, 2013). Such LSTM cell contains gates that determine when the input is significant enough to remember, when it should continue to remember or forget the value, and when it should output the value. The objective of this memory cell is to substitute the hidden layer in traditional RNN. The hidden layer is thus

implemented by the following composite functions:

$$\mathbf{i}_t = \sigma\big(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1}\big) \tag{2.47}$$

$$\mathbf{f}_t = \sigma\big(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1}\big) \tag{2.48}$$

$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \mathrm{Tanh}\big(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1}\big) \tag{2.49}$$

$$\mathbf{o}_t = \sigma\big(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t\big) \tag{2.50}$$

$$\mathbf{h}_t = \mathbf{o}_t \mathrm{Tanh}(\mathbf{c}_t) \tag{2.51}$$

where $\sigma(\cdot)$ and $\mathrm{Tanh}(\cdot)$ are activation functions (see Section 2.1.2), and $\mathbf{i}$, $\mathbf{f}$, $\mathbf{o}$ and $\mathbf{c}$ are respectively the *input gate*, *forget gate*, *output gate*, *cell* and *cell input* activation vectors, all of which are the same size as the hidden vector $\mathbf{h}$. The weight matrix subscripts have the obvious meaning, for example $\mathbf{W}_{hi}$ is the hidden-input gate matrix, $\mathbf{W}_{xo}$ is the input-output gate matrix, etc. The weight matrices from the cell to gate vectors (e.g.$\mathbf{W}_{ci}$) are diagonal, so element m in each gate vector only receives input from element m of the cell vector. The bias terms (which are added to i, f, c and o) have been omitted for clarity.

2.3.5 Evaluation Metrics

Perplexity

A very common method to measure the quality of a language model is to evaluate the perplexity of the model on some held-out data. Given a new test sentence $s = w_1, \ldots, w_T$, we can measure its probability $P(s)$ under the language model.

The cross-entropy over a sentence is then:

$$\begin{aligned} H(P) &= -\frac{1}{T}\log P(s) \\ &= -\frac{1}{T}\sum_{t=1}^{T}\log P(w_t|w_1,\ldots,w_{t-1}) \end{aligned} \tag{2.52}$$

The perplexity for that sentence is $2^{H(P)}$.

Perplexity can be interpreted as a measure of on average how many different equally most probable words can follow any given word. Lower perplexities represent better language models. However, evaluating language models only with perplexity is not optimal, as data may not be drawn from the same distribution than the test set in real world applications.

BLEU metric

BLEU (bilingual evaluation understudy) is an algorithm that has been first introduced for evaluating machine translation systems (Papineni et al., 2002). But this measure can also be used for comparing candidate sentences against reference sentences in a same language.

BLEU is about n-gram precision. It measures the proportion of n-grams in the candidate text that also appear in a reference text.

To deal with over-generation of frequent words, *BLEU* uses a *clipped* notion of precision that only accepts as many instances of a word as actually appear in some reference text. Give a set of candidate sentences $C \in \{\text{Candidates}\}$, a modified n-gram precision score p_n is computed as follows:

$$p_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{n\text{-gram} \in C} Count_{clip}(n\text{-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{n\text{-gram}' \in C'} Count(n\text{-gram}')}, \tag{2.53}$$

where $Count_{clip} = \min\big(Count, Max\_Reference\_Count\big)$.

A high-scoring candidate sentence must also match the reference sentence in length. Candidate sentences longer than their references are already penalized by the modified n-gram precision measure. Consequently, only a brevity penalty (BP) factor is introduced. Let c be the length of the candidate sentence and r be the effective reference corpus length. The brevity penalty is computed as follows:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases}. \tag{2.54}$$

Then, the *BLEU* score is the geometric mean of different sizes of n-gram (usually up to $N = 4$):

$$BLEU = BP \cdot \exp\big(\frac{1}{N} \sum_{n=1}^{N} \log p_n\big). \tag{2.55}$$

Part I
Building Word Embeddings from Large Text Corpora

3 State-of-the-art Word Representations

In this chapter, we introduce the state-of-the-art methods for computing word embeddings. Clustering techniques have been employed in the past to obtain word representations in discrete form. A first section is devoted to describing Brown clustering which is the main clustering algorithm for word representations. More recently, models based on NNLM have been proposed to compute distributed word representations. In a second section, we present the major methods going from the pioneer work of Collobert and Weston (2008) to the most popular technique, the Skip-gram model from Mikolov et al. (2013a).

3.1 Brown Clustering

This algorithm is first introduced in (Brown et al., 1992) with the idea of partitioning a corpus vocabulary into clusters, where clusters (ideally) include semantically similar words.

Initially, each word in the vocabulary is considered to be in its distinct cluster. The algorithm then repeatedly merges the pair of clusters which causes the smallest decrease in the likelihood of the text corpus, according to a class-based bigram language model defined on the word clusters (Liang, 2005), as illustrated in Figure 3.1.

Given a sequence of words $w_1, \ldots, w_T \in \mathcal{W}$ and a partition function C of the vocabulary into k

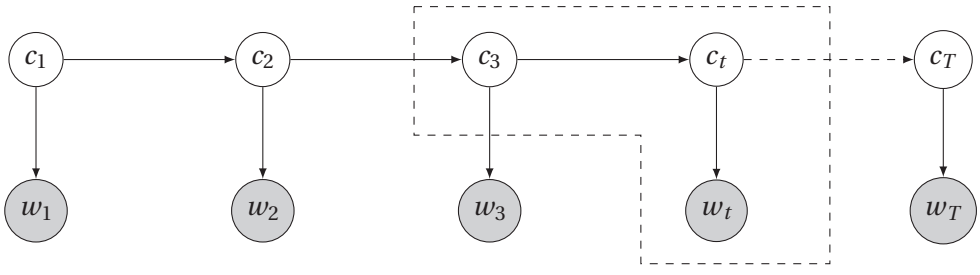


Figure 3.1 – The class-based bigram language model, which defines the quality of a clustering, represented as a Bayesian network.

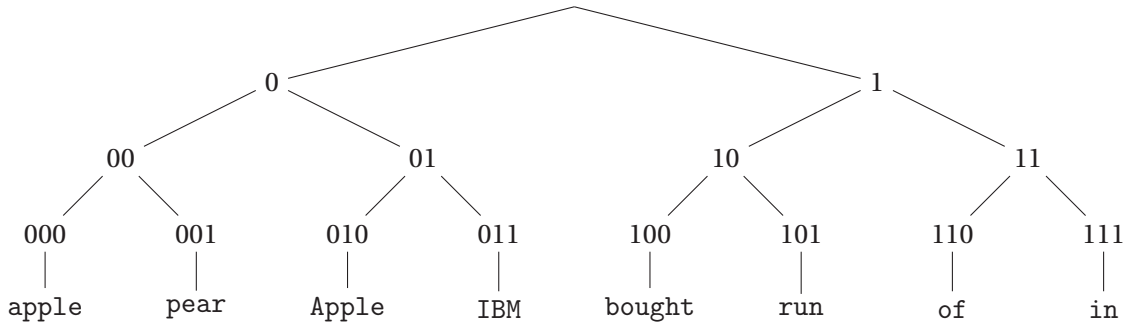


Figure 3.2 – An example of a Brown word-cluster hierarchy. Each node in the tree is labeled with a bit string indicating the path from the root node to that node, where 0 indicates a left branch and 1 indicates a right branch.

classes, the likelihood model is defined as

$$P(w_1,\ldots,w_T) = \prod_{t=1}^{T} P\big(w_t|C(w_t)\big)P\big(C(w_t)|C(w_{t-1})\big). \tag{3.1}$$

More conveniently,

$$\log P(w_1,\ldots,w_T) = \sum_{t=1}^{T} \log P\big(w_t|C(w_t)\big)P\big(C(w_t)|C(w_{t-1})\big). \tag{3.2}$$

Let $n(w,w')$ be the number of times that word w precedes w' in the corpus, and $n(w)$ the number of times we see word w. The quality of a cluster C is measured as

$$\begin{aligned}
\text{Quality}(C) &= \frac{1}{T}\sum_{t=1}^{T} \log P\big(C(w_t)|C(w_{t-1})\big)P\big(w_t|C(w_t)\big) \\
&= \sum_{w,w'} \frac{n(w,w')}{T} \log P\big(C(w')|C(w)\big)P\big(w'|C(w')\big) \\
&= \sum_{w,w'} \frac{n(w,w')}{T} \log \frac{n\big(C(w),C(w')\big)}{n\big(C(w)\big)} \frac{n(w')}{n\big(C(w')\big)} \\
&= \sum_{w,w'} \frac{n(w,w')}{T} \log \frac{n\big(C(w),C(w')\big)\times T}{n\big(C(w)\big)\times n\big(C(w')\big)} + \sum_{w,w'} \frac{n(w,w')}{T} \log \frac{n(w')}{T} \\
&= \sum_{c,c'} \frac{n(c,c')}{T} \log \frac{n(c,c')\times T}{n(c)\times n(c')} + \sum_{w'} \frac{n(w')}{T} \log \frac{n(w')}{T} \\
&= \sum_{c,c'} P(c,c') \log \frac{p(c,c')}{p(c)\times p(c')} + \sum_{w} P(w)\log P(w) \\
&= I(C) - H
\end{aligned} \tag{3.3}$$

where $P(c,c') = \frac{n(c,c')}{T}$, $P(w) = \frac{n(w)}{T}$ and $P(c) = \frac{n(c)}{T}$. The first term $I(C)$ is the mutual information between adjacent clusters and the second term H is the entropy of the word distribution.

At the end of the algorithm, we obtain a hierarchy of word types which can be represented as a binary tree as in Figure 3.2. Within this tree, each word is uniquely identified by its path from the root, and this path can be compactly represented with a bit string.

3.2 Through Neural Network Language Models

Word embeddings can be induced using neural network language models (NNLM), which use neural networks as the underlying predictive model. However, training of NNLM is slow, scaling as the size of the vocabulary for each model computation (see Section 2.3.3). On the other hand, for feature learning, we do not necessarily need a full probabilistic model. In recent years, several authors have thus proposed to eliminate that linear dependency on vocabulary size and allow scaling to very large training corpora.

3.2.1 Pairwise Ranking Approach

Collobert and Weston (2008) consider a standard NNLM, but they train the model to discriminate a two-class classification task: if the word w_t in the middle of a sequence of words is related to its context c_t or not. By considering all possible sequences of words from the corpus, they build a training set of all context/word pairs $(w_t, c_t) \in \mathcal{T}$. Those pairs are the positive examples. A negative example is then the same sequence but where the middle word w_t has been replaced by a random word $w_i \in \mathcal{W}$, with $w_i \neq w_t$.

They train this problem with a ranking-type cost:

$$\mathcal{L}(\theta) = \sum_{(w_t, c_t) \in \mathcal{T}} \sum_{\substack{i=1 \\ w_i \neq w_t}}^{|\mathcal{W}|} \max\big(0, 1 - \phi_\theta(w_t, c_t) + \phi_\theta(w_i, c_t)\big), \tag{3.4}$$

where $\mathcal{W}$ is the vocabulary of words, and ϕ_θ is the scoring function without the softmax activation function. $\phi_\theta(w_i, c_t)$ is thus the score for a corrupted sequence where the middle word is replaced by the random word w_i.

3.2.2 Scalable Log-Bilinear Model

In Mnih and Hinton (2007), the authors introduce a purely linear NNLM. Given a context c_t, the model combines the embeddings of the $n-1$ context words:

$$\phi^2_\theta(c_t) = \sum_{i=1}^{n-1} \mathbf{C}^i \phi^1_\theta(w_i), \tag{3.5}$$

where $\mathbf{C}^i$ is the weight matrix associated with the context position i, and $\phi^1_\theta(\cdot)$ is the embedding layer, as described in Section 2.1.3. It then learns a linear model to predict the embedding of

the next word w_t. The similarity between the context embedding and the embedding for each word w_t in the vocabulary is computed using the inner product:

$$\phi_\theta(w_t, c_t) = \phi_\theta^2(c_t) \cdot \phi_\theta^1(w_t) + b_{w_t}, \tag{3.6}$$

where b_{w_t} is the bias for word w_t, which is used to capture the context-independent word frequency. The softmax activation function can then be used to obtain the distribution of the next word $P(w_t|c_t)$. Best word embeddings are obtained by learning high-dimensional embeddings from very large quantities of data, which makes scalability of the training method a critical factor. The same authors have thus proposed two methods for scaling up to a large vocabulary, with hierarchical softmax or noise-contrastive estimation.

Hierarchical Softmax

Based on the hierarchical model from Morin and Bengio (2005), Mnih and Hinton (2009) propose to speed up this model by using a hierarchy to exponentially filter down the number of computations that are performed. The model, combined with this optimization, is called the hierarchical log-bilinear (HLBL) model. As in Brown clustering described in Section 3.1, this model organizes the output vocabulary into a binary tree where the leaves are the words. This allows each word to be represented by a binary string which it is called a *code*. At each node, the HLBL model then uses a probabilistic model for making the decisions. Each of the non-leaf nodes in the tree is represented with an embedding that is used for discriminating the words in the left sub-tree form the words in the right sub-tree of the node. The probability of the next word being w_t is the probability of making the sequences of binary decisions specified by the word's code, given the context. Since the probability of making a decision at a node depends only on the context embedding and the embedding for that node, the probability of the next word is expressed as a product of probabilities of the binary decisions:

$$P(w_t|c_t) = \prod_i P(d_i|q_i, c_t), \tag{3.7}$$

where d_i is the i^{th} digit in the code for word w_t, and q_i is the embedding for the i^{th} node in the path corresponding to that code. The probability of each decision is given by:

$$P(d_i|q_i, c_t) = \sigma\left(\phi_\theta^2(c_t) \cdot q_i + b_i\right), \tag{3.8}$$

where $\sigma(x)$ is the logistic function and $\phi_\theta^2(c_t)$ is the context embedding computed using Equation 3.5. b_i in the equation is the node's bias that captures the context-independent tendency to visit the left child when leaving this node. If the tree is perfectly balanced, this can reduce the complexity from $\mathcal{O}(V)$ to $\mathcal{O}(\log V)$.

Noise-Contrastive Estimation

Noise-Contrastive Estimation (NCE) is another approach which enables fast training without the complexity of working with tree-structured models, such as hierarchical softmax (Mnih and Teh, 2012). It reduces the language model estimation problem to the problem of estimating the parameters of a probabilistic binary classifier that uses the same parameters to distinguish samples from the empirical distribution from samples generated by the "noise" distribution Q (Gutmann and Hyvärinen, 2012). In practice, Q is the empirical context-independent (i.e. unigram) distribution, $Q(w)$. We denote $\tilde{P}(w_t|c_t)$ and $\tilde{P}(c)$ the empirical distributions. We are interested in fitting the context-dependent $P_\theta(w_t|c_t)$ to $\tilde{P}(w_t|c_t)$. The two-class training data is generated as follows:

1. sample a context c_t from $\tilde{P}(c)$,
2. sample one "true" sample from $\tilde{P}(w_t|c_t)$, with auxiliary label $D = 1$ indicating the data point is drawn from the true distribution,
3. sample k "noise" samples from $Q(w)$, with auxiliary label $D = 0$ indicating these data points are noise.

Thus, given c_t, the joint probability of (d, w_t) in the two-class data has the form of the mixture of two distributions:

$$P(d, w_t|c_t) = \begin{cases} \frac{k}{1+k} \times Q(w) & \text{if } d = 0 \\ \frac{1}{1+k} \times \tilde{P}(w_t|c_t) & \text{if } d = 1 \end{cases}. \tag{3.9}$$

Using the definition of conditional probability, this can be turned into a conditional probability of d having observed w_t and c_t:

$$\begin{aligned} p(D = 0|c_t, w_t) &= \frac{\frac{k}{1+k} \times Q(w)}{\frac{1}{1+k} \times \tilde{P}(w_t|c_t) + \frac{k}{1+k} \times Q(w)} \\ &= \frac{k \times Q(w)}{\tilde{P}(w_t|c_t) + k \times Q(w)} \end{aligned} \tag{3.10}$$

$$p(D = 1|c_t, w_t) = \frac{\tilde{P}(w_t|c_t)}{\tilde{P}(w_t|c_t) + k \times Q(w)}. \tag{3.11}$$

Note that these probabilities are written in terms of the empirical distribution.

NCE replaces the empirical distribution $\tilde{P}(w_t|c_t)$ with the model distribution $P_\theta(w_t|c_t)$, and θ is chosen to maximize the conditional likelihood of the "proxy corpus" created as described above. To avoid the expense of evaluating the partition function, NCE makes two further assumptions:

1. it proposes partition function value $Z(c_t)$ be estimated as parameter z_{c_t} (thus, for every

empirical c_t, classic NCE introduces one parameter);

2. for neural networks with lots of parameters, it turns out that fixing $z_{c_t} = 1$ for all c_t is effective (Mnih and Teh, 2012). This latter assumption both reduces the number of parameters and encourages the model to have "self-normalized" outputs (i.e., $Z(c_t) \approx 1$).

Consequently, the normalizing factor from Equation 2.3.3 can be dropped, and we simply use $\exp\big(\phi_\theta(w_t, c_t)\big)$ in place of $P_\theta(w_t|c_t)$ during training. Making these assumptions, we can now write the conditional likelihood of being a noise sample or true distribution sample in terms of θ as

$$
\begin{aligned}
P(D=0|c_t, w_t) &= \frac{k \times Q(w)}{\exp\big(\phi_\theta(w_t, c_t)\big) + k \times Q(w)} \\
&= \frac{k \times Q(w)}{\Big(k \times Q(w)\Big) \times \Big(\frac{\exp\big(\phi_\theta(w_t, c_t)\big)}{k \times Q(w)} + 1\Big)} \\
&= \frac{1}{\frac{1}{k \times Q(w)} \exp\big(\phi_\theta(w_t, c_t)\big) + 1} \\
&= \frac{1}{\exp\Big(\phi_\theta(w_t, c_t) - \log\big(k \times Q(w)\big)\Big) + 1} \\
&= 1 - \frac{1}{1 + \exp\Big(-\phi_\theta(w_t, c_t) + \log\big(k \times Q(w)\big)\Big)} \\
&= 1 - \sigma\big(\Delta\phi_\theta(w_t, c_t)\big) \qquad (3.12) \\
p(D=1|c_t, w_t) &= \frac{\exp\big(\phi_\theta(w_t, c_t)\big)}{\exp\big(\phi_\theta(w_t, c_t)\big) + k \times Q(w)} \\
&= \frac{\exp\big(-\phi_\theta(w_t, c_t)\big) \times \exp\big(\phi_\theta(w_t, c_t)\big)}{\exp\big(-\phi_\theta(w_t, c_t)\big) \times \Big(\exp\big(\phi_\theta(w_t, c_t)\big) + k \times Q(w)\Big)} \\
&= \frac{1}{1 + \exp\big(-\phi_\theta(w_t, c_t)\big) \times k \times Q(w)} \\
&= \frac{1}{1 + \exp\Big(-\phi_\theta(w_t, c_t) + \log\big(k \times Q(w)\big)\Big)} \\
&= \sigma\big(\Delta\phi_\theta(w_t, c_t)\big), \qquad (3.13)
\end{aligned}
$$

where $\sigma(x)$ is the logistic function and $\Delta\phi_\theta(w_t, c_t) = \phi_\theta(w_t, c_t) - \log\big(k \times Q(w)\big)$ is the difference in the scores of word w_t under the model and the (scaled) noise distribution. The scaling factor k in front of $Q(w)$ accounts for the fact that noise samples are k times more frequent than data samples.

We now have a binary classification problem with parameters θ that can be trained to maximize

conditional log-likelihood of the training set $\mathcal{T}$, with k negative samples chosen:

$$\mathcal{L}_{NCE_k} = \sum_{(w_t,c_t)\in\mathcal{T}} \left(\log P(D=1|c_t,w_t) + kE_{w_i\sim Q}\big[\log P(D=0|c_t,w_i)\big] \right). \tag{3.14}$$

Unfortunately, the expectation of the second term in this summation is still a difficult summation – it is k times the expected log probability (according to the current model) of producing a negative label under the noise distribution over all words in $\mathcal{W}$ in a context c_t. We still have a loop over the entire vocabulary. The final step is therefore to replace this expectation with its Monte Carlo approximation:

$$\begin{aligned}
\mathcal{L}^{\text{MC}}_{\text{NCE}_k} &= \sum_{(w_t,c_t)\in\mathcal{T}} \left(\log P(D=1|c_t,w_t) + k \times \sum_{i=1,w_i\sim Q}^{k} \frac{1}{k} \times \log P(D=0|c_t,w_i) \right) \\
&= \sum_{(w_t,c_t)\in\mathcal{T}} \left(\log P(D=1|c_t,w_t) + \sum_{i=1,w_i\sim Q}^{k} \log P(D=0|c_t,w_i) \right) \\
\mathcal{L}^{\text{MC}}_{\text{NCE}_k}(\theta) &= \sum_{(w_t,c_t)\in\mathcal{T}} \left(\log\sigma\big(\Delta\phi_\theta(w_t,c_t)\big) + \sum_{i=1,w_i\sim Q}^{k} \log\Big(1-\sigma\big(\Delta\phi_\theta(w_i,c_t)\big)\Big) \right).
\end{aligned} \tag{3.15}$$

This objective is maximized when the model assigns high probabilities to the real words, and low probabilities to noise words. Technically, this is called *Negative Sampling*, and there is good mathematical motivation for using this loss function. The updates it proposes approximate the updates of the softmax function in the limit. But computationally it is especially appealing because computing the loss function now scales only with the number of noise words that we select (k), and not all words in the vocabulary ($\mathcal{W}$). This makes it much faster to train.

3.2.3 Skip-gram Model

The Skip-gram model is also a purely linear NNLM (Mikolov et al., 2013a). Given a training sample (w_t, c_t), the model predicts source context-words $w_i \in c_t$ from the target word w_t (usually the word in the middle of a sequence):

$$\sum_{w_i\in c_t} \log P(w_i|w_t). \tag{3.16}$$

By defining two embedding layers, an input embedding layer ϕ^{in}_θ and an output embedding layer ϕ^{out}_θ, a score between a context word w_i and a target word w_t is computed with an inner product:

$$\phi_\theta(w_t,w_i) = \phi^{\text{in}}_\theta(w_t)\cdot\phi^{\text{out}}_\theta(w_i). \tag{3.17}$$

Negative Sampling

For training the Skip-gram, the authors use a variation of NCE, where the conditional probabilities are defined as follows:

$$P(D=0|w_i,w_t) = \frac{1}{\phi_\theta(w_t,w_i)+1} = \sigma\big(-\phi_\theta(w_t,w_i)\big) \tag{3.18}$$

$$P(D=1|w_i,w_t) = \frac{\phi_\theta(w_t,w_i)}{\phi_\theta(w_t,w_i)+1} = \sigma\big(\phi_\theta(w_t,w_i)\big). \tag{3.19}$$

The objective to maximize is thus defined as

$$\mathcal{L}_{\mathrm{NEG}_k}(\theta) = \sum_{(w_t,c_t)\in\mathcal{T}} \sum_{w_i\in c_t} \Big(\log\sigma\big(\phi_\theta(w_t,w_i)\big) + \sum_{j=1,w_j\sim Q}^{k} \log\sigma\big(-\phi_\theta(w_t,w_j)\big)\Big). \tag{3.20}$$

This negative sampling does not have the same asymptotic consistency guarantees that NCE has, but it works well for learning word embeddings.

Sub-Sampling of Frequent Words

In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., “in”, “the”, and “a”). Such words usually provide less information value than the rare words. To counter the imbalance between the rare and frequent words, Mikolov et al. (2013b) introduce a sub-sampling method that randomly removes words w_t that are more frequent than some threshold t with a probability

$$P(w_t) = 1 - \sqrt{\frac{t}{\mathrm{tf}_{w_t}}}, \tag{3.21}$$

where tf_{w_t} is the word frequency of w_t in the corpus. The recommended value for t is typically around 10^{-5}. Although this sub-sampling formula has been chosen heuristically, it works well in practice. It accelerates learning and seems to help learning better embeddings for rare words.

4 Word Embeddings through Hellinger PCA

Building word embeddings has always generated much interest for linguists. Popular approaches such as Brown clustering algorithm (see Section 3.1) have been used with success in a wide variety of NLP tasks (Schütze, 1995; Koo et al., 2008; Ratinov and Roth, 2009). Recently, distributed approaches based on neural network language models (NNLM) have revived the field of learning word embeddings (Collobert and Weston, 2008; Huang and Yates, 2009; Turian et al., 2010; Collobert et al., 2011; Mikolov et al., 2013b; Mnih and Kavukcuoglu, 2013). However, a neural network architecture can be hard to train. Finding the right hyper-parameters to tune the model is often a challenging task and the training phase is in general computationally expensive.

A NNLM learns word embeddings by predicting the words among the vocabulary that are likely to appear in the surrounding of sequences of words. More formally, it learns the word co-occurrence probability distributions. Instead, simply counting words on a large corpus of unlabeled text can be performed to retrieve those word distributions and to represent words (Turney and Pantel, 2010). In this chapter, we thus show that similar word embeddings can be computed using the word co-occurrence statistics and a well-known dimensionality reduction operation such as Principal Component Analysis (PCA). In contrast with NNLM, PCA is a simple, non-parametric method of extracting relevant information from confusing data sets. Previous work has therefore attempted to build word representations with such techniques with varying degrees of success (Schütze, 1993; Bengio et al., 2001). We claim that, assuming an appropriate metric, this simple spectral method can however generate word embeddings as good as with NNLM architectures.

4.1 Word Co-Occurrence Probabilities

“You shall know a word by the company it keeps" (Firth, 1957). Keeping this famous quote in mind, word co-occurrence probabilities are computed by counting the number of times each

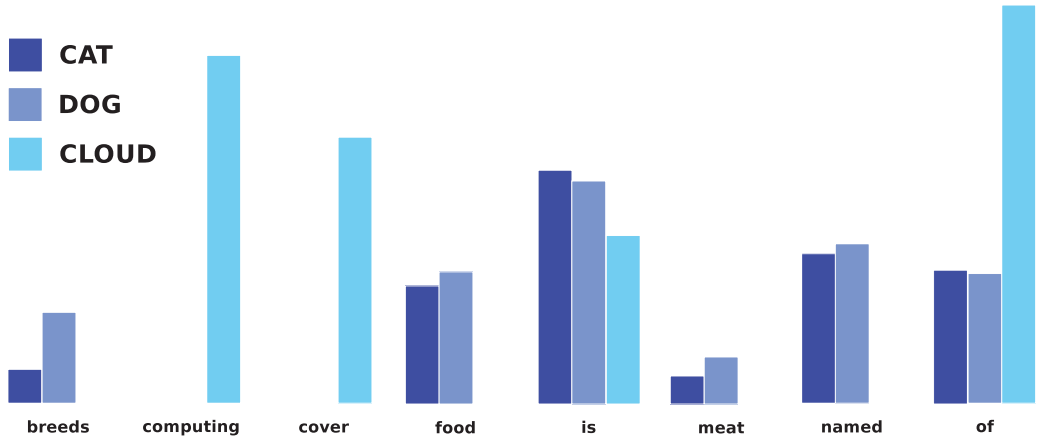


Figure 4.1 – A toy example illustrating that related words have similar word co-occurrence probability distributions.

context word $c_t \in \mathcal{D}$ (where $\mathcal{D} \subseteq \mathcal{W}$) occurs around a word $w_t \in \mathcal{W}$:

$$P(c_t|w_t) = \frac{P(c_t, w_t)}{P(w_t)} = \frac{n(c_t, w_t)}{\sum_{i=1}^{|\mathcal{D}|} n(c_i, w_t)}, \tag{4.1}$$

where $n(c_t, w_t)$ is the number of times a context word c_t occurs in the surrounding of the word w_t. The number of context words to consider around each word is variable and can be either symmetric or asymmetric. A multinomial distribution of $|\mathcal{D}|$ classes (words) is thus obtained for each word w_t:

$$P(c_1, \ldots, c_{|\mathcal{D}|}|w_t) = \left\{P(c_1|w_t), \ldots, P(c_{|\mathcal{D}|}|w_t)\right\}. \tag{4.2}$$

This distribution becomes less sparse when the window of context words is high.

For illustrative purposes, we denote both target and context vocabularies as follows

$$\mathcal{W} = \Big\{\text{cat}, \text{dog}, \text{cloud}\Big\}, \tag{4.3}$$

$$\mathcal{D} = \Big\{\text{breeds}, \text{computing}, \text{cover}, \text{food}, \text{is}, \text{meat}, \text{named}, \text{of}\Big\}. \tag{4.4}$$

We see in Figure 4.1 that related words (*cat* and *dog*) have similar word co-occurrence probabilities, while a non-related word (*cloud*) has a completely different probability distribution over this specific context vocabulary $\mathcal{D}$.

Because we are facing discrete probability distributions, the Hellinger distance (Hellinger, 1909) seems appropriate to calculate similarities between these word representations.

4.2 Hellinger Distance

Similarities between words can be derived by computing a distance between their corresponding word distributions. Several distances (or metrics) over discrete distributions exist, such as the *Bhattacharyya distance*, the *Hellinger distance* or *Kullback-Leibler divergence*. We chose here the Hellinger distance for its simplicity and symmetry property (as it is a true distance). Considering two discrete probability distributions $P = (p_1, \ldots, p_k)$ and $Q = (q_1, \ldots, q_k)$, the Hellinger distance is formally defined as:

$$H(P,Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2}, \tag{4.5}$$

which is directly related to the Euclidean norm of the difference of the square root vectors:

$$H(P,Q) = \frac{1}{\sqrt{2}} \| \sqrt{P} - \sqrt{Q} \|_2 . \tag{4.6}$$

Note that it makes more sense to take the Hellinger distance rather than the Euclidean distance for comparing discrete distributions, as P and Q are unit vectors according to the Hellinger distance ($\sqrt{P}$ and $\sqrt{Q}$ are units vector according to the ℓ_2 norm). The Hellinger distance thus forms a bounded metric on the space of probability distributions over a given probability space. The maximum distance 1 is achieved when P assigns probability zero to every set to which Q assigns a positive probability, and vice versa. Finally, the Hellinger distance is related to the *Bhattacharyya coefficient* (Bhattacharyya, 1943) $BC(P,Q)$ as it can be defined as

$$H(P,Q) = \sqrt{1 - BC(P,Q)}, \tag{4.7}$$

The Bhattacharyya coefficient is another divergence-type measure which is defined as

$$BC(P,Q) = \sum_{i=1}^{k} \sqrt{p_i q_i} . \tag{4.8}$$

4.3 Experimental Setup

After building our word vector representations over the English Wikipedia, we use benchmark datasets for evaluating them and selecting the right hyper-parameters accordingly.

4.3.1 Building Word Representations over Large Corpora

A large amount of text is necessary to build the word co-occurrence probability distributions. Our English corpus is thus composed of the entire English Wikipedia[1] (where all MediaWiki

[1] Available at http://download.wikimedia.org. We took the January 2014 version.

markups have been removed[2]). We consider lower case words to limit the number of words in the vocabulary. Additionally, all occurrences of sequences of numbers within a word are replaced with the special token "0". The resulting text is tokenized using the Stanford tokenizer[3]. The final dataset $\mathcal{S}$ contains about 1.6 billion words. As vocabulary $\mathcal{W}$, we consider all the words within our corpus which appear at least one hundred times. This results in a 191,268 words dictionary.

4.3.2 Evaluating Word Representations

Word analogies

The word analogy task consists of questions like, "a is to b as a^* is to ?". It was introduced in Mikolov et al. (2013a) and contains 19,544 such questions, divided into a semantic subset (SEM) and a syntactic subset (SYN). The 8,869 semantic questions are analogies about places, like "*Bern* is to *Switzerland* as *Paris* is to ?", or family relationship, like "*uncle* is to *aunt* as *boy* is to ?". The 10,675 syntactic questions are grammatical analogies, involving plural and adjectives forms, superlatives, verb tenses, etc. To correctly answer the question, the model should uniquely identify the missing term, with only an exact correspondence counted as a correct match. With word vector representations, we thus want the hidden vector b^* to be similar to the vector $b-a+a^*$. The analogy question can be solved by optimizing:

$$\underset{b^* \in V}{\operatorname{argmax}} \big(\operatorname{sim}(b^*, b-a+a^*)\big), \tag{4.9}$$

where V is the question vocabulary excluding the question words b, a and a^*, and $\operatorname{sim}(\cdot)$ is a similarity measure (usually the cosine similarity measure). As the number of questions is different in each category of analogy, the macro-averaged accuracy is reported for these two tasks.

Word Similarities

Word vector representations are also evaluated on two word similarity datasets: the WordSimilarity-353 (WS-353) Test Collection (Finkelstein et al., 2002) and the Stanford Rare Word (RW) (Luong et al., 2013). They both contain sets of English word pairs along with human-assigned similarity judgments. WS-353 contains 353 word pairs of relatively common words, like *computer:internet* or *football:tennis*. On the other hand, the RW dataset focuses on rare words. It contains 2,034 pairs where one of the two words is rare or morphologically complex, such as *brigadier:general* or *cognizance:knowing*. Performance is measured by Spearman's rank correlation coefficient.

[2] We used Wikipedia Extractor available at http://medialab.di.unipi.it/wiki/Wikipedia_Extractor.
[3] Available at http://nlp.stanford.edu/software/tokenizer.shtml

4.4 Analysis of the Context

As regards the context, two main hyper-parameters are involved:

1. The type of context to use, *i.e.* which words are to be chosen for defining the context dictionary $\mathcal{D}$. Do we need all the words, the most frequent ones or, on the contrary, the rare ones?
2. The context window size to consider, *i.e.* the number of context words c_t to count surrounding a given word w_t (symmetric context window).

4.4.1 Type of Context

Several types of context vocabularies $\mathcal{D}$ are considered to build the word co-occurrence probabilities. Mikolov et al. (2013b) have shown that better word representations can be obtained by sub-sampling of the frequent words. We thus want to find the right balance between frequent and rare words, while maintaining a reasonable number of context words in $\mathcal{D}$.

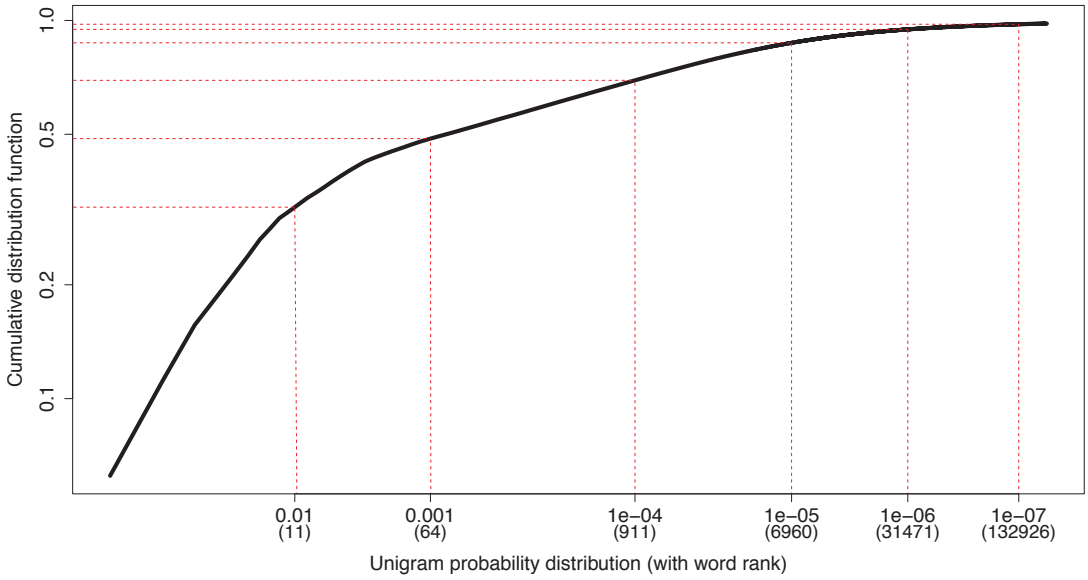


Figure 4.2 – Cumulative probability distribution of unigrams that occur at least a hundred times in the English Wikipedia.

Figure 4.2 reports the analysis of the unigram distribution of our corpus of text, with

$$P(w_t) = \frac{n(w_t)}{|\mathcal{S}|}, \quad \forall\, w_t \in \mathcal{W}. \tag{4.10}$$

We see that the 64 most frequent words with $P(w_t) > 10^{-3}$ represent about half of the entire

corpus. These words are considered as *stop words* and can be removed from the context vocabulary. Because the choice of stop words is arbitrary, this list can also include words with $P(w_t) > 10^{-4}$ which represent less than a thousand words (911). By selecting all words with $P(w_t) > 10^{-6}$, we see that we cover about 95% of our corpus. The size of context vocabulary becomes however quite large, as 31,471 words are above this threshold. A practical alternative is therefore to select words with $P(w_t) > 10^{-5}$, which already covers about 87% of the corpus. Finally, we observe a long tail of rare words which results in very sparse word representation when selecting large context vocabularies.

For a complete analysis, we thus define the following types of context vocabularies:

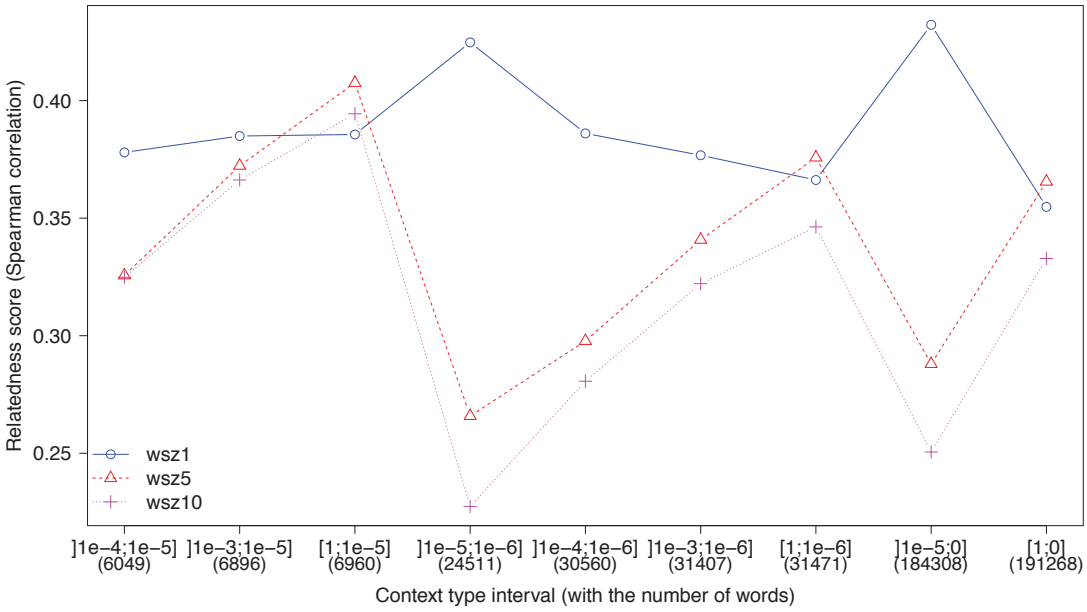
1. Only the most frequent words, where we consider all words with $P(w_t) > 10^{-5}$. This results in a context vocabulary of size $|\mathcal{D}| = 6960$.
2. Same vocabulary as 1 where we remove words with $P(w_t) > 10^{-3}$; $|\mathcal{D}| = 6896$.
3. Same vocabulary as 1 where we remove words with $P(w_t) > 10^{-4}$; $|\mathcal{D}| = 6049$.
4. All words with $P(w_t) > 10^{-6}$; $|\mathcal{D}| = 31471$.
5. Same vocabulary as 4 where we remove words with $P(w_t) > 10^{-3}$; $|\mathcal{D}| = 31407$.
6. Same vocabulary as 4 where we remove words with $P(w_t) > 10^{-4}$; $|\mathcal{D}| = 30560$.
7. Same vocabulary as 4 where we remove words with $P(w_t) > 10^{-5}$; $|\mathcal{D}| = 24511$.
8. Only rare words, where we consider all words with $P(w_t) < 10^{-5}$; $|\mathcal{D}| = 184308$.
9. Context vocabulary $\mathcal{D}$ is the same as the target vocabulary $\mathcal{W}$; $|\mathcal{D}| = 191268$.

Figures 4.3 and 4.4 present the performance obtained on the benchmark datasets for all these nine scenarios with different sizes of context. No dimensionality reduction has been applied in this analysis. Similarities between words are calculated with the Hellinger distance between the word probability distributions. For the word analogy task, we used the objective function 3CosMul defined by Levy and Goldberg (2014), as we are dealing with explicit word representations in this case:

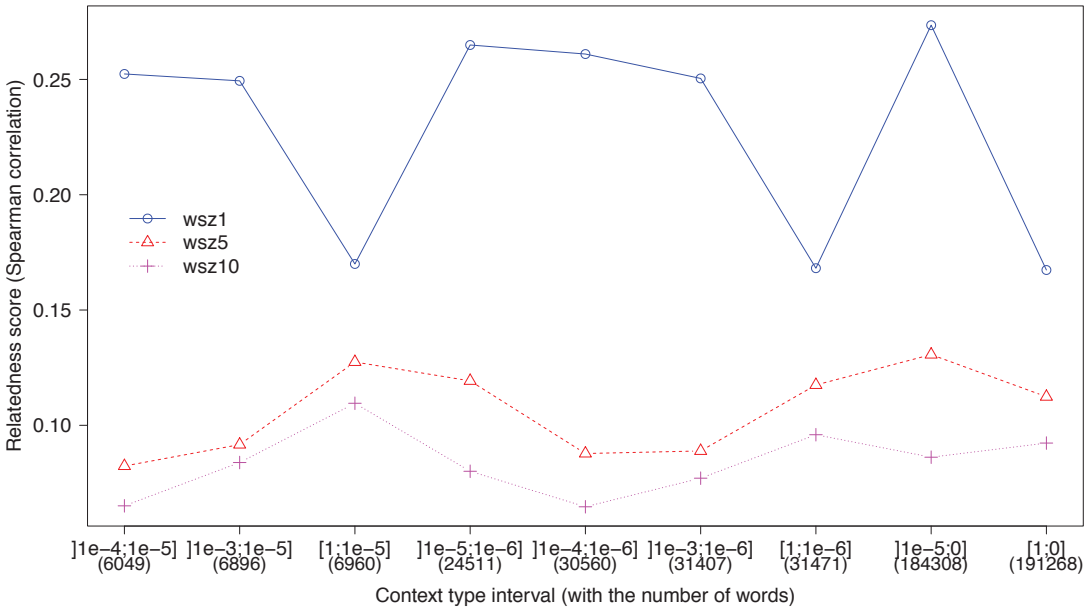
$$\underset{b^* \in V}{\operatorname{argmax}} \frac{\cos(b^*, b)\cos(b^*, a^*)}{\cos(b^*, a) + \epsilon}, \tag{4.11}$$

where $\epsilon = 0.001$ is used to prevent division by zero.

First, using all words as context does not imply to reach the best performance. With our smallest context vocabularies, performance is fairly similar than with all words. An in-between situation with words whose appearance frequency is greater than 10^{-6} gives also quite similar performance. Secondly, discarding the most frequent words from the context distributions helps to increase performance when using only one word of context. With five or ten context

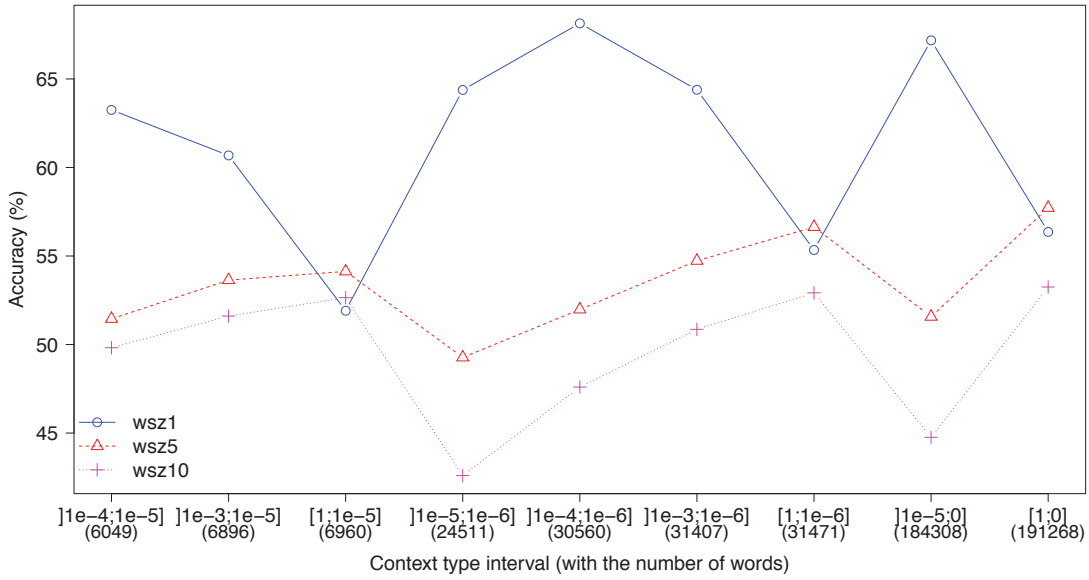


(a) WordSim-353 dataset

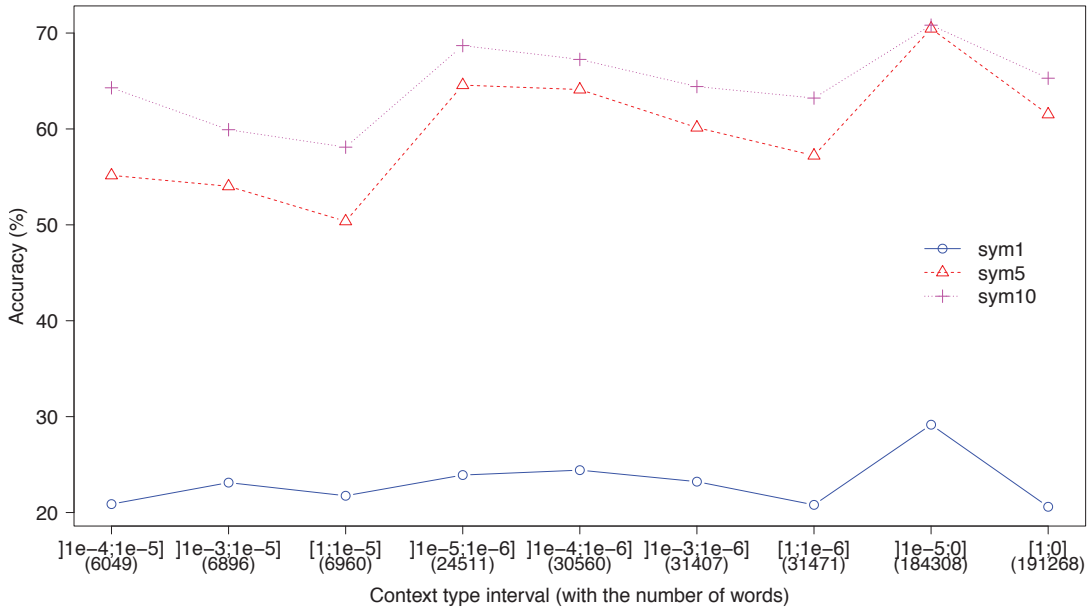


(b) Rare Word dataset

Figure 4.3 – Performance on word similarity datasets with different types of context word vocabularies $\mathcal{D}$ (in the ascending order of their number of words), and different window sizes (in the legend, *wsz1* is for symmetric window of 1 context word, etc.). Spearman rank correlation is reported.



(a) Mikolov's syntactic



(b) Mikolov's semantic

Figure 4.4 – Performance on word analogy datasets with different types of context word vocabularies $\mathcal{D}$ (in the ascending order of their number of words), and different window sizes (in the legend, *wsz1* is for symmetric window of 1 context word, etc.). Accuracy is reported.

words, the most frequent words seem helpful, except in the Mikolov's semantic task. On this semantic task, adding rare words in the context vocabulary helps improving the general performance, since results with words whose appearance frequency is less than 10^{-5} are the best. However, these observations might be explained by the sparsity of the probability distributions.

TYPE	DIM.	WINDOW SIZE		
		1	5	10
FROM 10^{-4} TO 10^{-5}	6049	151	660	946
FROM 10^{-3} TO 10^{-5}	6896	230	942	1319
UP TO 10^{-5}	6960	264	998	1380
FROM 10^{-5} TO 10^{-6}	24511	134	1615	1028
FROM 10^{-4} TO 10^{-6}	30560	282	1333	1974
FROM 10^{-3} TO 10^{-6}	31407	362	998	2347
UP TO 10^{-6}	31471	396	1672	2408
FROM 10^{-5}	184308	250	1305	2050
ALL	191268	514	2303	3430

Table 4.1 – The average number of context words according to the type and the size of context.

Counts in Table 4.1 show significant differences in terms of sparsity depending on the type of context. Similarities between words seem to be easier to find with sparse distributions. Consequently, adding the most frequent words hurt the performance when the distributions are very sparse since it introduces noise. The overlap between two word distributions on the most frequent words is probably high which eliminates the information coming from the less frequent words. When the number of context words is higher (5 or 10), the opposite occurs. Frequent words as context increase the general performance. As the number of counts for frequent words becomes higher with larger contexts, both most common and most rare words provide relevant information.

The average number of context words (i.e. features) whose appearance frequency is less than 10^{-5} and greater than 10^{-6} with a symmetric window of size 1 is extremely low (134). Performance with these hyper-parameters are still highly competitive on word similarities and syntactic analogies. Within this framework, it then becomes a good option for representing words in a low and sparse dimension. But overall the context vocabulary with only the most frequent (words with $P(w_t) > 10^{-5}$) appears like the ideal solution when considering larger context windows. The overall performance is very competitive yet with a far smaller number of context words. We thus choose this context vocabulary for performing the dimensionality reduction in the next section.

	WINDOW SIZE	
	1	10
BAIKAL	MÄLAREN	LAKE
(n^o37415)	TITICACA	SIBERIA
	BALATON	AMUR
	LADOGA	BASIN
	ILMEN	VOLGA
SPECIAL-NEED	AT-RISK	PRESCHOOL
(n^o165996)	SCHOOL-AGE	KINDERGARTEN
	LOW-INCOME	TEACHERS
	HEARING-IMPAIRED	SCHOOLS
	GRADE-SCHOOL	VOCATIONAL

Table 4.2 – Two rare words with their rank and their 5 nearest words with respect to the Hellinger distance, for a symmetric window of 1 and 10 context words.

4.4.2 Context Window Size

Except for semantic analogy questions, best performance are always obtained with symmetric context window of size 1. However, performance dramatically drop with this window size on the latter. It seems that a limited window size helps to find syntactic similarities, but a large window is needed to detect the semantic aspects. The best results are thus obtained with a symmetric window of ten words on the semantic analogy questions task. This intuition is confirmed by looking at the nearest neighbors of certain rare words with different sizes of context. In Table 4.2, we can observe that a window of one context word brings together words that occur in a same syntactic structure, while a window of ten context words will go beyond that and adds semantic information. With only one word of context, Lake *Baikal* is therefore neighbor to other lakes, and the word *special-needs* is close to other words composed of two words. With ten words of context, the nearest neighbors of *Baikal* are words in direct relation to this location, *i.e.* these words cannot match with other lakes, like Lake *Titicaca*. This also applies for the word *special-needs*, where we find words related to the educational meaning of this word. This could explain why the symmetric window of one context word gives the best results on the word similarity and syntactic tasks, but performs very poorly on the semantic task.

4.4.3 Analysis Findings

The analysis of the context reveals that word similarities can even be found with extremely sparse word vector representations. But these representations lack semantic information since they perform poorly on the word analogy task involving semantic questions. A symmetric

window of five or ten context words seems to be the best option to capture both syntactic and semantic information about words. The average number of context words is much larger within these parameters, which justifies the need of dimensionality reduction. Furthermore, this analysis show that a large vocabulary of context words is not necessary to achieve significant improvements. Good performance on syntactic and similarity tasks are reached with our smallest vocabularies of context words. Using instead a distribution of a limited number of rare words increases performance on the semantic task while reducing performance on syntactic and similarity tasks.

4.5 Principal Component Analysis (PCA)

As discrete distributions are vocabulary size-dependent, using directly the distribution as a word representation is, in general, not really tractable for large vocabularies. This is even more true in the case of a large number of context words, distributions becoming less sparse. As dimensionality reduction technique, we choose to perform a principal component analysis (PCA) of the word co-occurrence probability matrix $\mathbf{C} \in \mathbb{R}^{|\mathcal{W}|\times|\mathcal{D}|}$. The word co-occurrence matrix $\mathbf{C}$ is defined as follows

$$\mathbf{C} = \begin{pmatrix} P(c_1|w_1) & \cdots & P(c_{|\mathcal{D}|}|w_1) \\ P(c_1|w_2) & \cdots & P(c_{|\mathcal{D}|}|w_2) \\ \vdots & \ddots & \vdots \\ P(c_1|w_{|\mathcal{W}|}) & \cdots & P(c_{|\mathcal{D}|}|w_{|\mathcal{W}|}) \end{pmatrix} = \begin{pmatrix} P_{w_1} \\ P_{w_2} \\ \vdots \\ P_{w_{|\mathcal{W}|}} \end{pmatrix}. \tag{4.12}$$

PCA uses an orthogonal transformation to convert a set of observations (the target words, i.e. rows of $\mathbf{C}$) of possibly correlated variables (the context words, i.e. columns of $\mathbf{C}$) into a set of values of linearly uncorrelated variables called *principal components* $\bar{\mathbf{C}}$ defined as

$$\bar{\mathbf{C}} = \mathbf{CA}, \tag{4.13}$$

where columns of $\mathbf{A} \in \mathbb{R}^{|\mathcal{D}|\times|\mathcal{D}|}$ are orthonormal vectors. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible). Keeping only the first d principal components gives a truncated transformation of $\mathbf{C}$, leading to $\bar{\mathbf{C}}_d \in \mathbb{R}^{|\mathcal{W}|\times d}$ where $d \ll |\mathcal{D}|$. Reducing dimensions means that redundancy in the data is eliminated. Redundancy doesn't mean that the variables are identical; it means that there is a strong correlation between them. PCA is usually done by *eigenvalue decomposition* (ED) of a data covariance (or correlation) matrix $\mathbf{C}^T\mathbf{C}$ or *singular value decomposition* (SVD) of $\mathbf{C}$, after normalizing the variables. Because we are dealing with discrete probability distributions, we use the *Bhattacharyya coefficient* to quantify the redundancy between two word context distributions. Taking the square root of $\mathbf{C}$, PCA thus learns to project word co-occurrence probability distributions to a lower dimensional manifold, while minimizing the reconstruction error according to the Hellinger

distance

$$\min_{\mathbf{A}_d \in \mathbb{R}^{|\mathcal{D}| \times d}} \sum_{i=1}^{|\mathcal{W}|} \left\| \sqrt{P_{w_i}} - \mathbf{A}_d \mathbf{A}_d^T \sqrt{P_{w_i}} \right\|_2 . \tag{4.14}$$

Covariance-based PCA of high-dimensional matrices can lead to round-off errors, and thus fails to properly approximate these high-dimensional matrices in low-rank matrices. And SVD will generally requires a large amount of memory to factorize such huge matrices. To overcome these barriers, we propose a dimensionality reduction based on a fast randomized SVD.

4.5.1 Eigen Decomposition (ED)

PCA can be done by ED of the covariance (or correlation) matrix $\mathbf{R}$. The symmetric, positive semi-definite matrix $\mathbf{R}$ can be rewritten as

$$\mathbf{R} = \mathbf{A}\mathbf{D}\mathbf{A}^T , \tag{4.15}$$

where $\mathbf{D}$ is the diagonal matrix of *eigenvalues* of $\mathbf{R}$ and $\mathbf{A}$ is the orthogonal matrix of *eigenvectors* of $\mathbf{R}$. As we use the *Bhattacharyya coefficient* to quantify the redundancy between context distributions, the matrix $\mathbf{R}$ is obtained as follows

$$\mathbf{R} = \sqrt{\mathbf{C}^T}\sqrt{\mathbf{C}}, \quad \mathbf{R} \in \mathbb{R}^{|\mathcal{D}| \times |\mathcal{D}|} . \tag{4.16}$$

With a limited size of context word dictionary $\mathcal{D}$ (thousands of words), this operation can be performed very quickly since it is highly parallelizable. Word embeddings are then obtained by projecting the transformed word distributions $\sqrt{P_{w_t}}$ into the d first principal components

$$\bar{\mathbf{C}} = \sqrt{\mathbf{C}}\mathbf{A}_d , \tag{4.17}$$

where the columns of $\mathbf{A}_d \in \mathbb{R}^{|\mathcal{D}| \times d}$ are the first d eigenvectors, and each row of $\bar{\mathbf{C}} \in \mathbb{R}^{|\mathcal{W}| \times d}$ is a word embedding.

4.5.2 Singular Value Decomposition (SVD)

The principal components transformation can also be associated with another matrix factorization, the SVD of $\sqrt{\mathbf{C}}$,

$$\sqrt{\mathbf{C}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4.18}$$

Here $\mathbf{\Sigma} \in \mathbf{R}^{|\mathcal{W}| \times |\mathcal{D}|}$ is a rectangular diagonal matrix of positive numbers, called the singular values. The columns of $\mathbf{U} \in \mathbf{R}^{|\mathcal{W}| \times |\mathcal{W}|}$ are orthogonal unit vectors called the left singular vectors, and the columns of $\mathbf{V} \in \mathbf{R}^{|\mathcal{D}| \times |\mathcal{D}|}$ are the right singular vectors. In terms of this factorization,

the matrix $\mathbf{R} = \sqrt{\mathbf{C}}^T\sqrt{\mathbf{C}}$ can be written as

$$\mathbf{R} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \tag{4.19}$$
$$= \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^T. \tag{4.20}$$

Consequently, the right singular vectors $\mathbf{V}$ are equivalent to the eigenvectors of $\mathbf{R}$, while the singular values are equal to the square roots of the eigenvalues. A truncated transformation of $\mathbf{C}$ can be obtained by considering only the first d largest singular values and their left singular vectors:

$$\bar{\mathbf{C}} = \sqrt{\mathbf{C}}\mathbf{V}_d \tag{4.21}$$
$$= \mathbf{U}_d\boldsymbol{\Sigma}_d\mathbf{V}_d^T\mathbf{V}_d \tag{4.22}$$
$$= \mathbf{U}_d\boldsymbol{\Sigma}_d. \tag{4.23}$$

Computing the SVD can be extremely time-consuming for the large-scale problems. Thus, we turn to randomized methods which offer significant speedups over classical methods.

Fast Randomized SVD

Halko et al. (2011) propose a two-stage algorithm that uses randomized techniques for computing a low-rank approximation of $\sqrt{\mathbf{C}}$.

Stage A We seek to find a matrix $\mathbf{Q}$ which approximates the range of the input matrix $\sqrt{\mathbf{C}}$, where the number of columns should be as few as possible. $\mathbf{Q}$ has orthonormal columns and

$$\sqrt{\mathbf{C}} \approx \mathbf{Q}\mathbf{Q}^T\sqrt{\mathbf{C}}. \tag{4.24}$$

Stage B Assuming we have found such a $\mathbf{Q}$, we can then compute an SVD of $\sqrt{\mathbf{C}}$ as follows:

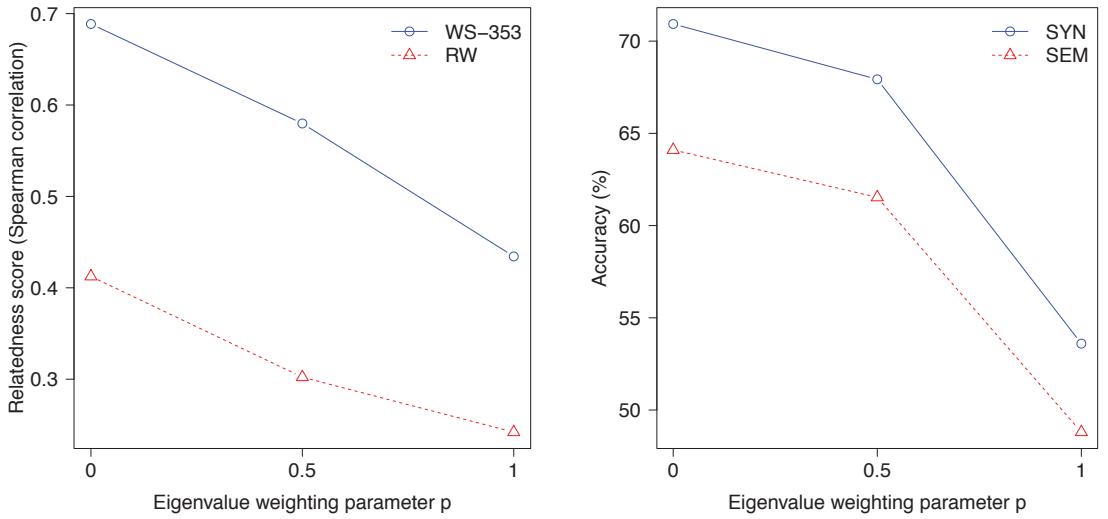
1. construct $\mathbf{B} = \mathbf{Q}^T\sqrt{\mathbf{C}}$,
2. compute SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \mathbf{S}\boldsymbol{\Sigma}\mathbf{V}^T$,
3. as $\sqrt{\mathbf{C}} \approx \mathbf{Q}\mathbf{Q}^T\sqrt{\mathbf{C}} = \mathbf{Q}\left(\mathbf{S}\boldsymbol{\Sigma}\mathbf{V}^T\right)$, we see that taking $\mathbf{U} = \mathbf{Q}\mathbf{S}$, we have computed a low rank approximation $\sqrt{\mathbf{C}} \approx \mathbf{Q}\mathbf{Q}^T\sqrt{\mathbf{C}}$.

When $\mathbf{Q}$ has few columns, this procedure is efficient because we can easily construct the reduced matrix $\mathbf{B}$ and rapidly compute its SVD.

4.5.3 Experimental Analysis

As seen in Section 4.4.1, we are dealing with very sparse distributions. PCA is about reducing dimensionality by removing redundant variables. It makes thus sense to use context vocabularies which provide variables with the less sparsity for performing Hellinger PCA over word probability distributions. In our experiments, we perform the dimensionality reduction over a context of words with probability $P(w_t) > 10^{-5}$, following the findings of Section 4.4.1.

Eigenvalue Weighting



(a) Word similarities datasets

(b) Word analogies datasets

Figure 4.5 – Performance on datasets with different eigenvalue weighting parameter p. Word embeddings dimension is 512. Context vocabulary interval is $]1; 10^{-5}]$ with a symmetric window of 10 words. Spearman rank correlation is reported on word similarity tasks. Accuracy is reported on word analogy tasks.

It has been shown that adding a parameter p to control the eigenvalues matrix $\mathbf{\Sigma}$ helps to get better word representations after SVD (Caron, 2001),

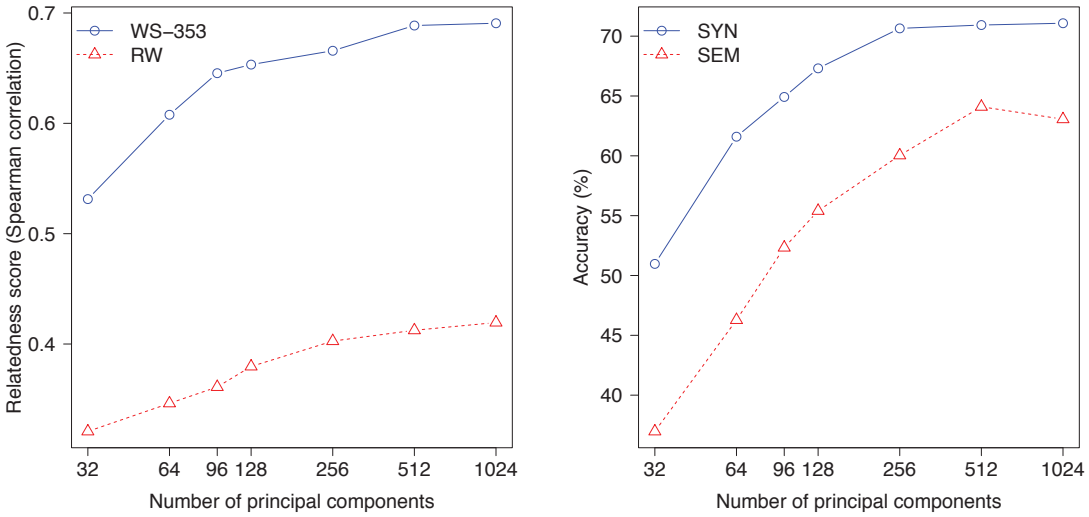
$$\tilde{\mathbf{C}}^p = \mathbf{U}_d \mathbf{\Sigma}_d^p. \tag{4.25}$$

Since $\mathbf{\Sigma}$ is diagonal and the eigenvalues are sorted in ascending order, setting $p < 1$ gives more emphasis to the later components of $\mathbf{U}$. We thus evaluate word embeddings with $p \in \{0, 0.5, 1\}$, where $p = 1$ corresponds to the traditional factorization, and $p = 0$ means that the eigenvalue matrix is dismissed. Results reported in Figure 4.5 show that the best performance is achieved when discarding the eigenvalue matrix. We also observe a significant drop when we use the traditional principal components (with $p = 1$). For the next sections, we will therefore define

the word embeddings as

$$\bar{\mathbf{C}} = \mathbf{U}_d \,. \tag{4.26}$$

Number of Dimensions



(a) Word similarities datasets

(b) Word analogies datasets

Figure 4.6 – Performance on datasets with different dimensions using context interval $]1;10^{-5}]$ with a symmetric window of 10 words. Dimensionality reduction has been obtained with the Hellinger PCA using randomized SVD. Spearman rank correlation is reported on word similarity tasks. Accuracy is reported on word analogy tasks.

When a dimensionality reduction method is applied, a number of dimensions needs to be chosen. This number has to be large enough to retain the maximum variability. It also has to be small enough for the dimensionality reduction to be truly meaningful and effective. We thus analyze the impact of the number of principal components from the Hellinger PCA of the co-occurrence matrix. Figure 4.6 reports performance on the benchmark datasets for different numbers of dimensions. On both tasks, we observe that performance is optimal with 512 dimensions. However, performance with about a hundred dimensions gives already competitive results.

Dense vs Sparse Word Representations

In Table 4.3, we compare performance on the benchmark datasets described in Section 4.3.2 between sparse word representations and the dense word representations obtained after Hellinger PCA. The ability of the PCA to summarize the information compactly leads to improved results on the both tasks, where performance is better than with no dimensionality

	SPARSE			DENSE		
Context Window Size	1	5	10	1	5	10
WORDSIMILARITY-353	0.39	0.41	0.39	0.53	0.67	0.69
RARE WORD	0.17	0.13	0.11	0.38	0.42	0.41
SYNTACTIC ANALOGIES	51.9	54.1	52.6	50.7	68.8	70.9
SEMANTIC ANALOGIES	21.8	50.3	58.1	18.3	54.5	64.1

Table 4.3 – Performance on word similarity and word analogy datasets using a context word vocabulary with $P(w_t) > 10^{-5}$, and different window sizes. For dense representations, we report results using the first 512 principal components after Hellinger PCA with SVD. Spearman rank correlation is reported for similarities. Accuracy is reported for analogies.

reduction. This observation is especially true for finding word similarities, where the improvement is the most significant.

Qualitative Analysis

ASSOCIATED (n°866)	**ABBEY** (n°2980)	**BAIKAL** (n°37415)	**ZIDANE** (n°49155)	**SPECIAL-NEEDS** (n°165997)
CONSISTENT	PRIORY	AMUR	ZINEDINE	DAYCARE
CLOSELY	MONASTERY	SIBERIAN	RONALDINHO	HEARING-IMPAIRED
ALONG	FRIARY	URAL	MARADONA	PRESCHOOL
DEALT	ABBOT	TITICACA	FIGO	SCHOOL-AGE
DEALING	CLUNY	ALTAI	MESSI	SCHOOLS
FAMILIAR	NUNNERY	URALS	RONALDO	KINDERGARTENS
CONTRASTED	CISTERCIAN	SIBERIA	CANTONA	SCHOOLCHILDREN
INTIMATELY	BENEDICTINE	YAKUTIA	PLATINI	VOCATIONAL
SYNONYMOUS	CASTLE	VOLGA	CRUYFF	BOARDING
ASSOCIATING	CARTHUSIAN	KAMCHATKA	PELÉ	MIDDLE-SCHOOL

Table 4.4 – Words with their rank in $\mathcal{W}$ and their 10 nearest neighbors in the word embedding space (according the the Euclidean metric). Dimension of word embeddings is 128. A window of 10 context words has been used to build the word co-occurrence matrix.

Table 4.4 shows the ten nearest neighbors of a few randomly chosen words in the word embedding space. Word embeddings are the 128 first principal components of the Hellinger PCA of the word co-occurrence matrix built with a context window of ten words. First, we see that our method produces appealing word embeddings for both frequent and rare words. Furthermore, we see that both syntactic and semantic information about words are captured. For instance, nearest neighbors of *associated* are other similar verbs in the past particle form,

but also adverbs and adjectives with the same meaning. For *baikal* and *special-needs* that have been analyzed in Table 4.2, we see that the nearest neighbors are now a combination of syntactically and semantically related words. *Abbey* is close to other buildings whether or not they are religious ones (e.g. *castle*), and also close to words related to religion that are not buildings (e.g. *abbot* or *benedictine*). For a person like *zidane*, both nationality and occupation have been captured as we see that other French football players are in the nearest neighbors (e.g. *platini* or *cantona*), along with other well famous players.

4.6 Supervised Evaluation Tasks

While benchmark datasets described in Section 4.3.2 are useful for evaluating the quality of word embeddings, these tasks are not relevant in a real word context. Using word embeddings as feature proved that it can improve the generalization performance on several NLP tasks (Turian et al., 2010; Collobert et al., 2011; Chen et al., 2013). Good word embeddings should be those which give the best performance in such real world applications. We thus evaluate our word embeddings on two standard word tagging tasks: chunking (CHUNK) and Named Entity Recognition (NER). For that purpose, we trained a neural network with a window approach as in Collobert et al. (2011).

4.6.1 Tasks Description

Chunking

Also called shallow parsing, chunking aims at labeling segments of a sentence with syntactic constituents such as noun or verbal phrases (NP or VP). Each word is assigned only one unique tag, often encoded as a begin-chunk (e.g., B-NP) or inside-chunk tag (e.g., I-NP). Chunking is often evaluated using the CoNLL 2000 shared task[4]. Sections 15–18 of Wall Street Journal data are used for training and section 20 for testing. Validation is achieved by splitting the training set. As a benchmark, we report a F-1 score around 94.3% coming from several systems based on second-order random fields (Sha and Pereira, 2003; McDonald et al., 2005; Sun et al., 2008). These systems use features composed of words, part-of-speech tags, and other tags.

Named Entity Recognition

This task labels atomic elements in the sentence into categories such as “PERSON" or “LOCATION". The CoNLL 2003 setup[5] is a NER benchmark data set based on Reuters data. The contest provides training, validation and testing sets. As a benchmark, we report the system of Ando et al. (2005) which reached 89.31% F1 with a semi-supervised approach. Their system uses many hand-crafted features (words, part-of-speech tags, suffixes and prefixes and

[4] See http://www.cnts.ua.ac.be/conll2000/chunking.
[5] See http://www.cnts.ua.ac.be/conll2003/ner/.

CHUNK tags), but overall is less specialized than CoNLL 2003 challengers.

For both tasks, we adopt the BIO2 annotation standard.

4.6.2 Neural Network Approach

We learn a neural network approach for tagging with a label each word in a given sentence. Using a sliding window approach, embeddings of the word to tag with some context words are given to a nonlinear classification model. We train the model by maximizing the log-likelihood at the sentence level.

Sliding window

Context is crucial to characterize word meanings. We thus consider n context words around each word w_t to be tagged, leading to a window of $N = (2n+1)$ words, $\{w_{t-n}, \ldots, w_t, \ldots, w_{t+n}\}$. We first define an embedding layer ϕ^1_θ (see Section 2.1.3) which maps each word in a given window to a d^{wrd}-dimensional vector. By concatenating the resulting vectors, we obtain a $d^{\text{wrd}} \times N$ vector, which aims at characterizing the middle word w_t in this window:

$$\phi^1_\theta(w_t) = \left[\mathbf{E}_{w_{t-n}}; \ldots; \mathbf{E}_{w_t}; \ldots; \mathbf{E}_{w_{t+n}}\right] \in \mathbb{R}^{(d^{\text{wrd}} \times N)}. \tag{4.27}$$

Given a complete sentence of T words $s = \{w_1, \ldots, w_T\}$, we can obtain for each word w_t a context-dependent representation by sliding over all the possible windows in the sentence. Each window representation is then given to a nonlinear classifier which gives a score for each possible tag y_k

$$\phi_\theta(w_t) = \mathbf{W}^2 h\Big(\mathbf{W}^1 \phi^1_\theta(w_t) + \mathbf{b}^1\Big) + \mathbf{b}^2 \quad \in \mathbb{R}^K \tag{4.28}$$

where $\theta = \left\{\mathbf{E}, \mathbf{W}^1, \mathbf{W}^2, \mathbf{b}^1, \mathbf{b}^2\right\}$ are the trainable parameters of the network, and $h(\cdot)$ is the activation function. $\mathbf{E}$ is the word embedding matrix which is initialized with our pre-trained embeddings, and $\mathbf{W}^1 \in \mathbb{R}^{n_{\text{hu}} \times (d^{\text{wrd}} \times N)}$, $\mathbf{W}^2 \in \mathbb{R}^{K \times n_{\text{hu}}}$, $\mathbf{b}^1 \in \mathbb{R}^{n_{\text{hu}}}$ and $\mathbf{b}^2 \in \mathbb{R}^K$ are the classifier parameters, with K the number of classes.

CRF-type inference

There exists strong dependencies between tags in a sentence: some tags cannot follow other tags. To take the sentence structure into account, we want to encourage valid paths of tags during training, while discouraging all other paths. Considering the matrix of scores outputted by the network $\phi_\theta(s) \in \mathbb{R}^{T \times K}$ for a given sentence s of T words, we train a conditional random field (CRF). At test time, the best path minimizing the sentence score is inferred with the Viterbi algorithm (Viterbi, 1967). The element $[\phi_\theta]_{y_k, w_t}$ of the matrix is the score outputted by the network for the tag y_k at the word $w_t \in s$. We introduce a transition matrix $\mathbf{Z}$, where

$[\mathbf{Z}]_{y_i,y_j}$ is the score for jumping from y_i to y_j tags in successive words, and an initial score $[\mathbf{Z}]_{y_i,0}$ for starting from the tag y_i. As the transition scores are going to be trained, we define $\tilde{\theta} = \theta \cup \mathbf{Z}$. The score of a sentence s along a path of tags $y = \{y_1, \ldots, y_T\}$ is then given by the sum of transition scores and classification scores:

$$\phi_{\tilde{\theta}}(s, y) = \sum_{t=1}^{T} \left([\mathbf{Z}]_{y_{t-1},y_t} + [\phi_\theta]_{y_t,w_t} \right). \tag{4.29}$$

We normalize this score over all possible tag paths $\bar{y}$ using a softmax, and we interpret the resulting ratio as a conditional tag path probability. Taking the log, the conditional probability of the true path y is therefore given by:

$$\log P_{\tilde{\theta}}(y|s) = \phi_{\tilde{\theta}}(s, y) - \underset{\forall \bar{y}}{\text{logadd}}\, \phi_{\tilde{\theta}}(s, \bar{y}), \tag{4.30}$$

where we adopt the notation

$$\underset{i}{\text{logadd}}\, z_i = \log\left(\sum_i e^{z_i}\right). \tag{4.31}$$

Computing the log-likelihood efficiently is not straightforward, as the number of terms in the logadd grows exponentially with the length of the sentence. It can be computed in linear time with the Forward algorithm, which derives a recursion similar to the Viterbi algorithm (see Rabiner (1989)). This allows for maximizing the log-likelihood over all the training pairs (s, y). In contrast to classical CRF, all parameters $\tilde{\theta}$ are trained in a end-to-end manner, by backpropagation through the Forward recursion, following Collobert et al. (2011). At inference time, the Viterbi algorithm is used to find the best path for a given sentence s among all the possible path $\bar{y}$:

$$\underset{\bar{y}}{\text{argmax}}\, \phi_{\tilde{\theta}}(s, \bar{y}). \tag{4.32}$$

4.6.3 Experimental Setup

Word Embeddings

For the purpose of a fair comparison, we choose to compare only with other word embeddings that have been trained on the same corpus than our word embeddings (i.e. the entire English Wikipedia). We thus compare with word embeddings from SENNA[6] and word embeddings trained with a Skip-gram model. For all embeddings, we train two versions of our system. In the first version, word embeddings are considered as fixed inputs, as a classical approach would use them. In the second version, we leverage the deep architecture of our system and we tune the word embeddings for the given task.

[6] Available at http://ml.nec-labs.com/senna/

SENNA SENNA's embeddings covers 130,000 words with 50 dimensions for each word. They were trained for about two months, over Wikipedia, using a NNLM with a pairwise ranking approach (see Section 3.2.1). These embeddings have, furthermore, been fine-tuned for NER and chunking. This makes those embeddings highly competitive for these two tasks.

Skip-gram Nowadays, the Skip-gram model (see Section 3.2.3 for details) is considered as the state-of-the-art method for getting word embeddings. This method has become popular thanks to the `word2vec`[7] toolkit, which provides a user-friendly and efficient implementation of the method. We thus use this toolkit to compute word embeddings on the same Wikipedia corpus using the same context window sizes (1, 5 and 10 words). Embeddings have been trained with negative sampling (with 5 negative samples) and sub-sampling of frequent words (with $t = 10^{-5}$).

As SENNA's embeddings are only available with $d^{\text{wrd}} = 50$ dimensions, we thus compute word embeddings with Skip-gram and with Hellinger PCA (H-PCA) in that dimension for this experiment. To highlight the importance of the Hellinger metric, we also compute word embeddings using the matrix $\mathbf{C}$ instead of $\sqrt{\mathbf{C}}$. These embeddings are named Euclidean PCA (E-PCA).

Other Features and Hyper-parameters

For chunking, the networks are fed only one raw feature: the word embeddings. In NER where the goal is to tag entities, we use an additional raw feature: a capital letter feature. The "caps" feature tells if each word was in lowercase, was all uppercase, had first letter capital, or had at least one non-initial capital letter. Each caps feature is mapped to an embeddings of size 5, which is learned during training as the other parameters. No other feature has been used to tune the models. This is a main difference with other systems which usually include more features, such as part-of-speech tags, prefixes and suffixes or gazetteers (only for NER). We also introduce a special "PADDING" word for context at the beginning and the end of each sentence. Hyper-parameters were tuned by early-stopping on the validation set. We selected $n = 2$ context words leading to a window of 5 words. The number of hidden units is nhu = 300. For chunking, the best context window size is 1. It makes sense since it is purely a syntactic task. For NER, the best context window size is 5. This task combines both syntactic and semantic aspects, it is thus not surprising that this context window size gives the best performance. Since the capacity of our tagging model mainly comes from the word embeddings, we use word embedding dropout as a regularization, following (Legrand and Collobert, 2014).

[7] Available at https://code.google.com/archive/p/word2vec/

Embeddings Normalization

Word embeddings are continuous vector spaces that are not necessarily in a bounded range. To avoid saturation issues in the network architectures, embeddings need to be properly normalized. Considering the matrix of word embeddings **E**, we normalize the rows of **E**.

4.6.4 Results

	CHUNK		NER	
BENCHMARK	94.29		89.31	
Embeddings	*Fixed*	*Tuned*	*Fixed*	*Tuned*
SENNA	93.35 ± 0.06	94.18 ± 0.05	88.54 ± 0.13	89.55 ± 0.15
SKIP-GRAM	91.83 ± 0.09	93.66 ± 0.05	87.50 ± 0.20	88.96 ± 0.24
E-PCA	90.46 ± 0.09	93.18 ± 0.07	85.56 ± 0.16	87.50 ± 0.07
H-PCA	92.74 ± 0.10	94.19 ± 0.07	88.07 ± 0.25	89.45 ± 0.09

Table 4.5 – Performance comparison on named entity recognition (NER) and chunking (CHUNK) tasks with different embeddings. The first column reports results with the original embeddings. The second column reports results after fine-tuning the embeddings for the task. Results are reported in F1 score (mean ± standard deviation of ten training runs with different initialization). H-PCA is for Hellinger PCA, while E-PCA stands for Euclidean PCA.

H-PCA's embeddings

Results summarized in Table 4.5 reveal that performance on both tasks can be as good with word embeddings from a word co-occurrence matrix decomposition as with a NNLM. The F1 scores with the H-PCA tuned embeddings are as good as SENNA tuned embeddings, which yields state-of-the-art results on both tasks. When the embeddings are not tuned, H-PCA's embeddings are slightly outperformed by the SENNA's embeddings, which is not surprising as the latter are already fine-tuned for those tasks. However, HPCA's embeddings always achieve better results than Skip-gram's embeddings on both tasks. It is worth mentioning that on both tasks, H-PCA's embeddings outperform the E-PCA's embeddings, demonstrating the value of the Hellinger distance.

Embedding Fine-Tuning

By leveraging the deep architecture of our system, we can tune the word embeddings by backpropagating the error through the embedding layer. Results in Table 4.5 show that tuning the embeddings increases the general performance on both chunking and NER tasks. This is a great advantage compared to conventional approaches where these embeddings remain unchanged, as they are not structurally able to fine-tune them.

CPU Time

Embedding size	$d^{\text{wrd}} = 50$		$d^{\text{wrd}} = 512$	
SENNA	2 months		-	
Context Window Size	1	5	1	5
SKIP-GRAM	39 min	62 min	73 min	138 min
H-PCA	24 sec	77 sec	229 sec	660 sec

Table 4.6 – CPU time for computing word embeddings. *min* is for minutes and *sec* for seconds.

The Hellinger PCA is very fast to compute[8]. We report in Table 4.6 the time needed to compute the embeddings used for this experiment (50 dimensions). For this benchmark we used Intel i7 3770K 3.5GHz CPU. Although word embeddings with `word2vec` are also quickly computed, we see that a randomized SVD is done in a few seconds for getting the 50 first principal components. When setting a larger number of components, this operation is a bit longer but it provides all embedding sizes in a single operation. This is not the case with the Skip-gram model, where `word2vec` needs to be relaunched for each new embedding size. We intentionally do not mention the time for building the word co-occurrence matrix in Table 4.6. The sparse matrix $\mathbf{C}$ is stored as a set of triplets $\{w_t, c_t, n(w_t, c_t)\}$, making our model much more scalable than NNLM where each pair $\{w_t, c_t\}$ is treated separately. Counts $n(w_t, c_t)$ can easily be aggregated in time over multiple corpora of text. This counting operation is furthermore highly parallelizable. For each new matrix $\mathbf{C}$, just the randomized SVD needs to be run for extracting new word embeddings, while NNLM need to run a complete process.

4.7 Embedding Inference

While inference is not possible with NNLM-based methods, one main advantage of computing word embeddings through Hellinger PCA is the possibility to infer embeddings for unseen words. Given a new word w_{new}, one only needs to count its context words over a large corpus of text to build the distribution $\sqrt{P_{w_{\text{new}}}}$. Embedding for that word is then computed with the eigenvectors by projecting its word co-occurrence distribution $\sqrt{P_{w_{\text{new}}}}$ into a dimensionally reduced feature space,

$$\mathbf{e}_{w_{\text{new}}} = \mathbf{A}_d^T \sqrt{P_{w_{\text{new}}}} \tag{4.33}$$

where $\mathbf{A}_d$ are the d first eigenvectors, and $\mathbf{e}_{w_{\text{new}}} \in \mathbb{R}^d$ is the resulting embedding for w_{new}. The few examples in Table 4.7 show that this nice feature can be extrapolated to phrases. By building word co-occurrence distributions for phrases in the same way, phrase embeddings are easily computed. It thus becomes a valuable asset which offers a simple approach for

[8]The randomized SVD is done with MATLAB. We use the implementation of Mark Tygert available at http://tygert.com/software.html.

NEW YORK CITY	PRESIDENT OF THE UNITED STATES	HOME PLATE
MANHATTAN	LINCOLN	INFIELD
BROOKLYN	REAGAN	DUGOUT
MINNEAPOLIS	TRUMAN	BATTER
HARLEM	APPOINTEE	FIELDERS
CHICAGO	PRESIDENT-ELECT	OUTFIELD
BOSTON	PRESIDENT	CREASE
D.C.	NIXON	ELBOW
WASHINGTON	NOMINATING	SIDELINE
NYC	EISENHOWER	GOALPOST
THEATER	SENATOR	BALL

Table 4.7 – Three phrases with their 10 nearest words with respect to the Euclidean distance between the inferred phrase embeddings and the pre-computed word embeddings. Word co-occurrence statistics for these phrases are built using a context window of 10 words with a vocabulary containing the 6961 most frequent words.

embedding sequences of words, such as entities or multiword expressions.

4.8 Implementation

We implemented a standalone version of the Hellinger PCA for computing word embeddings, written in the C++ language[9]. The runtime version contains about 2,800 lines of C++ code, and it has been designed to run on any standard computer. The toolkit provides 7 different tools for computing the following step:

1. **Corpus pre-processing**. Given a tokenized corpus of text, `preprocess` implements lowercase conversion and/or replaces all numbers with a special token (0).

2. **Vocabulary extraction**. Given a pre-processed corpus, `vocab` extracts all words with their respective frequency.

3. **Getting co-occurrence probability matrix**. Given the pre-processed corpus and the extracted vocabulary, `cooccurrence` constructs word-word co-occurrence statistics from the corpus. Several options are available for setting the word context vocabulary, the context window size, and for discarding target words with a low frequency of occurrence.

4. **Performing Hellinger PCA**. When the co-occurrence matrix is ready and given the number of components to keep, `pca` runs the randomized SVD with respect to the Hellinger distance. This tool uses the external `redsvd` library[10], which implements the

[9] Available at https://github.com/rlebret/hpca
[10] Available at https://code.google.com/p/redsvd/.

randomized SVD using `Eigen3`[11].

5. **Extracting word embeddings**. `embeddings` generates word embeddings from the Hellinger PCA for a given dimension. An option for eigenvalue weighting is available, as well as embeddings normalization.

6. **Evaluating word embeddings**. `eval` provides a quick evaluation of the word embeddings produced by `embeddings` for an English corpus. It includes all datasets described in Section 4.3.2.

7. **Computing word embeddings nearest neighbors**. `neighbors` is an exploratory tool to evaluate word embeddings quality. Given a word, it computes its nearest neighbors according to the Euclidean distance between their embeddings.

The three first steps which led to the creation of the word co-occurrence matrix are all highly parallelizable. POSIX Threads is thus used to allow parallel execution and speed up the process. Users can also control the memory usage when getting the co-occurrence matrix.

4.9 The Revival of Count-based Methods

After introducing our count-based model for getting word embeddings, other new count-based approaches have emerged. In this section, we introduce two recent works that propose alternative approaches to build the word co-occurrence matrix.

4.9.1 SVD over Shifted Positive Point Mutual Information

Levy and Goldberg (2014) show that the Skip-gram model implicitly factorizes a word co-occurrence matrix, where each entry is the pointwise mutual information (PMI) of the word and context pairs, shifted by a global constant. PMI measures the association between a word w_t and a context c_t,

$$PMI(w_t, c_t) = \log \frac{P(w_t, c_t)}{P(w_t)P(c_t)} = \log \frac{n(w_t, c_t) \cdot |\mathcal{S}|}{n(w_t)n(c_t)}. \tag{4.34}$$

The positive PMI (PPMI) has shown to be a better alternative for word representation (Bullinaria and Levy, 2007), as it allows sparsity and consistency,

$$PPMI(w_t, c_t) = \max\big(PMI(w_t, c_t), 0\big). \tag{4.35}$$

Inspired by the success of negative sampling in Skip-gram, the authors then propose a shifted version of PPMI,

$$SPPMI(w_t, c_t) = \max\big(PMI(w_t, c_t) - \log(k), 0\big), \tag{4.36}$$

[11] See http://eigen.tuxfamily.org/.

where k is a prior on the probability of observing a positive example (an actual occurrence of (w_t, c_t) in the corpus $\mathcal{S}$) versus a negative example. After building the Shifted PPMI matrix, SVD is used for dimensionality reduction.

4.9.2 Global Vectors (GloVe)

In contrast with our model, Pennington et al. (2014) suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves. A training objective is then defined to learn word embeddings such that their dot product equals the logarithm of the their co-occurrence probability,

$$\mathbf{e}_{w_i} \cdot \tilde{\mathbf{e}}_{w_j} + b_{w_i} + \tilde{b}_{w_j} = \log\big(n(w_i, w_j)\big) \quad \forall (w_i, w_j) \in \mathcal{S}\,, \tag{4.37}$$

where $\mathbf{e}_{w_i}$ and $\tilde{\mathbf{e}}_{w_j} \in \mathbb{R}^{d^{\text{wrd}}}$ are word embeddings, b_{w_i} and $\tilde{b}_{w_j}$ are additional scalars. Because the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. The model is fit to minimize a weighted least square loss, giving more weight to frequent (w_i, w_j) pairs. At the end, each word w_t has two different embeddings $\mathbf{e}_{w_t}$ and $\tilde{\mathbf{e}}_{w_t}$. The authors uses the summation of the two word vectors as final embedding. As the Skip-gram model, the main drawbacks of this model are that it does not allow inference of unseen words, and it trains one embedding size at the time.

4.10 Conclusion

We have demonstrated that appealing word embeddings can be obtained by computing a *Hellinger PCA* of the word co-occurrence matrix. While a NNLM can be painful and long to train, we can get a word co-occurrence matrix by simply counting words over a large corpus of text. The resulting embeddings give similar results on NLP tasks, even from a word co-occurrence matrix computed with only a relatively small context vocabulary (i.e. a small number of columns). It reveals that having a significant, but not too large set of common words, seems sufficient for capturing most of the syntactic and semantic characteristics of words. As PCA of a such matrix is really fast to compute, our method gives an interesting and practical alternative to NNLM for generating word embeddings. By leveraging the deep architecture of neural networks, we also show that existing embeddings can be fine-tuned to a specific task which leads to improve general performance for this task. Last but not least, this method enables inference of unseen words or phrases.

5 Towards Phrase Embeddings

While there has been a lot of effort to capture the meaning of words, distributed representations of phrases still remain a challenge. Many recent works are however based on distributed representations of phrases to tackle a wide range of applications in NLP: machine translation (Bahdanau et al., 2015; Zhao et al., 2015), constituency parsing (Legrand and Collobert, 2014) or sentiment analysis (Socher et al., 2013). In all these works, phrase representations are learned from composition of word embeddings. There is therefore a clear need for word embeddings that can be easily extrapolated to meaningful phrase representations.

We argue that distributed representation and composition must go hand in hand, i.e., they must be mutually learned. We present a model that learns to capture meaning of words in distributed representations using a low-rank approximation of a large word co-occurrence matrix. We choose to stochastically perform this low-rank approximation (with an autoencoder network) which enables the model to simultaneously train these representations to compose for producing representations of phrases (see Figure 5.1). As composition function, we choose a simple weighted addition for its simplicity and for enabling sequences of words with different lengths to be represented in a common vector space. Aside from generating distributed representations of words and phrases, this model gives an encoding function (represented by a matrix) which can be used to encode new words or even phrases based on their co-occurrence counts. This offers two different alternatives for phrase representations: (1) representation for a query phrase can be inferred by averaging vector representations of its words (only if they all were in the training set), or (2) by using its word co-occurrence statistics.

Evaluation on the popular word similarity and analogy tasks demonstrate the capability of our joint model for capturing as good distributed representations as with Hellinger PCA. We then introduce a novel task for evaluating phrase representations. Given a phrase representation, the objective is to retrieve the words that compose the phrase. We compare our model against other state-of-the-art methods for distributed word representations which capture meaningful linear substructures (Mikolov et al., 2013a; Pennington et al., 2014). We show that our model achieves similar performance on word evaluation tasks, but that it outperforms other methods on the phrase evaluation task.

5.1 Related Work

Given representations of words in a vector space, techniques for combining them have been proposed to get representations of phrases or sentences. These compositional models involve vector addition or multiplication (Mitchell and Lapata, 2010). Such simple compositions have shown to perform competitively on the paraphrase detection and phrase similarity tasks (Blacoe and Lapata, 2012). More sophisticated approaches use techniques from logic, category theory, and quantum information (Clark et al., 2008). Others use the syntactic relations between words to treat certain words as functions and other as arguments such as adjective-noun composition (Baroni and Zamparelli, 2010) or noun-verb composition (Grefenstette et al., 2013). Recursive neural network model for semantic compositionality has also been proposed (Socher et al., 2012), where each word has a matrix-vector representation: the vector captures its meaning (as it is initialized with a pre-trained distributed representation), while the matrix learns through a parse tree how it modifies the meaning of the other word that it combines with.

All these methods learn to compose pre-trained word embeddings. In contrast, we propose to simultaneously learn word embeddings and the composition function.

5.2 Hellinger PCA with Autoencoder

Inspired by the success of Hellinger PCA for computing meaningful word embeddings (see Chapter 4), we propose to stochastically perform the low-rank approximation. For this purpose, we use an autoencoder with only linear activation to find an optimal solution related to the Hellinger PCA (Bourlard and Kamp, 1988). Replacing the PCA by an autoencoder allows us to learn jointly a cost function which constrains the word information to be kept by summation. An autoencoder is employed to represent words in a lower dimensional space. It takes a distribution $\sqrt{P_{w_t}}$ as input, encodes it in a more compact representation, and is trained to reconstruct its own input from that representation:

$$\left\| \sqrt{P_{w_t}} - g\left(f\left(\sqrt{P_{w_t}}\right)\right) \right\|_2, \tag{5.1}$$

where $\phi_\theta = g\left(f\left(\sqrt{P_{w_t}}\right)\right)$ is the output of the network, $f(\cdot)$ is the encoding function which maps distributions in a d^{wrd}-dimension (with $d^{\text{wrd}} << |\mathcal{D}|$), and $g(\cdot)$ is the decoding function. $f\left(\sqrt{P_{w_t}}\right)$ is a distributed representation that captures the main factors of variation in the data as the Hellinger PCA does. Here, encoder $f(\cdot)$ and decoder $g(\cdot)$ are defined as follows:

$$f(\mathbf{x}) = \mathbf{U}\mathbf{x}, \; g(\mathbf{x}) = \mathbf{V}^T\mathbf{x}, \tag{5.2}$$

where $\mathbf{x} \in \mathbb{R}^{|\mathcal{D}|}$, and $\mathbf{U}$ and $\mathbf{V} \in \mathbb{R}^{d^{\text{wrd}} \times |\mathcal{D}|}$. We see that we can reformulate Equation 5.1 as follows:

$$\left\| \sqrt{P_{w_t}} - \mathbf{V}^T \mathbf{U} \sqrt{P_{w_t}} \right\|_2 , \tag{5.3}$$

which correspond to the same minimization function than with the Hellinger PCA (see Equation 4.14). The autoencoder parameters $\theta = \{\mathbf{U}, \mathbf{V}\}$ are trained by backpropagation using stochastic gradient descent.

5.3 Joint Learning with Summation

Interesting compositionality properties have been observed from models based on the addition of representations (Mikolov et al., 2013b). An exhaustive comparison of different composition functions has indeed revealed that an additive model performs well on pre-trained word representations (Mitchell and Lapata, 2010). Because our word representations are learned from linear operations, the inherent structure of these representations is linear. To combine a sequence of words into a common vector space, we then simply apply an element-wise addition of their vector representations. This approach makes sense and works well when the meaning of a text is literally "the sum of its parts". This is usually the case with noun and verb phrase chunks. For example, into phrases such as "the red cat" or "struggle to deal", each word independently has its proper meaning. Distributed representations for such phrase chunks must retain information from the individual words. An objective function is thus defined to learn how to combine the word vector representations, while keeping the maximum information from the original vectors. An operation as simple as a weighted sum will probably fail for sequences where individual words act as operators that modify the meaning of another word, or for multiword expressions. Other more complex functions could be chosen to also include such cases, but we choose to propose a much simpler model (i.e., averaging the word representations) to get phrase chunk representations with unsupervised learning. In this work, we therefore focus on noun and verb phrase chunks.

5.3.1 Training an Additive Model

We define $s = \{w_1, \ldots, w_T\} \in \mathcal{S}$ a phrase chunk of T words, with $\mathcal{S}$ a set of phrase chunks. By feeding all $\sqrt{P_{w_t}}$ into the autoencoder, a representation $\mathbf{x}_{w_t} \in \mathbb{R}^{d^{\text{wrd}}}$ of each word $w_t \in \mathcal{W}$ is obtained:

$$\mathbf{x}_{w_t} = f\left(\sqrt{P_{w_t}}\right). \tag{5.4}$$

By an element-wise addition, a representation of the phrase chunk s can be calculated as:

$$\mathbf{x}_s = \frac{1}{T} \sum_{w_t \in s} \mathbf{x}_{w_t} . \tag{5.5}$$

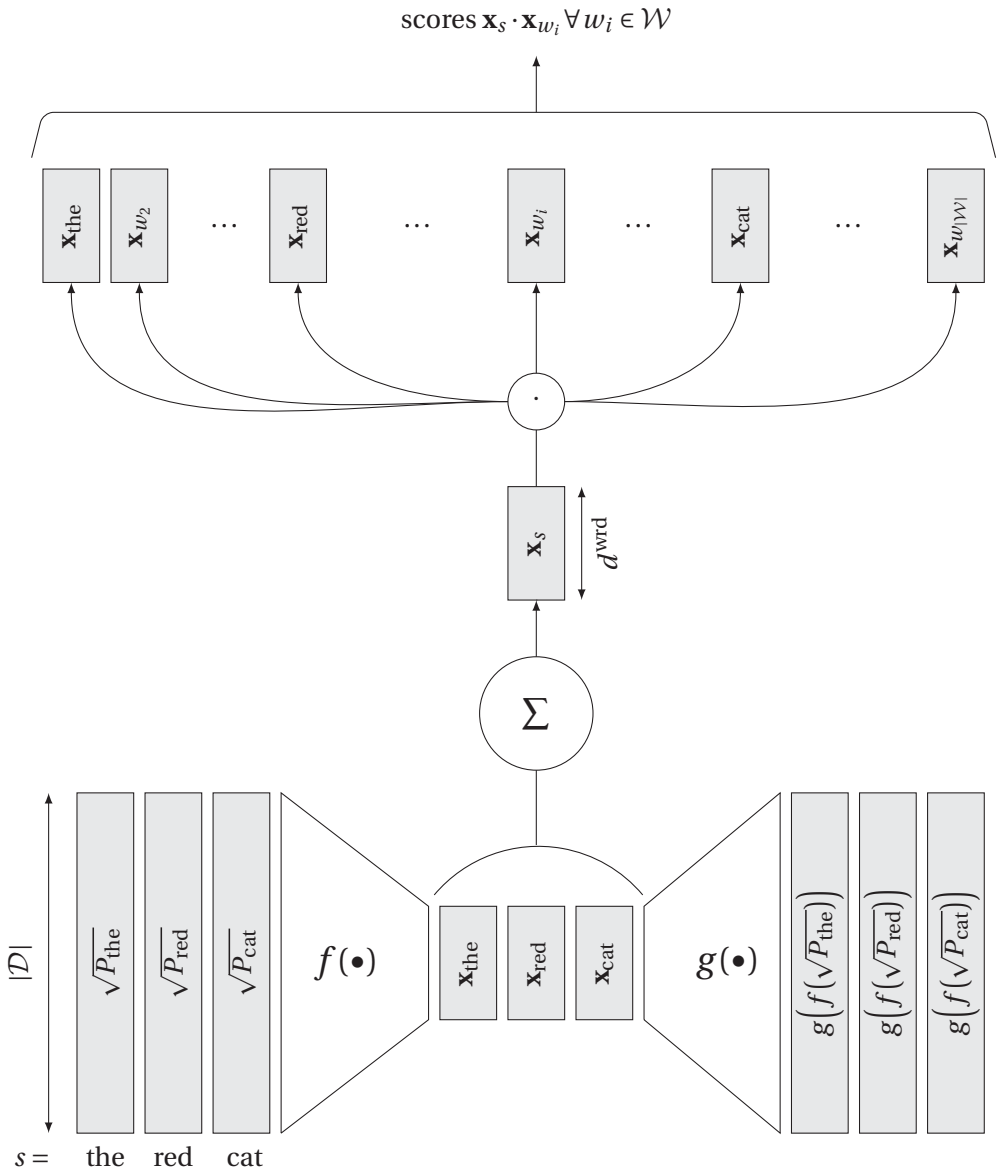


Figure 5.1 – Architecture for the joint learning of word representations and their summation. Considering the noun phrase $s =$ `the red cat`, each word $w_t \in s$ is represented as the square root of its co-occurrence probability distribution $\sqrt{P_{w_t}}$. These are the inputs given to an autoencoder which encodes them in a lower dimension $\mathbf{x}_{w_t} \in \mathbb{R}^{d^{\text{wrd}}}$. These new representations are then given to a decoder which is trained to reconstruct the initial inputs. This is the first objective function. The second objective is to keep information when words are summed. All $\mathbf{x}_{w_t}$ are averaged together to represent s in the same space as w_t. A dot product between the phrase representation $\mathbf{x}_s$ and all the other word representations from the dictionary $\mathcal{W}$ is calculated. These scores are trained to be high for words that appear in s and low for the others.

In predictive-based model, such as the Skip-gram model, the objective is to maximize the likelihood of a word based on other words in the same sequence (see Section 3.2.3). Instead, our training is slightly different in the sense that we aim at discriminating whether words are in the phrase chunk or not. An objective function is thus defined to encourage words w_t which appear in the chunk s to give high scores when calculating the dot product between $\mathbf{x}_{w_t}$ and $\mathbf{x}_s$. On the other hand, these scores must be low for words $w_i \notin s$ that do not appear in the chunk. We train this problem with a ranking-type cost:

$$\sum_{s \in \mathcal{S}} \sum_{w_t \in s} \sum_{\substack{w_i \in \mathcal{W} \\ w_i \notin s}} \max\left(0, 1 - \mathbf{x}_s \cdot \mathbf{x}_{w_t} + \mathbf{x}_s \cdot \mathbf{x}_{w_i}\right). \tag{5.6}$$

5.3.2 Joint Learning with Negative Sampling

In contrast with other methods which have subsequently found nice compositionality properties by simple summation, the novelty of our method is the explicit learning of word representations suitable for summation. The system is then designed to force words with similar context to be close in a d^{wrd}-dimensional space, while these dimensions are learned to be combined with other related words. This joint learning is illustrated in Figure 5.1.

Due to the large size of $\mathcal{W}$, a negative sampling approach is used to speed up the training. In Equation 5.6, the whole dictionary $\mathcal{W}$ is thus replaced by a subset $\mathcal{W}^- \subseteq \mathcal{W}$ with N randomly chosen negative samples $w_i \notin s$. A new set $\mathcal{W}^-$ is randomly picked at each iteration during the training. The whole system is trained by minimizing both objective functions (5.1) and (5.6) over the training data using stochastic gradient descent. For a training sample $s \in \mathcal{S}$, the loss function is thus as follows:

$$L(s;\theta) = \sum_{w_t \in s} \left\| \sqrt{P_{w_t}} - g\left(f\left(\sqrt{P_{w_t}}\right)\right) \right\|_2 + \sum_{\substack{w_i \in \mathcal{W}^- \\ w_i \notin s}} \max\left(0, 1 - \mathbf{x}_s \cdot \mathbf{x}_{w_t} + \mathbf{x}_s \cdot \mathbf{x}_{w_i}\right). \tag{5.7}$$

5.4 Experimental Results

For our experiments, the corpus of text is the entire English Wikipedia described in Section 4.3.1.

5.4.1 Phrase Dataset

To learn the summation of words that appear frequently together, we choose to consider only the noun and verb phrase chunks to build $\mathcal{S}$. We extract these chunks with a phrase chunking approach by using the SENNA software[1]. By retaining only the phrase chunks appearing at least ten times, this results in 1,823,259 noun phrase chunks and 255,232 verb phrase chunks,

[1] Available at http://ml.nec-labs.com/senna/

for a total of 2,078,491 phrase chunks. We divided this set of phrases into three sets: 1,000 phrases for validation, 5,000 phrases for testing, and the rest for training (2,072,491 phrases). An unsupervised framework requires a large amount of data. Because our primary focus is to provide good word representations, validation and testing sets are intentionally kept small to retain as much phrases as possible in the training set.

5.4.2 Other Methods

We compare our distributed representations with other available models for computing vector representations of words:

1. the GloVe model which is also based on co-occurrence statistics of corpora (Pennington et al., 2014)[2] (see Section 4.9.2),

2. the Skip-gram (SG) model which learns representations from prediction-based models (Mikolov et al., 2013b)[3] (see Section 3.2.3).

The same corpus and dictionary $\mathcal{W}$ as the ones described in Section 4.3.1 are used to train 100-dimensional word vector representations. We use a symmetric context window of ten words, and the default values set by the authors for the other hyper-parameters.

5.4.3 Evaluating Word Representations

	SVD	AUTOENCODER
WORDSIMILARITY-353	0.64	0.64
RARE WORD	0.37	0.39
SYNTACTIC ANALOGIES	65.6	68.0
SEMANTIC ANALOGIES	52.7	51.3

Table 5.1 – Evalution of word representations on both similarity and analogy tasks. Comparison of performance between Hellinger PCA with randomized SVD and with autoencoder. We use 100-dimensional word vector representations. Spearman rank correlation is reported on word similarity tasks. Accuracy is reported on word analogy tasks.

The first objective of the model is to learn word embeddings as good as with Hellinger PCA through randomized SVD. To evaluate this, we use both analogy and similarity tasks described in Section 4.3.2. As expected, results reported in Table 5.1 show that our model gives similar results than with the randomized SVD. Even with the addition of a second objective function,

[2] Code available at http://www-nlp.stanford.edu/software/glove.tar.gz.
[3] Code available at http://word2vec.googlecode.com/svn/trunk/.

the autoencoder approach has captured the same syntactic and semantic information about words.

5.4.4 Evaluating Phrase Representations

We aim at learning to sum word representations to generate phrase representations, while keeping the original information coming from the words. We thus introduce a novel task to evaluate the phrase representations.

Description of the Task

As dataset, we use the collection of test phrases described in Section 5.4.1. It contains 5000 phrases (noun phrases and verb phrases) extracted from Wikipedia with a chunking approach. Among them, 2244, 2030 and 547 are, respectively, composed of two, three and four words. The remaining 179 are composed of at least five words with a maximum of eight words. For a given phrase $s = \{w_1, \ldots, w_T\} \in \mathcal{S}$ of T words, the objective is to retrieve the T words from its distributed representation $\mathbf{x}_s$. Scores between the phrase s and all the possible words $w_i \in \mathcal{W}$ are calculated using the dot product between their distributed representations $\mathbf{x}_s \cdot \mathbf{x}_{w_i}$, as illustrated in Figure 5.1. The top T scores are considered as the words composing the phrase s.

Results

To evaluate whether words making a given phrase can be retrieved from the distributed phrase representation, we use Recall $@K$, which measures the fraction of times a correct word was found among the top K results. K is proportional to the number of words per phrase, e.g. for a 3-word phrase with a Recall@5, the correct words are found among the top 15 results. Higher Recall $@K$ means better retrieval performance. Since we care most about the top-ranked retrieved results, the Recall $@K$ with small K are more important.

	R@1	**R@5**	**R@10**
SKIP-GRAM	7.96	22.26	30.04
GLOVE	54.97	79.97	86.54
SVD	17.42	32.87	40.72
OUR MODEL	**64.22**	**91.72**	**95.85**

Table 5.2 – Evaluation of phrase representations. Comparison of performance across all models with 100-dimensional phrase vector representations on word retrieval. R@K is Recall@K, with $K = \{1, 5, 10\}$.

Results reported in Table 5.2 show that our distributed word representations can be averaged

together to produce meaningful phrase representations, since the words are retrieved with a high recall. Our model significantly outperforms other methods on this task. The comparison with SVD results is particularly interesting since we observe a significant gap. By introducing the second objective, we learn word embeddings distributed differently, which allows the summation without discarding the information carried by the original vectors.

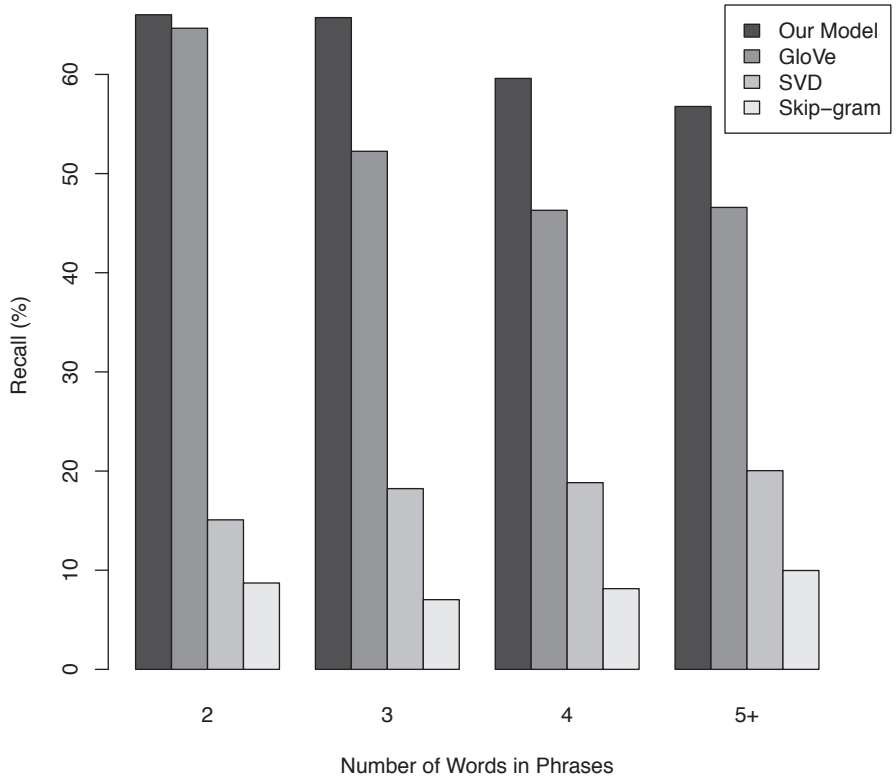


Figure 5.2 – Recall@1 based on the number of words per phrases. Comparison of performance across all models with 100-dimensional word vector representations.

In Figure 5.2, a more detailed analysis of results reveals that the GloVe model competes with ours for the 2-word phrases. However GloVe's representations cannot maintain this performance for longer phrases. It is probably not too surprising as this model is trained using ratios of co-occurrence probabilities for two target words. Consequently, it well learns linear substructures for pairs of words. In contrast, our joint model can learn more complex substructures which make possible the aggregation of multiple words within a low-dimensional vector space.

5.4.5 Inferring New Phrase Representations

Representations for new phrases can thus be generated by simply averaging its word representations, assuming that all words are in the vocabulary $\mathcal{W}$. Considering that $\mathcal{W}^n$ tends to grow exponentially with n, it gives a nice framework to produce the huge variety of possible

QUERY PHRASES	NEAREST PHRASES	
	ENCODING FUNCTION $f(\cdot)$	AVERAGING WORDS
AMERICAN AIRLINES	BRANIFF AIRLINES	AMERICAN AIRWAYS
	ALOHA AIRLINES	PAN AMERICAN AIRLINES
	BRANIFF AIRWAYS	AMERICAN EAGLE AIRLINES
	JETBLUE AIRWAYS	NORTH AMERICAN AIRLINES
	BRANIFF INTERNATIONAL AIRWAYS	AMERICAN OVERSEAS AIRLINES
CHICAGO BULLS	DENVER NUGGETS	CHICAGO COLTS
	SEATTLE SUPERSONICS	CHICAGO HORNETS
	CLEVELAND CAVALIERS	CHICAGO STAGS
	BOSTON CELTICS	BUFFALO BULLS
	DALLAS MAVERICKS	CHICAGO CARDINALS
HOME PLATE	RIGHT FIELDER	THE HOME PLATE UMPIRE
	CENTER FIELDERS	THE HOME PLATE AREA
	THE OUTFIELD FENCE	THE HOME LEG
	LEADOFF BATTER	THE BALL HOME
	THE INFIELD	THE DIAMOND STATE BASE BALL CLUB
PRESIDENT OF THE UNITED STATES	PRESIDENT COOLIDGE	THE UNITED STATES PRESIDENT
	PRESIDENT EISENHOWER	THE UNITED STATES PRESIDENCY
	U.S. PRESIDENT DWIGHT EISENHOWER	THE FIRST UNITED STATES SECRETARY
	PRESIDENT TRUMAN	THE UNITED STATES MINISTER
	PRESIDENT REAGAN	THE FIRST UNITED STATES SENATOR

Table 5.3 – Examples of phrases and five of their ten nearest phrases from the collection of phrases. Representations for the collection of phrases have been computed by averaging the word representations. Query phrase representations are inferred using the two different alternatives: (1) with the encoding function f using counts from a symmetric window of ten context words around the query phrase, (2) by averaging the representations of the words that compose the query phrase. All distributed representations are 100-dimensional vectors.

sequences of n words in a timely and efficient manner with low memory consumption, unlike other methods. Relying on word co-occurrence statistics to represent words in vector space also provides a framework to easily generate representations for unseen words or phrases, as described in Section 4.7. Table 5.3 presents some examples of phrases, where we use both alternatives to compute their distributed representations. It can be seen that both alternatives give distinct representations. For instance, by using the encoding function $f(\cdot)$, our model infers a representation for the entity *Chicago Bulls* which is close to other NBA teams, like the *Denver Nuggets* or the *Seattle Supersonics*. By averaging the representations of both words *Chicago* and *Bulls*, our model infers a representation which is close to other Chicago's sport teams. Both representations are meaningful, but they carry different information. Relying on co-occurrence statistics gives entities that occur in a similar context, while the summation tries to find entities containing the maximum amount of similar information. This also works with longer phrases, such as *President of the United States*. The first alternative gives men who served as president, when the second gives related positions.

5.5 Conclusion

We introduce a model that combines both count-based methods and predictive-based methods for generating distributed representations of words and phrases. Using a chunking approach, a collection of noun phrases and verb phrases is extracted from Wikipedia. For a given n-word phrase, we train our model to generate a low-dimensional representation for each word based on its co-occurrence probability distribution. These n representations are averaged together to generate a distributed phrase representation in the same semantic space. Thanks to an autoencoder approach, we can simultaneously train the model to retrieve the original n words from the phrase representation, and therefore learn complex linear substructures. Furthermore, we show that the autoencoder learns word embeddings as good as with the conventional SVD. Performance on a novel task for evaluating phrase representations confirm the ability of our model to learn complex substructures, which make possible the aggregation of multiple words within a low-dimensional vector space. Better still, inference of new phrase representations is also easily feasible when relying on counts. Some qualitative examples demonstrate that both alternatives can give different but meaningful information about phrases.

Document Classification Part II

6 Sentiment Classification with Convolutional Neural Network

Successful methods for document classification are traditionally based on *bag-of-words* (BOW), where words are transformed into numeric values. These are considered as features for training a classifier, such as Naive Bayes, Maximum Entropy, or Support Vector Machine (see Section 2.2). Such methods work well in practice, since finding the most discriminative keywords is generally enough to classify documents. However, sentiments might be harder to detect when relying only on keywords. An obvious example is when "not" is used with an adjective. Then, a negative expression is likely to be predicted as a positive sentiment. Adding n-grams could be a solution to overcome this limitation, but the number of features grows exponentially with n, causing an increase in the computational cost. We thus propose to address this issue with an approach based on a convolutional neural network (CNN). CNN are powerful models for classification, and are already a great success in computer vision. Thanks to word embeddings, CNN-based models can also be designed for tackling NLP problems. Words are represented as dense vectors that can be fed to a convolutional layer. Stacking many layers has as consequence a high computational cost, which requires the use of Graphical Processing Unit (GPU). However, traditional models in text document classification are relatively simple, leading to a rapid classification. In this chapter, we therefore introduce a simple CNN with only one convolutional layer followed by one max layer for classifying documents with sentiments. We evaluate our approach on short and long documents, using tweets and movie reviews.

6.1 Convolutional Neural Network for Sentiment Classification

In sentiment classification, we are given a document $d \in \mathcal{D}$ and a class $y \in \{-1;1\}$ where $y = -1$ is negative sentiment and $y = 1$ is positive sentiment. Traditional NLP approaches extract a rich set of hand-designed features from documents which are then fed to a standard classification algorithm. In contrast, we want to pre-process our features as little as possible. In that respect, a convolutional neural network architecture seems appropriate as it can be trained in an end-to-end fashion on the task of interest (Collobert et al., 2011). A convolutional layer receives pre-trained word embeddings as inputs, and learns which sequences of words

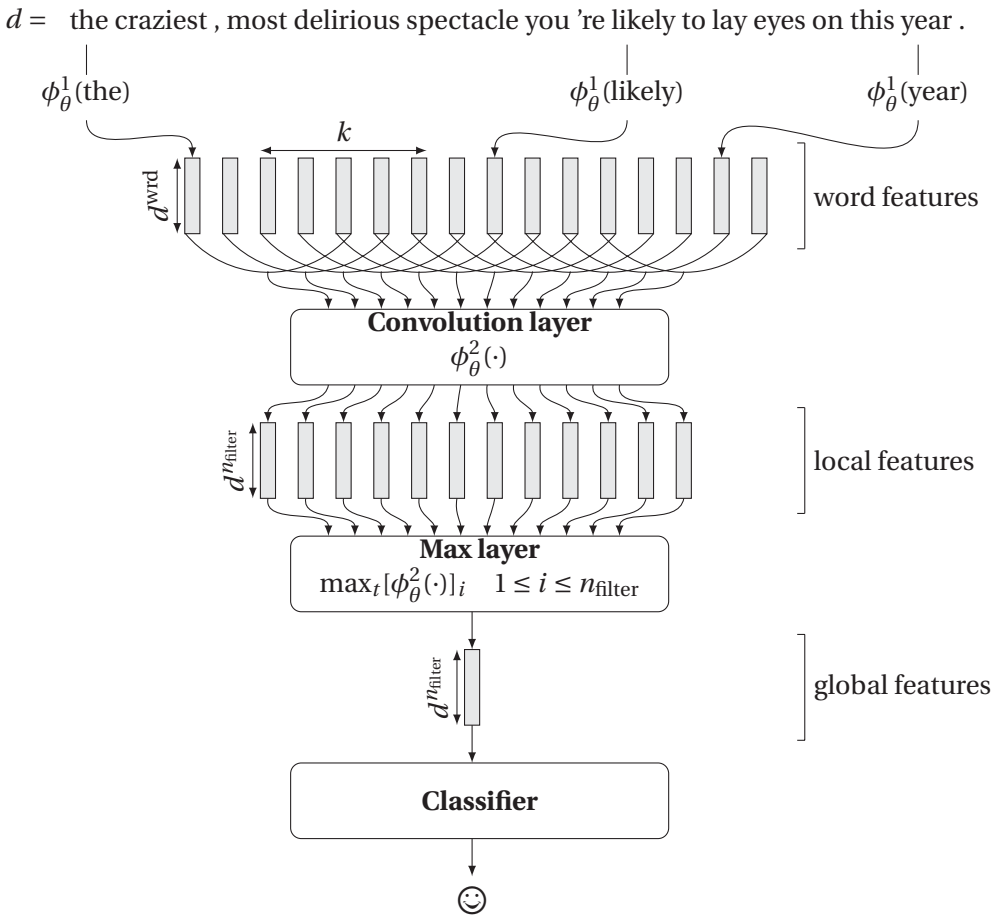


Figure 6.1 – Convolutional neural network for sentiment classification.

are good indicators of sentiments. Documents might contain multiple sentiments with various levels of polarity, especially for long documents such as movie reviews. As we want to extract the most discriminative sentiments, we use a max layer which summarizes all local features into a global document representation. This final representation is used to train a classifier. The architecture is illustrated in Figure 6.1.

6.1.1 Embedding Layer

Given a document of T words $\{w_1, w_2, \ldots, w_T\}$, each word $w_t \in \mathcal{W}$ is first embedded into a d^{wrd}-dimensional vector space, using an embedding layer ϕ_θ^1 as described in Section 2.1.3. The embedding layer produces the following matrix by applying this operation for all T words in d:

$$\phi_\theta^1(w_1, w_2, \ldots, w_T) = \begin{pmatrix} \mathbf{E}_{w_1} & \mathbf{E}_{w_2} & \ldots & \mathbf{E}_{w_T} \end{pmatrix} \in \mathbb{R}^{d^{\text{wrd}} \times T}, \tag{6.1}$$

where $\mathbf{E} \in \mathbb{R}^{d^{\text{wrd}} \times |\mathcal{W}|}$ is the word embedding matrix which is initialized with pre-trained embeddings using Hellinger PCA (see Chapter 4). As sentiment classification is clearly a semantic task, we select embeddings obtained with a context window size of 10 words.

6.1.2 Convolutional Layer

The convolutional layer takes the complete document d and successively produces local features by applying a nonlinear transformation to all sequences of words in d. We define a kernel size k (a hyper-parameter), which corresponds to a fixed window size of words. The layer then applies a k-word sliding window over the matrix of embeddings. It first concatenates each column vector to produce a $(d^{wrd} \times k)$-dimensional vector. This vector is then fed to a nonlinear hidden layer,

$$\phi_\theta^2(w_t, \ldots, w_{t+k}) = \mathbf{W}^2 h\Big(\mathbf{W}^1 \phi_\theta^1(w_t, \ldots, w_{t+k}) + \mathbf{b}^1\Big) + \mathbf{b}^2 \quad \in \mathbb{R}^{n_{\text{filter}}}. \tag{6.2}$$

The hyper-parameter n_{filter} is the *number of filters* of the convolution layer. The weight matrices $\mathbf{W}^1 \in \mathbb{R}^{n_{\text{hu}} \times (d^{\text{wrd}} \times k)}$ and $\mathbf{W}^2 \in \mathbb{R}^{n_{\text{filter}} \times n_{\text{hu}}}$, and the biases $\mathbf{b}^1 \in \mathbb{R}^{n_{\text{hu}}}$ and $\mathbf{b}^2 \in \mathbb{R}^{n_{\text{filter}}}$ are the same across all windows in the document.

6.1.3 Global Document Representation

Each window of words in a document d is represented as a set of (trained) filters produced by the convolutional layer. We now aim at focusing on the most important filters in the document, regardless of their location. The maximum value obtained by the i^{th} filter over the whole document is:

$$\big[\phi_\theta^3(d)\big]_i = \max_{1 \le t \le T-k+1} \big[\phi_\theta^2(w_t, \ldots, w_{t+k})\big]_i \quad 1 \le i \le n_{\text{filter}}, \tag{6.3}$$

where $\phi_\theta^3(d) \in \mathbb{R}^{n_{\text{filter}}}$ is a global document representation. It can be seen as a way to measure if the information represented by the filter has been captured in the document or not.

6.1.4 Binary Classification

Finally, we feed all these intermediate scores to a linear classifier, leading to the following simple model:

$$\phi_\theta(d) = \boldsymbol{\alpha} \cdot \phi_\theta^3(d), \quad \boldsymbol{\alpha} \in \mathbb{R}^{n_{\text{filter}}}. \tag{6.4}$$

The i^{th} filter might capture positive or negative sentiment depending on the sign of $[\boldsymbol{\alpha}]_i$.

6.1.5 Training

The neural network is trained using stochastic gradient descent. We denote $\theta = \{\mathbf{E}, \mathbf{W}^1, \mathbf{W}^2, \mathbf{b}^1, \mathbf{b}^2, \boldsymbol{\alpha}\}$ all the trainable parameters of the network. Using a training set $\mathcal{D}$, we minimize the following logistic loss function with respect to θ:

$$\mathcal{L}(\theta) = \sum_{(d,y)\in\mathcal{D}} \log\left(1 + e^{-y\phi_\theta(d)}\right). \tag{6.5}$$

6.2 Short Document Classification

As a first experiment, we want to evaluate whether our approach can detect sentiments in short documents. For that purpose, we use Twitter as a data source. Twitter is an online social networking service that enables users to send and read short 140-character messages called *tweets*. Users post messages where they can express opinions about different topics, which includes products or services. These tweets are publicly visible by default, which makes Twitter a gold mine for consumers, marketers or companies. Consumers can analyze the opinions of Twitter users about products or services before making a purchase. Marketers can analyze customer satisfaction or research public opinion of their company and products. Companies can gather critical feedback about problems in newly released products. Identifying and extracting this subjective information has therefore become a key point.

6.2.1 Dataset Description

Twitter data posses many unique properties that make sentiment classification much more challenging than in other domains:

- Maximum length of a tweet is 140 characters. The dataset considered has in average 14 words and 78 characters.
- The quantity of misspelling, slang and informal language is much higher than in other types of data.
- Twitter users post an infinitude of different subjects. This differs from classical sentiment classification datasets, which are usually focused on a specific domain (such as movie reviews).

For our experiments, we consider the same dataset used by Go et al. (2009). For the training data, the tweets were extracted using the official Twitter Application Programming Interface (API)[1]. The sentiment of Twitter posts have been predicted using distant supervision. The positive tweets were selected with a query for tweets containing ":)", ":-)", ":)", ":D", "=)". The negative ones with a query for tweets containing ":(", ":-(", ": (". The tweets in the training set

[1] http://apiwiki.twitter.com

are from the time period between April 6, 2009 to June 25, 2009. The following filtering were applied on the data:

- emoticons were removed from the tweets,
- tweets containing both positive and negative emoticons were removed,
- retweets were removed to avoid giving extra weight to a particular weight.

Stripping out the emoticons causes the classifier to learn from the other features (the words in our case) present in the tweet. The final training data consists of a total of 1.6M tweets, half labeled as positive and half labeled as negative. The test data is manually collected, using the web application. A set of 177 negative tweets and 182 positive tweets were manually marked. Not all the test tweets have emoticons. We also consider the particularity of Twitter language to reduce the vocabulary size and make the data more concise. This is achieved with the following data pre-processing:

- **Target**: words started with the character @ are replaced by the special token "TARGET".
- **Link**: every URL (http://...) is replaced by the special token "URL".
- **Hashtag**: words started with the character # are replaced by the token "HASHTAG".
- **Repeated Letters**: every letter occurring more than two times is replaced with two occurrences (*e.g.*, 'huuuuuuuungry' is replaced by 'huungry')
- **Digit**: all occurrences of sequences of numbers within a word are replaced with the special token "NUMBER".

Finally, all words are lowercased. The resulting tweets have been tokenized using the CMU ARK Twitter NLP tools[2] (Gimpel et al., 2011). This results in a 323,393 words vocabulary $\mathcal{W}$.

6.2.2 Related Work

Go et al. (2009) proposed a model to automatically extract sentiment from tweets. They consider three different feature-based classical machine learning classifiers to infer sentiment on tweets: (i) Naive Bayes (NB), (ii) Max-Entropy (MaxEnt) and (iii) Support Vector Machine (SVM). They report results for different set of features: Unigram, Unigram+Bigram and Unigram+Part-of-Speech (POS). More recently, Poria et al. (2014) outperform these baseline methods by employing lexical resources to provide polarity scores (from *SenticNet*) or emotion labels (from *WordNet-Affect*) for words and concepts. Kalchbrenner et al. (2014) have proposed a dynamic convolutional neural network (DCNN) for modeling sentences. While we

[2] http://www.ark.cs.cmu.edu/TweetNLP/

propose to simply extract a global feature vector with a max approach after one convolutional layer, they used multiple layers of convolution followed by dynamic k-max pooling to induce a structured feature graph over a given tweet. This approach is therefore much more complex (deeper) than our proposed model.

6.2.3 Experimental Results

Tweets are inherently very short documents. We thus use a small window size of $k = 3$ words for the convolutional layer, along with $n_{\text{filter}} = 30$ filters. For the nonlinear hidden layer, we use $n_{\text{hu}} = 50$ hidden units. All these hyper-parameters were chosen considering a validation set extracted from the training data. Word embeddings dimension is $d^{\text{wrd}} = 50$. Because each input (a tweet) contains a limited number of words, we choose not to introduce a special embeddings for unknown words. For each word in $\mathcal{W}$ where its embedding is not available, we use instead a random initialization. These new embeddings are then learned, while the existing ones are tuned during the training.

MODEL	**ACCURACY** (%)
SVM	81.6
NB	82.7
MAXENT	83.0
EMOSENTICSPACE	85.1
DCNN	87.4
OUR MODEL	**88.3**

Table 6.1 – Accuracy on the Twitter sentiment classification test set. The three classical models (SVM, BiNB and MaxEnt) are based on unigram and bigram features; the results are reported from Go et al. (2009).

Results reported in Table 6.1 show that our model significantly outperforms the baseline models, and a model with a prior knowledge on sentiments (EmoSenticSpace). It also slightly outperforms a much deeper convolutional neural network (DCNN), which indicates that there is no need for multiple convolutional layers in sentiment classification of short documents. The number of filters used in our model is very low, $n_{\text{filter}} = 30$. This means that tweets are represented in 30-dimensional global representations. Conversely, traditional bag-of-words based classifiers will represent tweets with as many features as there are words in the vocabulary. Considering that our vocabulary $\mathcal{W}$ contains 323,393 words, this is about ten thousand times higher than our tweet representations.

At inference time, our model can also output polarity scores for each k-word window in a given tweet by simply removing the max layer. This is a valuable asset to detect which parts of a tweet are positive or negative, as illustrated in Figure 6.2. This also helps to understand why certain tweets are misclassified. Some examples of misclassified tweets are in Figure 6.2 where

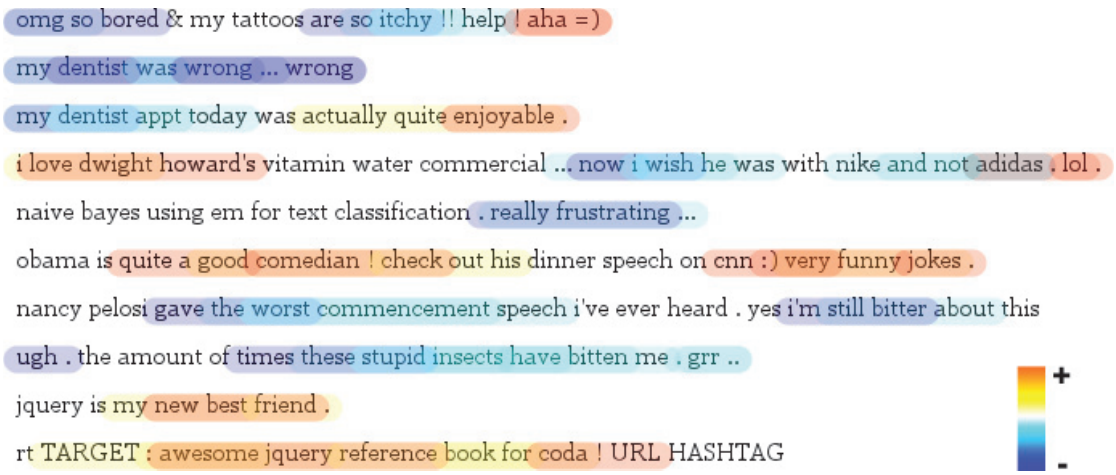


Figure 6.2 – Selection of tweets from the test set where sentiments are highlighted using our model outputs. The blue color scale indicates negative sentiment, the red one indicates positive sentiments. Best viewed in colors.

both sentiments have been detected.

6.3 Long Document Classification

Our second experiment focuses on long text documents. In that respect, we consider movie reviews as data source. As Twitter, movie reviews are widely available online. Many websites offer a platform for expressing opinions on movies, such as IMDB (www.imdb.com) or Rotten Tomatoes (http://www.rottentomatoes.com/). But unlike Twitter, those reviews are, in general, well written as they are subject to moderation by the website, and they contain a polarity score. Distance supervision is therefore not needed.

6.3.1 Dataset Description

We used a collection of 50,000 reviews from IMDB introduced in Maas et al. (2011)[3]. This dataset contains no more than 30 reviews per movie, with an even number of positive and negative reviews, so randomly guessing yields 50% accuracy. Reviewers from IMDB give a score from 1 to 10 in addition to their reviews, which allows supervised learning. Only highly polarized reviews have been considered. A review is considered as negative if the score ≤ 4, and as positive if the score ≥ 7. The final dataset has been evenly divided into training and test sets (25,000 reviews each). In contrast with tweets, IMDB movie reviews are long documents since each review contains on average 271 words. IMBD reviews guidelines indeed say that the minimum length for reviews is 10 lines of text, with a recommended lengths of 200 to 500 words. As data pre-processing, we just replace all digits with a special token and lowercase all words.

[3]Available at http://www.andrew-maas.net/data/sentiment

6.3.2 Related Work

As a baseline system, we report the best model from Maas et al. (2011). By mixing unsupervised and supervised techniques, they learn word vectors capturing general semantic information, as well as rich sentiment content. Then they combine these word representations with BOW representations for classifying sentiment of movie reviews. Later, Wang and Manning (2012) explored variants of Naive Bayes (NB) and Support Vector Machines (SVM) for movie reviews. They propose to combine generative and discriminative classifiers to introduce a simple model where an SVM is built over NB log-count ratios as feature values (see Section 7.2.3 for details). These baselines are proving to be highly competitive, as they provide state-of-the-art performance on this task.

6.3.3 Experimental Results

When dealing with movie reviews, the model has to classify much longer documents than in Twitter. The convolutional layer therefore uses a window of $k = 5$ words and $n_{\text{filter}} = 1000$ filters. The number of hidden units is $n_{\text{hu}} = 300$. A simple cross-validation has been performed on the training set to choose these optimal hyper-parameters. Word embeddings dimension remains $d^{\text{wrd}} = 50$, but this time, we introduce a special embedding for unknown words.

MODEL	**ACCURACY** (%)
MAAS ET AL. (2011)	88.9
SVM	86.9
BISVM	89.2
NB	83.5
BINB	86.6
NBSVM	88.3
BINBSVM	91.2
OUR MODEL	90.2

Table 6.2 – Accuracy on the IMDB test set for sentiment classification. When BI is used as prefix, models include bigram features.

Results in Table 6.2 show that our model outperforms the baseline from Maas et al. (2011) and other classical approaches based on models with unigrams. We note that including bigrams to those classical approaches significantly helps to increase the general performance. This confirms the need of considering longer sequences of words for sentiment classification, which legitimizes the use of our convolutional model to tackle this task. However, it is interesting to see that the best result is obtained with SVM classifier built over NB features from unigrams and bigrams. For long document classification, those simple models are highly competitive which raises the question of how relevant CNN models are for this task. One answer to this question is that CNN can help fighting the curse of dimensionality, as they provide global document representations in much lower dimensional space than BOW models.

Embeddings fine-tuning

BORING		BAD		AWESOME	
before	*after*	*before*	*after*	*before*	*after*
SAD	CRAP	HORRIBLE	TERRIBLE	SPOOKY	TERRIFIC
SILLY	LAME	TERRIBLE	STUPID	AWFUL	TIMELESS
SUBLIME	MESS	DREADFUL	BORING	SILLY	FANTASTIC
FANCY	STUPID	UNFORTUNATE	DULL	SUMMERTIME	LOVELY
SOBER	DULL	AMAZING	CRAP	NASTY	FLAWLESS
TRASH	HORRIBLE	AWFUL	WRONG	MACABRE	MARVELOUS
LOUD	RUBBISH	MARVELOUS	TRASH	CRAZY	EERIE
RIDICULOUS	SHAME	WONDERFUL	SHAME	ROTTEN	LIVELY
RUDE	AWFUL	GOOD	KINDA	OUTRAGEOUS	FANTASY
MAGIC	ANNOYING	FANTASTIC	JOKE	SCARY	SURREAL

Table 6.3 – Set of words with their 10 nearest neighbors before and after fine-tuning for the movie review task (using the Euclidean metric in the embedding space). Before tuning, antonyms are highlighted in blue. After tuning, antonyms have been replaced by some task-specific words which are highlighted in red. Best viewed in colors.

As we have seen in Section 4.6.4, tuning word embeddings for the given task helps to increase the general performance. In sentiment classification, it is even more important since antonyms tend to be close in the original embedding space. We see in Table 6.3 that *bad* is, for instance, close to antonyms such as *good* or *fantastic*. After fine-tuning, antonyms have been removed from the nearest neighbors, and replaced by words that are related to the task of interest. In terms of results, the accuracy drops from 90.2% to 88.0% when the word embeddings remain fixed during the training. Fine-tuning is therefore quite important in sentiment classification.

Sentiment Detection

As our method takes the whole review as input, we can extract windows of words having the most discriminative power: it is a major advantage of our method compared to conventional bag-of-words based methods. We report in Table 6.4 some examples of windows of words extracted from the most discriminative filters $[\boldsymbol{\alpha}]_i$ (positive and negative). Note that there is about the same number of positive and negative filters after learning.

As with Twitter, each k-word window can receive a polarity score for highlighting which parts of the movie reviews contains positive and negative sentiments. This feature makes more sense when dealing with long document as it gives a nice visualization tool to quickly explore the most interesting sequences of words, as seen in Figure 6.3.

6.4 Conclusion

As we have built embeddings carrying meaningful semantic information about words, we propose to tackle sentiment classification with a convolutional neural network. A convolu-

k-WORD WINDOW	
$[\boldsymbol{\alpha}]_i < 0$	$[\boldsymbol{\alpha}]_i > 0$
THE WORST FILM THIS YEAR	BOTH REALLY JUST WONDERFUL .
VERY WORST FILM I 'VE	. A TRULY EXCELLENT FILM
VERY WORST MOVIE I 'VE	. A REALLY GREAT FILM
WATCH THIS UNFUNNY STINKER .	EXCELLENT FILM WITH GREAT PERFORMANCES
, EXTREMELY UNFUNNY DRIVEL COME	EXCELLENT FILM WITH A GREAT
, THIS LUDICROUS SCRIPT GETS	EXCELLENT MOVIE WITH A STELLAR
IT WAS POINTLESS AND BORING	INCREDIBLE . JUST INCREDIBLE .
IT IS UNFUNNY . UNFUNNY	PERFORMANCES AND JUST AMAZING .
FILM ARE AWFUL AND EMBARRASSING	ONE WAS REALLY GREAT .

Table 6.4 – The top 3 positive and negative filters $[\boldsymbol{\alpha}]_i$ and their respective top 3 windows of words within the whole IMDB review dataset.

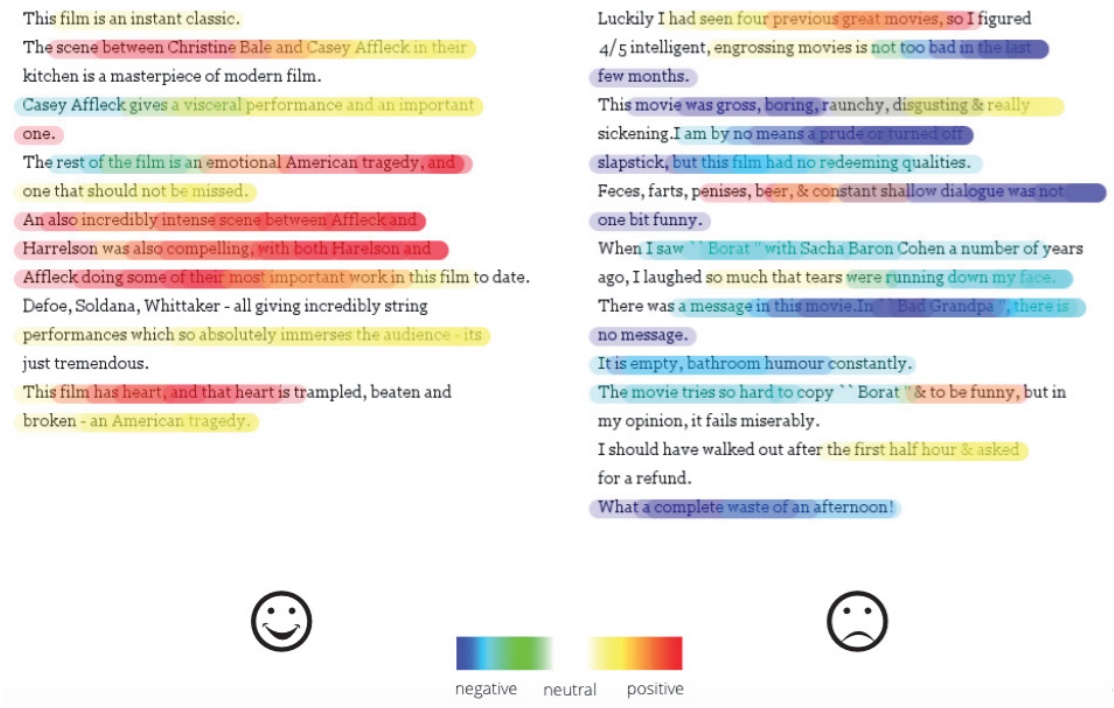


Figure 6.3 – Highlighting sentiments in IMDB movie reviews. The blue color scale indicates negative sentiment, the red one indicates positive sentiments. Best viewed in colors.

tional layer learns local features from windows of k-word embeddings. A global (and compact) document representation is then obtained by extracting the most discriminative local features with a max layer. This global representation is finally used for training a linear classifier which gives better performance than classical approaches in short documents classification. Bag-of-words based models with bigram features are still highly competitive in long documents classification, but we show that the proposed model is a good alternative for fighting the curse of dimensionality, while offering a framework for sentiment visualization.

7 N-gram-Based Model for Compact Document Representation

Day after day, the amount of text documents available online is growing. Effective text mining is getting worse without efficient organization, summarization and indexing of document content. Traditional representation of documents known as *bag-of-words* (BOW) considers every document as a vector in a very high dimensional space where each element of this vector represents one term appeared in the document collection. One limitation of this model is that the discriminative words are usually not the most frequent ones. A large vocabulary of words needs to be defined to obtain a robust model. Classification or text clustering then must deal with a huge number of features, and it becomes time-consuming and memory-hungry.

Furthermore, such models are based on words alone, which raises another limitation. A collection of words cannot capture phrases or multiword expressions, while n-grams have shown to be helpful features in several natural language processing tasks (Tan et al., 2002; Lin and Wu, 2009; Wang and Manning, 2012). N-gram features are not commonly used in text mining, probably because the vocabulary $\mathcal{W}^n$ tends to grow exponentially with n. Phrase structure extraction can be used to identify only n-grams which are phrase patterns, and thus limit the vocabulary size. However, this adds another step to the model, making it more complex. To overcome these barriers, we propose that documents be represented as a *bag of semantic concepts*, where n-grams are considered instead of only words. By leveraging the ability of word vector representations to compose (see Chapter 5), representations for n-grams are easily computed with an element-wise addition. Using a clustering algorithm such as K-means, those representations are grouped into K clusters which can be viewed as *semantic concepts*. Text documents are now represented as bag of semantic concepts, with each feature corresponding to the presence or not of n-grams from the resulting clusters. By setting a small K, semantic information is captured while remaining in a low-dimensional space as the number of features is dramatically reduced. We evaluate the proposed model in the classification of movie reviews and news.

7.1 Related Work

Some techniques have been proposed to reduce the dimensionality and represent documents in a low-dimensional semantic space. Latent Semantic Analysis (LSA) (Deerwester et al., 1990) uses the term-document matrix and a singular value decomposition (SVD) to represent terms and documents in a new low-dimensional space. Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is a generative probabilistic model of a corpus. Each document is represented as a mixture of latent topics, where each topic is characterized by a distribution over words. By defining K topics, documents can then be represented as K-dimensional vectors. Pessiot et al. (2010) also proposed probabilistic models for unsupervised dimensionality reduction in the context of document clustering. They make the hypothesis that words occurring with the same frequencies in the same document are semantically related. Based on this assumption, words are partitioned into word topics. Document are then represented by a vector where each feature corresponds to a word-topic representing the number of occurrences of words from that word-topic in the document. Other techniques have tried to improve text document clustering by taking into account relationships between important terms. Some have enriched document representations by integrating core ontologies as background knowledge (Staab and Hotho, 2003), or with Wikipedia concepts and category information (Hu et al., 2009). Part-of-speech tags have also been used to disambiguate words (Sedding and Kazakov, 2004).

7.2 A Bag of Semantic Concepts Model

The model is divided into three steps:

1. vector representations of n-grams are obtained by averaging pre-trained representations of its individual words;

2. n-grams are grouped into K semantic concepts by performing K-means clustering on all n-gram representations;

3. documents are represented by a bag of K semantic concepts, where each entry depends on the presence of n-grams from the concepts defined in the previous step.

7.2.1 N-gram Representation

The first step of the model is to generate continuous vector representations for each n-gram. Leveraging the model described in Chapter 5, word representations are summed to generate n-gram representations:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_{w_i}\,. \tag{7.1}$$

These representations are vectors which keep the semantic information of n-grams with different n in the same dimensionality. Distances between them are thus computable. It allows the use of a K-means clustering for grouping all n-grams into K classes.

7.2.2 K-means Clustering

K-means is an unsupervised learning algorithm commonly used to automatically partition a data set into K clusters. Considering a set of n-gram representations $\mathbf{x}_i \in \mathbb{R}^{d^{\text{wrd}}}$, the algorithm will determine a set of K centroids $\boldsymbol{\gamma}_k \in \mathbb{R}^{d^{\text{wrd}}}$, so as to minimize the average distance from each representation to its nearest centroid:

$$\sum_i \left\| \mathbf{x}_i - \boldsymbol{\gamma}_{\sigma_i} \right\|_2, \quad \text{where } \sigma_i = \underset{k}{\operatorname{argmin}} \left\| \mathbf{x}_i - \boldsymbol{\gamma}_k \right\|_2. \tag{7.2}$$

The limitation due to the size of the vocabulary is therefore overcome. By setting K to a low value, documents can also be represented by more compact vectors than with a bag-of-words model, while keeping all the meaningful information.

7.2.3 Document Representation

Denoting $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$ a set of text documents, where each document d_i contains a set of n-grams. First, each n-gram is embedded into a common vector space by averaging its word vector representations. The resulting n-grams representations are assigned to clusters using the centroids $\boldsymbol{\gamma}_k$ defined by the K-means clustering. Documents d_i are then represented by a vector of K features, $\mathbf{v}_i \in \mathbb{R}^K$. Each entry $[\mathbf{v}_i]_k$ usually corresponds to the frequency of n-grams from the k^{th} cluster within the document d_i. The set of text documents is then defined as $\bar{\mathcal{D}} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m\}$.

With Naive Bayes Features

For certain type of document, such as movie reviews in Section 6.3, the use of Naive Bayes features can improve the general performance (Wang and Manning, 2012). Success in sentiment classification relies mostly on the capability of the models to detect negative and positive n-grams in a document. A proper normalization is then calculated to determine how important each n-gram is for a given class $y \in \{-1, 1\}$. We first define a set of count vectors for all n-grams contained in $\mathcal{D}$, $\{\mathbf{f}_1, \ldots, \mathbf{f}_N\}$ where $\mathbf{f}_t \in \mathbb{R}^m$ is the frequencies of the t^{th} n-gram. $[\mathbf{f}_t]_i$ represents the number of occurrences of the t^{th} n-gram in the training document d_i. We then define count vectors for each sentiment. $\mathbf{p} \in \mathbb{R}^N$ is for positive sentiment, where each entry corresponds to a n-gram such as

$$[\mathbf{p}]_t = 1 + \sum_{i:y_i=1}^{m} [\mathbf{f}_t]_i \tag{7.3}$$

is the number of occurrence of the t^{th} n-gram in positive documents. The same operation is applies to define a count vector $\mathbf{q} \in \mathbb{R}^N$ for negative sentiment, where

$$[\mathbf{q}]_t = 1 + \sum_{i:y_i=-1}^{m} [\mathbf{f}_t]_i \tag{7.4}$$

is the number of occurrence of the t^{th} n-gram in negative documents.

A log-count ratio is then calculated to determine how important n-grams are for the sentiments (classes y):

$$\mathbf{r} = \log\left(\frac{\mathbf{p}/||\mathbf{p}||_1}{\mathbf{q}/||\mathbf{q}||_1}\right), \text{with } \mathbf{r} \in \mathbb{R}^N. \tag{7.5}$$

Because n-grams are in clusters, we extract the maximum absolute log-count ratio for every cluster $k \in \{1, \ldots, K\}$:

$$[\tilde{\mathbf{v}}_i]_k = \underset{[\mathbf{r}]_t}{\text{argmax}} \left|[\mathbf{r}]_t\right|, \quad \forall n\text{-gram } t \in k \quad \text{where} \quad [\mathbf{f}_t]_i > 0 \tag{7.6}$$

These document representations can then be used for several NLP tasks such as classification or information retrieval. As for BOW-based models, this model is particularly suitable for linear SVM.

7.3 Experiments with Sentiment Classification

Sentiments can have a completely different meaning if n-grams are considered instead of words. A classifier might leverage a bigram such as “not good” to classify a document as negative, while this would probably fail if only unigrams (words) were considered. We thus benchmark the bag of semantic concepts model on sentiment classification.

7.3.1 IMDB Movie Reviews Datasets

Datasets from IMDB have the nice property of containing long documents. It is thus valuable to consider n-grams in such a framework. We did experiments with small and large collections of reviews. We can thus analyze how well our model competes against classical models, for different dataset sizes.

Pang and Lee (2004)

The collection consists of 1,000 positive and 1,000 negative processed reviews[1]. So a random guess yields 50% accuracy. The authors selected only reviews where rating was expressed either with stars or some numerical value. To avoid domination of the corpus by a small number of prolific reviewers, they imposed a limit of fewer than 20 reviews per author per sentiment category. As there is no test set, we used 10-fold cross-validation.

Maas et al. (2011)

The collection consists of 100,000 reviews[2]. It has been divided into three datasets: training and test sets (25,000 labeled reviews each), and 50,000 unlabeled training reviews. See Section 6.3.1 for more details.

7.3.2 Building Bag of Semantic Concepts for Movie Reviews

In the following experiments, we use the word representations trained with the model described in Chapter 5. By following the three steps described in Section 7.2, movie reviews are then represented as bags of semantic concepts.

Computing n-gram representations.

We consider n-grams up to $n = 3$. Only n-grams with words from our vocabulary $\mathcal{W}$ are considered for both datasets[3]. This results in a set of 34,360 1-gram representations, 419,918 2-gram representations, and 921,837 3-gram representations for the Pang and Lee's dataset. And 67,847 1-gram representations, 1,842,461 2-gram representations, and 5,724,871 3-gram representations for the Maas et al.'s dataset. Because n-gram representations are computed by averaging representations of its word, all n-grams are also represented in a 100-dimensional vector.

Partitioning n-grams into semantic concepts.

Because n-grams are represented in a common vector space, similarities between n-grams of different length can be computed. To evaluate the benefit of adding n-grams for sentiment analysis, we define semantic concepts with different combinations of n-grams:

- only 1-grams (i.e. clusters of words),
- only 2-grams,

[1] Available at http://www.cs.cornell.edu/people/pabo/movie-review-data/.

[2] Available at http://www.andrew-maas.net/data/sentiment.

[3] Our English corpus is not large enough to cover all the words present in the IMDB datasets. We thus use the same 1-gram vocabulary with the other methods.

- only 3-grams,
- with 1-grams and 2-grams,
- with 1-grams, 2-grams and 3-grams.

Each of these five sets of n-gram representations are then partitioned in $K = \{100, 200, 300\}$ clusters with the K-means clustering. The centroids $\boldsymbol{\gamma}_k \in \mathbb{R}^{100}$ are obtained after K-means convergence (usually after 10 iterations of the algorithm).

Movie review representations.

Movie reviews are then represented as bags of semantic concepts with Naive Bayes features as described in Section 7.2.3. The log-count ratio for each n-gram is calculated on the training set for both datasets.

7.3.3 Comparison with Other Methods

We compare our models with two classical techniques for representing text documents in a low-dimensional vector space: LSA and LDA. Both methods use the same 1-gram vocabulary than with the bag of semantic concepts model with $K = \{100, 200, 300\}$. In the framework of Maas et al.'s dataset, LSA and LDA benefit from the large set of unlabeled reviews.

Latent Sentiment Analysis (LSA) (Deerwester et al., 1990).

Let $\mathbf{X} \in \mathbb{R}^{|\mathcal{W}| \times m}$ be a matrix where each element $[\mathbf{X}]_{i,j}$ describes the log count ratio of words i in document j, with m the number of training documents and $\mathcal{W}$ the vocabulary of words (i.e. 34,360 for Pang and Lee's dataset, 67,847 for Maas et al's dataset). By applying truncated SVD to the log-count ratio matrix $\mathbf{X}$, we thus obtain semantic representations in a K-dimensional space for movie reviews.

Latent Dirichlet Allocation (LDA) (Blei et al., 2003).

We train the K-topics LDA model using the code released by Blei et al. (2003)[4]. We leave the LDA hyper-parameters at their default values. Like our model, LDA extracts K topics (i.e. semantic concepts) and assigns words to these topics. Considering only the words in documents, we thus apply the method described in Section 7.2.3 to get document representations. A movie review d_i is then represented in a K-dimensional vector, where each feature $[\tilde{\mathbf{v}}_i]_k$ is the maximum absolute log-count ratio for the k^{th} topic.

[4] Available at http://www.cs.princeton.edu/~blei/lda-c/.

7.3.4 Classification using SVM

Having representations of movie reviews in a K-dimensional vector, a classifier is trained to determine whether a given review is positive or negative. Given the set of training documents $\tilde{D} = \{(\tilde{\mathbf{v}}_i, y_i) | \tilde{\mathbf{v}}_i \in \mathbb{R}^K, y_i \in \{-1, 1\}\}_{i=1}^{m}$, we picked a linear SVM as a classifier, trained using the LIBLINEAR library (Fan et al., 2008):

$$\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\tilde{\mathbf{v}}_i\right)^2, \tag{7.7}$$

with $\mathbf{w}$ the weight vector, and C a penalty parameter.

7.3.5 Results

	PANG AND LEE, 2004			MAAS ET AL., 2011		
K =	100	200	300	100	200	300
LDA	76.20	77.10	76.80	85.43	85.45	84.40
LSA	81.60	82.55	83.75	85.82	86.63	86.88
1-GRAM	81.60	82.60	82.70	84.51	84.76	85.54
2-GRAM	82.30	82.25	83.15	88.02	88.06	87.87
3-GRAM	73.85	73.05	72.65	87.41	87.46	87.22
1+2-GRAM	83.85	**84.00**	**84.00**	88.10	88.19	88.18
1+2+3-GRAM	82.45	83.05	83.05	88.39	88.46	**88.55**

Table 7.1 – Classification accuracy on both movie review tasks with $K = \{100, 200, 300\}$ number of features.

The overall results summarized in Table 7.1 show that the bag of semantic concepts approach outperforms the traditional LDA and LSA approaches to represent documents in a low-dimensional space. Good performance is achieved even with only 100 clusters, where LSA needs more clusters to improve. We also denote that our approach performs well on a small dataset, where LDA fails. A significant increase is observed when using 2-grams instead of 1-grams. However, using only 3-grams hurts the performance. The best results are obtained using a combination of n-grams, which confirms the benefit of the method. That also means that word vector representations can be combined while keeping relevant semantic information.

This is illustrated in Table 7.2 where semantically close n-grams are in the same cluster. We can see that the model is furthermore able to clearly separate antonyms, which is a good asset for sentiment classification. The results are also very competitive with a traditional BOW-model.

GOOD	NOT GOOD	ENJOY	DID N'T ENJOY
$k = 269$	$k = 297$	$k = 160$	$k = 108$
NICE ONE	SUFFICIENTLY BAD	ENTERTAIN	SCEPTICS
LIKED HERE	NOT LIKED	ADORED THEM	DID N'T LIKE
IS PRETTY NICE	IS FAR WORSE	ENJOYING	N'T ENJOY ANY
THE GREATEST THING	NOT THAT GREATEST	WATCHED AND ENJOY	VALUELESS

Table 7.2 – Selected pairs of antonyms and their cluster number. Here, n-grams from Maas et al's dataset have been partitioned into 300 clusters. Each n-gram is accompanied with a selection of others from its cluster.

Using the same 1-gram vocabulary and a linear SVM classifier with the Naive Bayes features, BOW-model achieves 83% accuracy for Pang and Lee's dataset, and 88.58% for Maas et al's dataset. Our model therefore performs better with about 344 times less features for the first dataset, and yields similar result with about 678 times less features for the second one.

7.3.6 Computation Time

	1-GRAM	2-GRAM	3-GRAM	1+2-GRAM	1+2+3-GRAM
N-GRAM	0	43.00	164.34	43.00	207.34
K-MEANS	14.18	291.62	747.90	302.34	1203.99
DOCUMENT	36.45	173.48	494.06	343.29	949.01
TOTAL	50.63	508.10	1406.30	688.63	2360.34

Table 7.3 – Computation time for building movie review representations with $K = 300$ semantic concepts. Time is reported in seconds.

The bag of semantic concepts model can leverage information coming from n-grams to improve sentiment classification of documents. This model has also the nice property to build document representations in an efficient and timely manner. The most time-consuming and costly process step in the model is the K-means clustering, especially when dealing with millions of n-gram representations. However, this step can be done very quickly with low memory by using mini-batch K-means method. Computation times for generating 300-dimensional representations are reported in Table 7.3. All experiments have been run on single CPU core Intel i7 2600K 3.4 GHz. Despite the fact that single CPU has been used for this benchmark, the three steps of the model are highly parallelizable. The recorded times could thus be divided by the number of CPU available. We see that representations can be computed in less than one minute with only 1-gram vocabulary. About 10 minutes are

necessary when adding 2-grams, and about 40 minutes by adding 3-grams. In comparison, LDA needs six hours for extracting 100 topics and three days for 300 topics. Our model is also very competitive with LSA which takes 540 seconds to generate 300-dimensional document representations. However, adding 2-grams and 3-grams to perform a LSA would be extremely time-consuming and memory-hungry while our model can handle it.

7.3.7 Inferring Semantic Concepts for Unseen N-grams

Another drawback of classical models is that they cannot deal with unseen words. Only words present in the training documents are used to infer representations for a new text document. Unlike these models, our model can easily assign semantic concepts for new n-grams. Because n-gram representations are based on its word vector representations, a new n-gram vector representation can be calculated if a representation is available for each of its words. This new representation is then assigned to the nearest centroid $\boldsymbol{\gamma}_k$, which determines its semantic concept. With a small training set, this is a valuable asset when compared to other models.

7.4 Experiments with Text News Classification

The classification of full-text news is another important application in natural language processing. Given the huge amount of this type of document, an unsupervised method for obtaining low-dimensional document representations is therefore interesting. In this section, we then use our method to represent text news in a low-dimensional vector space.

7.4.1 Reuters-27000 Dataset

We used the Reuters-27000 corpus recently released by Mourino-García et al. (2015)[5]. This corpus comprises 23,722 online news from Reuters agency, belonging to only one category. There are about 3,000 news for each of the 8 following categories: Health, Art, Politics, Sports, Science, Technology, Economy and Business. We divided this corpus into two parts: (1) we randomly selected 1,000 news from each category to compose a test set, (2) the remaining 15,722 documents are the training set.

7.4.2 Experimental Setup

One of the findings from the experiments on sentiment classification in Section 7.3 is that our method becomes a good alternative to classical approaches when the training size is relatively small. In this framework of text news classification, we aim to exploit this potential. We compare our method to LSA and BOW models with different training sizes $L = \{50, 100, 500, 1000, 1500, 2000\}$ [6]. Given the overall results on the Pang and Lee's dataset, we

[5] Available at http://www.itec-sde.net/reuters_27000.zip

[6] Here, LDA is deliberately left out as it gives poor performance with small datasets.

choose to group 1-grams and 2-grams into $K = 300$ clusters.

Bag of semantic concepts for text news

For each training size L, we consider only the 1-grams and 2-grams that appear in the training set to learn the centroids $\boldsymbol{\gamma}_k \in \mathbb{R}^{100}$. The resulting number of n-grams is reported in Table 7.4.

SIZE	1-GRAM	2-GRAM	1+2-GRAM
50	14839	90870	105709
100	20755	156142	176897
500	41220	506027	547247
1000	52292	801121	853413
1500	59483	1038236	1097719
2000	64010	1221578	1285588

Table 7.4 – N-gram frequencies on Reuters-27000 according to the number of documents in the training size.

As for sentiment classification, the word vector representations trained in Chapter 5 are used in this experiment. Once the centroids are defined, each news document is represented as a bag of semantic concepts by assigning n-gram representations to their nearest centroid. As described in Section 7.3.7, this method provides a way to easily assign n-grams from the test set which are not in the training. The popular TF-IDF (term frequency-inverse document frequency) is then used as weighting factors, where terms are clusters.

Other methods

In this framework, we compared our method to LSA and BOW models for each training size L. For both models, we consider words (1-grams) with TF-IDF weighting.

Classification using multiclass SVM

Reuters-27000 contains news for eight different categories. We are thus facing a multiclass problem. A common approach to solve this problem is to reduce the single multiclass problem into a multiple binary classification problems. We thus train a linear SVM classifier with a "one-versus-all" strategy, using the LIBLINEAR library (Fan et al., 2008).

7.4.3 Results

Results reported in Figure 7.1 show that our method is also competitive for representing text news in a low-dimensional vector space. It performs as well as a BOW model with the

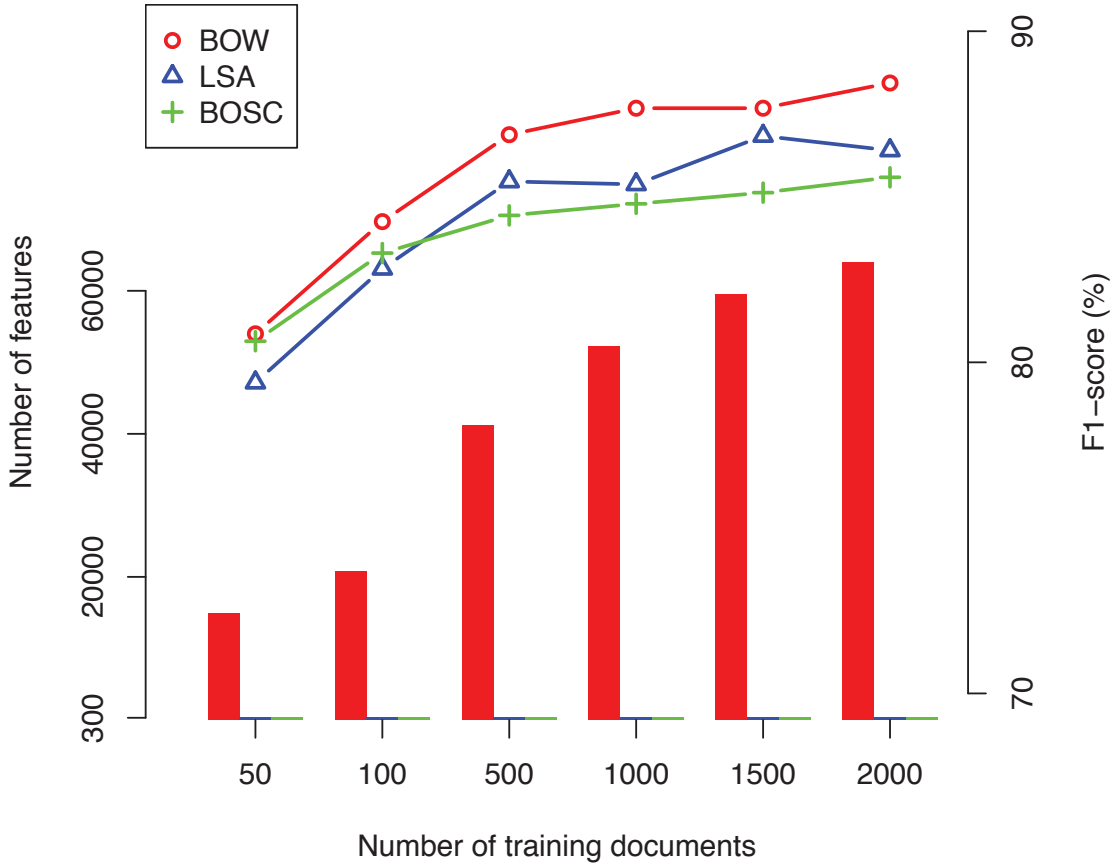


Figure 7.1 – Performance of the three models on Reuters-27000 with different training sizes. Bar chart (left axis) is the number of features. Line chart (right axis) reports the F1-scores. Our method is denoted by BOSC. BOSC and LSA have a fixed number of features, $K = 300$.

smallest training set $L = 50$, where LSA gives a poorer result. Increasing the number of training documents helps the BOW model to outperform our method, but the number of features also increases dramatically. LSA also gives slightly better results than our method when the training set contains at least 500 documents. Below this number, our method outperforms LSA. A larger training set has, however, an impact on the computational cost while our method does not suffer from that. Those results also confirm that using n-grams with $n > 1$ in text news classification is not particularly interesting. Finding the right keywords seems enough for predicting news categories.

7.5 Conclusion

In this chapter, we leverage our model described in Chapter 5 to propose an unsupervised method for producing compact representations of text documents. As word embeddings can be summed together, n-grams with different lengths n can then be embedded in a same dimensional vector space with a simple element-wise addition. This makes it possible to

compute distances between n-grams, which can have many applications in natural language processing. We therefore propose a bag of semantic concepts model to represent documents in a low-dimensional space. Semantic concepts are obtained by performing a K-means clustering which partitions all n-grams into K clusters. This model has several advantages over classical approaches for representing documents in a low-dimensional space: it leverages semantic information coming from n-grams; it builds document representations with low resource consumption (time and memory); it can infer semantic concepts for unseen n-grams; and finally, it is capable of providing relevant document representations even with a small set of documents. We have shown that such model is suitable for document classification, both for text news or movie reviews. Competitive performance has been reached on binary sentiment classification tasks, where this model outperforms traditional approaches. It also attained similar results to traditional bag-of-words with considerably fewer features.

Sentence Generation Part III

8 Phrase-based Image Captioning

Being able to automatically generate a description from an image is a fundamental problem in artificial intelligence, connecting computer vision and natural language processing. The problem is particularly challenging because it requires to correctly recognize different objects in images and how they interact. Another challenge is that an image description generator needs to express these interactions in a natural language (*e.g.* English). Therefore, a language model is implicitly required in addition to visual understanding. Recently, this problem has been studied by many different authors. Most of the attempts are based on recurrent neural networks to generate sentences. These models leverage the power of neural networks to transform image and sentence representations into a common space (Mao et al., 2015; Donahue et al., 2014; Karpathy and Fei-Fei, 2015; Vinyals et al., 2015).

In this chapter, we propose a different approach to the problem that does not rely on complex recurrent neural networks. An exploratory analysis of two large datasets of image descriptions reveals that their syntax is quite simple. The ground-truth descriptions can be represented as a collection of noun, verb and prepositional phrases. The different entities in a given image are described by the noun phrases, while the interactions or events between these entities are encoded by both the verb and the prepositional phrases. We thus train a model that predicts the set of phrases present in the sentences used to describe the images. By leveraging word vector representations defined in Chapter 5, each phrase can be represented by the average of the representations of the words that compose the phrase. Vector representations for images can also be easily obtained from some pre-trained convolutional neural networks. The model then learns a common embedding between phrase and image representations (see Figure 8.3).

Given a test image, a bilinear model is trained to predict a set of top-ranked phrases that best describe it. Several noun phrases, verb phrases and prepositional phrases are in this set. The objective is therefore to generate syntactically correct sentences from (possibly different) subsets of these phrases. We introduce a trigram constrained language model based on our knowledge about how the sentence descriptions are structured in the training set. With a very constrained decoding scheme, sentences are inferred with a beam search. Because these sentences are not conditioned to the given image (apart with the initial phrases selection),

a re-ranking is used to pick the sentence that is closest to the sample image (according to the learned metric). The quality of our sentence generation is evaluated on two very popular datasets for the task: Flickr30k (Hodosh et al., 2013) and COCO (Lin et al., 2014).

8.1 Related Work

The classical approach to sentence generation is to pose the problem as a retrieval problem: a given test image will be described with the highest ranked annotation in the training set (Hodosh et al., 2013; Socher et al., 2014; Srivastava and Salakhutdinov, 2014). These matching methods may not generate proper descriptions for a new combination of objects. Due to this limitation, several generative approaches have been proposed. Many of them use syntactic and semantic constraints in the generation process (Yao et al., 2010; Mitchell et al., 2012; Kuznetsova et al., 2012; Kulkarni et al., 2013). These approaches benefit from visual recognition systems to infer words or phrases, but in contrast to the proposed model they do not leverage a multimodal metric between images and phrases.

More recently, automatic image sentence description approaches based on deep neural networks have emerged with the release of new large datasets. As starting point, these solutions use the rich representation of images generated by Convolutional Neural Networks (LeCun et al., 1998) (CNN) that were previously trained for object recognition tasks. These CNN are generally followed by recurrent neural networks (RNN) in order to generate full sentence descriptions (Donahue et al., 2014; Chen and Zitnick, 2015; Mao et al., 2015; Venugopalan et al., 2014; Kiros et al., 2014; Karpathy and Fei-Fei, 2015; Vinyals et al., 2015). Among these recent works, long short-term memory (LSTM) is often chosen as RNN. In such approaches, the key point is to learn a common space between images and words or between images and sentences, i.e. a multimodal embedding.

Vinyals et al. (2015) consider the problem in a similar way as a machine translation problem. The authors propose an encoder/decoder (CNN/LSTM networks) system that is trained to maximize the likelihood of the target description sentence given a training image. Karpathy and Fei-Fei (2015) propose an approach that is a combination of CNN, bidirectional RNN over sentences and a structured objective responsible for a multimodal embedding. They then propose a second RNN architecture to generate new sentences. Similarly, Mao et al. (2015) and Donahue et al. (2014) propose a system that uses a CNN to extract image features and a RNN for sentences. The two networks interact with each other in a multimodal common layer.

Our model shares some similarities with these recent proposed approaches. We also use a pre-trained CNN to extract image features. However, thanks to the phrase-based approach, our model does not rely on complex recurrent networks for sentence generation, and we do not fine-tune the image features.

8.2 Syntax Analysis of Image Descriptions

The art of writing sentences can vary a lot according to the domain. When reporting news or reviewing an item, not only the choice of the words might vary, but also the general structure of the sentence. In this section, we wish to analyze the syntax of image descriptions to identify whether captions have their own structures. We therefore proceed to an exploratory analysis of two datasets containing a large amount of images with descriptions: Flickr30k (Hodosh et al., 2013) and COCO (Lin et al., 2014).

8.2.1 Datasets

The Flickr30k dataset contains 31,014 images where 1,014 images are for validation, 1,000 for testing and the rest for training (i.e. 29,000 images). The COCO dataset contains 123,287 images, 82,783 training images and 40,504 validation images[1]. We use two sets of 5,000 images from the validation images for validation and test, as in Karpathy and Fei-Fei (2015)[2]. In both datasets, images are given with five (or six) sentence descriptions annotated using Amazon Mechanical Turk (see Figure 8.3). This results in 559,113 sentences when combining both training datasets.

8.2.2 Chunking-based Approach

A quick overview over these sentence descriptions reveals that they all share a common structure, usually describing the different entities present in the image and how they interact between each other. This interaction among entities is described as actions or relative position between different objects. The sentence can be short or long, but it generally respects this process. To confirm this claim and better understand the description structures, we used a chunking (also called shallow parsing) approach which identifies the phrase chunks of a sentence (i.e., the non-recursive cores of various phrase types in text). These chunks are usually noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). We extract them from the training sentences with the SENNA software[3]. Pre-verbal and post-verbal adverb phrases are merged with verb phrases to limit the number of phrase types. Table 8.1 presents an example sentence with its chunking analysis.

a man	is grinding	a ramp	on	a skateboard	.
NP	VP	NP	PP	NP	O

Table 8.1 – Chunking analysis of an image description.

Statistics reported in Figure 8.1 and Figure 8.2 confirm that image descriptions possess a simple and distinct structure.

[1] The testing images were not released at the time of the experiment.
[2] Available at http://cs.stanford.edu/people/karpathy/deepimagesent/
[3] Available at http://ml.nec-labs.com/senna/

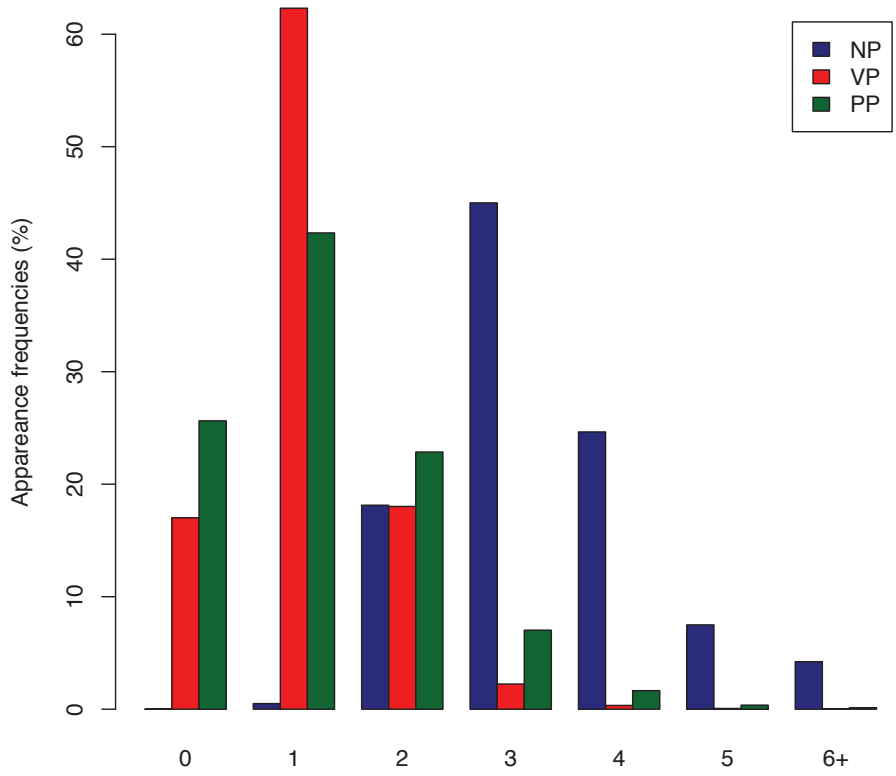


Figure 8.1 – Statistics on the number of phrase chunks (NP, VP, PP) per ground-truth descriptions in Flickr30k and COCO training datasets. Best viewed in colors.

These sentences do not have much variability. All the key elements in a given image are usually described with a noun phrase (NP). Interactions between these elements can then be explained using prepositional phrases (PP) or verb phrases (VP). A large majority of sentences contains from two to four noun phrases. Two noun phrases then interact using a verb or prepositional phrase. Describing an image is therefore just a matter of identifying these chunks. We thus propose to train a model which can predict the phrases which are likely to be in a given image.

8.3 Phrase-based Model for Image Descriptions

By leveraging pre-trained word and image representations, we propose a simple model which can predict the phrases that best describe a given image. For this purpose, a metric between images and phrases is trained, as illustrated in Figure 8.3. The proposed architecture is then just a low-rank bilinear model $\mathbf{U}^T\mathbf{V}$.

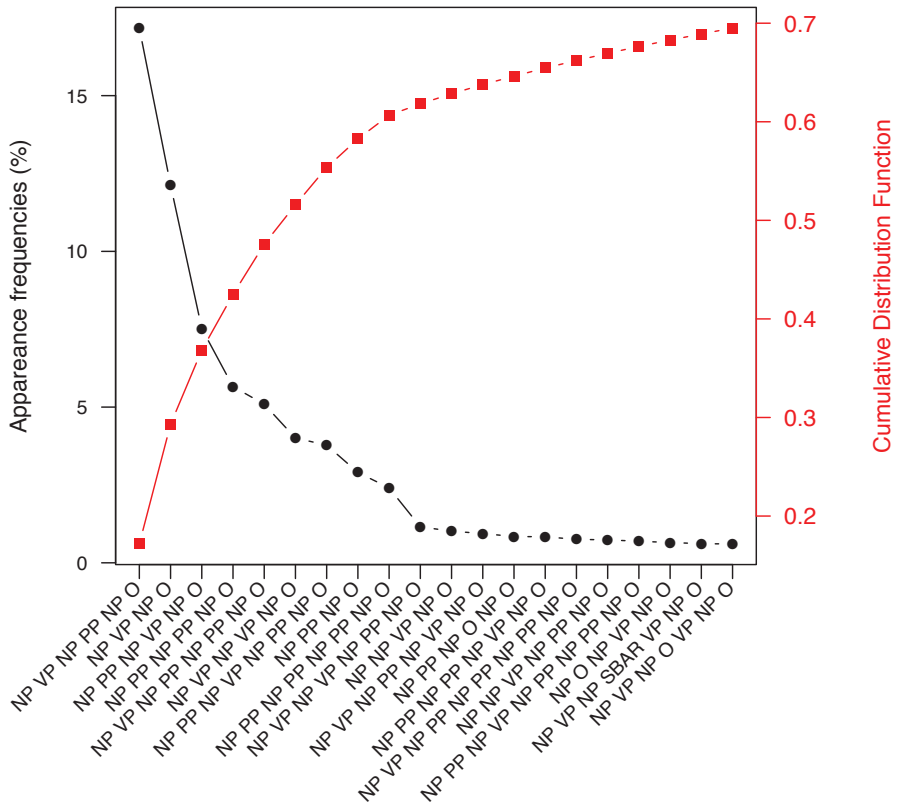


Figure 8.2 – The 20 most frequent sentence structures in Flickr30k and COCO training datasets. The black line is the appearance frequency for each structure, the red line is the cumulative distribution. Best viewed in colors.

8.3.1 Image Representations

For the representation of images, we choose to use a Convolutional Neural Network. CNN have been widely used in different vision domains and are currently the state-of-the-art in many object recognition tasks. We consider a CNN that has been pre-trained for the task of object classification (Simonyan and Zisserman, 2014). We use a CNN solely to the purpose of feature extraction, that is, no learning is done in the CNN layers.

8.3.2 Learning a Common Space for Image and Phrase Representations

Let $\mathcal{I}$ be the set of training images, $\mathcal{C}$ the set of all phrases used to describe $\mathcal{I}$, and θ the trainable parameters of the model. By representing each image $i \in \mathcal{I}$ with a vector $\mathbf{z}_i \in \mathbb{R}^n$ thanks to the pre-trained CNN, we define a metric between the image i and a phrase c as a bilinear operation:

$$\phi_\theta(c, i) = \mathbf{u}_c^T \mathbf{V} \mathbf{z}_i \,, \tag{8.1}$$

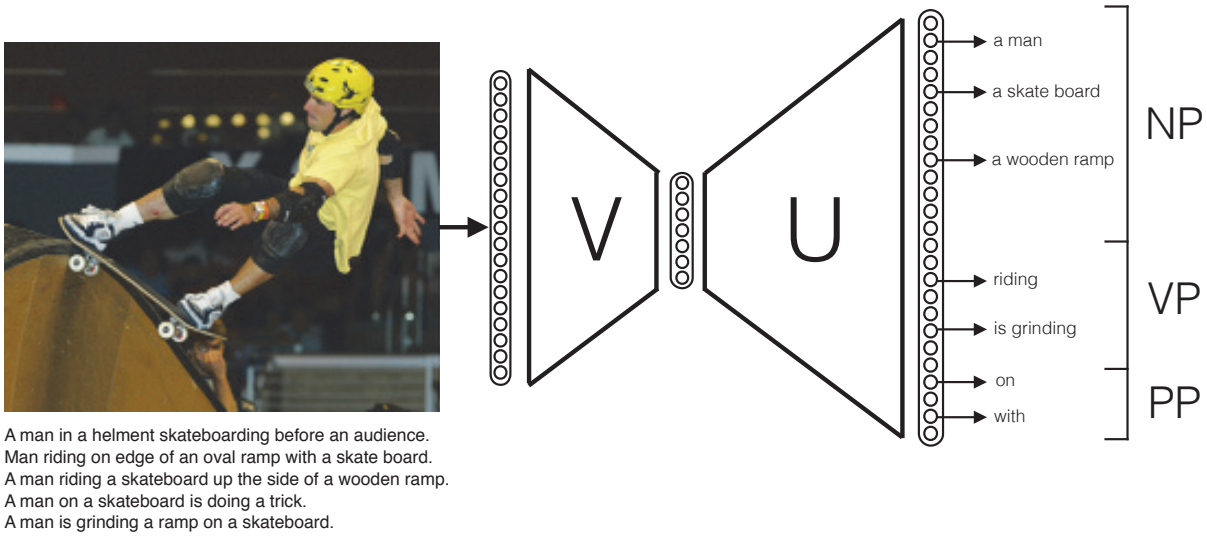


Figure 8.3 – Schematic illustration of our phrase-based model for image descriptions.

with $\mathbf{U} = (\mathbf{u}_{c_1}, \ldots, \mathbf{u}_{c_{|\mathcal{C}|}}) \in \mathbb{R}^{m \times |\mathcal{C}|}$ and $\mathbf{V} \in \mathbb{R}^{m \times n}$ being the trainable parameters θ. Note that $\mathbf{U}^T\mathbf{V}$ could be a full matrix, but a low-rank setting eases the capacity control.

8.3.3 Phrase Representations Initialization

Noun phrases or verb phrases are often a combination of several words. By leveraging the ability of our word vector representations to compose by simple summation (see Chapter 5), representations for phrases are easily computed with an element-wise addition. A vector representation $\mathbf{u}_c \in \mathbb{R}^m$ for a phrase $c = \{w_1, \ldots, w_K\}$ is then calculated by averaging its word vector representations pre-trained on Wikipedia:

$$\mathbf{u}_c = \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_{w_k} . \tag{8.2}$$

Vector representations for all phrases $c \in \mathcal{C}$ can thus be obtained to initialized the matrix $\mathbf{U} \in \mathbb{R}^{m \times |\mathcal{C}|}$. $\mathbf{V} \in \mathbb{R}^{m \times n}$ is initialized randomly and trained to encode images in the same vector space than the phrases used for their descriptions.

8.3.4 Training with Negative Sampling

Each image i is described by a multitude of possible phrases $\mathcal{C}^i \subseteq \mathcal{C}$. We consider $|\mathcal{C}|$ classifiers attributing a score for each phrase. We train our model to discriminate a target phrase c_j from a set of negative phrases $c_k \in \mathcal{C}^- \subseteq \mathcal{C}$, with $c_k \neq c_j$. With $\theta = \{\mathbf{U}, \mathbf{V}\}$, we minimize the following logistic loss function with respect to θ:

$$\mathcal{L}(\theta) = \sum_{i \in \mathcal{I}} \sum_{c_j \in \mathcal{C}^i} \left(\log\left(1 + e^{-\phi_\theta(c_j, i)}\right) + \sum_{c_k \in \mathcal{C}^-} \log\left(1 + e^{+\phi_\theta(c_k, i)}\right) \right). \tag{8.3}$$

The model is trained using stochastic gradient descent. A new set of negative phrases $\mathcal{C}^-$ is randomly picked from the training set at each iteration.

8.4 From Phrases to Sentence

After identifying the L most likely constituents c_j in the image i, we propose to generate sentences out of them. From this set, $l \in \{1,\ldots,L\}$ phrases are used to compose a syntactically correct description.

8.4.1 Sentence Generation

Using a statistical language modeling framework, the likelihood of a certain sentence is given by:

$$P(c_1, c_2, \ldots, c_l) = \prod_{j=1}^{l} P(c_j | c_1, \ldots, c_{j-1}) \tag{8.4}$$

Keeping this system as simple as possible and using the second order Markov property, we approximate Equation 9.2 with a trigram language model:

$$P(c_1, c_2, \ldots, c_l) \approx \prod_{j=1}^{l} P(c_j | c_{j-2}, c_{j-1}) \,. \tag{8.5}$$

The best candidate corresponds to the sentence $P(c_1, c_2, \ldots, c_l)$ which maximizes the likelihood of Equation 8.5 over all the possible sizes of sentence. Because we want to constrain the decoding algorithm to include prior knowledge on chunking tags $t \in \{NP, VP, PP\}$, we rewrite Equation 8.5 as:

$$\begin{aligned} P(c_1, c_2, \ldots, c_l) &= \prod_{j=1}^{l} \sum_{t} P(c_j | t_j = t, c_{j-2}, c_{j-1}) P(t_j = t | c_{j-2}, c_{j-1}) \\ &= \prod_{j=1}^{l} P(c_j | t_j, c_{j-2}, c_{j-1}) P(t_j | c_{j-2}, c_{j-1}) \,. \end{aligned} \tag{8.6}$$

Both conditions $P(c_j | t_j, c_{j-2}, c_{j-1})$ and $P(t_j | c_{j-2}, c_{j-1})$ are probabilities estimated by counting trigrams in the training datasets.

8.4.2 Sentence Decoding

At decoding time, we prune the graph of all possible sentences made out of the top L phrases with a beam search, according to three heuristics:

- we consider only the transitions which are likely to happen (we discard any sentence

which would have a trigram transition probability inferior to 0.01). This thresholding helps to discard sentences that are semantically incorrect;

- each predicted phrases c_j may appear only once[4];
- we add syntactic constraints which are illustrated in Figure 8.4.

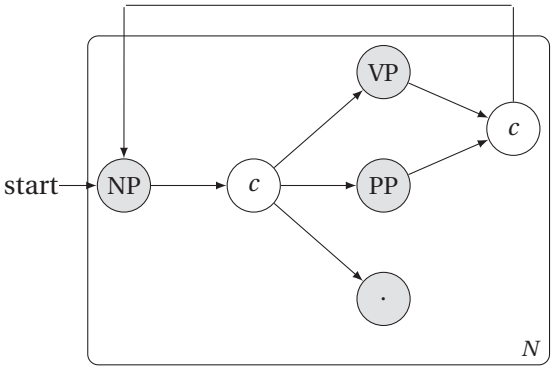


Figure 8.4 – The constrained language model for generating description given the predicted phrases for an image.

The last heuristic is based on the analysis of syntax in Section 8.2. In Figure 8.2, we see that a noun phrase is, in general, always followed by a verb phrase or a prepositional phrase, and both are then followed by another noun phrase. A large majority of the sentences contain three noun phrases interleaved with verb phrases or prepositional phrases. According the statistics reported in Figure 8.1, sentences with two or four noun phrases are also common, but sentences with more than four noun phrases are marginal. We thus repeat this process $N = \{2,3,4\}$ times until reaching the end of a sentence (characterized by a period).

8.4.3 Sentence Re-ranking

For each test image i, the proposed model will generate a set of M sentences. Sentence generation is not conditioned on the image, apart from phrases which are selected beforehand. Some phrase sequences might be syntactically good, but have low match with the image. Consider, for instance, an image with a cat and a dog. Both sentences *"a cat sitting on a mat and a dog eating a bone"* and *"a cat sitting on a mat"* are correct, but the second is missing an important part of the image. A ranking of the generated sentences is therefore necessary to choose the one that has the best match with the image.

Because a generated sentence is composed from l phrases predicted by our system, we simply average the phrase scores given by Equation 8.1. For a generated sentence s composed of l

[4]This is easy to implement with a beam search, but intractable with a full search.

phrases c_j, a score between s and i is calculated as:

$$\phi_\theta(s,i) = \frac{1}{l}\sum_{c_j \in s} \phi_\theta(c_j, i)\,. \tag{8.7}$$

The best candidate is the sentence which has the highest score out of the M generated sentences. This ranking helps the system to chose the sentence which is closer to the sample image.

8.5 Experiments

8.5.1 Experimental Setup

Feature Selection

Following Karpathy and Fei-Fei (2015), the image features are extracted using VGG CNN (Simonyan and Zisserman, 2014). This model generates image representations of dimension 4096 from RGB input images.

For each training set, only phrases occurring at least ten times are considered. This threshold is chosen to fulfill two objectives: (i) limit the number of phrases $\mathcal{C}$ and therefore the size of the matrix $\mathbf{U}$ and (ii) exclude rare phrases to better generalize the descriptions. Statistics on the number of phrases are reported in Table 8.2.

	FLICKR30K	COCO
NOUN PHRASE (NP)	4818	8982
VERB PHRASE (VP)	2109	3083
PREPOSITIONAL PHRASE (PP)	128	189
TOTAL $\lvert\mathcal{C}\rvert$	7055	12254

Table 8.2 – Statistics of phrases appearing at least ten times.

For Flickr30k, this threshold covers about 81% of NP, 83% of VP and 99% of PP. For COCO, it covers about 73% of NP, 75% of VP and 99% of PP. Phrase representations are computed by averaging vector representations of their words, trained with the model described in Chapter 5 in $d^{\text{wrd}} = 400$-dimensional vectors.

Learning the Multimodal Metric

The parameters θ are $\mathbf{V} \in \mathbb{R}^{400\times 4096}$ (initialized randomly) and $\mathbf{U} \in \mathbb{R}^{400\times|\mathcal{C}|}$ (initialized with the phrase representations), which are trained with 15 randomly chosen negative samples. It takes about 2.5 hours on single CPU (Intel i7 4930K 3.4 GHz) to train on the COCO training dataset,

which is much faster than models based on deep neural networks.

Generating Sentences from the Predicted Phrases

Transition probabilities for our constrained language model (see Figure 8.4) are calculated independently for each training set. No smoothing has been used in the experiments. Concerning the set of top-ranked phrases for a given test image, we select only the top five predicted verb phrases and the top five predicted prepositional phrases. Since the average number of noun phrases is higher than for the two other types of phrases (see Figure 8.1), more noun phrases are needed. The top twenty predicted noun phrases are thus selected.

8.5.2 Experimental Results

	FLICKR30K	COCO
NOUN PHRASE (NP)	38.14	45.44
VERB PHRASE (VP)	20.61	27.83
PREPOSTIONAL PHRASE (PP)	81.70	84.49
TOTAL	44.92	52.49

Table 8.3 – Recall on phrase retrieval. For each test image, we take the top 20 predicted NP, the top 5 predicted VP, and the top 5 predicted PP.

As a first evaluation, we consider the task of retrieving the ground-truth phrases from test image descriptions. Results reported in Table 8.3 show that our system achieves a recall of around 50% on this task on the test set of both datasets, assuming the threshold considered for each type of phrase (see 8.5.1). Note that this task is extremely difficult, as semantically similar phrases (*the women* / *women* / *the little girls*) are classified separately. Despite the possible number of noun phrases being higher, results in Table 8.3 reveal that noun phrases are better retrieved than verb phrases. This shows that our system is able to detect different objects in the image. However, finding the right verb phrase seems to be more difficult. A possible explanation could be that there exists a wide choice of verb phrases to describe interactions between the noun phrases. For instance, we see in Figure 8.3 that two annotators have used the same noun phrases (*a man*, *a skateboard* and *a (wooden) ramp*) to describe the scene, but they have then chosen a different verb phrase to link them (*riding* versus *is grinding*). Therefore, we suspect that a low recall for verb phrases does not necessarily mean that the predictions are wrong. Finding the right prepositional phrase seems, on the contrary, much easier. The high recall for prepositional phrase can be explained by much lower variability of this type of phrase compared to the two others (see Table 8.2).

As a second evaluation, we consider the task of generating full descriptions. We measure the

	FLICKR30K				COCO			
	B-1	B-2	B-3	B-4	B-1	B-2	B-3	B-4
HUMAN AGREEMENT	0.55	0.35	0.23	0.15	0.68	0.45	0.30	0.20
MAO ET AL. (2015)	0.60	0.41	0.28	0.19	0.67	0.49	0.35	0.25
KARPATHY AND FEI-FEI (2015)	0.57	0.37	0.24	0.16	0.62	0.45	0.32	0.23
VINYALS ET AL. (2015)	0.66	0.42	0.28	0.18	0.67	-	-	-
DONAHUE ET AL. (2014)	0.59	0.39	0.25	0.16	0.63	0.44	0.30	0.21
OUR MODEL	0.60	0.37	0.22	0.14	0.73	0.50	0.34	0.23

Table 8.4 – Comparison between human agreement scores, state-of-the-art models and our model on both datasets. Note that there are slight variations between the test sets chosen in each paper. Scores are reported in terms of BLEU metric.

quality of the generated sentences using the popular, yet controversial, BLEU score (Papineni et al., 2002). Table 8.4 shows our sentence generation results on the two datasets considered. BLEU scores are reported up to 4-gram. Human agreement scores are computed by comparing the first ground-truth description against the four others[5]. For comparison, we include results from recently proposed models. Our model, despite being simpler, achieves similar results to state-of-the-art results. It is interesting to note that our results are very close to the human agreement scores.

We show examples of full automatic generated sentences in Figure 8.5. The simple language model used is able to generate sentences that are in general syntactically correct. Our model produces sensible descriptions with variable complexity for different test samples. Due to the generative aspect of the model, it can occur that the sentence generated is very different from the ground-truth and still provides a descent description. The last row of Figure 8.5 illustrates failure samples. We can see in these failure samples that our system has however outputted relevant phrases. There is still room for improvement for generating the final description. We deliberately choose a simple language model to show that competitive results can be achieved with a simple approach. A more complex language model could probably avoid these failure samples by considering a larger context. The probability for *a dog* to stand on top of *a wave* is obviously very low, but this kind of mistake cannot be detected with a simple trigram language model.

8.5.3 Diversity of Image Descriptions

In contrast to RNN-based models, our model is not trained to match a given image i with its ground-truth descriptions s, i.e., to give $P(s|i)$. Because our model outputs instead a set

[5]For all models, BLEU scores are computed against five reference sentences which give a slight advantage compared to human scores.

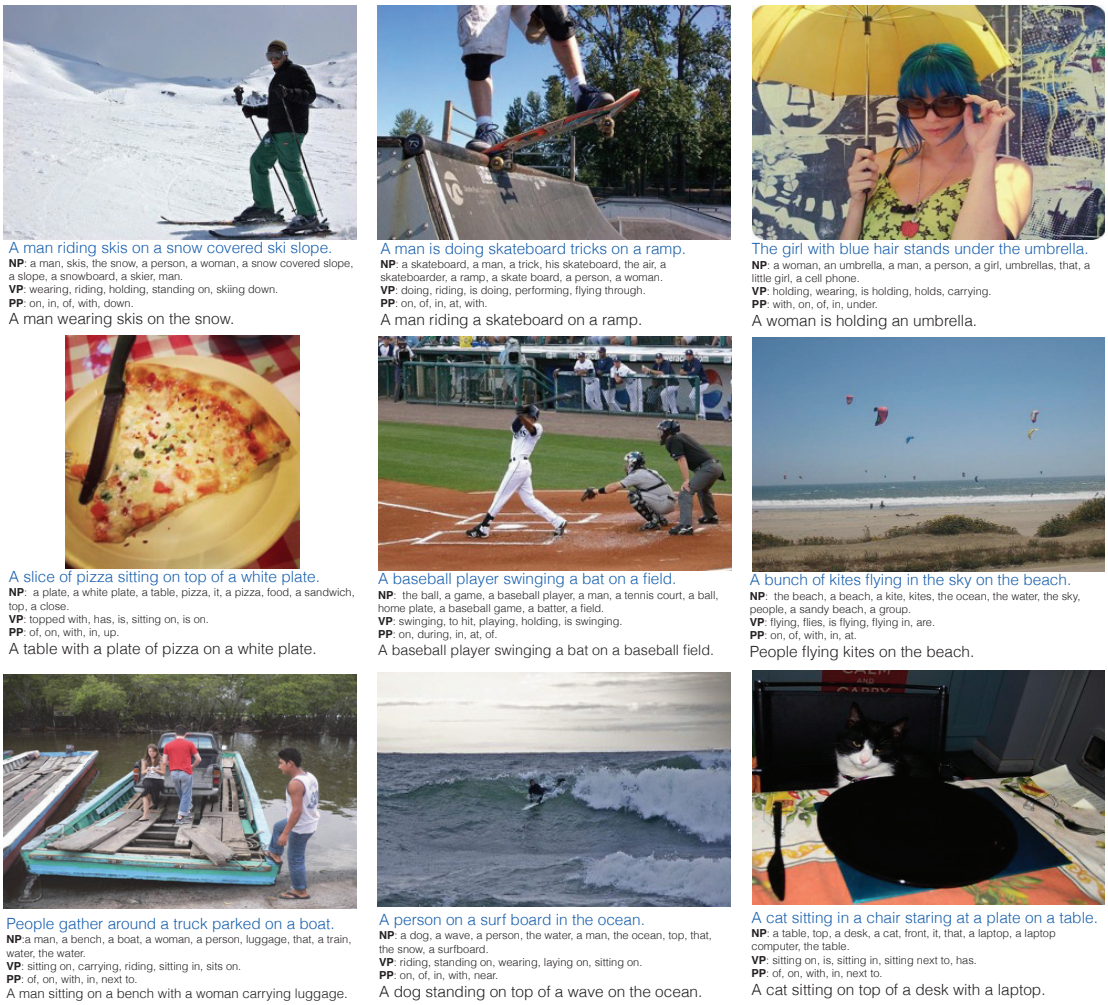


Figure 8.5 – Qualitative results for images on the COCO dataset. Ground-truth annotation (in blue, at the top), the NP, VP and PP predicted from the model and generated annotation (in black, at the bottom) are shown for each image. The last row are failure samples.

of phrases, this is not really surprising that only 1% of our generated descriptions are in the training set for Flickr30k, and 9.7% for COCO. While a RNN-based model is generative, it might easily over-fit a small training data. Vinyals et al. (2015) report, for instance, that the generated sentence is present in the training set 80% of the time. Our model therefore offers a good alternative with the possibility of producing unseen descriptions with a combination of phrases from the training set.

8.5.4 Phrase Representation Fine-Tuning

Before training the model, the matrix **U** is initialized with phrase representations obtained from the whole English Wikipedia. This corpus of unlabeled text is well structured and large

PHRASES		NEAREST NEIGHBORS	
	#	*before*	*after*
A GREY CAT	1	A GREY DOG	A GRAY CAT
	2	A GREY AND BLACK CAT	A GREY AND BLACK CAT
	3	A GRAY CAT	A BROWN CAT
	4	A GREY ELEPHANT	A GREY AND WHITE CAT
	10	A YELLOW CAT	GREY AND WHITE CAT
HOME PLATE	1	A HOME PLATE	A HOME PLATE
	4	A PLATE	HOME BASE
	6	ANOTHER PLATE	THE PITCH
	9	A RED PLATE	THE BATTER
	10	A DINNER PLATE	A BASEBALL PITCH
A HALF PIPE	1	A PIPE	A PIPE
	2	A HALF	THE RAMP
	5	A SMALL CLOCK	A HAND RAIL
	9	A LARGE CLOCK	A SKATE BOARD RAMP
	10	A SMALL PLATE	AN EMPTY POOL

Table 8.5 – Examples of three noun phrases from the COCO dataset with five of their nearest neighbors before and after learning.

enough to provide good word vector representations, which can then produce good phrase representations. However, the content of Wikipedia is clearly different from the content of the image descriptions. Some words used for describing images might be used in different contexts in Wikipedia, which can lead to out-of-domain representations for certain phrases. This becomes thus crucial to adapt these phrase representations by fine-tuning the matrix **U** during the training[6]. Some examples of noun phrases are reported in Table 8.5 with their nearest neighbors before and after the training. These confirm the importance of fine-tuning to incorporate visual features. In Wikipedia, *cat* seems to occur in the same context than *dog* or other animals. When looking at the nearest neighbors of a phrase such as *a grey cat*, other *grey* animals arise. After training on images, the word *cat* becomes the important feature of that phrase. And we see that the nearest neighbors are now cats with different colors. In some cases, averaging word vectors to represent phrases is not enough to capture the semantic meaning. Fine-tuning is thus also important to better learn specific phrases. Images related to baseball games, for example, have enabled the phrase *home plate* to be better defined. This is also true for the phrase *a half pipe* with images about skateboarding. This leads to interesting

[6]Experiments with a fixed **U** phrase representations matrix significantly hurt the general performance. We observe about a 50% decrease in both datasets with the BLEU metric. Since the number of trainable parameters is reduced, the capacity of **V** should be increased to guarantee a fair comparison.

phrase representations, grounded in the visual world, which could be possibly used in natural language applications in future work.

8.6 Conclusion

With this model, we propose to infer different phrases from image samples by leveraging pre-trained word and image representations. From the phrases predicted, our model is able to automatically generate sentences using a statistical language model. We show that the problem of sentence generation can be effectively achieved without the use of complex recurrent networks. Our algorithm, despite being simpler than state-of-the-art models, achieves similar results on this task. Also, our model generate new sentences which are not generally present in training set. Future research directions will go towards leveraging unsupervised data and more complex language models to improve sentence generation. Another interest is assessing the impact of visually grounded phrase representations into existing natural language processing systems.

9 Generating Text from Structured Data

Concept-to-text generation addresses the problem of rendering structured records into natural language (Reiter et al., 2000). A typical application is to generate a weather forecast based on a set of structured records of meteorological measurements. While previous work has experimented with datasets that contain only a few tens of thousands of records such as WEATHERGOV reports, or the ROBOCUP dataset, we scale to the large and very diverse problem of generating biographies for personalities based on the structured data contained in Wikipedia infoboxes. Similar applications include the generation of product descriptions based on a product catalog which may contain millions of items with dozens of attributes each.

To tackle this problem we introduce a statistical generation model conditioned on a Wikipedia infobox. Such diversity makes it difficult for classical count-based models to estimate probabilities of rare events due to data sparsity. We address this issue by parameterizing words and fields as vectors (embeddings), along with a neural language model operating on these embeddings. This factorization allows us to scale to a large number of words and fields compared to Liang et al. (2009) and Kim and Mooney (2010) where the number of parameters grows as the product of the number of words and fields. Moreover, our approach does not restrict the relations between the record content and the generated text. This contrasts with less flexible strategies that assume the generation to follow an hybrid alignment tree (Kim and Mooney, 2010), a probabilistic context-free grammar (Konstas and Lapata, 2013), or a tree adjoining grammar (Gyawali and Gardent, 2014).

Our model exploits structured data both globally and locally. Global conditioning summarizes all information about a personality to understand high-level themes such as that the biography is about a scientist or an artist, while as local conditioning, or attention, describes the previously generated tokens in terms of the their relationship to the infobox. We analyze the effectiveness of each and demonstrate their complementarity.

9.1 Related Work

Traditionally, generation systems relied on rules and hand-crafted specifications (Dale et al., 2003; Reiter et al., 2005; Green, 2006; Galanis and Androutsopoulos, 2007; Turner et al., 2010). Generation is divided into modular, yet highly interdependent, decisions: (1) *content planning* defines which parts of the input fields or meaning representations should be selected; (2) *sentence planning* determines which selected fields are to be dealt with in each output sentence; and (3) *surface realization* generates those sentences.

Data-driven approaches have been proposed to automatically learn the individual modules. One approach first aligns records and sentences and then learns a content selection model (Duboue and McKeown, 2002; Barzilay and Lapata, 2005). Hierarchical hidden semi-Markov generative models have also been used to first determine which facts to discuss and then to generate words from the predicates and arguments of the chosen facts (Liang et al., 2009). Sentence planning has been formulated as a supervised set partitioning problem over facts where each partition corresponds to a sentence (Barzilay and Lapata, 2006). End-to-end approaches have combined sentence planning and surface realization by using explicitly aligned sentence/meaning pairs as training data (Ratnaparkhi, 2002; Wong and Mooney, 2007; Belz, 2008; Lu and Ng, 2011). More recently, content selection and surface realization have been combined (Angeli et al., 2010; Kim and Mooney, 2010; Konstas and Lapata, 2013).

Our model is probably most similar to Mei et al. (2016) who use an encoder-decoder style neural network model to tackle the limited WEATHERGOV and ROBOCUP tasks. Their architecture relies on LSTM units and a fairly complicated attention mechanism which reduces it's scalability compared to our much simpler design.

Our approach is inspired by the recent success of neural language models in image captioning (Kiros et al., 2014; Karpathy and Fei-Fei, 2015; Vinyals et al., 2015; Fang et al., 2015; Xu et al., 2015), neural machine translation (Devlin et al., 2014; Bahdanau et al., 2015; Luong et al., 2015), and in modeling conversations and dialogues (Shang et al., 2015; Wen et al., 2015; Yao et al., 2015).

9.2 Language Modeling for Constrained Sentence Generation

Conditional language models are a popular choice to generate sentences. We introduce a table-conditioned language model for constraining the sentence generation to include elements from fact tables.

9.2.1 Language Model

Given a sentence $s = w_1, \ldots, w_T$ composed of T words from a vocabulary $\mathcal{W}$, a language model estimates:

$$P(s) = \prod_{t=1}^{T} P(w_t | w_1, \ldots, w_{t-1}) . \tag{9.1}$$

Let $c_t = w_{t-(n-1)}, \ldots, w_{t-1}$ be the sequence of $n-1$ context words preceding $w_t \in s$. In an n-gram language model, Equation 9.1 is approximated as

$$P(s) \approx \prod_{t=1}^{T} P(w_t | c_t) , \tag{9.2}$$

assuming an order n Markov property.

9.2.2 Language Model Conditioned on Tables

Frederick Parker-Rhodes	
Born	21 November 1914 Newington, Yorkshire
Died	2 March 1987 (aged 72)
Residence	UK
Nationality	British
Fields	Mycology, Plant Pathology, Mathematics, Linguistics, Computer Science
Known for	Contributions to computational linguistics, combinatorial physics, bit-string physics, plant pathology, and mycology
Author abbrev. (botany)	Park.-Rhodes

Figure 9.1 – Wikipedia infobox of Frederick Parker-Rhodes. The introduction of his article reads: “Frederick Parker-Rhodes (21 March 1914 – 21 November 1987) was an English linguist, plant pathologist, computer scientist, mathematician, mystic, and mycologist.”.

As seen in Figure 9.1, a table consists of a set of field/value pairs, where values are sequences of words. We therefore propose language models that are conditioned on these pairs.

Local conditioning

The table allows us to describe each word not only by its string (or index in the vocabulary) but also by a descriptor of its occurrence in the table. Let $\mathcal{F}$ define the set of all possible fields

f. The occurrence of a word w in the table is described by a set of (field, position) pairs.

$$z_w = \{(f_i, p_i)\}_{i=1}^{m}, \tag{9.3}$$

where m is the number of occurrences of w. Each pair (f, p) indicates that w occurs in field f at position p. In this scheme, most words are described by the empty set as they do not occur in the table. For example, the word *linguistics* in the table of Figure 9.1 is described as follows:

$$z_{\text{linguistics}} = \{(\text{fields}, 8); (\text{known for}, 4)\}, \tag{9.4}$$

assuming words are lower-cased and commas are treated as separate tokens.

Conditioning both on the field type and the position within the field allows the model to encode field-specific regularities, e.g., a number token in a date field is likely followed by a month token; knowing that the number is the first token in the date field makes this even more likely.

The (field, position) description scheme of the table does not allow to express that a token terminates a field which can be useful to capture field transitions. For biographies, the last token of the name field is often followed by an introduction of the birth date like '(' or 'was born'. We hence extend our descriptor to a triplet that includes the position of the token counted from the end of the field:

$$z_w = \{(f_i, p_i^+, p_i^-)\}_{i=1}^{m}, \tag{9.5}$$

where our example becomes:

$$z_{\text{linguistics}} = \{(\text{fields}, 8, 4); (\text{known for}, 4, 13)\}. \tag{9.6}$$

We extend Equation 9.2 to use the above information as additional conditioning context when generating a sentence s:

$$P(s|z) = \prod_{t=1}^{T} P(w_t|c_t, z_{c_t}), \tag{9.7}$$

where $z_{c_t} = z_{w_{t-(n-1)}}, \ldots, z_{w_{t-1}}$ are referred to as the local conditioning variables since they describe the local context (previous word) relations with the table.

Global conditioning

The set of fields available in a table often impacts the structure of the generation. For biographies, the fields used to describe a politician are different from the ones for an actor or an athlete. Knowing which fields are available in the table provides type information and helps to determine which fields should be mentioned, both of which greatly influence sentence

structure. We introduce global conditioning on the fields g_f as

$$P(s|z, g_f) = \prod_{t=1}^{T} P(w_t|c_t, z_{c_t}, g_f). \tag{9.8}$$

Similarly, global conditioning g_w on the words occurring in the table is introduced:

$$P(s|z, g_f, g_w) = \prod_{t=1}^{T} P(w_t|c_t, z_{c_t}, g_f, g_w). \tag{9.9}$$

Words provide information complementary to fields. For example, it may be hard to distinguish a basketball player from a hockey player by looking only at the field names, e.g. teams, league, position, weight and height, etc. However the actual field values such as team names, league name, player's position can help the model to give a better prediction. Here, $g_f \in \{0,1\}^{\mathcal{F}}$ and $g_w \in \{0,1\}^{\mathcal{W}}$ are binary indicators over fixed field and word vocabularies.

Figure 9.2 illustrates the model with a schematic example. For predicting the next word w_t after a given context c_t, we see that the language model is conditioned with sets of triplets for each word occurring in the table, along with all fields and words from this table.

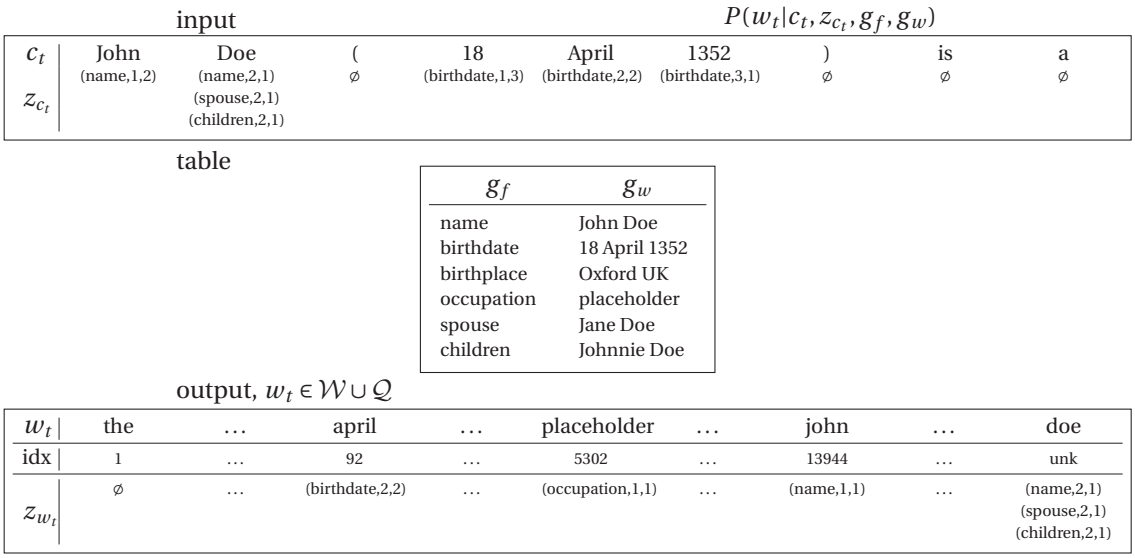


Figure 9.2 – Schematic example of the language model conditioned on tables.

9.2.3 Copy Actions

So far, we have extended context description with table information. The scoring of each potential output word can also leverage table information. In particular, knowing that a word appears in the table is valuable. For instance, sentences which expresses facts from a given table often copies words from the table. We therefore extend our language model to enable copy operations. Like for context/input conditioning, we describe each word w with both

its string (or index in a vocabulary) and its table descriptor z_w. Our model reads a table and defines an output domain $\mathcal{Q} \cup \mathcal{W}$ which encompases all vocabulary words $\mathcal{W}$ as well as all table tokens $\mathcal{Q}$, as illustrated in Figure 9.2. A side effect is that we can generate words which are outside our vocabulary, for instance a word like *Park-Rhodes* from the table of Figure 9.1 is unlikely to be in the vocabulary $\mathcal{W}$, but its table descriptor expresses that it occurs as the second token of the name field. Therefore the output space of each decision $\mathcal{Q} \cup \mathcal{W}$ is often larger than $\mathcal{W}$.

9.3 A Neural Language Model Approach

As described in Section 2.3.3, a feed-forward neural network language model (NNLM) estimates $P(w_t|c_t)$ in Equation 9.1 with a parametric function ϕ_θ which results from the composition of simple differentiable functions or *layers*. Given a context input c_t, it outputs a score for each next word $w_t \in \mathcal{W}$, $\phi_\theta(w_t, c_t)$.

9.3.1 Embeddings as Inputs

A key aspect of neural language models is the use of word embeddings as inputs. Similar words have generally similar embeddings, as they share latent features. Because the probability estimates are smooth functions of the continuous word embeddings, a small change in the features results in a small change in the probability estimation (Bengio et al., 2003). Therefore, the neural language model can achieve better generalization for unseen n-grams. Just as the discrete feature representations of words are mapped into continuous word embeddings, the discrete feature representations of tables can be mapped into continuous vector spaces, as illustrated in Figure 9.3.

Word embeddings

As described in Section 2.1.3, the embedding layer maps each context word index to a continuous d^{wrd}-dimensional vector. It relies on a parameter matrix $\mathbf{E} \in \mathbb{R}^{d^{\text{wrd}} \times |\mathcal{W}|}$ to convert the input c_t into $n-1$ vectors of dimension d^{wrd}:

$$\mathbf{c}_t = \left[\mathbf{E}_{w_{t-(n-1)}}; \ldots; \mathbf{E}_{w_{t-1}}\right] \in \mathbb{R}^{(d^{\text{wrd}} \times (n-1))}, \tag{9.10}$$

where $\mathbf{E}$ is initialized with pre-trained word embeddings obtained with Hellinger PCA (see Chapter 4).

Embeddings for tables

As described in Section 9.2.2, the language model is conditioned with elements coming from the tables. Embedding matrices are therefore defined to model both local and global

conditioning information. For local conditioning, we denote l the maximum length that a sequence of words. Each field $f_j \in \mathcal{F}$ is associated with $2 \times l$ vectors of d dimensions, a first set of l vectors covers each possible starting position in $1, \ldots, l$ and a similar set covers ending positions. This results in a parameter matrix $\mathbf{Z} \in \mathbb{R}^{|\mathcal{F}| \times (2 \times l) \times d}$. For a given triplet (f_j, p_i^+, p_i^-), $\mathbf{Z}_{j,p_i^+}$ and $\mathbf{Z}_{j,p_i^-}$ respectively refer to the embedding vectors of the start and end position for field f_j.

Finally, the global conditioning is handled with two other parameter matrices $\mathbf{G}^f \in \mathbb{R}^{|\mathcal{F}| \times g}$ and $\mathbf{G}^w \in \mathbb{R}^{|\mathcal{W}| \times g}$. Each row $\mathbf{G}_j^f$ maps a table field f_j into a vector of dimension g, while each row $\mathbf{G}_t^w$ maps a word w_t into a vector of the same dimension. In general, $\mathbf{G}^w$ shares its parameters with $\mathbf{E}$, provided that d^{wrd} is equal to g.

Aggregating embeddings

We represent each occurrence of a word w_t as a triplet (field, start, end) where we have embeddings for the start and end position as described above. Often times a particular word w_t occurs multiple times in a table, e.g., 'linguistics' has two instances in Figure 9.1. In this case, we perform a component-wise max over the start embeddings of all instances of w_t to obtain the best features across all occurrences of w_t. We do the same for end position embeddings:

$$\mathbf{z}_{w_t} = \Big[\max\{\mathbf{Z}_{j,p_i^+}, \forall (f_j, p_i^+, p_i^-) \in z_{w_t}\}; \max\{\mathbf{Z}_{j,p_i^-}, \forall (f_j, p_i^+, p_i^-) \in z_{w_t}\} \Big] \tag{9.11}$$

Note that a special no-field embedding is assigned to w_t, when that word is not associated with any fields.

For global conditioning, we define $\mathcal{F}^q \subset \mathcal{F}$ as the set of all the fields in a given table q, and $\mathcal{Q}$ as the set of all words in q. We also perform max aggregation. This yields the vectors

$$\mathbf{g}_f = \max\{\mathbf{G}^f, \forall f \in \mathcal{F}^q\}, \tag{9.12}$$

and

$$\mathbf{g}_w = \max\{\mathbf{G}^w, \forall w \in \mathcal{Q}\}. \tag{9.13}$$

The final context input is then the concatenation of these vectors:

$$\mathbf{x}_{c_t} = \left[\mathbf{c}_t; \mathbf{z}_{c_t}; \mathbf{g}_f; \mathbf{g}_w\right] \in \mathbb{R}^{d^1}, \tag{9.14}$$

with $d^1 = (n-1) \times (2 \times d + d^{\text{wrd}}) + (2 \times g)$.

This input is mapped to a latent context representation using a linear operation followed by a non-linear activation function $h(\cdot)$,

$$\mathbf{h}_{c_t} = h\big(\mathbf{W}^1 \mathbf{x}_{c_t} + \mathbf{b}^1\big). \tag{9.15}$$

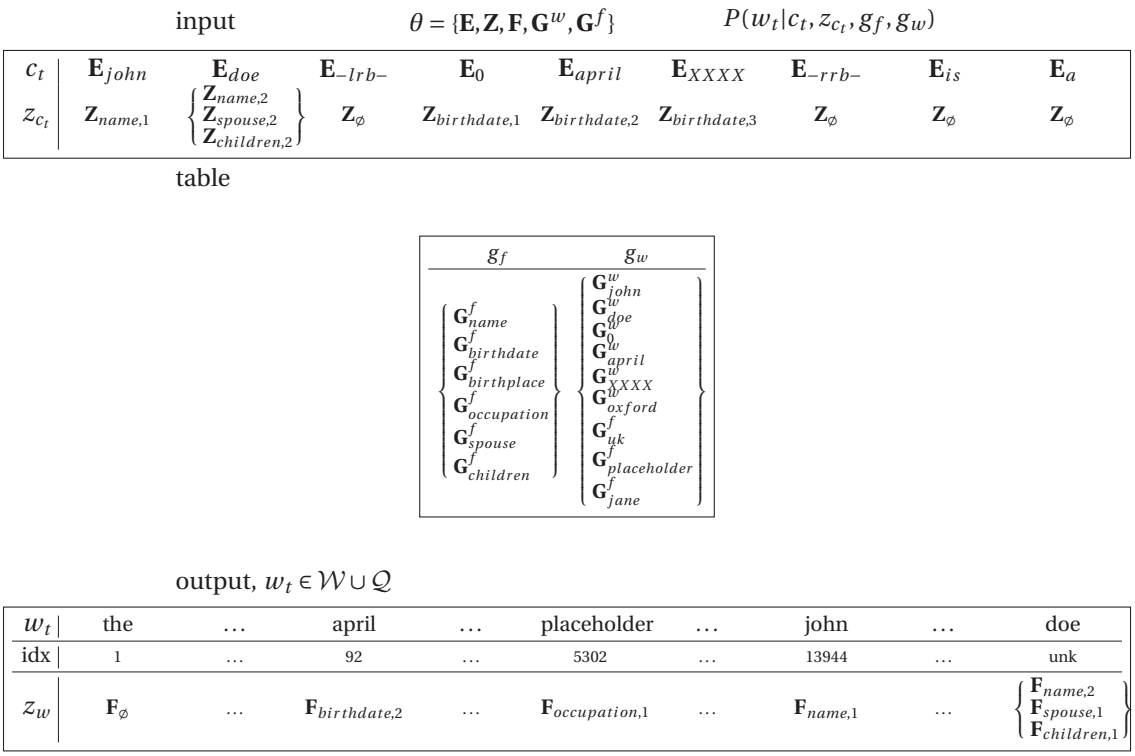


Figure 9.3 – Schematic example of the embedding-based language model. $\{\cdot\}$ symbolizes the max operation over the embeddings.

9.3.2 In-Vocabulary Outputs

The representation of the context $\mathbf{h}_{c_t}$ is then multiplied by a matrix with one row per word, this produces a real value score for each word in the vocabulary,

$$\phi_\theta^{\mathcal{W}}(c_t) = \mathbf{W}^{\text{out}}\mathbf{h}_{c_t} + \mathbf{b}^{\text{out}} \in \mathbb{R}^{|\mathcal{W}|}, \tag{9.16}$$

where $\mathbf{W}^1 \in \mathbb{R}^{n_{\text{hu}} \times d^1}$, $\mathbf{W}^{\text{out}} \in \mathbb{R}^{|\mathcal{W}| \times n_{\text{hu}}}$, $\mathbf{b}^1 \in \mathbb{R}^{n_{\text{hu}}}$, and $\mathbf{b}^{\text{out}} \in \mathbb{R}^{|\mathcal{W}|}$ are learnable weights and biases.

9.3.3 Mixing Outputs for Better Copying

Section 9.2.3 explains that each word w_t is also associated with z_{w_t}, the set of fields in which it occurs, along with the position of w_t in that field. Similarly to local conditioning, we represent each field, position pair (j, i) with an embedding $\mathbf{F}_{j,i}$. These embeddings are then projected into the same space than the latent representation of a context input $\mathbf{h}_{c_t}$. Using the max operation over the embeddings dimension, each word is finally embedded into a unique vector:

$$\mathbf{q}_{w_t} = \max\left\{h\left(\mathbf{W}^2\mathbf{F}_{j,i} + \mathbf{b}^2\right), \forall \mathbf{F}_{j,i} \in z_{w_t}\right\} \tag{9.17}$$

where $\mathbf{W}^2 \in \mathbb{R}^{n_{\text{hu}} \times d}$, and $\mathbf{b}^2 \in \mathbb{R}^{n_{\text{hu}}}$ are learnable weights and biases, and $\mathbf{q}_{w_t} \in \mathbb{R}^{n_{\text{hu}}}$. A dot product with the context vector $\mathbf{h}_{c_t}$ produces a real value score for each word w_t in the table,

$$\phi_\theta^{\mathcal{Q}}(w_t, c_t) = \mathbf{h}_{c_t} \cdot \mathbf{q}_{w_t}. \tag{9.18}$$

Each word $w_t \in \mathcal{W} \cup \mathcal{Q}$ then receive a final score by summing its vocabulary score and its field score:

$$\phi_\theta(w_t, c_t) = \phi_\theta^{\mathcal{W}}(w_t, c_t) + \phi_\theta^{\mathcal{Q}}(w_t, c_t), \tag{9.19}$$

where $\phi_\theta^{\mathcal{Q}}(w_t, c_t) = 0$ when $w_t \notin \mathcal{Q}$.

9.3.4 Training

The neural language model is trained to minimize the negative log-likelihood of a training sentence s with stochastic gradient descent:

$$L_\theta(s) = -\sum_{t=1}^{T} \log P(w_t | c_t, z_{c_t}, g_f, g_w), \tag{9.20}$$

with $\theta = \{\mathbf{E}, \mathbf{Z}, \mathbf{G}^f, \mathbf{G}^w, \mathbf{F}, \mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^2, \mathbf{b}^2, \mathbf{W}^{\text{out}}, \mathbf{b}^{\text{out}}\}$.

9.4 Experiments

Our neural network model (Section 9.3) is designed to generate sentences from tables for large-scale problems, where a diverse set of sentence types need to be generated. Biographies are therefore a good framework to evaluate our model, with Wikipedia offering a large and diverse dataset.

9.4.1 Biography Dataset

The corpus consists of 728,321 biography articles extracted from English Wikipedia (dump of September 2015). These biographies have been detected using "WikiProject Biography" [1]. For each biography article, only the introduction section and the infobox are kept. Introductions are split into sentences and tokenized with the Stanford CoreNLP toolkit (Manning et al., 2014). All numbers are mapped to a special token '0', except for years which are mapped to another special token 'XXXX'. Infobox values have also been tokenized, templates for birth dates and death dates have been formatted in natural language[2]. All tokens in introductions and infoboxes have been lowercased. The final corpus has been divided into three sub-parts to provide training (80%), validation (10%) and test sets (10%). We will release this data with

[1] See https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Biography
[2] See https://en.wikipedia.org/wiki/Template:Birth_date

the camera-ready.

9.4.2 Baseline

Our baseline is an interpolated Kneser-Ney (KN) language model and we use the KenLM toolkit to train 5-gram models without pruning (Heafield et al., 2013). We equip the baseline with *copy actions* of words from tables to sentences by pre-processing words occurring in both as follows: each copied word w is replaced by a special token reflecting its table descriptor z_w (Equation 9.3). When words occur in multiple fields the longest common field sequence is chosen. In case of ties the most frequent field is preferred. The introduction section of the table in Figure 9.1 may look as follows under this scheme: "`name_1 name_2 ( birthdate_1 birthdate_2 birthdate_3 – deathdate_1 deathdate_2 deathdate_3 )` was an english linguist , `fields_3` pathologist , `fields_10` scientist , mathematician , mystic and mycologist ."

At decoding time, we copy words from the tables when those special tokens are emitted. This baseline can be seen as a template model, and it is called *Template KN* in the next sections.

9.4.3 Training Setup

For our neural models, we train 11-gram language models ($n = 11$) with the following hyper-parameters:

- number of word types: $|\mathcal{W}| = 20000$,
- number of field types: $|\mathcal{F}| = 1740$,
- maximum number of occurrences of a word in a table: $m = 10$,
- maximum length of a word sequence in a field, $l = 10$,
- word embeddings size: $d^{\text{wrd}} = 64$,
- field embeddings size: $d = 64$,
- global embedding size: $g = 128$,
- number of hidden units: $n_{\text{hu}} = 256$,

All fields that occur at least 100 times in the training data are included in the set of fields $\mathcal{F}$. We include the 20,000 most frequent words in the vocabulary. The other hyper-parameters are set through validation, maximizing BLEU over a validation subset of 1,000 sentences. Similarly, early stopping is applied: training ends when BLEU stops improving on the same validation subset. One should note that the maximum number of tokens in a field $l = 10$ means that we encode only 10 positions: for longer field values the final tokens are not dropped but their position is capped to 10.

MODEL	PERPLEXITY		BLEU	
	valid	*test*	*valid*	*test*
KN	10.54	10.51	2.2	2.2
NNLM	9.43 ± 0.01	9.40 ± 0.01	2.6 ± 0.4	2.4 ± 0.3
+ LOCAL (FIELD, START, END)	8.63 ± 0.01	8.61 ± 0.01	4.4 ± 0.2	4.2 ± 0.5
TEMPLATE KN	$7.48^{\star}$	$7.46^{\star}$	19.7	19.8
TABLE NNLM W/ LOCAL (FIELD, START)	$4.59 \pm 0.01^{\dagger}$	$4.60 \pm 0.01^{\dagger}$	26.0 ± 0.4	26.0 ± 0.4
+ LOCAL (FIELD, START, END)	$4.60 \pm 0.01^{\dagger}$	$4.60 \pm 0.01^{\dagger}$	26.7 ± 0.4	26.6 ± 0.4
+ GLOBAL (FIELD)	$4.30 \pm 0.01^{\dagger}$	$4.30 \pm 0.01^{\dagger}$	33.4 ± 0.2	33.4 ± 0.2
+ GLOBAL (FIELD & WORD)	$4.40 \pm 0.02^{\dagger}$	$4.40 \pm 0.02^{\dagger}$	34.7 ± 0.3	34.7 ± 0.4

Table 9.1 – Valid and test perplexity for all models. Valid and test BLEU for table-conditioned models. For neural network language models (NNLM) we report the mean with standard deviation of five training runs with different initialization. BLEU scores are computed over sentences generated with a beam search (beam size is 5). $\star$ and † are not directly comparable as the output vocabulary is slightly different.

9.4.4 Evaluation Metrics

We use two different metrics to evaluate our models. Performance is first evaluated in terms of perplexity, as this is the standard metric for language modeling (see Section 2.3.5). We report perplexity results on a per sentence basis. For models with copy actions, we measure the quality of the generated sentences using the BLEU score (see Section 2.3.5).

9.5 Results

This section describes our results and discusses the impact of the different conditioning variables.

9.5.1 The More, The Better

The results (Table 9.1) show that more conditioning information helps to improve the performance of our models.

Without copy actions.

In terms of perplexity the (i) neural network language model (NNLM) is slightly better than an interpolated KN language model, and (ii) adding local conditioning on the field start and end position further improves performance. BLEU over the model generations is generally very low but there is a clear improvement when using local conditioning because it allows learning transitions between fields and links past model predictions to the table unlike KN or plain

NNLM.

With copy actions.

For experiments with copy actions we use the full local conditioning (Equation 9.4) in the neural language models. Perplexity can only be compared between variants of Table NNLM models as described above and improvements are less clear when adding features in terms of this measure. However, BLEU clearly improves when moving from Template KN to Table NNLM and more features successively improve accuracy. Global conditioning on the fields improves the model by over 7 BLEU and adding words gives 1.3 points. This is a total improvement of nearly 15 BLEU over the template KN baseline.

9.5.2 Attention Mechanism

Our model implements attention over input table fields. For each word w_t in the table, Equation (9.19) takes the language model score $\phi_\theta^{\mathcal{W}}(w_t, c_t)$ and adds a bias $\phi_\theta^{\mathcal{Q}}(w_t, c_t)$. The bias is the dot-product between a representation of the table field in which w_t occurs and a representation of the context, Equation (9.18) that summarizes the previously generated fields and words.

Figure 9.2 shows that this mechanism adds a large bias to continue a field if it has not generated all tokens from the table, e.g., it emits the word occurring in `name_2` after generating `name_1`. It also nicely handles transitions between field types, e.g., the model adds a large bias to the words occurring in the occupation field after emitting the birth date.

9.5.3 Sentence Decoding

We use a standard beam search to explore a larger set of sentences compared to simple greedy search. This allows us to explore K times more paths which comes at a linear increase in the number of forward computation steps for our language model. We compare various beam settings for the baseline Template KN and our Table NNLM (Figure 9.4). The best validation BLEU can be obtained with a beam size of $K = 5$. Our model is also several times faster than the baseline, requiring only about 200 ms per sentence with $K = 5$. Beam search generates many n-gram lookups for Kneser-Ney which requires many random memory accesses; while neural models perform scoring through matrix-matrix products, an operation which is more local and can be performed in a block parallel manner where modern graphic processors shine (Kindratenko, 2014).

9.5.4 Qualitative Analysis

Table 9.3 shows generations for different conditioning information from the Wikipedia table shown in Figure 9.1. First of all, comparing the reference to the fact table reveals that our train-

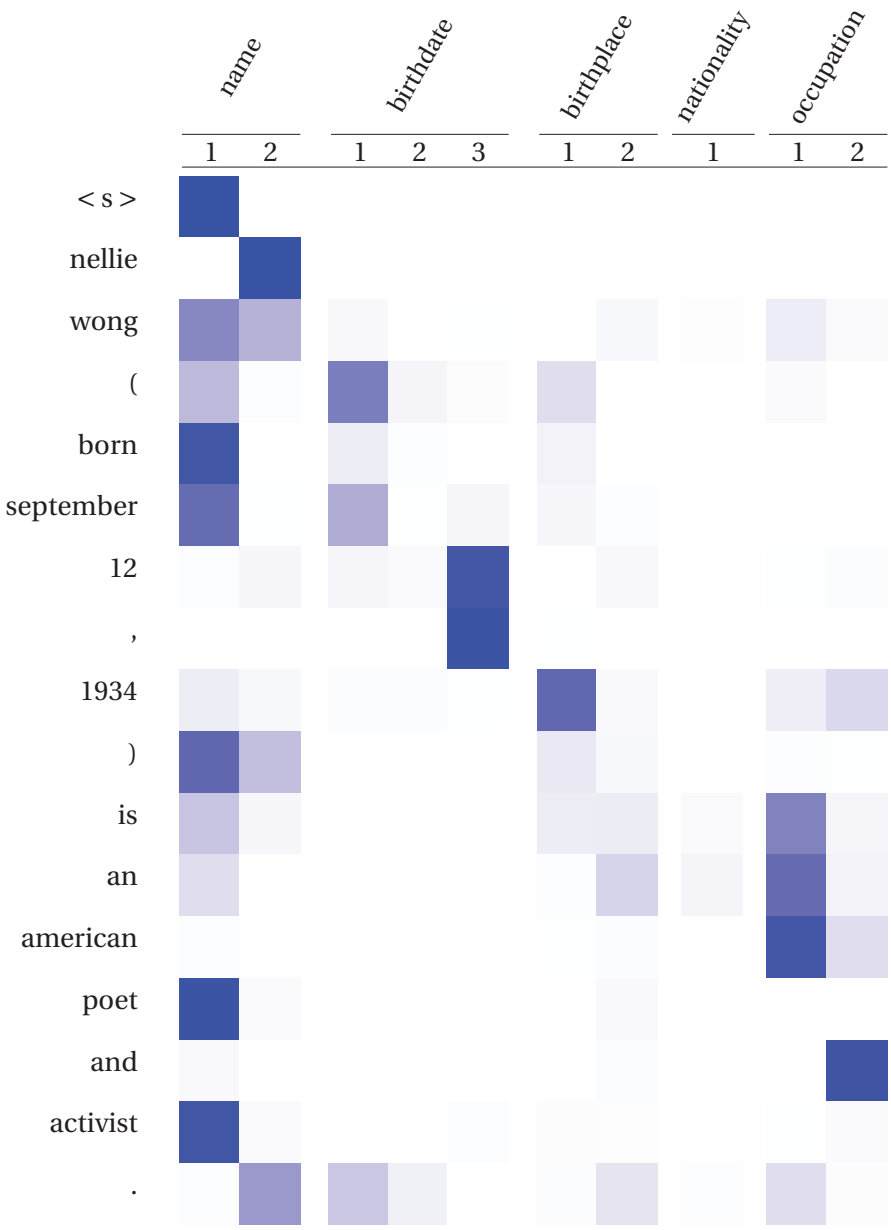


Table 9.2 – Visualization of attention scores for Nellie Wong's Wikipedia infobox. Rows represent the probability distribution over (field, position) pairs from the table after generating each word. The columns represent the conditioning context, e.g., the model takes $n-1$ words as context. The darker the color, the higher the probability. Best viewed in colors.

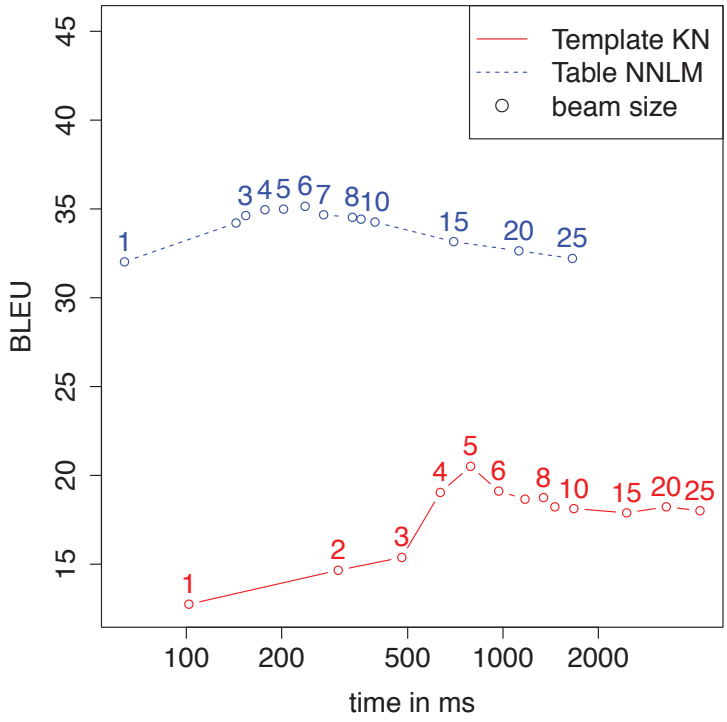


Figure 9.4 – Comparison between our best model (Table NNLM) and the baseline (Template KN) for different beam sizes. The x-axis is the average timing (in milliseconds) for generating one sentence. The y-axis is the BLEU score. All results are measured on a subset of 1,000 samples of the validation set.

ing data is not perfect. The month of birth mentioned in the fact table and the first sentence of the Wikipedia article are different; this may have been introduced by one contributor editing the article and not keeping the information consistent.

All three versions of our model correctly generate the beginning of the sentence by copying the name, the birth date and the death date from the table. Knowing that the person has died, the past tense is used. Frederick Parker-Rhodes was a scientist, but this occupation is not directly mentioned in the table. The model without global conditioning can therefore not predict the right occupation, and it continues the generation with the most common occupation (in Wikipedia) for a person who has died. In contrast, the global conditioning over the fields helps the model to understand that this person was indeed a scientist. However, it is only with the global conditioning on the words that the model can infer the correct occupation, i.e., *computer scientist*.

MODEL	GENERATED SENTENCE
REFERENCE	Frederick Parker-Rhodes (21 March 1914 – 21 November 1987) was an English linguist, plant pathologist, computer scientist, mathematician, mystic, and mycologist.
TABLE NNLM W/ LOCAL	frederick parker-rhodes (21 november 1914 – 2 march 1987) was an australian rules footballer who played with carlton in the victorian football league (vfl) during the XXXXs and XXXXs .
+ GLOBAL (FIELD)	frederick parker-rhodes (21 november 1914 – 2 march 1987) was an english mycology and plant pathology , mathematics at the university of uk .
+ GLOBAL (FIELD & WORD)	frederick parker-rhodes (21 november 1914 – 2 march 1987) was a british computer scientist , best known for his contributions to computational linguistics .

Table 9.3 – First sentence from the current Wikipedia article about Frederick Parker-Rhodes and the sentences generated from the three versions of our table-conditioned neural language model (Table NNLM) using the Wikipedia infobox seen in Figure 9.1.

9.6 Conclusion

We have shown that our embeddings-based model can generate fluent descriptions of arbitrary people based on structured data. Local and global conditioning improves our model by a large margin and we outperform a Kneser-Ney language model by nearly 15 BLEU. Our task uses an order of magnitude more data than previous work and has a vocabulary that is three orders of magnitude larger.

In this chapter, we have only focused on generating the first sentence, but this model can be extended for the generation of longer biographies. Furthermore, the current training loss function does not explicitly penalize the model from generating incorrect facts, e.g. predicting the wrong nationality or a wrong occupation is not currently considered worse than choosing the wrong determiner. A loss function that could assess factual accuracy would certainly improve sentence generation by avoiding such mistakes.

10 Conclusion

Natural language is complex and subtle, which makes its understanding an AI-hard problem. For solving NLP tasks, computers need to become as intelligent as people and thus, apprehend the underlying mechanisms that determine human language. This thesis does not have the ambition to solve NLP but aims at bringing efficient and effective solutions for computers to understand the basics of human language: words and phrases. Just as humans know that *cat* and *dog* are close semantically, we propose an approach that allows computers to understand that too, by capturing syntactic and semantic properties of words into vector space models (aka *word embeddings*). Because a *red persian cat* is also close to *cat*, we then propose a model that enables phrases with different length to be encoded into the same semantic space. The model is trained in such a manner that word vector representations can be aggregated together while keeping the maximum information from the original vectors so that computers know that a *red persian cat* is a composition of three features. Finally, we show that "intelligent" systems can be built upon the understanding of these basics for classifying text documents or generating natural language.

10.1 Achievements

This thesis proposes techniques for:

- **Building word embeddings from large text corpora.** In Chapter 4, we demonstrate that appealing word embeddings can be obtained by computing a *Hellinger PCA* of the word co-occurrence matrix built from the English Wikipedia. While PCA can be computationally expensive with huge matrices, our findings reveal that having a significant, but not too large set of context words, is sufficient for capturing most of the syntactic and semantic properties of words. With a limited number of columns and a randomized SVD, PCA is fast to compute and becomes a good alternative to NNLM.

  In Chapter 5, we show that Hellinger PCA can be performed with an autoencoder network, which enables to construct a joint learning architecture. Given an input phrase

composed of n words, the network learns embeddings for the n words, while keeping the maximum information of each individual word when they are summed together. Considering that the word vocabulary grows exponentially with n, this approach gives a nice framework to produce the huge variety of possible sequences of n words in a timely and efficient manner with low memory consumption. Relying on word co-occurrence statistics to represent words in vector space also provides a framework to easily generate representations for unseen words or phrases.

Regarding technology transfer, we have released a complete toolkit in C++ for computing word embeddings from a large corpus of text with Hellinger PCA[1]. An open source software for the autoencoder approach is also available on github.com[2]. Pre-trained word embeddings and the phrase dataset from Chapter 5 are both available online[3].

- **Document classification.** As we have built embeddings carrying meaningful semantic information about words, we propose to tackle sentiment classification with a convolutional neural network in Chapter 6. The objective of sentiment classification is to classify a text document according to the sentimental polarities of opinions it contains (e.g. positive or negative). Traditional BOW models usually need bigram features not to misclassify documents containing bigrams such as *not good* or *not funny*, leading to high-dimensional yet sparse document representations. We show that the proposed model is a good alternative for fighting the curse of dimensionality, while offering good performance and an interesting framework for sentiment visualization.

  In Chapter 7, we leverage our model described in Chapter 5 to propose an unsupervised method for producing compact representations of text documents, while adding more semantic information. As word embeddings can be summed together, we group n-grams with different lengths n into semantic concepts. Documents are then represented as bags of semantic concepts. Such model has several advantages over classical approaches for representing documents in a low-dimensional space: it leverages semantic information coming from n-grams; it builds document representations with low resource consumption (time and memory); it can infer semantic concepts for unseen n-grams; and finally, it is capable of providing relevant document representations even with a small set of documents. We show that such model is suitable for the classification of text news and movie reviews, reaching similar performance than traditional BOW models with considerably fewer features.

- **Sentence generation.** Chapter 8 proposes a model for generating image descriptions, leveraging our pre-trained word embeddings from Chapter 5 and image representations from pre-trained CNN. Exploiting such rich representations enables the design of an effective model that achieves good performance without the use of complex recurrent networks. In this chapter, we also show that the multimodal learning leads to improve the quality of phrase vector representations, by incorporating visual features.

[1] Available at http://github.com/rlebret/hpca.
[2] Available at http://github.com/ParallelDots/WordEmbeddingAutoencoder.
[3] Available at http://www.lebret.ch/words.

Chapter 9 addresses the problem of concept-to-text generation, rendering Wikipedia infoboxes into natural language. To tackle this problem we propose to parameterize words and fields from infoboxes as embeddings, along with a neural network language model operating on these embeddings. We show that our embeddings-based model can generate fluent descriptions of arbitrary people based on structured data, outperforming a standard template-based model.

Regarding technology transfer, we have released the code of our phrase-based model for image captioning[4]. All biography articles and infoboxes from Chapter 9 will also be released online soon.

10.2 Perspectives for Future Work

As mentioned in the introduction, natural language is based on three mechanisms: compositionality, hierarchy and recursion. Even if not optimal, this thesis provides a solution for the first mechanism. The two other mechanisms are however not addressed in this thesis. Perspectives for future work are thus the following:

- **Considering word order for compositionality.** Our approach for compositionality makes sense and works well when the meaning of a phrase is literally "the sum of its parts", where each word independently has its proper meaning. However, Landauer (2002) estimates that 80% of the meaning of English text comes from word choice and the remaining 20% comes from word order. It, therefore, seems important to preserve word order, but commutative or associative operations ignore word order and thus the syntactic structure of expressions. To overcome this shortcoming, connectionist approaches have been proposed to develop distributed representations which encode the structural relationships between words (Hinton, 1986; Pollack, 1990; Elman, 1991). Future work is to investigate these recurrent architectures, which could convert sequences of words of variable sizes into a same semantic space while retaining the word order.

- **Hierarchical structure for higher-level phrase embeddings.** Knowing the hierarchical structure, those approaches based on recurrent neural network architectures can produce representations of syntactic sentence structures in an efficient way (Kwasny and Kalman, 1995). When operating over a parse tree, good performance can be achieved on tasks such as sentiment detection or semantic relationships classification (Socher et al., 2012). Future work would be to learn higher-level phrase embeddings by combining both compositionality and hierarchical structure in an unsupervised manner.

- **Recursion for sentence embeddings.** With recurrent architectures, the recursion theory could then be addressed by compressing recursively higher-level phrase embeddings within another higher-level phrase embeddings to compute sentence embeddings.

[4] Available at http://github.com/rlebret/phrase-based_image_captioning.

Sentence embeddings would be helpful for document classification and natural language generation. For instance, we could define the semantic concepts from Chapter 7 by clustering sentences instead of *n*-grams. After the generation of the first sentence in Chapter 9, we could use the embedding of this sentence as conditioning for generating the next one, and thus produce a short paragraph summary for a given Wikipedia infobox.

References

R. K. Ando, T. Zhang, and P. Bartlett. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*.

G. Angeli, P. Liang, and D. Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512. Association for Computational Linguistics.

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*.

M. Baroni and R. Zamparelli. 2010. Nouns are Vectors, Adjectives are Matrices: Representing Adjective-Noun Constructions in Semantic Space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1193. Association for Computational Linguistics.

R. Barzilay and M. Lapata. 2005. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 331–338. Association for Computational Linguistics.

R. Barzilay and M. Lapata. 2006. Aggregation via set partitioning for natural language generation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 359–366. Association for Computational Linguistics.

A. Belz. 2008. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(04):431–455.

Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Y. Bengio, R. Ducharme, and P. Vincent. 2001. A Neural Probabilistic Language Model. In *NIPS Workshop*.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.

A. Bhattacharyya. 1943. On a Measure of Divergence Between Two Statistical Populations Defined by Probability Distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–110.

W. Blacoe and M. Lapata. 2012. A Comparison of Vector-based Representations for Semantic Composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 546–556. Association for Computational Linguistics.

D. M. Blei, A. Y. Ng, and M. I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*.

H. Bourlard and Y. Kamp. 1988. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics*, 59(4):291–294.

J. S. Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*.

J. A. Bullinaria and J. P. Levy. 2007. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526.

J. Caron. 2001. Computational information retrieval. chapter Experiments with LSA Scoring: Optimal Rank and Basis, pages 157–169. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

S. F. Chen and J. Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics.

X. Chen and C. L. Zitnick. 2015. Mind's Eye: A Recurrent Visual Representation for Image Caption Generation. In *IEEE International Concference on Computer Vision and Patter Recognition (CVPR)*.

Y. Chen, B. Perozzi, R. Al-Rfou', and S. Skiena. 2013. The Expressive Power of Word Embeddings. *CoRR*, abs/1301.3226.

N. Chomsky. 1957. *Syntactic Structures*. Mouton.

S. Clark, B. Coecke, and M. Sadrzadeh. 2008. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*, pages 133–140.

R. Collobert and J. Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *ICML*.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*.

R. Collobert. 2004. *Large Scale Machine Learning*. Ph.D. thesis, Université Paris VI.

C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

R. Dale, S. Geldof, and J.-P. Prost. 2003. Coral: Using natural language generation for navigational assistance. In *Proceedings of the 26th Australasian computer science conference-Volume 16*, pages 35–44. Australian Computer Society, Inc.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1370–1380.

J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. 2014. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. *arXiv preprint arXiv:1411.4389*.

P. A. Duboue and K. R. McKeown. 2002. Content planner construction via evolutionary algorithms and a corpus-based fitness function. In *Proceedings of INLG 2002*, pages 89–96.

J. L. Elman. 1991. Distributed Representations, Simple Recurrent Networks, and Grammatical Structure. *Machine Learning*.

R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*.

H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollar, J. Gao, X. He, M. Mitchell, J. C. Platt, L. C. Zitnick, and G. Zweig. 2015. From Captions to Visual Concepts and Back. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. 2002. Placing Search in Context: The Concept Revisited. *ACM Transactions on Information Systems*.

J. R. Firth. 1957. A Synopsis of Linguistic Theory 1930-55.

G. Frege. 1892. über sinn und bedeutung. In Mark Textor, editor, *Funktion - Begriff - Bedeutung*, Sammlung Philosophie. Vandenhoeck & Ruprecht, Göttingen.

D. Galanis and I. Androutsopoulos. 2007. Generating multilingual descriptions from linguistically annotated owl ontologies: the naturalowl system. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 143–146. Association for Computational Linguistics.

K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. 2011. Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proceedings of the 49nd Annual Meeting of the Association for Computational Linguistics*.

A. Go, R. Bhayani, and L. Huang. 2009. Twitter Sentiment Classification using Distant Supervision. *CS224N Project Report*.

Irving J Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264.

A. Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

N. Green. 2006. Generation of biomedical arguments for lay readers. In *Proceedings of the Fourth International Natural Language Generation Conference*, pages 114–121. Association for Computational Linguistics.

E. Grefenstette, G. Dinu, Y. Zhang, M. Sadrzadeh, and M. Baroni, 2013. *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013) – Long Papers*, chapter Multi-Step Regression Learning for Compositional Distributional Semantics, pages 131–142. Association for Computational Linguistics.

M. U. Gutmann and A. Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361.

B. Gyawali and C. Gardent. 2014. Surface Realisation from Knowledge-Bases. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 424–434.

N. Halko, P.-G. Martinsson, and J. A. Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Z. Harris. 1954. Distributional Structure. 10.

M. D. Hauser, N. Chomsky, and W. T. Fitch. 2002. The Faculty of Language: What Is It, Who Has It, and How Did It Evolve? *Science*.

K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn. 2013. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria, August.

E. Hellinger. 1909. Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271.

G. E. Hinton. 1986. Learning Distributed Representations of Concepts. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pages 1–12. Hillsdale, NJ: Erlbaum.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

M. Hodosh, P. Young, and J. Hockenmaier. 2013. Framing image description as a ranking task: data, models and evaluation metrics. *Journal of Artificial Intelligence Research*.

X. Hu, X. Zhang, C. Lu, E. K. Park, and X. Zhou. 2009. Exploiting Wikipedia As External Knowledge for Document Clustering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 389–396, New York, NY, USA. ACM.

F. Huang and A. Yates. 2009. Distributional Representations for Handling Sparsity in Supervised Sequence-Labeling. In *ACL*.

W. Humboldt. 1836. *Über die Verschiedenheit des menschlichen Sprachbaues: Und ihren Einfluss auf die geistige Entwickelung des Menschengeschlechts*. Druckerei der Königlichen Akademie der Wissenschaften.

T. Joachims. 1998. Text categorization with Support Vector Machines: Learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Computer Science*, pages 137–142. Springer Berlin Heidelberg.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 655–665.

A. Karpathy and L. Fei-Fei. 2015. Deep Visual-Semantic Alignments for Generating Image Descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

J. Kim and R. J. Mooney. 2010. Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 543–551. Association for Computational Linguistics.

V. Kindratenko. 2014. *Numerical Computations with GPUs*. Springer.

R. Kiros, R. Salakhutdinov, and R. S. Zemel. 2014. Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*.

I. Konstas and M. Lapata. 2013. A global model for concept-to-text generation. *Journal of Artificial Intilligence Research*, 48(1):305–346, October.

T. Koo, X. Carreras, and M. Collins. 2008. Simple semi-supervised dependency parsing. In *ACL*.

G. Kulkarni, V. Premraj, S. Dhar, Siming Li, Yejin Choi, A. C. Berg, and T. L. Berg. 2013. Baby Talk: Understanding and Generating Simple Image Descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2891–2903.

P. Kuznetsova, V. Ordonez, A. C. Berg, T. L. Berg, and Y. Choi. 2012. Collective Generation of Natural Image Descriptions. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 359–368. Association for Computational Linguistics, July.

S. C. Kwasny and B. L. Kalman. 1995. Tail-Recursive Distributed Representations and Simple Recurrent Networks. *Connection Science*, 7:61–80.

T. K. Landauer and S. T. Dumais. 1997. A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*.

T. K. Landauer. 2002. On the computational basis of learning and cognition: Arguments from LSA. In *The psychology of learning and motivation*. Academic Press.

P.-S. Laplace. 1820. *Théorie analytique des probabilités*. V. Courcier.

R. Lebret and R. Collobert. 2013. Is Deep Learning Really Necessary for Word Embeddings? In *NIPS Deep Learning Workshop*.

R. Lebret and R. Collobert. 2014. Word Embeddings through Hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 482–490. Association for Computational Linguistics.

R. Lebret and R. Collobert. 2015a. N-gram-Based Low-Dimensional Representation for Document Classification. In *3rd International Conference on Learning Representations (ICLR) – Workshop Track*.

R. Lebret and R. Collobert. 2015b. Rehabilitation of Count-Based Models for Word Vector Representations. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 9041 of *Lecture Notes in Computer Science*, pages 417–429. Springer International Publishing.

R. Lebret and R. Collobert. 2015c. "The Sum of Its Parts": Joint Learning of Word and Phrase Representations with Autoencoders. In *ICML Deep Learning Workshop*.

R. Lebret, P. O. Pinheiro, and R. Collobert. 2015a. Phrase-Based Image Captioning. In *Proceedings of the 32st International Conference on Machine Learning (ICML)*.

R. Lebret, P. O. Pinheiro, and R. Collobert. 2015b. Simple Image Description Generator via a Linear Phrase-Based Model. In *3rd International Conference on Learning Representations (ICLR) – Workshop Track*.

R. Lebret, D. Grangier, and M. Auli. 2016a. Neural Text Generation from Structured Data with Application to the Biography Domain . In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

R. Lebret, P. O. Pinheiro, and R. Collobert. 2016b. Twitter Sentiment Analysis (Almost) from Scratch. Idiap-RR Idiap-RR-15-2016, Idiap, 5.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

J. Legrand and R. Collobert. 2014. Joint RNN-based Greedy Parsing and Word Composition. *arXiv preprint arXiv:1412.7028*.

O. Levy and Y. Goldberg. 2014. Linguistic Regularities in Sparse and Explicit Word Representations. In *CoNLL*, pages 171–180.

P. Liang, M. I. Jordan, and D. Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 91–99. Association for Computational Linguistics.

P. Liang. 2005. *Semi-supervised learning for natural language*. Ph.D. thesis, Citeseer.

D. Lin and X. Wu. 2009. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1030–1038. Association for Computational Linguistics.

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision–ECCV 2014*, pages 740–755. Springer.

W. Lu and H. T. Ng. 2011. A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1611–1622. Association for Computational Linguistics.

M. Luong, R. Socher, and C. D. Manning. 2013. Better Word Representations with Recursive Neural Networks for Morphology. In *CoNLL*, pages 104–113. Citeseer.

M.-T. Luong, I. Sutskever, Q. V Le, O. Vinyals, and W. Zaremba. 2015. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics*, pages 11–19.

A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. 2011. 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.

C. D. Manning and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. L. Yuille. 2015. Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN). In *3rd International Conference on Learning Representations (ICLR)*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Flexible text segmentation with structured multilabel classification. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 987–994. Association for Computational Linguistics.

H. Mei, M. Bansal, and M. R. Walter. 2016. What to talk about and how? Selective Generation using LSTMs with Coarse-to-Fine Alignment. In *Proceedings of Human Language Technologies: The 2016 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *1st International Conference on Learning Representations (ICLR) – Workshop Track*.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

J. Mitchell and M. Lapata. 2010. Composition in Distributional Models of Semantics. *Cognitive Science*.

M. Mitchell, X. Han, J. Dodge, A. Mensch, A. Goyal, A. Berg, K. Yamaguchi, T. Berg, K. Stratos, and H. Daumé, III. 2012. Midge: Generating Image Descriptions from Computer Vision Detections. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 747–756. Association for Computational Linguistics.

A. Mnih and G. Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, pages 641–648.

A. Mnih and G. Hinton. 2009. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems*, pages 1081–1088.

A. Mnih and K. Kavukcuoglu. 2013. Learning Word Embeddings Efficiently with Noise-Contrastive Estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273.

A. Mnih and Y. W. Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 1751–1758, New York, NY, USA, July. Omnipress.

Richard Montague. 1974. English as a Formal Language. In Richmond H. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*, pages 188–222. Yale University Press, New Haven, London.

F. Morin and Y. Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model. In *Aistats*, volume 5, pages 246–252. Citeseer.

M. Mourino-García, R. Pérez-Rodríguez, and L. Anido-Rifón. 2015. Bag-of-Concepts Document Representation for Textual News Classification. In *the 16th International Conference on Intelligent Text Processing and Computational Linguistics*.

K. Nigam. 1999. Using maximum entropy for text classification. In *In IJCAI-99 Workshop on Machine Learning for Information Filtering*, pages 61–67.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

J.-F. Pessiot, Y.-M. Kim, M.-R. Amini, and P. Gallinari. 2010. Improving Document Clustering in a Learned Concept Space. *Information Processing & Management*, 46(2):180–192.

J. B. Pollack. 1990. Recursive Distributed Representations. *Artificial Intelligence*.

S. Poria, A. F. Gelbukh, E. Cambria, A. Hussain, and G.-B. Huang. 2014. EmoSenticSpace: A novel framework for affective common-sense reasoning. *Knowledge-Based Systems*.

L. R. Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286.

L. Ratinov and D. Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics.

A. Ratnaparkhi. 2002. Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech & Language*, 16(3):435–455.

E. Reiter, R. Dale, and Z. Feng. 2000. *Building natural language generation systems*, volume 33. MIT Press.

E. Reiter, S. Sripada, J. Hunter, J. Yu, and I. Davy. 2005. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167(1):137–169.

H. Schütze. 1993. Word Space. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 895–902. Morgan-Kaufmann.

H. Schütze. 1995. Distributional part-of-speech tagging. In *Proceedings of the 7th conference on European Chapter of the Association for Computational Linguistics*, pages 141–148. Morgan Kaufmann Publishers Inc.

J. Sedding and D. Kazakov. 2004. WordNet-based Text Document Clustering. In *Proceedings of the 3rd Workshop on RObust Methods in Analysis of Natural Language Data*, pages 104–113, Stroudsburg, PA, USA. Association for Computational Linguistics.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics.

L. Shang, Z. Lu, and H. Li. 2015. Neural Responding Machine for Short-Text Conversation. *arXiv preprint arXiv:1503.02364*.

K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR*.

R. Socher, B. Huval, C. Manning, and A. Ng. 2012. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211. Association for Computational Linguistics.

R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y Ng, and C. Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1631, page 1642. Association for Computational Linguistics.

R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng. 2014. Grounded Compositional Semantics for Finding and Describing Images with Sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218.

N. Srivastava and R. Salakhutdinov. 2014. Multimodal Learning with Deep Boltzmann Machines. *Journal of Machine Learning Research*.

S. Staab and A. Hotho. 2003. Ontology-based Text Document Clustering. In *Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM'03 Conference held in Zakopane*, pages 451–452.

X. Sun, L.-P. Morency, D. Okanohara, and J. Tsujii. 2008. Modeling latent-dynamic in shallow parsing: a latent conditional model with improved inference. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 841–848. Association for Computational Linguistics.

C. Tan, Y. Wang, and C. Lee. 2002. The Use of Bigrams to Enhance Text Categorization. *Journal of Information Processing and Management*.

J. Turian, L. Ratinov, and Y. Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

R. Turner, S. Sripada, and E. Reiter. 2010. Generating approximate geographic descriptions. In *Empirical methods in natural language generation*, pages 121–140. Springer.

P. Turney and P. Pantel. 2010. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*.

S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. J. Mooney, and K. Saenko. 2014. Translating Videos to Natural Language Using Deep Recurrent Neural Networks. *arXiv preprint arXiv:1412.4729*.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. 2015. Show and Tell: A Neural Image Caption Generator. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*.

S. I. Wang and C. D. Manning. 2012. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics.

T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young. 2015. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. *arXiv preprint arXiv:1508.01745*.

Y. W. Wong and R. J. Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *HLT-NAACL*, pages 172–179.

K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of The 32nd International Conference on Machine Learning*, volume 37, July.

Y. Yang and X. Liu. 1999. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49. ACM.

B. Z. Yao, X. Yang, L. Lin, M. W. Lee, and S. C. Zhu. 2010. I2T: Image Parsing to Text Description. *Proceedings of the IEEE*, 98(8):1485–1508.

K. Yao, G. Zweig, and B. Peng. 2015. Attention with Intention for a Neural Network Conversation Model. *arXiv preprint arXiv:1510.08565*.

K. Zhao, H. Hassan, and M. Auli. 2015. Learning Translation Models from Monolingual Continuous Representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1527–1536, Denver, Colorado, May–June. Association for Computational Linguistics.

Rémi Lebret

Ph.D.
Machine Learning / Natural Language Processing
`http://lebret.ch`

Rue du Village, 14
1923 Le Trétien, SUISSE
+41 (0) 76 592 26 12
remi@lebret.ch
DOB: 26th December 1985
French nationality

Professional Experience

May 2012 – June 2016 **Research Assistant**, *Idiap Research Institute*, Martigny, Valais, Switzerland.
Working on deep learning approaches for Natural Language Processing (NLP) under the supervision of Ronan Collobert.

September 2015 – March 2016 **Research Intern**, *Facebook Artificial Intelligence Research*, Menlo Park, California, USA.
Working on concept-to-text generation under the supervision of David Grangier and Michael Auli.

September 2011 – May 2012 **CNRS Research Engineer**, *Heudiasyc laboratory of CNRS at University of Technology in Compiègne. Paul Painlevé Laboratory, within the Probability and Statistics team, University Lille 1. MODAL Team, INRIA Lille - Nord Europe*, Lille, France.
Developing softwares in R and C++ for supervised and unsupervised classifications under the supervision of Yves Grandvalet and Christophe Biernacki.

March 2011 – August 2011 **CNRS Studies Engineer in Biostatistics**, *Institute Pasteur of Lille, Genomics and metabolic diseases*, Lille, France.
Supporting the researchers in the design of statistical genetic analyses. Creating new tools for high-speed analyses.

January 2009 – March 2011 **Statistical Programmer/Analyst**, *PharmacoGenomic Innovative Solutions (PGXIS). Paul Painlevé Laboratory, within the Probability and Statistics team, University Lille 1*, High Wycombe, UK.
Designing a software for the multivariate analysis of large-scale datasets in genetic.

Education

July 2016 **École Polytechnique Fédéral de Lausanne (EPFL)**, Switzerland.
Ph.D. in Electrical Engineering.

September 2009 **École Polytechnique Universitaire de Lille (Polytech'Lille)**, France.
"Diplôme d'ingénieur" (Master's degree in Engineering) in Software Engineering and Statistics

Fall 2008 **Oklahoma State University**, USA.
Exchange student for a semester in the department of Statistics

Computer Skills

Programming languages

C++/C	Software Design	Torch	Deep Learning Framework
R	Creation of packages	Lua/Perl	Script for data management
SQL	Data management with mySQL	PHP/HTML	Website design
Parallel	POSIX Threads, Open MPI	Other	LaTeX. Multi-platform system administration under Linux, Mac OS X, Unix, Windows.

Languages spoken

French (native), English (fluent, TOEIC: 895), German (basic knowledge)

Publications

Journal

R. Lebret, S. Iovleff, F. Langrognet, C. Biernacki, G. Celeux, and G. Govaert. Rmixmod: The R Package of the Model-Based Unsupervised, Supervised, and Semi-Supervised Classification Mixmod Library. *Journal of Statistical Software*, 67(1):1–29, 2015.

Conference

Lebret R., Grangier D., and Auli M. Neural Text Generation from Structured Data with Application to the Biography Domain . In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Austin, USA, November 2016.

Lebret R., Pinheiro P. O., and Collobert R. Phrase-based Image Captioning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, pages 2085–2094, Lille, France, July 2015.

Lebret R. and Collobert R. Rehabilitation of Count-based Models for Word Vector Representations. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing (CICLing): Proceedings of the 16th International Conference*, volume 9041 of *Lecture Notes in Computer Science*, pages 417–429, Cairo, Egypt, April 2015. Springer.

Lebret R. and Collobert R. Word Embeddings through Hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 482–490, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

Workshop

Lebret R. and Collobert R. "The Sum of Its Parts": Joint Learning of Word and Phrase Representations with Autoencoders. In ICML Deep Learning Workshop, July 2015.

Lebret R., Pinheiro P. O., and Collobert R. Simple Image Description Generator via a Linear Phrase-Based Model. In ICLR Workshop, May 2015.

Lebret R. and Collobert R. N-gram-Based Low-Dimensional Representation for Document Classification. In ICLR Workshop, May 2015.

Lebret R., Legrand J., and Collobert R. Is Deep Learning Really Necessary for Word Embeddings? In NIPS Workshop on Deep Learning, December 2013.

Lebret R., Iovleff S., and Langrogner F. Rmixmod: A MIXture MODelling R Package. Statlearn'12 - Workshop on Challenging problems in Statistical Learning, Lille, France, avril 2012.

Lebret R., Biernacki C., Iovleff S., Jacques J., Preda C., McCarthy A., and Delrieu O. Genetic epistasis analysis using 'taxonomy3'. 2nd International BIO-SI Workshop, Rennes, France, october 2011.

Lebret R., Iovleff S., Biernacki C., Jacques J., Preda C., McCarthy A., and Delrieu O. Rapid multivariate analysis of 269 hapmap subjects and 1 million snps using 'taxonomy3'. Cold Spring Harbor/Wellcome Trust meeting on Pharmacogenomics, Hinxton, UK, september 2009.

Softwares and Packages

Lebret R. *HPCA*, 2015. C/C++ implementation of the Hellinger PCA for computing word embeddings.

Lebret R., Iovleff S., and Langrognet F. *Rmixmod: MIXture MODelling Package*, 2012. R package for Model-Based supervised and unsupervised classification on qualitative and quantitative data.

Lebret R. and Iovleff S. *Taxonomy3*, 2011. C/C++ implementation of a statistical method providing an analytical framework for high dimensional datasets and complex problems combining several variable types: genetics, genomics, biomarkers and phenotypes.