# Notation

This section provides a concise reference describing notation used throughout this document.

**Numbers and Arrays**

| | |
|---|---|
| $a$ | A scalar (integer or real) |
| $\boldsymbol{a}$ | A vector |
| $\boldsymbol{A}$ | A matrix |
| $\mathsf{A}$ | A tensor |
| $\boldsymbol{I}_n$ | Identity matrix with $n$ rows and $n$ columns |
| $\boldsymbol{I}$ | Identity matrix with dimensionality implied by context |
| $\boldsymbol{e}^{(i)}$ | Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position $i$ |
| $\text{diag}(\boldsymbol{a})$ | A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$ |
| $\mathrm{a}$ | A scalar random variable |
| $\mathbf{a}$ | A vector-valued random variable |
| $\mathbf{A}$ | A matrix-valued random variable |

**Sets and Graphs**

| | |
|---|---|
| $\mathbb{A}$ | A set |
| $\mathbb{R}$ | The set of real numbers |
| $\{0, 1\}$ | The set containing 0 and 1 |
| $\{0, 1, \dots, n\}$ | The set of all integers between 0 and $n$ |
| $[a, b]$ | The real interval including $a$ and $b$ |
| $(a, b]$ | The real interval excluding $a$ but including $b$ |
| $\mathbb{A} \backslash \mathbb{B}$ | Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$ |
| $\mathcal{G}$ | A graph |
| $Pa_{\mathcal{G}}(\mathrm{x}_i)$ | The parents of $\mathrm{x}_i$ in $\mathcal{G}$ |

### Indexing

| | |
|---|---|
| $a_i$ | Element $i$ of vector $\boldsymbol{a}$, with indexing starting at 1 |
| $a_{-i}$ | All elements of vector $\boldsymbol{a}$ except for element $i$ |
| $A_{i,j}$ | Element $i, j$ of matrix $\boldsymbol{A}$ |
| $\boldsymbol{A}_{i,:}$ | Row $i$ of matrix $\boldsymbol{A}$ |
| $\boldsymbol{A}_{:,i}$ | Column $i$ of matrix $\boldsymbol{A}$ |
| $\mathsf{A}_{i,j,k}$ | Element $(i, j, k)$ of a 3-D tensor $\mathsf{A}$ |
| $\mathsf{A}_{:,:,i}$ | 2-D slice of a 3-D tensor |
| $\mathrm{a}_i$ | Element $i$ of the random vector $\mathbf{a}$ |

### Linear Algebra Operations

| | |
|---|---|
| $\boldsymbol{A}^\top$ | Transpose of matrix $\boldsymbol{A}$ |
| $\boldsymbol{A}^*$ | Hermitian transpose of matrix $\boldsymbol{A}$ |
| $\boldsymbol{A}^+$ | Moore-Penrose pseudoinverse of $\boldsymbol{A}$ |
| $\boldsymbol{A} \odot \boldsymbol{B}$ | Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$ |
| $\det(\boldsymbol{A})$ | Determinant of $\boldsymbol{A}$ |

### Calculus

| | |
|---|---|
| $\frac{dy}{dx}$ | Derivative of $y$ with respect to $x$ |
| $\frac{\partial y}{\partial x}$ | Partial derivative of $y$ with respect to $x$ |
| $\nabla_{\boldsymbol{x}} y$ | Gradient of $y$ with respect to $\boldsymbol{x}$ |
| $\nabla_{\boldsymbol{X}} y$ | Matrix derivatives of $y$ with respect to $\boldsymbol{X}$ |
| $\nabla_{\mathsf{X}} y$ | Tensor containing derivatives of $y$ with respect to $\mathsf{X}$ |
| $\frac{\partial f}{\partial \boldsymbol{x}}$ | Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$ |
| $\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$ | The Hessian matrix of $f$ at input point $\boldsymbol{x}$ |
| $\int f(\boldsymbol{x}) d\boldsymbol{x}$ | Definite integral over the entire domain of $\boldsymbol{x}$ |
| $\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$ | Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$ |

### Probability and Information Theory

| | |
|---|---|
| $\mathrm{a} \perp \mathrm{b}$ | The random variables a and b are independent |
| $\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$ | They are conditionally independent given c |
| $P(\mathrm{a})$ | A probability distribution over a discrete variable |
| $p(\mathrm{a})$ | A probability distribution over a continuous variable, or over a variable whose type has not been specified |
| $\mathrm{a} \sim P$ | Random variable a has distribution $P$ |
| $\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E}f(x)$ | Expectation of $f(x)$ with respect to $P(\mathrm{x})$ |
| $\mathrm{Var}(f(x))$ | Variance of $f(x)$ under $P(\mathrm{x})$ |
| $\mathrm{Cov}(f(x), g(x))$ | Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$ |
| $H(\mathrm{x})$ | Shannon entropy of the random variable x |
| $D_{\mathrm{KL}}(P \Vert Q)$ | Kullback-Leibler divergence of P and Q |
| $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ | Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ |

### Functions

| | |
|---|---|
| $f : \mathbb{A} \to \mathbb{B}$ | The function $f$ with domain $\mathbb{A}$ and range $\mathbb{B}$ |
| $f \circ g$ | Composition of the functions $f$ and $g$ |
| $f(\boldsymbol{x}; \boldsymbol{\theta})$ | A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation) |
| $\operatorname{sgn} x$ | Signum function of $x$ |
| $\log_p x$ | Logarithm of $x$ to base $p$ |
| $\log x$ | Natural logarithm of $x$ |
| $\sigma(x)$ | Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$ |
| $\zeta(x)$ | Softplus, $\log(1+\exp(x))$ |
| $\Vert\boldsymbol{x}\Vert_p$, $\Vert\boldsymbol{A}\Vert_p$ | $L^p$ norm of $\boldsymbol{x}$, $\boldsymbol{A}$ |
| $\Vert\boldsymbol{x}\Vert$, $\Vert\boldsymbol{A}\Vert$ | $L^2$ norm of $\boldsymbol{x}$, $\boldsymbol{A}$ |
| $\Vert\boldsymbol{A}\Vert_F$ | Frobenius norm of $\boldsymbol{A}$ |
| $x^+$ | Positive part of $x$, i.e., $\max(0, x)$ |
| $\mathbf{1}_{\text{condition}}$ | is 1 if the condition is true, 0 otherwise |
| $\#(w_i, c_j)$ | Number of times when a word $w_i$ and a context $c_j$ appeared together |
| $\lvert\mathbb{A}\rvert$ | Number of elements in $\mathbb{A}$ |

Sometimes we use a function $f$ whose argument is a scalar but apply it to a vector,

matrix, or tensor: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathsf{X})$. This denotes the application of $f$ to the array element-wise. For example, if $\mathsf{C} = \sigma(\mathsf{X})$, then $\mathsf{C}_{i,j,k} = \sigma(\mathsf{X}_{i,j,k})$ for all valid values of $i$, $j$ and $k$.

**Datasets and Distributions**

| | |
|---|---|
| $p_{\text{data}}$ | The data generating distribution |
| $\hat{p}_{\text{data}}$ | The empirical distribution defined by the training set |
| $\mathbb{X}$ | A set of training examples |
| $\mathcal{W}$ | A vocabulary |
| $\mathcal{C}$ | A set of conexts |
| $\mathcal{D}$ | A document corpus |
| $\boldsymbol{x}^{(i)}$ | The $i$-th example (input) from a dataset |
| $y^{(i)}$ or $\boldsymbol{y}^{(i)}$ | The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning |
| $\boldsymbol{X}$ | The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$ |

# Chapter 1

# Introduction

**Keywords:** lexicology, lexical semantics, lexicography, wordnet, distributional hypothesis, distributional semantics, vector space models.

## 1.1 What Is This Course About?

Distributional semantics is one of the most important notions in contemporary computational linguistics and NLP: representations of meaning exploiting distributional semantics framework are used in almost any NLP system. Therefore, deep understanding of distributional semantics and models based on it is crucial for resolving cutting-edge NLP tasks. The main idea of this course is to give such understanding to the listener.

It is important to understand that this course is not about NLP and/or semantics in general. Many aspects crucial for a NLP (or semantics) course will be omitted, and this course is not recommended for a listener seeking for a good introductory NLP course. Instead of it, this course will give an exhaustive (and unique) introduction to a much more narrower NLP field called distributional semantics (with a context of modern NLP techniques and linguistic theories). So, the course follows three main aims:

1. to give an understanding of distributional hypothesis and its role in contemporary language studies/technology;
2. to survey of state-of-the-art models and formalisms exploiting distributional hypothesis;
3. to describe application basis of distributional hypothesis.

It is also possible to consider this course as a very comprehensive overview of the field which also explains some basic NLP concepts. Despite this course proposes some very theoretical views on semantic theory, the main focus of the course would be held on computational and practical applications of distributional semantics. The course also requires a basic understanding of calculus, linear algebra and probability theory. No exceptional linguistic or engineering background is required.

To sump up: this course is about a narrow, but a very ubiquitous field of computational linguistics and NLP. It will give a deep understanding of what could be done with the most outstanding semantic theory of modern linguistics.

## 1.2 Basic Lexical Semantics

Lexical semantics is a science that tries to define the meaning of words, explaining its flexibility in context and how it contributes to the construction of the meaning of sentences. Lexical semantics is a subfield of lexicology, the science that studies words in the language (lexicon). One of the ultimate goals of lexicology is to develop a formal system representing the variety of lexicon.

Lexical semantics proposes different theories that try to explain the nature of word's meaning, – and, moreover, the nature of concept 'word' and the nature of concept 'meaning'. Distributional semantics is one of such theories, and it is important to understand the object of this study before going deeper.

### 1.2.1 A Notion of Word

The first thing essential for the concept 'word' is that this concept is very vague. Finding a formal definition refers to the philosophy of a language, and for the convenience in this course we will refer to word as a linguistic structure that requires to 3 following conditions:

1. a character set associated with it separates space characters from other words (we will omit languages that do not have a written tradition);

2. a phoneme set associated with it can be pronounced in separation from other phonemes;

3. it has a unitary meaning (it is also a vague concept) which could not be split into several meaning components. From this perspective, we should not rely only on semantic criterion while separating words from non-words (since phrases like 'airbag' could be referred as word and not-word at the same time). We will exploit definition of unitary meaning (referred as a 'semantic component') of (Cruse 1986) to deal with this stumbling stone.

There are also different types of words, for example, Incorporated compounds, Juxtaposed compounds, etc. Fixed phrases. Words are divided according to the typology of languages (inflectional, isolating, etc.). At the same time, one word can have many forms (for simplicity, you can agree that all word forms are different words). We will omit different linguistic theories of words.

A stricter concept for analysis is a lexeme. Lexeme is a set of word forms that have the same meaning. The lemma corresponds to the lexeme in a dictionary form (roughly speaking, the normal form of a word).

### 1.2.2 A Notion of Meaning

The theory of linguistic meaning has two related goals: to establish what words mean (this is the goal of lexical semantics) and explain how complex expressions acquire their meaning based on the meaning of their constituent parts (this is the purpose of a phrase or sentence semantics).

When we speak of the lexicon, we mean mental representation. Lexical information is embedded in the lexicon without representation (i.e., without definitions, as in a dictionary). The word is a lexical form, combined with concept and meaning.

What is included in the concept of 'value' is an open question relating to the prerogative of the philosophy of language. Nevertheless, we know for sure that the lexicon, like any

semiotic system, has two dimensions - the form and the content. Moreover, if there is content that does not have certain forms, then they can be viewed as concepts - mental categories with informational content, as if language-independent. Establishing the correspondence of a concept with a lexical form is called lexicalization - in the process of this a word is formed.

What distinguishes the meaning of a word from the meaning of syntactic and morphological units is that the meaning of a word can be perceived and described by the rapporteur more directly. That is, it is necessary to distinguish lexical meaning and grammatical meaning.

### 1.2.3 Theories of Lexical Semantics

We can distinguish 5 primary meaning theories that propose their own view on what is lexical semantics:

1. Referential hypothesis
2. Conceptual hypothesis
3. Structural hypothesis
4. Prototypical hypothesis
5. Distributional hypothesis

#### Referential hypothesis

Firstly introduced in 1982 (**?** ).

Referential theory says that the words are used to refer on objects and events of the real word. So the **meaning of the word** is the ability to make a reference with the real world. This idea could be formalized by applying logical induction, and in formal semantics it is called **set theory**. In other words, we can describe each word meaning through the set of rules that create references through other words and objects. For example, by saying word 'cat' we refer to a certain object with certain properties, and its extension includes all cats in the world.

The referential hypothesis exploits the concept of intension, which means mapping of all possible worlds to their extensions. In other words, intension is a function which, given a word, returns the things denoted by that world in a particular world. It allows us to make sense of the fact that objects like 'my cat' and 'the pet that lives in my house' have different connotations despite they refer to the same objects.

#### Conceptual hypothesis

Conceptual theory is based on the idea that the reference established between the word and the object is mediated by our mental representation of this object. In other words, the word obtain meaning only through this representation, and its called concept. And when we say 'cat', we are talking not about the real cat, but only about our metal representation of this object.

The framework that exploits conceptual hypothesis is called conceptual semantics, and it is ubiquitous among cognitivists and psycholinguists.

#### Structural hypothesis

Structural hypothesis says that words meaning is not only limited by the ability to make a reference with the real world or some mental representation. The meaning of the word also includes a certain value, which is determined relatively to other words, particularly, the most similar words. So we could say that words meaning is defined by its semantic field.

#### Prototypical hypothesis

Prototypical hypothesis is based on the notions of a category and a best exemplar of the category. Best exemplar is called prototype, and the meaning of the word is the set of the prototype's properties as well as set of objects in order for descending their closeness to the prototype.

This hypothesis exploits different interpretations of prototype, and it could figure as an analogue of a concept as well as some kind of core embedded into meaning and a word.

## 1.3 Computational Lexical Semantics

*e.g. what tasks could be solved.*

### 1.3.1 Historical Approaches

Levenshtein, dictionaries, etc.

### 1.3.2 Word Similarity

Thesauri, WordNet.

## 1.4 Distributional Hypothesis

Distributional hypothesis was first formulated by Harris, and by other linguists. It is the idea that word meaning is determined by its contexts, or 'You shall know a word by the company it keeps.', as Firth, 1957 puts it. One can see that the words to the right and to the left of the words (their neighbors) in natural texts tend to be to some extent similar. These are the words like etc. It seems that semantically similar words share similar contexts.

## 1.5 Connection with Corpus Linguistics

# Bibliography

[1] Geeraerts, D. (2010). *Theories of lexical semantics.* Oxford University Press.

[2] Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

[3] Jezek, E. and Ježek, E. (2016). *The lexicon: An introduction.* Oxford University Press.

[4] Sahlgren, M. (2008). The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53.

# Chapter 2

# Count-based Distributional Models

**Keywords:** count-based models, co-occurrence matrix, similarity measure, sparse vectors, bag-of-words, TF-IDF, (P)PMI, matrix factorization, SVD, LSA, LSI, PCA.

In this chapter, we will consider the models of distributional semantics which were originally introduced in the 1950s, but became widely used in 1970s when computers made it possible for researchers to process text data. In that time several programming languages such as C, Pascal, Fortran, etc. were developed. Also, first information systems were created, and that entailed the need in vector semantic space models.

## 2.1 First Vector Space Models: Matrix of Word Co-occurrence Counts

Vector or distributional models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur. This matrix can be constructed in various ways; let's s begin by looking at one such co-occurrence matrix.

### 2.1.1 Term-document Matrix

In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents. Table 2.1 shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.

| | **As You Like It** | **Twelfth Night** | **Julius Caesar** | **Henry V** |
|---|---|---|---|---|
| battle | 1 | 0 | 7 | 13 |
| good | 115 | 81 | 71 | 91 |
| fool | 37 | 58 | 2 | 5 |
| wit | 21 | 15 | 2 | 3 |

Table 2.1: The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

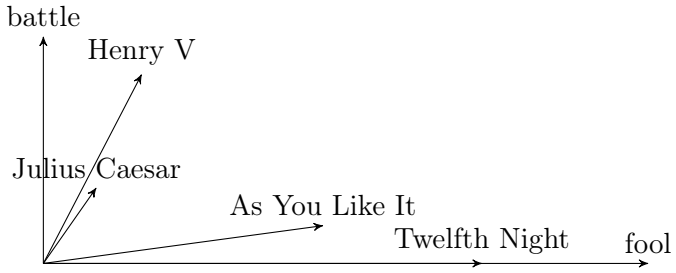



Figure 2.1: Vector space projection on 2-dimensional fool-battle space

The term-document matrix of Table 2.1 was first defined as part of the vector space model of information retrieval (Salton, 1971). In this model, a document is vector space model represented as a count vector, a column in Table 2.2.

So *As You Like It* is represented as the list $[1, 115, 37, 21]$ and *Julius Caesar* is represented as the list $[7, 71, 2, 2]$. In the example in Table 2.2, the vectors are of dimension dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality $|\mathcal{W}|$, the vocabulary size.

| | **As You Like It** | **Twelfth Night** | **Julius Caesar** | **Henry V** |
|---|---|---|---|---|
| battle | 1 | 0 | 7 | 13 |
| good | 115 | 81 | 71 | 91 |
| fool | 37 | 58 | 2 | 5 |
| wit | 21 | 15 | 2 | 3 |

Table 2.2: The term-document matrix for four words in four Shakespeare plays. The colored boxes show that each document is represented as a column vector of length four.

We can think of the vector for a document as identifying a point in $|\mathcal{W}|$-dimensional space; thus the documents in Table 2.2 are points in 4-dimensional space. Since 4-dimensional spaces are hard to draw in textbooks, Figure 2.1 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.

Term-document matrices were originally defined as a means of finding similar documents for the task of document information retrieval. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar. The vectors for the comedies *As You Like It* $[1, 115, 37, 21]$ and *Twelfth Night* $[0, 81, 58, 15]$ look a lot more like each other (more fools and wit than battles) than they do like *Julius Caesar* $[7, 71, 2, 2]$ or *Henry V* $[13, 91, 5, 3]$. We can see the intuition with the raw numbers; in the first dimension (*battle*) the comedies have low numbers and the others have high numbers, and we can see it visually in Pic. 2.3; we'll see very shortly how to quantify this intuition more formally.

A real term-document matrix, of course, wouldn't just have 4 rows and columns, let alone 2. More generally, the term-document matrix $\boldsymbol{W}$ has $|\mathcal{W}|$ rows (one for each word type in the vocabulary) and $|\mathcal{D}|$ columns (one for each document in the collection); as we'll see, vocabulary sizes are generally at least in the tens of thousands, and the number of documents can be enormous (think about all the pages on the web).

### 2.1.2 Term-term and Document-document Matrices

We have seen that documents can be represented as vectors in a vector space. However, semantic vectors can also be used to represent the meaning of words, by associating to each a vector.

The word vector is now a row vector rather than a column vector, and hence the row vector dimensions of the vector are different. The four dimensions of the vector for *fool*, $[37, 58, 2, 5]$, correspond to the four Shakespeare plays. The same four dimensions are used to form the vectors for the other 3 words: *wit*, $[21, 15, 2, 3]$; *battle*, $[1, 0, 7, 13]$; and *good* $[115, 81, 71, 91]$. Each entry in the vector thus represents the counts of the word's occurrence in the document corresponding to that dimension.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

However, it is most common to use a different kind of context for the dimensions of a word's vector representation. Rather than the term-document matrix we use the **term-term matrix**, more commonly called the **word-word matrix**, the **word-context matrix** or the **term-context matrix**, in which the columns are labeled by words rather than documents.

This matrix is thus of dimensionality $|\mathcal{W}| \times |\mathcal{W}|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, generally a window around the word, for example of 4 words to the left and 4 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a $\pm 4$ word window around the row word.

## 2.2 Measures of Word Co-occurrence

In order to build the word-context matrices, different possibilities are available and all of them do not require obligatory the use of $W_{i,j} = \#(w^{(i)}, c^{(j)})$ where $w^{(i)}$ denotes the $i$th word in our ordered or enumerated vocabulary $\mathcal{W}$, and the same can be said about the context $c^{(j)}$. Sometimes we associate words or contexts with their indices, so to obtain the simplicity in formulas we will write $W_{w,c} = \#(w, c)$.

In the following paragraph, the context is used in its general definition; it can be either a word, a sentence, a document, etc.

In this section, we will introduce several modifications and alternative ways of computing co-occurrence matrix.

### 2.2.1 Binary Matrix

In a binary matrix,

$$W_{w,c} = \operatorname{sgn} \#(w, c).$$

This means that in this case we are interested only in whether a word and a context appeared together. For instance, it can be used in hip-hop artists lexicon analysis.

### 2.2.2 TF-IDF

The term frequency—inverse document frequency or TF-IDF is a numerical statistics that tries to point out how important is a word in a given document. It is linked with the number of occurring and the number of documents contained in the corpora. The use of this function is very simple leading it to be often implemented in search engines.

The **term frequency (TF)** is proportional to the number of times a given query word $w$ is found in the corpora. The simplest and most preferred choice is to use simple raw counts of a word $w$ and a document $d$ $\#(w, d)$:

$$TF_{w,d} = \#(w, d).$$

As it may appear, some words such as articles or prepositions can be used too often. In order to deal with this and not being flood by unnecessary information, an **inverse document frequency (IDF)** is also implemented. It is a factor increasing weights of elements and words seldom seen in a document. Thus for the elements of TF-IDF matrix we have the following formula:

$$W_{w,d} = TF_{w,d} \cdot IDF_w$$

where

$$IDF_w = \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : w \in d\}|}.$$

To simplify further formulas, we will denote $\mathcal{D}_{|w} := \{d \in \mathcal{D} : w \in d\}$. So,

$$IDF_w = \log \frac{|\mathcal{D}|}{|\mathcal{D}_{|w}|}.$$

The inverse document frequency was originally conceived by Karen Spärck Jones in 1972. See HIS PUBLICATION.

There are a lot of variants of this formula. For instance, $TF_{w,d}$ may be any reasonable function of $\#(w, d)$, e. g. $TF_{w,d} = \log(1 + \#(w, d))$, etc. Hence the formula has to be adapted. Another possibility is modification of IDF-term. Sometimes you can see formulas with shifted numerator and denominator. In general, the smoothed IDF can be computed as follows:

$$IDF_w^{\alpha,\beta} = \log \frac{|\mathcal{D}| + \alpha}{|\mathcal{D}_{|w}| + \beta}.$$

Anyway, the simplest version of TF-IDF weights has a probabilistic sense. Consider a model in which the probability of finding a word $w$ in an arbitrary document $d$ is

$$P(w) = \frac{|\mathcal{D}|}{|\mathcal{D}_{|w}|}.$$

Consider a query $q$ which contains words $w_1, \ldots, w_Q$. To estimate the probability of finding words $w_1, \ldots, w_Q$ from a query $q$ we can use a heuristic formula:

$$P(q, d) = \prod_{k=1}^{Q} P(w_k, d) = \prod_{k=1}^{Q} P(w_k)^{\#(w_k,d)} = \prod_{k=1}^{Q} \left( \frac{|\mathcal{D}|}{|\mathcal{D}_{|w_k}|} \right)^{\#(w_k,d)}.$$

Now let's take a logarithm of it:

$$\log P(q,d) = \sum_{k=1}^{Q} \#(w_k,d) \log \frac{|\mathcal{D}|}{|\mathcal{D}_{|w_k}|}$$

and multiply the result by $-1$. We will obtain

$$-\log P(q,d) = \sum_{k=1}^{Q} \#(w_k,d) \log \frac{|\mathcal{D}_{|w_k}|}{|\mathcal{D}|} = \sum_{k=1}^{Q} TF_{w_k,d} \cdot IDF_{w_k}.$$

Under the sum, the first term is a document frequency and the second term is an inverse document frequency.

While the implementation is easy, this algorithm does have some drawbacks. Basically, there is no semantic match to a given query. All synonyms or related terms can not be reached without modifications. Moreover, certain words may give some noisy results such as "apple" or other common words used by brands.

### 2.2.3 BM25

Okapi BM25 is function based on bag of words. Its role is to rank documents according to contained terms and relations between them. The name of "Okapi" refers to the first place where this function was implemented. Mainly used by searching engines, this algorithm overtakes the classic TD-IDF function from the speed side and the accuracy side. In 2015, Lucene, an open-source search engine, moved to BM25 with some modifications to increase its performances.

The way this function works is strongly linked with TF-IDF, however some important modifications are presented. For a given query $q$ in a document $d$, all words are associated to some scoring functions and merged together at the end. The formula is as follows:

$$\begin{aligned}\text{BM25}(q,d) &= \sum_{k=1}^{Q} IDF_{w_k} \frac{\#(w_k,d)(k_1+1)}{\#(w_k,d) + k_1(1-b+b\frac{|\mathcal{D}|}{avgdl})} = \\ &= \sum_{k=1}^{Q} IDF_{w_k} \frac{TF_{w_k,d}(k_1+1)}{TF_{w_k,d} + k_1(1-b+b\frac{|\mathcal{D}|}{avgdl})}\end{aligned}$$

where $avgdl$ is the average document length and $k_1$ and $b$ are two optimization parameters, usually $k_1 = 2.0$ and $b = 0.75$.

Nowadays, some modifications have been made to BM25 such has BM25F which splits documents into fields (head, core, etc.) before searching inside or BM25+ which adds one parameter to help BM25 to work with long documents.

### 2.2.4 PMI. Collocation Extraction

An alternative weighting function to TF-IDF is called PPMI (positive pointwise mutual information). PPMI draws on the intuition that best way to weigh the association between two words is to ask how much more the two words co-occur in our corpus than we would have a priori expected them to appear by chance. **Pointwise mutual information (PMI)** (Fano, 1961) is one of the most important concepts in NLP. It is a measure of how often

two random variables x and y occur, compared with what we would expect if they were independent:

$$I(\mathrm{x},\mathrm{y}) = \log_2 \frac{P(\mathrm{x},\mathrm{y})}{P(\mathrm{x})P(\mathrm{y})} = \log_2 \frac{P(\mathrm{x} \mid \mathrm{y})P(\mathrm{y})}{P(\mathrm{x})P(\mathrm{y})} = \log_2 \frac{P(\mathrm{x} \mid \mathrm{y})}{P(\mathrm{x})}$$
$$= \log_2 \frac{P(\mathrm{y} \mid \mathrm{x})P(\mathrm{x})}{P(\mathrm{x})P(\mathrm{y})} = \log_2 \frac{P(\mathrm{y} \mid \mathrm{x})}{P(\mathrm{y})}.$$

If $\mathrm{x} \perp \mathrm{y}$ then $P(\mathrm{x} \mid \mathrm{y}) = P(\mathrm{x})$ and $P(\mathrm{y} \mid \mathrm{x}) = P(\mathrm{y})$, and thus $I(\mathrm{x},\mathrm{y}) = 0$.

The pointwise mutual information between a target word $w$ and a context word $c$ (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$PMI_{w,c} = \log_2 \frac{P(w,c)}{P(w)P(c)}.$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would expect the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated.

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each $10^{-6}$ occur together more often than chance, we would need to be certain that the probability of the two occurring together is significantly different than $10^{-12}$, and this kind of granularity would require an enormous corpus. Furthermore it's not clear whether it's even possible to evaluate such scores of "unrelatedness" with human judgments. For this reason it is more common to use **positive PMI (PPMI)** which replaces all negative PMI values with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994):

$$PPMI_{w,c} = \max(PMI_{w,c}, 0) = PMI^{+}_{w,c}.$$

More formally, let's assume we have a co-occurrence matrix $\boldsymbol{W}$ with $|\mathcal{W}|$ rows (words) and $|\mathcal{C}|$ columns (contexts), where $W_{w,c} = \#(w,c)$. This can be turned into a PPMI matrix where

$$PPMI_{w,c} = \max(\log_2 \frac{p_{w,c}}{p_{w,:} \cdot p_{:,c}}, 0)$$

where

$$p_{w,c} = \frac{\#(w,c)}{\sum_{w' \in \mathcal{W}} \sum_{c' \in \mathcal{C}} \#(w',c')}, \; p_{w,:} = \sum_{c' \in \mathcal{C}} p_{w,c'}, \; p_{:,c} = \sum_{w' \in \mathcal{W}} p_{w',c}.$$

In computational linguistics, PMI has been used for finding collocations and associations between words. The following Table 2.3 shows counts of pairs of words getting the most and the least PMI scores in the first 50 millions of words in Wikipedia (dump of October 2015) filtering by 1,000 or more co-occurrences. The frequency of each count can be obtained by dividing its value by 50,000,952. (Note: natural log is used to calculate the PMI values in this example, instead of log base 2)

| $w$ | $c$ | $\#(w)$ | $\#(c)$ | $\#(w,c)$ | PMI |
|---|---|---|---|---|---|
| puerto | rico | 1938 | 1311 | 1159 | 10.0349081703 |
| hong | kong | 2438 | 2694 | 2205 | 9.72831972408 |
| los | angeles | 3501 | 2808 | 2791 | 9.56067615065 |
| carbon | dioxide | 4265 | 1353 | 1032 | 9.09852946116 |
| prize | laureate | 5131 | 1676 | 1210 | 8.85870710982 |
| san | francisco | 5237 | 2477 | 1779 | 8.83305176711 |
| nobel | prize | 4098 | 5131 | 2498 | 8.68948811416 |
| ice | hockey | 5607 | 3002 | 1933 | 8.6555759741 |
| star | trek | 8264 | 1594 | 1489 | 8.63974676575 |
| car | driver | 5578 | 2749 | 1384 | 8.41470768304 |
| it | the | 283891 | 3293296 | 3347 | -1.72037278119 |
| are | of | 234458 | 1761436 | 1019 | -2.09254205335 |
| this | the | 199882 | 3293296 | 1211 | -2.38612756961 |
| is | of | 565679 | 1761436 | 1562 | -2.54614706831 |
| and | of | 1375396 | 1761436 | 2949 | -2.79911817902 |
| a | and | 984442 | 1375396 | 1457 | -2.92239510038 |
| in | and | 1187652 | 1375396 | 1537 | -3.05660070757 |
| to | and | 1025659 | 1375396 | 1286 | -3.08825363041 |
| to | in | 1025659 | 1187652 | 1066 | -3.12911348956 |
| of | and | 1761436 | 1375396 | 1190 | -3.70663100173 |

Table 2.3: The results of applying PMI to search collocations

Good collocation pairs have high PMI because the probability of co-occurrence is only slightly lower than the probabilities of occurrence of each word. Conversely, a pair of words whose probabilities of occurrence are considerably higher than their probability of co-occurrence gets a small PMI score.

PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values. One way to reduce this bias toward low frequency events is to slightly change the computation for $P(c)$, using a different function $P_\alpha(c)$ that raises contexts to the power of $\alpha$:

$$PMI^\alpha_{w,c} = \log_2 \frac{P(w,c)}{P(w)P_\alpha(c)},\ P_\alpha(c) = \frac{\#(c)^\alpha}{\sum_{c' \in \mathcal{C}} \#(c')^\alpha}.$$

Levy et al. (2015) found that a setting of $\alpha = 0.75$ improved performance of embeddings on a wide range of tasks (drawing on a similar weighting used for skip-grams described below in Section 4). This works because raising the probability to $\alpha = 0.75$ increases the probability assigned to rare contexts, and hence lowers their PMI ($P_\alpha(c) > P(c)$ when $c$ is rare).

Another possible solution is Laplace smoothing: Before computing PMI, a small constant $k$ (values of $0.1 - 3$ are common) is added to each of the counts, shrinking (discounting) all the non-zero values. The larger the $k$, the more the non-zero counts are discounted.

## 2.3 Similarity Measures

### 2.3.1 Cosine Similarity

To define similarity between two target words $u$ and $v$, we need a measure for taking two such vectors and giving a measure of vector similarity. By far the most common similarity metric is the **cosine** of the angle between the vectors.

$$\cos(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|} = \frac{\sum_{i=1}^{N} u_i v_i}{\sqrt{\sum_{i=1}^{N} u_i^2} \sqrt{\sum_{i=1}^{N} v_i^2}}.$$

Sometimes, to satisfy the first axiom of distance, we can use an equivalent formula:

$$\text{cosine}(\boldsymbol{u}, \boldsymbol{v}) = 1 - \cos(\boldsymbol{u}, \boldsymbol{v}).$$

### 2.3.2 Jaccard Index

The **Jaccard coefficient** measures similarity between finite sample sets $\mathbb{A}$ and $\mathbb{B}$, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(\mathbb{A}, \mathbb{B}) = \frac{|\mathbb{A} \cap \mathbb{B}|}{|\mathbb{A} \cup \mathbb{B}|}.$$

If $\mathbb{A}$ and $\mathbb{B}$ are both empty, we define $J(\mathbb{A}, \mathbb{B}) = 1$.

The Jaccard distance, which measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1:

$$d_J(\mathbb{A}, \mathbb{B}) = 1 - J(\mathbb{A}, \mathbb{B}).$$

Given two objects, A and B, each with $n$ binary attributes, the Jaccard coefficient is a useful measure of the overlap that A and B share with their attributes. Each attribute of A and B can either be 0 or 1. The total number of each combination of attributes for both A and B are specified as follows:

- $M_{11}$ represents the total number of attributes where A and B both have a value of 1.
- $M_{01}$ represents the total number of attributes where the attribute of A is 0 and the attribute of B is 1.
- $M_{10}$ represents the total number of attributes where the attribute of A is 1 and the attribute of B is 0.
- $M_{00}$ represents the total number of attributes where A and B both have a value of 0.

Then the Jaccard coefficient can be calculated as:

$$J(A, B) = \frac{M_{11}}{M_{11} + M_{01} + M_{10}}$$

and follows the Jaccard distance $d_J$ :

$$d_J(A,B) = \frac{M_{10} + M_{01}}{M_{11} + M_{01} + M_{10}} = 1 - J(A,B).$$

Grefenstette in 19?? generalized these formulas on a case of non-negative real valued vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

$$\text{Jaccard}(\boldsymbol{u},\boldsymbol{v}) = \frac{\sum_{i=1}^{N} \min(u_i, v_i)}{\sum_{i=1}^{N} \max(u_i, v_i)}.$$

### 2.3.3 Dice Coefficient

The **Dice coefficient** is similar to the Jaccard index except that the denominator is the total weight of non zero entries within the two vectors compared.

$$\text{Dice}(\boldsymbol{u},\boldsymbol{v}) = \frac{2\sum_{i=1}^{N} \min(u_i, v_i)}{\sum_{i=1}^{N} u_i + v_i}.$$

An important point with the Dice coefficient is the use of wrong values. Instead of dealing only with positive ones, wrong values penalize the result. This leads to a different behavior than some other metrics.

### 2.3.4 Jensen—Shannon Divergence

TODO

## 2.4 Matrix Factorization. SVD, PCA

### 2.4.1 PCA

**Principal component analysis (PCA)** is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. If there are $n$ observations with $p$ variables, then the number of distinct principal components is $\min(n-1, p)$. 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), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing $n$ observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA was invented in 1901 by Karl Pearson; it was later independently developed and named by Harold Hotelling in the 1930s.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualised as a set of

coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint[citation needed]. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced.

PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset.

### 2.4.2 SVD

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.

Perhaps the most known and widely used matrix decomposition method is the **singular-value decomposition (SVD)**. All matrices have an SVD, which makes it more stable than other methods, such as the eigendecomposition. As such, it is often used in a wide array of applications including compressing, denoising, and data reduction.

**SVD Theorem**. Suppose $\boldsymbol{M}$ is a $m \times n$ matrix whose entries come from the field $\mathbb{K}$, which is either the field of real numbers $\mathbb{R}$ or the field of complex numbers $\mathbb{C}$. Then there exists a factorization, called a "singular value decomposition" of $\boldsymbol{M}$, of the form

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$$

where $\boldsymbol{U}$ is an $m \times m$ unitary matrix over $\mathbb{K}$ (if $\mathbb{K} = \mathbb{R}$, unitary matrices are orthogonal matrices), $\boldsymbol{\Sigma}$ is a diagonal $m \times n$ matrix with non-negative real numbers on the diagonal, $\boldsymbol{V}$ is an $n \times n$ unitary matrix over $\mathbb{K}$, and $\boldsymbol{V}^*$ is the conjugate transpose of $\boldsymbol{V}$.

The diagonal entries $\sigma_i$ of $\boldsymbol{\Sigma}$ are known as the singular values of $\boldsymbol{M}$. A common convention is to list the singular values in descending order. In this case, the diagonal matrix, $\boldsymbol{\Sigma}$, is uniquely determined by $\boldsymbol{M}$ (though not the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ if $\boldsymbol{M}$ is not square).

Some practical applications need to solve the problem of approximating a matrix $\boldsymbol{M}$ with another matrix $\tilde{\boldsymbol{M}}$, said truncated, which has a specific rank $r$. In the case that the approximation is based on minimizing the Frobenius norm of the difference between $\boldsymbol{M}$ and $\tilde{\boldsymbol{M}}$ under the constraint that rank$\tilde{\boldsymbol{M}} = r$ it turns out that the solution is given by the SVD of $\boldsymbol{M}$, namely

$$\tilde{\boldsymbol{M}} = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^*$$

where $\tilde{\boldsymbol{\Sigma}}$ is the same matrix as $\boldsymbol{\Sigma}$ except that it contains only the $r$ largest singular values (the other singular values are replaced by zero). This is known as the Eckart–Young theorem, as it was proved by those two authors in 1936 (although it was later found to have been known to earlier authors; see Stewart 1993).

For the case of simplicity we will focus on the SVD for real-valued matrices and ignore the case for complex numbers. Thus for every rectaingular matrix $\boldsymbol{M}$ there exist orthogonal matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ and a diagonal matrix $\boldsymbol{\Sigma}$ such that

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top.$$

## 2.5 Count-based Distributional Models Based on Matrix Factorization: LSA, LSI

### 2.5.1 LSA

**Latent semantic analysis (LSA)** is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text (Landauer and Dumais, 1997). The underlying idea is that the aggregate of all the word contexts in which a given word does and does not appear provides a set of mutual constraints that largely determines the similarity of meaning of words and sets of words to each other. The adequacy of LSA's reflection of human knowledge has been established in a variety of ways. Its performance has been assessed more or less rigorously in several ways. LSA was assessed as

1. a predictor of query-document topic similarity judgments.
2. a simulation of agreed upon word-word relations and of human vocabulary test synonym judgments.
3. a simulation of human choices on subject-matter multiple choice tests.
4. a predictor of text coherence and resulting comprehension.
5. a simulation of word-word and passage-word relations found in lexical priming experiments.
6. a predictor of subjective ratings of text properties, i.e. grades assigned to essays.
7. a predictor of appropriate matches of instructional text to learners.
8. LSA has been used with good results to mimic synonym, antonym, singular-plural and compound-component word relations, aspects of some classical word sorting studies, to simulate aspects of imputed human representation of single digits, and, in pilot studies, to replicate semantic categorical clusterings of words found in certain neuropsychological deficits.

### 2.5.2 LSI

J. R. Anderson (1990) has called attention to the analogy between **information retrieval** and human semantic memory processes. One way of expressing their commonality is to think of a searcher as having in mind a certain meaning, which he or she expresses in words, and the system as trying to find a text with the same meaning. Success, then, depends on the system representing query and text meaning in a manner that correctly reflects their similarity for the human. **Latent semantic indexing** (**LSI**; LSA's alias in this application) does this better than systems that depend on literal matches between terms in queries and documents. Its superiority can often be traced to its ability to correctly match queries to (and only to) documents of similar topical meaning when query and document use different words. In the text-processing problem to which it was first applied, automatic matching of information requests to document abstracts, SVD provides a significant improvement over prior methods. In this application, the text of the document database is first represented as a matrix of terms by documents (documents are usually represented by a surrogate such as a title, abstract and/or keyword list) and subjected to SVD, and each word and document

is represented as a reduced dimensionality vector, usually with 50-400 dimensions. A query is represented as a "pseudo-document" a weighted average of the vectors of the words it contains. (A document vector in the SVD solution is also a weighted average of the vectors of words it contains, and a word vector a weighted average of vectors of the documents in which it appears.)

The first tests of LSI were against standard collections of documents for which representative queries have been obtained and knowledgeable humans have more or less exhaustively examined the whole database and judged which abstracts are and are not relevant to the topic described in each query statement. In these standard collections LSI's performance ranged from just equivalent to the best prior methods up to about 30% better.

## 2.6 Clustering

In data analysis, **clustering**, or cluster analysis, is the process of creating sets of similar elements. This concept was made by Driver and Kroeber in anthropology and then used for classification of traits in psychology by Cattell in 1943.

Once clusters are made, the idea afterwards is to consider them either as references or as group of objects for data operations.

In the following lines, the uses of some clusters in natural language processing are going to be described.

### 2.6.1 Brown Clusters

The first model of cluster presented here is the **Brown clustering** model.

Consider a corpora as a single sequence of words $w_1, \ldots, w_T$. The Brown clustering algorithm groups elements of a vocabulary $\mathcal{W}$ into $k$ classes $1, \ldots, k$ (0 is used for denoting the start of the sequence). It was initially proposed, in the natural language processing field, by Peter Brown et al. in (1). The main idea of Brown's model is that the words from one cluster often appear after words from other clusters.

Let $C$ be the function which takes the word $w \in \mathcal{W}$ as input and returns its cluster number $0, \ldots, k$. Let $E$ be a conditional probability distribution over words $\mathcal{W}$ given a cluster number of a current word, and $Q$ be a probability distribution over cluster numbers given a cluster number of a previous word.

The algorithm tries to maximize the joint probability

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} E(w_t \mid C(w_t)) Q(C(w_t) \mid C(w_{t-1}))$$

(we will talk about generative models in the Chapter 3). The distributions $E$ and $Q$ and a function $C$ are learned during the training procedure.

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays

June March July April January December October November September August

people guys folks fellows CEOs chaps doubters commies unfortunates blokes

down backwards ashore sideways southward northward overboard aloft downwards adrift

water gas coal liquid acid sand carbon steam shale iron

great big vast sudden mere sheer gigantic lifelong scant colossal

man woman boy girl lawyer doctor guy farmer teacher citizen

American Indian European Japanese German African Catholic Israeli Italian Arab

pressure temperature permeability density porosity stress velocity viscosity gravity tension

mother wife father son husband brother daughter sister boss uncle

machine device controller processor CPU printer spindle subsystem compiler plotter

John George James Bob Robert Paul William Jim David Mike

anyone someone anybody somebody

feet miles pounds degrees inches barrels tons acres meters bytes

director chief professor commissioner commander treasurer founder superintendent dean custodian

Figure 2.2: Brown clusters

The original algorithm has a number of modifications. For instance, if you run it again for each separate cluster, you will get hierarchical Brown clusters. The final tree, or cluster, is composed of a reduce number of roots and a large number of leafs (where all elements of the corpora can be found). Thanks to this architecture, cluster merging is rather simple.

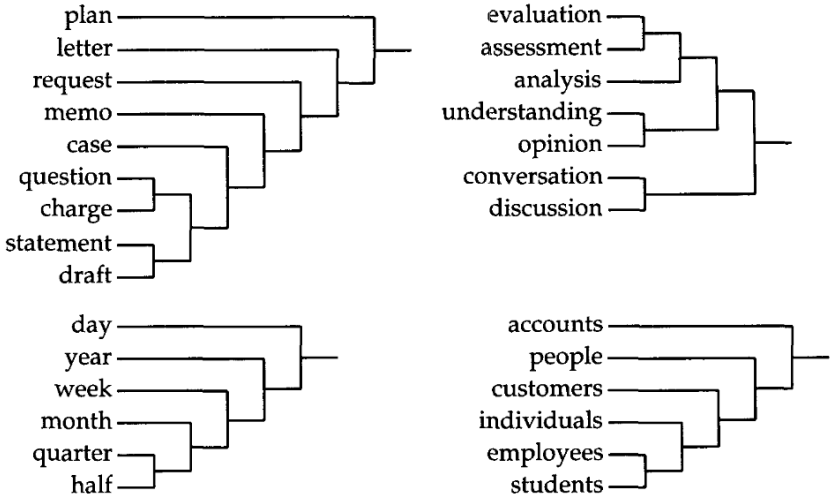



Figure 2.3: Hierarchical Brown clusters

One of the main drawbacks of this method is the limitation of range (original implementation requires $O(|\mathcal{W}|^5)$ operations, more efficient version—$O(|\mathcal{W}|^3)$, but it's still hard for real-world datasets). This can be seen if word-models are too simple, as it may be the case with bi-gram. For instance, it is mostly the case that only most common words from clusters

are going to be predicted. This limitation can be overtaken with more complex models but solutions stay in the field of greedy heuristic results.

### 2.6.2 Self-organizing Semantic Maps

**Self-organizing maps (SOMs)** (Ritter and Kohonen 1989, 1990) are a kind of neural network (see Chapter 4) that aims to reduce the dimension of a given input $\boldsymbol{x}$ through a mapping. This method is usually used as an unsupervised tool and works well for pattern recognition, statistical analysis and statistical similarities.

In natural language processing, the SOM can be used on contextual information to find similarities between words (the context in which they appear) and can organize them into grammatical and semantic categories represented as a two-dimensional array. These similarities of categories are calculated according to (normally Euclidean) distance of relationships within the array.

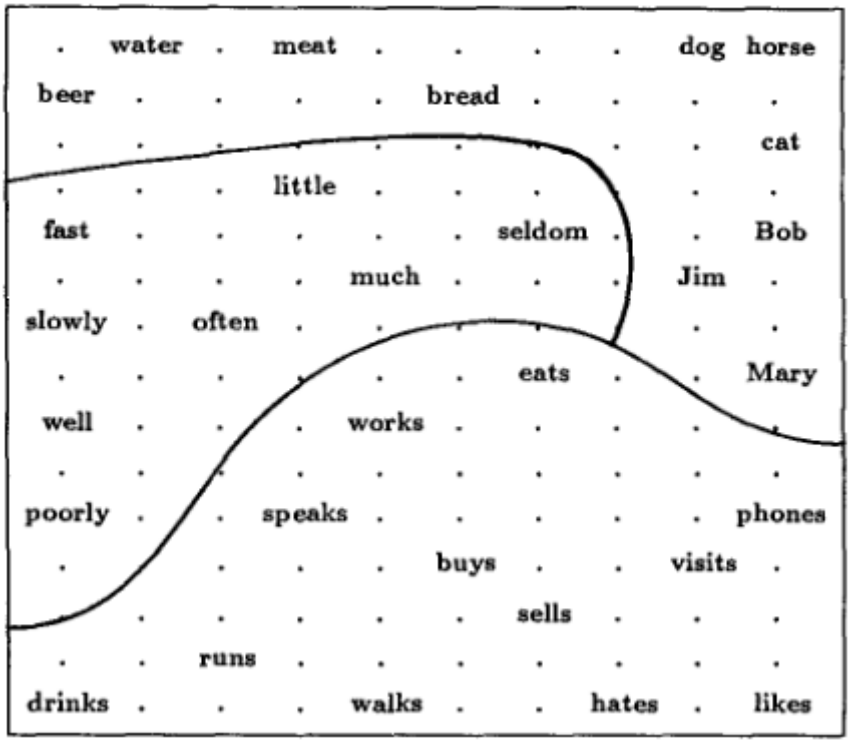



Figure 2.4: «Semantic map» obtained on a network of $10 \times 15$ cells after 2000 presentations of word-context-pairs derived from 10,000 random sentences of kind «Mary likes meat», «Jim speaks well», «Mary likes Jim», «Jim eats often»... Nouns, verbs and adverbs are segregated into different domains. Within each domain a further grouping according to aspects of meaning is discernible

In (4; 3), SOMs are reorganized according to word similarities and the SOM algorithm is based on competitive learning. The best matching neurons to inputs are selected and updated according to the adaptation rule.

Word category maps can be used in a large variety of tasks such information retrieval and text data mining as it has be shown that SOM can provide wield overviews of large corpora. Also, the competitive learning approach can be seen as a good basis for natural language interpretation.

### 2.6.3 Hyperspace Analogue to Language

The **hyperspace analogue to language (HAL)** also known as **semantic memory** was introduced by (8; 9). It is based on the idea that similar words may share similar meanings and may appear rather closely to each other.

The process is as follows. The goal is creation of a matrix where unique words are stored and represented as rows and columns. Number of occurrences close to the other unique words are counted and filled in the matrix.

As the result, co-occurring words have similar rows as weights are given according to the distance between the occurrence and the context: the smaller the distance, the higher is the weight.

For the sentence "The cat eats all cookies and all the meat", the co-occurring matrix, with a window of 3, is as follows:

| | the | cat | eats | all | cookies | and | meat |
|---|---|---|---|---|---|---|---|
| the | 0 | 0 | 0 | 0 | 1 | 2 | 0 |
| cat | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| eats | 2 | 3 | 0 | 0 | 0 | 0 | 0 |
| all | 1 | 2 | 3 | 1 | 2 | 3 | 0 |
| cookies | 0 | 1 | 2 | 3 | 0 | 0 | 0 |
| and | 0 | 0 | 1 | 2 | 3 | 0 | 0 |
| meat | 3 | 0 | 0 | 0 | 0 | 1 | 0 |

To extract vectors from this matrix, row and column of each words are joined. For instance, the vector for *cat* and *meat* are given by:

$$\boldsymbol{v}_{cat} = [3,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0;\ 0,\ 0,\ 3,\ 2,\ 1,\ 0,\ 0],$$
$$\boldsymbol{v}_{meat} = [3,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0;\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0].$$

With vectors as shown, it is simple to compare then thanks to their cosine or other geometrical metrics.

### 2.6.4 Phrase Clustering

With current algorithms and resources, clustering have helped to solve or improve results in many natural language processing tasks. However, in some particular fields, things seem stuck due to not strong enough model. It may be the case in named entity recognition or some information comparison tasks.

Lin and Wu in (7) describe another approach. Instead of using word based clusters, their idea is to consider sentences. Indeed, information contained inside these ones are bigger than information contained within words. However, with current ways of clustering, sentences can not be used. Hence, they decide, instead of recreating a complete model to adapt the K-Means one to sentences. This algorithm is a semi-supervised learning algorithm which takes as features word-clusters and phrase-clusters.

One strength of the algorithm is its possibility to scale easily first and to give good results. The main assumption was as for words; to consider that similar contexts give similar meanings.

The context feature for a phrase is based on the frequency counts of words in a given window of phrases. Then, frequency counts are translated into pointwise mutual information (PMI):

$$PMI(phrase, feature) = \log\left(\frac{P(phrase, feature)}{P(phrase)P(feature)}\right).$$

The clustering process is based on the K-Means clustering which assigns elements to a cluster according to a distance function.

## 2.7 Spectral Word Embeddings (advanced)

## 2.8 Pros and Cons of Count-based Models

To summarize previous parts, count-based models have been used for decades now and shown multiple strength as in their use as in their implementation. Hypothesis needed for such algorithms are very convenient and let a lot of people worked and explore possibilities. Different approaches are possible within this formalism and results are about a lot of topic.

In the following sections, main advantages and drawbacks of these models are going to be described.

### 2.8.1 Advantages

The first and biggest strength of count-based models is their age; they have been used and explored for decades. Hence, results found are for most of them very well known and accurate. The support for these algorithms covers an impressive range going from linear algebra to information theory and statistics.

Another strength is about data manipulation. Even if for large amount of texts and words these models may suffer, as soon as sets and corpora are small enough, for a given task, accuracy, efficiency and speed are going to be very high. For a large part of semantic analysis performed in industry, the use of these techniques remains against neural network based approaches.

### 2.8.2 Drawbacks

If the strength of advantages push the use of such models, some drawbacks still remain. The first one is linked with the interpretation of queries; words inside one are most of the time seen as bag-of-word. In other words, the order of entities in a sentence does not matter: "A cat is chasing a dog" is equivalent to "A dog is chasing a cat", "Bayern vs. PSG" is equivalent to "PSG vs. Bayern". This miss comprehension of the grammar structure is very important and blocks the access, with those algorithms, to some area of natural language processing.

Linked with the idea of "no grammar structure" encoded, issues about ambiguous words can be raised. Indeed, some words share the same writing for different senses as it is the case with polysemous words.

The second main drawback is about "known-words". During training of algorithms, a certain quantity of words is being processed leading to vectors and matrices. However, it may appear that some words do not belong to the training data set and then are considered as unknown during computational process. This issue is well known and is called out-of-vocabulary words. In such case, one variant is to recompute all matrices, however, it will

require some time. This problem is seen with new documents too. For a given set of texts, adding some others once computation done can lead to new expressions and issues. As said in the previous part, these models are working very well for closed dimension set of data where the amount of information is not too big. Otherwise, some out of range problems can appear.

## 2.9 Seminar

- Explanation of SVD.
- Hands-on tutorial with count-based models.
- **Homework:** implementation of a count-based model for information retrieval task.

# Bibliography


[1] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

[2] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

[3] Honkela, T. (1997). Self-organizing maps of words for natural language processing applications. In *Proceedings of the International ICSC Symposium on Soft Computing*, pages 401–407. Citeseer.

[4] Honkela, T., Pulkki, V., and Kohonen, T. (1995). Contextual relations of words in grimm tales analyzed by self-organizing map. In *Proceedings of ICANN-95, international conference on artificial neural networks*, volume 2, pages 3–7. EC2 et Cie Paris.

[5] Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*, volume 3. Pearson London.

[6] Landauer, T. K., Foltz, P. W., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284.

[7] Lin, D. and Wu, X. (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.

[8] Lund, K. (1995). Semantic and associative priming in high-dimensional semantic space. In *Proc. of the 17th Annual conferences of the Cognitive Science Society, 1995.*

[9] Lund, K. and Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208.

[10] Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188.

# Chapter 3

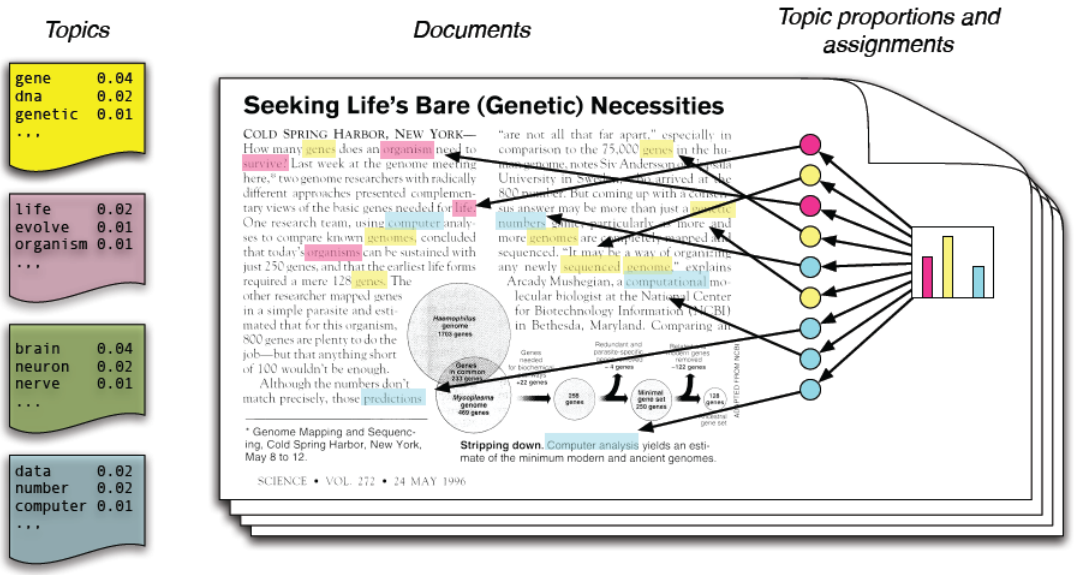
# Topic Modeling

**Keywords:** topic modeling, probabilistic topic modeling, barycentric coordinate system, Dirichlet distribution, EM-algorithm, LDA, ARTM.

## 3.1 Topic Modeling. What Is It, What Task Does It Resolve

In natural language processing, a **topic modeling** model is a statistical model which aims to highlight topics contained within documents. Its actions is about creating clusters of words targeting the same definition or concept. This process may be used to guess the subject of some documents or benches of texts regarding their content or to classify documents (in order to create some kind of search engines).

As simple used of topic modeling is only based on statistical approaches, it has to be understood that results may present biases according to the prepossessing of data. However, more complete structures and algorithms allow very accurate uses, such as latent Dirichlet allocation (LDA, 3.6).



## 3.2 Graphical Models

### 3.2.1 Introduction to Graphical Models

Graphical models bring together graph theory and probability theory in a powerful formalism for multivariate statistical modeling. In various applied fields including bioinformatics, speech processing, image processing and control theory, statistical models have long been formulated in terms of graphs, and algorithms for computing basic statistical quantities such as likelihoods and score functions have often been expressed in terms of recursions operating on these graphs; examples include phylogenies, pedigrees, hidden Markov models, Markov random fields, and Kalman filters. These ideas can be understood, unified, and generalized within the formalism of **graphical models**. Indeed, graphical models provide a natural tool for formulating variations on these classical architectures, as well as for exploring entirely new families of statistical models. Accordingly, in fields that involve the study of large numbers of interacting variables, graphical models are increasingly in evidence.

We begin with background on graphical models. The key idea is that of factorization: a graphical model consists of a collection of probability distributions that factorize according to the structure of an underlying graph. Here, we are using the terminology "distribution" loosely; our notation $p$ should be understood as a mass function (density with respect to counting measure) in the discrete case, and a density with respect to Lebesgue measure in the continuous case. There are two main kinds of structured probabilistic models: directed and undirected. Both kinds of graphical models use a graph $\mathcal{G}$ in which each node in the graph corresponds to a random variable, and an edge connecting two random variables means that the probability distribution is able to represent direct interactions between those two random variables.

### 3.2.2 Directed Models

**Directed models** use graphs with directed edges, and they represent factorizations into conditional probability distributions, as in the example above. Specifically, a directed model contains one factor for every random variable $\mathrm{x}_i$ in the distribution, and that factor consists of the conditional distribution over $\mathrm{x}_i$ given the parents of $\mathrm{x}_i$, denoted $Pa_{\mathcal{G}}(\mathrm{x}_i)$:

$$p(\mathbf{x}) = \prod_i p(\mathrm{x}_i \mid Pa_{\mathcal{G}}(\mathrm{x}_i)).$$

See figure 3.1 for an example of a directed graph and the factorization of probability distributions it represents. This graph allows us to quickly see some properties of the distribution. For example, a and c interact directly, but a and e interact only indirectly via c.

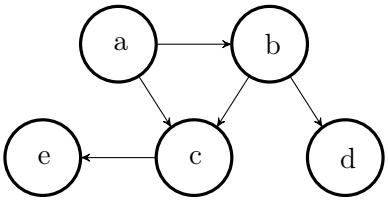



Figure 3.1: A directed graphical model over random variables a, b, c, d and e.

This graph corresponds to probability distributions that can be factored as follows:

$$p(\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}) = p(\mathrm{a})p(\mathrm{b} \mid \mathrm{a})p(\mathrm{c} \mid \mathrm{a}, \mathrm{b})p(\mathrm{d} \mid \mathrm{b})p(\mathrm{e} \mid \mathrm{c}).$$

### 3.2.3 Undirected Models

**Undirected models** use graphs with undirected edges, and they represent factorizations into a set of functions; unlike in the directed case, these functions are usually not probability distributions of any kind. Any set of nodes that are all connected to each other in $\mathcal{G}$ is called a clique. Each clique $\mathcal{C}^{(i)}$ in an undirected model is associated with a factor $\varphi^{(i)}\left(\mathcal{C}^{(i)}\right)$. These factors are just functions, not probability distributions. The output of each factor must be non-negative, but there is no constraint that the factor must sum or integrate to 1 like a probability distribution.

The probability of a configuration of random variables is proportional to the product of all of these factors-assignments that result in larger factor values are more likely. Of course, there is no guarantee that this product will sum to 1. We therefore divide by a normalizing constant $Z$, defined to be the sum or integral over all states of the product of the $\varphi$ functions, in order to obtain a normalized probability distribution:

$$p(\mathbf{x}) = \frac{1}{Z}\prod_i \varphi^{(i)}\left(\mathcal{C}^{(i)}\right).$$

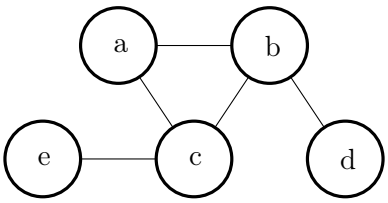



Figure 3.2: An undirected graphical model over random variables a, b, c, d and e.

The graph 3.2 corresponds to probability distributions that can be factored as

$$p(\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{d},\mathrm{e}) = \varphi^{(1)}(\mathrm{a},\mathrm{b},\mathrm{c})\varphi^{(2)}(\mathrm{b},\mathrm{d})\varphi^{(3)}(\mathrm{c},\mathrm{e}).$$

This graph allows us to quickly see some properties of the distribution. For example, a and c interact directly, but a and e interact only indirectly via c.

### 3.2.4 Observable and Latent Variables

Before going further, let us define two important statistical concepts.

**Observable variables** are variables that can be observed and directly measured.

**Latent (hidden) variables** are variables that are not directly observed but are rather inferred (through a mathematical model) from other variables that are observed (directly measured).

Sometimes latent variables correspond to aspects of physical reality, which could in principle be measured, but may not be for practical reasons. In this situation, the term hidden variables is commonly used (reflecting the fact that the variables are "really there", but hidden). Other times, latent variables correspond to abstract concepts, like categories, behavioral or mental states, or data structures. The terms **hypothetical variables** may be used in these situations.

To distinguish the observable variables from the hidden ones, we will use different background colors of graph vertices. Traditionally, vertices corresponding to observable variables are colored grey and vertices corresponding to latent variables are colored white.

Also, usually a number of variables behave in the same way. To simplify our pictures, we will use a rectangular border over the "repeated" variable with a number of such variables on the corner. 3.3

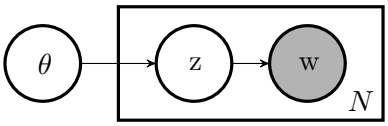



Figure 3.3: In this example $\theta$ is an unknown parameter, $\mathrm{z}_1, \ldots, \mathrm{z}_N$ are hidden (latent) variables, and $\mathrm{w}_1, \ldots, \mathrm{w}_N$ are observable variables.

Given $\theta$, the joint probability for such model can be computes as follows:

$$p(\mathbf{w}, \mathbf{z} \mid \theta) = \prod_{i=1}^{N} p(\mathrm{z}_i \mid \theta) p(\mathrm{w}_i \mid \mathrm{z}_i).$$

## 3.3 EM-algorithm

In this section we will introduce **EM-algorithm**, a powerful method that is widely used in probabilistic models. In general case, we will use $\mathbf{X}$ to denote the observable variables, $\mathbf{Z}$ to denote the latent variables, and $\mathbf{\Theta}$ to denote the model parameters (that we want to estimate).

Assume that we know how to compute a joint distribution over $\mathbf{X}$ and $\mathbf{Z}$ given $\mathbf{\Theta}$: $p(\mathbf{X}, \mathbf{Z} \mid \mathbf{\Theta})$.

Let's write a log-likelihood function as an expectation over $\mathbf{Z}$:

$$\log p(\mathbf{X} \mid \mathbf{\Theta}) = \int q(\mathbf{Z}) \log p(\mathbf{X} \mid \mathbf{\Theta}) d\mathbf{Z}$$

where $q(\mathbf{Z})$ is an arbitrary probability density for $\mathbf{Z}$.

Now that we will transform this formula.

$$\begin{aligned}
\log p(\mathbf{X} \mid \mathbf{\Theta}) &= \int q(\mathbf{Z}) \log p(\mathbf{X} \mid \mathbf{\Theta}) d\mathbf{Z} = \int q(\mathbf{Z}) \log \frac{p(\mathbf{X}, \mathbf{Z} \mid \mathbf{\Theta})}{p(\mathbf{Z} \mid \mathbf{X}, \mathbf{\Theta})} \frac{q(\mathbf{Z})}{q(\mathbf{Z})} d\mathbf{Z} \\
&= \int q(\mathbf{Z}) \log \frac{p(\mathbf{X}, \mathbf{Z} \mid \mathbf{\Theta})}{q(\mathbf{Z})} d\mathbf{Z} + \int q(\mathbf{Z}) \log \frac{q(\mathbf{Z})}{p(\mathbf{Z} \mid \mathbf{X}, \mathbf{\Theta})} d\mathbf{Z} \\
&= \mathscr{L}(q, \mathbf{\Theta}) + D_{\mathrm{KL}}(q(\mathbf{Z}) \| p(\mathbf{Z} \mid \mathbf{X}, \mathbf{\Theta})) \geq \mathscr{L}(q, \mathbf{\Theta}).
\end{aligned}$$

A full log-likelihood function is hard to optimize. Instead of maximizing $\log p(\mathbf{X} \mid \mathbf{\Theta})$ by $\mathbf{\Theta}$, we are going to maximize $\mathscr{L}(q, \mathbf{\Theta})$ by $q, \mathbf{\Theta}$. Notice that $\log p(\mathbf{X} \mid \mathbf{\Theta})$ does not depend on $q$ so we are not limited to choose $q$. Thus we can turn the last inequality into equality by putting $q(\mathbf{Z}) = p(\mathbf{Z} \mid \mathbf{X}, \mathbf{\Theta})$ so that $D_{\mathrm{KL}}(q(\mathbf{Z}) \| p(\mathbf{Z} \mid \mathbf{X}, \mathbf{\Theta})) = 0$.

These ideas lead us to an iterative EM-algorithm.

- **E-step** creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters:

$$q(\mathbf{Z})^{(n+1)} = \arg\max_{q'} \mathscr{L}(q', \mathbf{\Theta}^{(n)}) = p(\mathbf{Z} \mid \mathbf{X}, \mathbf{\Theta}^{(n)}).$$

- **M-step** computes parameters maximizing the expected log-likelihood found on the E-step:

$$\begin{aligned}\boldsymbol{\Theta}^{(n+1)} &= \arg\max_{\boldsymbol{\Theta}'} \mathscr{L}(q^{(n+1)}, \boldsymbol{\Theta}') \\ &= \arg\max_{\boldsymbol{\Theta}'} \int q(\mathbf{Z})^{(n+1)} \log \frac{p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\Theta}')}{q(\mathbf{Z})^{(n+1)}} d\mathbf{Z} \\ &= \arg\max_{\boldsymbol{\Theta}'} \int q(\mathbf{Z})^{(n+1)} \log p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\Theta}') d\mathbf{Z} \\ &= \arg\max_{\boldsymbol{\Theta}'} \mathbb{E}_{\mathbf{Z} \sim q(\mathbf{Z})^{(n+1)}} [\log p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\Theta}')].\end{aligned}$$

In several cases the formulas can be derived analytically.

At that point, all mandatory knowledge have been described which allows us to introduce yet some hidden variable models for topic modeling.

## 3.4 Probabilistic Generative Topic Model

The joint probability of a document $d$ and a word $w$ can be expressed with conditional probability formula as

$$P(d, w) = P(d)P(w \mid d).$$

Now we will define a model with hidden variables $\mathcal{Z} = \{z_1, \ldots, z_T\}$. We assume that each document has its on distribution over (latent) topics, and that each topic has its on distribution over (observable) words. So our goal is to find the matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Theta}$ such that

$$P(w \mid d) = \sum_{z \in \mathcal{Z}} P(w \mid z, d)P(z \mid d) = \sum_{z \in \mathcal{Z}} P(w \mid z)P(z \mid d) = \sum_{z \in \mathcal{Z}} \Phi_{w,z}\Theta_{z,d}.$$

In other words, we want to get a low-rank matrix factorization.

We can write a log-likelihood function for our model:

$$\begin{aligned}\log P(\mathcal{D}, \mathcal{W} \mid \boldsymbol{\Phi}, \boldsymbol{\Theta}) &= \log \prod_{d \in \mathcal{D}} \prod_{w \in \mathcal{W}} P(d, w)^{\#(w,d)} \\ &= \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} \#(w, d) \log P(d, w) \\ &= \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} \#(w, d) \log(P(d)P(w \mid d)) \\ &= \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} \#(w, d) \log \left( P(d) \sum_{z \in \mathcal{Z}} \Phi_{w,z}\Theta_{z,d} \right) \longrightarrow \max_{\boldsymbol{\Phi}, \boldsymbol{\Theta}}.\end{aligned}$$

which is an objective function with restrictions:

$$\sum_{w \in \mathcal{W}} \Phi_{w,z} = 1,\ \Phi_{w,z} \geq 0,\ \sum_{z \in \mathcal{Z}} \Theta_{z,d} = 1,\ \Theta_{z,d} \geq 0.$$

We can also add a non-negative regularizer to the objective function so that it becomes

$$\sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\#(w,d)\log\left(P(d)\sum_{z\in\mathcal{Z}}\Phi_{w,z}\Theta_{z,d}\right)+R(\boldsymbol{\Phi},\boldsymbol{\Theta})\longrightarrow\max_{\boldsymbol{\Phi},\boldsymbol{\Theta}}.$$

If $P(d)$ does not depend on $\boldsymbol{\Phi}$ or $\boldsymbol{\Theta}$, then the problem is equivalent to

$$\sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\#(w,d)\log\sum_{z\in\mathcal{Z}}\Phi_{w,z}\Theta_{z,d}+R(\boldsymbol{\Phi},\boldsymbol{\Theta})\longrightarrow\max_{\boldsymbol{\Phi},\boldsymbol{\Theta}}.$$

Let $R(\boldsymbol{\Phi},\boldsymbol{\Theta})$ be a continuously differentiable function. Then a point $(\boldsymbol{\Phi},\boldsymbol{\Theta})$ of a local extremum can be found as a solution of the equation system:

$$\begin{aligned}
P(z\mid d,w)&\propto\Phi_{w,z}\Theta_{z,d},\\
\Phi_{w,z}&\propto\sum_{d\in\mathcal{D}}\#(w,d)P(z\mid d,w)+\frac{\partial R}{\partial\Phi_{w,z}},\\
\Theta_{z,d}&\propto\sum_{w\in\mathcal{W}}\#(w,d)P(z\mid d,w)+\frac{\partial R}{\partial\Theta_{z,d}}.
\end{aligned}$$

It means that on an E-step we should compute $P(z\mid d,w)$, and on an M-step we should compute $P(w\mid z)=\Phi_{w,z}$ and $P(z\mid d)=\Theta_{z,d}$ according to the formulas obtained.

## 3.5 PLSA

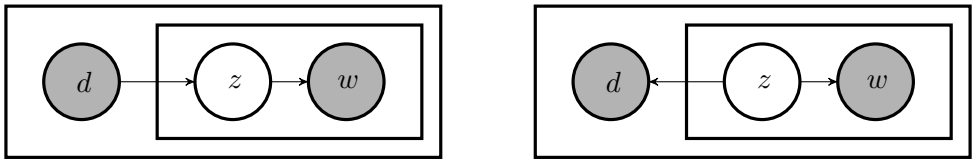
**Probabilistic latent semantic analysis (PLSA)**, proposed by Thomas Hofmann in 1999, was the first technique in history for probabilistic topic modeling. PLSA uses a statistical model which has been called aspect model. The aspect model is a latent variable model for co-occurrence data which associates an unobserved class variable $z\in\mathcal{Z}=\{z_1,\ldots,z_T\}$ with each observation. A joint probability model over $\mathcal{D}\times\mathcal{W}$ is defined by the mixture

$$P(d,w)=P(d)P(w\mid d),\ P(w\mid d)=\sum_{z\in\mathcal{Z}}P(w\mid z)P(z\mid d).$$

Like virtually all statistical latent variable models the aspect model introduces a conditional independence assumption, namely that $d$ and $w$ are independent conditioned on the state of the associated latent variable (the corresponding graphical model representation is depicted in Figure 3.4(a). Since the cardinality of $z$ is smaller than the number of documents/words in the collection, $z$ acts as a bottleneck variable in predicting words. It is worth noticing that the model can be equivalently parameterized by (cf. 3.4(b)):

$$P(d,w)=\sum_{z\in\mathcal{Z}}P(z)P(d\mid z)P(w\mid z)$$

which is perfectly symmetric in both entities, documents and words.




(a) non-symmetrical (b) symmetrical

Figure 3.4: Two versions of the aspect model.

Aspect model uses EM algorithm for maximum likelihood estimation.

- **E-step**. For each $z \in \mathcal{Z}$, $d \in \mathcal{D}$, $w \in \mathcal{W}$ compute

$$P(z \mid d, w) = \frac{P(d, z, w)}{P(d, w)} = \frac{P(z)P(d \mid z)P(w \mid z)}{\sum\limits_{z' \in \mathcal{Z}} P(z')P(d \mid z')P(w \mid z')}.$$

- **M-step** For each $z \in \mathcal{Z}$, $d \in \mathcal{D}$, $w \in \mathcal{W}$ compute

$$\begin{aligned} P(w \mid z) &\propto \sum_{d \in \mathcal{D}} \#(w, d) P(z \mid d, w), \\ P(d \mid z) &\propto \sum_{w \in \mathcal{W}} \#(w, d) P(z \mid d, w), \\ P(z) &\propto \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} \#(w, d) P(z \mid d, w). \end{aligned}$$

To clarify the relation to LSA, let us rewrite the aspect model as parameterized in matrix notation. Hence define matrices by $\hat{\boldsymbol{U}} : \hat{U}_{i,k} = P(w^{(i)} \mid z_k)$, $\hat{\boldsymbol{\Sigma}} : \hat{\Sigma}_{k,k} = P(z_k)$, $\hat{\boldsymbol{V}}^\top : \hat{V}^\top_{k,j} = P(d^{(j)} \mid z_k)$. The joint probability model $\boldsymbol{P}$ can then be written as a matrix product $\boldsymbol{P} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^\top$. Comparing this with SVD, one can make the following observations: (i) outer products between rows of $\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{V}}^\top$ reflect conditional independence in PLSA, (ii) the $K$ factors correspond to the mixture components in the aspect model, (iii) the mixing proportions in PLSA substitute the singular values. The crucial difference between PLSA and LSA, however, is the objective function utilized to determine the optimal decomposition/approximation. In LSA, this is the $L_2$ or Frobenius norm, which corresponds to an implicit additive Gaussian noise assumption on (possibly transformed) counts. In contrast, PLSA relies on the likelihood function of multinomial sampling and aims at an explicit maximization of the predictive power of the model.

## 3.6 LDA

**Latent Dirichlet allocation (LDA)** is a topic modeling algorithm based on a generative graphical model. It was firstly introduced in 2003 by David Blei, Andrew Ng and Michael I. Jordan in Journal of Machine Learning Research. In this section, we will consider the simplest version of LDA. For more complicated models, SEE SOMETHING. The concept of LDA model is to consider all documents as composed of set of topics or clusters of words and to observe the distribution of them.

More formally, we have a document collection $\mathcal{D}$ and a vocabulary $\mathcal{W}$. Each document $d \in \mathcal{D}$ consists of $N_d$ words $w_{d,1}, \ldots, w_{d,N_d}$. The probability distributions over topics $\mathcal{Z}$ are sampled for each document $d \in \mathcal{D}$ from a Dirichlet distribution with parameter $\boldsymbol{\alpha}$. Thus, $\Theta_{:,d} \sim Dir(\boldsymbol{\alpha})$. For each topic $z \in \mathcal{Z}$ we generate a probability distribution over all words $\mathcal{W}$ $\Phi_{:,z}$. After that, for each document $d$ we generate $N_d$ topics $z_{d,1}, \ldots, z_{d,N_d}$ from distribution $\Theta_{:,d}$, and for each of these topics $z_{d,k}$ generate per a single word from distribution $\Phi_{:,z_{d,k}}$.

### 3.6.1 Dirichlet Distribution

A $K$-dimensional Dirichlet random variable $\boldsymbol{\theta}$ can take values in the $(K-1)$-simplex (a $K$-vector $\boldsymbol{\theta}$ lies in the $(K-1)$-simplex if $\theta_k \geq 0$ and $\sum_{k=1}^{K} \theta_k = 1$), and has the following probability density on this simplex:

$$p(\boldsymbol{\theta} \mid \boldsymbol{\alpha}) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$

where the parameter $\boldsymbol{\alpha}$ is a $K$-vector with components $\alpha_k > 0$, and where $\Gamma(x)$ is the Gamma function.

The Dirichlet is a convenient distribution on the simplex—it is in the exponential family, has finite dimensional sufficient statistics, and is conjugate to the multinomial distribution. These properties facilitate the development of inference and parameter estimation algorithms for LDA.

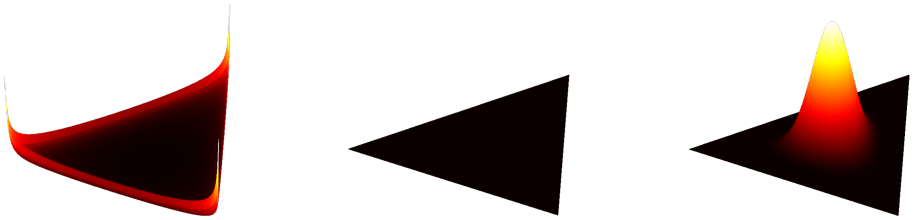


Figure 3.5: Dirichlet distribution plots for $\boldsymbol{\alpha} = (0.75, 0.75, 0.75), (1, 1, 1), (10, 10, 10)$

LDA posits that each word of both the observed and unseen documents is generated by a randomly chosen topic which is drawn from a distribution with a randomly chosen parameter. This parameter is sampled once per document from a smooth distribution on the topic simplex.

### 3.6.2 LDA Model

Consider the following graphical model.

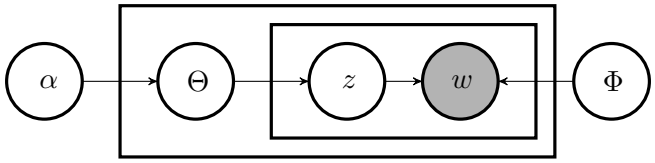



Figure 3.6: A graphical model for LDA.

In this model, $\boldsymbol{\alpha}$ and $\boldsymbol{\Phi}$ are parameters. $\boldsymbol{\Phi}$ is a matrix of size $|\mathcal{W}| \times |\mathcal{Z}|$, $\boldsymbol{\Theta}$ is a matrix of size $|\mathcal{Z}| \times |\mathcal{D}|$, $\boldsymbol{\alpha}$ is a vector of size $|\mathcal{Z}|$. Moreover, $\boldsymbol{\Theta}_{:,d} \sim Dir(\boldsymbol{\alpha})$.

The joint probability of corpora $(\mathcal{D}, \mathcal{W})$ and latent variables $\mathcal{Z}$ and $\boldsymbol{\Theta}$ can be expressed with the formula below.

$$
\begin{aligned}
P(\mathcal{D}, \mathcal{W}, \mathcal{Z}, \boldsymbol{\Theta} \mid \boldsymbol{\Phi}, \boldsymbol{\alpha}) &= \prod_{d=1}^{|\mathcal{D}|} P(\boldsymbol{\Theta}_{:,d} \mid \boldsymbol{\alpha}) \prod_{n=1}^{N_d} P(z_{d,n} \mid \boldsymbol{\Theta}_{:,d}) P(w_{d,n} \mid z_{d,n}, \boldsymbol{\Phi}) \\
&= \prod_{d=1}^{|\mathcal{D}|} \frac{\Gamma(\sum_{t=1}^{|\mathcal{Z}|} \alpha_t)}{\prod_{t=1}^{|\mathcal{Z}|} \Gamma(\alpha_t)} \prod_{t=1}^{|\mathcal{Z}|} \Theta_{t,d}^{\alpha_t - 1} \prod_{n=1}^{N_d} \Theta_{z_{d,n},d} \Phi_{w_{d,n},z_{d,n}} \\
&= \prod_{d=1}^{|\mathcal{D}|} \frac{\Gamma(\sum_{t=1}^{|\mathcal{Z}|} \alpha_t)}{\prod_{t=1}^{|\mathcal{Z}|} \Gamma(\alpha_t)} \prod_{t=1}^{|\mathcal{Z}|} \Theta_{t,d}^{\alpha_t - 1} \prod_{n=1}^{N_d} \prod_{t=1}^{|\mathcal{Z}|} \Theta_{t,d}^{[z_{d,n}=t]} \Phi_{w_{d,n},t}^{[z_{d,n}=t]}.
\end{aligned}
$$

Now we will take the logarithm of it.

$$
\log P(\mathcal{D}, \mathcal{W}, \mathcal{Z}, \boldsymbol{\Theta} \mid \boldsymbol{\Phi}, \boldsymbol{\alpha}) = \sum_{d=1}^{|\mathcal{D}|} \left[ \sum_{t=1}^{|\mathcal{Z}|} (\alpha_t - 1) \log \Theta_{t,d} + \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} [z_{d,n} = t] (\log \Theta_{t,d} + \log \Phi_{w_{d,n},t}) \right] + const.
$$

After that, we should apply EM algorithm.

- **E-step**.

  $$
  P(\boldsymbol{\Theta}, \mathcal{Z} \mid \mathcal{D}, \mathcal{W}, \boldsymbol{\Phi}, \boldsymbol{\alpha}) \approx q(\boldsymbol{\Theta}) q(\mathcal{Z}) = \arg \min_{q(\boldsymbol{\Theta}), q(\mathcal{Z})} D_{\mathrm{KL}}(q(\boldsymbol{\Theta}) q(\mathcal{Z}) || P(\boldsymbol{\Theta}, \mathcal{Z} \mid \mathcal{D}, \mathcal{W}, \boldsymbol{\Phi}, \boldsymbol{\alpha})).
  $$

- **M-step**.

  $$
  \boldsymbol{\Phi} = \arg \max_{\boldsymbol{\Phi}} \mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta}), \mathcal{Z} \sim q(\mathcal{Z})} \log P(\boldsymbol{\Theta}, \mathcal{Z} \mid \mathcal{D}, \mathcal{W}, \boldsymbol{\Phi}, \boldsymbol{\alpha}).
  $$

Here we are using mean field approximation. See ADVANCED BOOK ON BAYESIAN METHODS IF IT'S NOT ENOUGH FOR YOU.

$$
\begin{aligned}
\log q(\boldsymbol{\Theta}) &= \mathbb{E}_{\mathcal{Z} \sim q(\mathcal{Z})} \log P(\boldsymbol{\Theta}, \mathcal{Z} \mid \mathcal{D}, \mathcal{W}, \boldsymbol{\Phi}, \boldsymbol{\alpha}) + const \\
&= \mathbb{E}_{\mathcal{Z} \sim q(\mathcal{Z})} \log \frac{P(\boldsymbol{\Theta}, \mathcal{Z}, \mathcal{D}, \mathcal{W} \mid \boldsymbol{\Phi}, \boldsymbol{\alpha})}{P(\mathcal{D}, \mathcal{W} \mid \boldsymbol{\Phi}, \boldsymbol{\alpha})} + const \\
&= \mathbb{E}_{\mathcal{Z} \sim q(\mathcal{Z})} \log P(\boldsymbol{\Theta}, \mathcal{Z}, \mathcal{D}, \mathcal{W} \mid \boldsymbol{\Phi}, \boldsymbol{\alpha}) + const \\
&= \mathbb{E}_{\mathcal{Z} \sim q(\mathcal{Z})} \sum_{d=1}^{|\mathcal{D}|} \left[ \sum_{t=1}^{|\mathcal{Z}|} (\alpha_t - 1) \log \Theta_{t,d} + \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} [z_{d,n} = t] (\log \Theta_{t,d} + \log \Phi_{w_{d,n},t}) \right] + const \\
&= \mathbb{E}_{\mathcal{Z} \sim q(\mathcal{Z})} \sum_{d=1}^{|\mathcal{D}|} \left[ \sum_{t=1}^{|\mathcal{Z}|} (\alpha_t - 1) \log \Theta_{t,d} + \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} [z_{d,n} = t] \log \Theta_{t,d} \right] + const \\
&= \sum_{d=1}^{|\mathcal{D}|} \left[ \sum_{t=1}^{|\mathcal{Z}|} (\alpha_t - 1) \log \Theta_{t,d} + \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} \mathbb{E}_{\mathcal{Z} \sim q(\mathcal{Z})} [z_{d,n} = t] \log \Theta_{t,d} \right] + const \\
&= \sum_{d=1}^{|\mathcal{D}|} \left[ \sum_{t=1}^{|\mathcal{Z}|} (\alpha_t - 1) \log \Theta_{t,d} + \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} \gamma_{d,n}^t \log \Theta_{t,d} \right] + const \\
&= \sum_{d=1}^{|\mathcal{D}|} \left[ \sum_{t=1}^{|\mathcal{Z}|} (\alpha_t - 1) + \sum_{n=1}^{N_d} \gamma_{d,n}^t \right] \log \Theta_{t,d} + const.
\end{aligned}
$$

Thus,

$$q(\boldsymbol{\Theta}) = const \prod_{d=1}^{|\mathcal{D}|} \prod_{t=1}^{|\mathcal{Z}|} \Theta_{t,d}^{\alpha_t + \sum\limits_{n=1}^{N_d} \gamma_{d,n}^t - 1} = \prod_{d=1}^{|\mathcal{D}|} q(\boldsymbol{\Theta}_{:,d}),\ q(\boldsymbol{\Theta}_{:,d}) \sim Dir(\boldsymbol{\alpha} + \sum_{n=1}^{N_d} \boldsymbol{\gamma}_{d,n}),$$

where

$$\gamma_{d,n}^t = \mathbb{E}_{z_{d,n} \sim q(z_{d,n})}[z_{d,n} = t].$$

What does *const* equal to here?

These formulas allow us to compute $q(\boldsymbol{\Theta})$ on the E-step. Similarly, we compute $q(\mathcal{Z})$.

$$\begin{aligned}
\log q(\mathcal{Z}) &= \mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta})} \log P(\boldsymbol{\Theta}, \mathcal{Z}, \mathcal{D}, \mathcal{W} \mid \boldsymbol{\Phi}, \boldsymbol{\alpha}) + const \\
&= \mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta})} \sum_{d=1}^{|\mathcal{D}|} \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} [z_{d,n} = t] \left( \log \Theta_{t,d} + \log \Phi_{w_{d,n},t} \right) + const \\
&= \sum_{d=1}^{|\mathcal{D}|} \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} [z_{d,n} = t] \left( \mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta})} \log \Theta_{t,d} + \log \Phi_{w_{d,n},t} \right) + const.
\end{aligned}$$

And we can write

$$q(\mathcal{Z}) = \prod_{d=1}^{|\mathcal{D}|} \prod_{n=1}^{N_d} q(z_{d,n}),\ q(z_{d,n} = t) = \frac{\Phi_{w_{d,n},t} \exp(\mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta})} \log \Theta_{t,d})}{\sum_{t'=1}^{|\mathcal{Z}|} \Phi_{w_{d,n},t'} \exp(\mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta})} \log \Theta_{t',d})} = \gamma_{d,n}^t.$$

E-step is finished.

On the M-step, we need to construct a Lagrange function because of restrictions for all $t \in \{1, \ldots, |\mathcal{Z}|\} \sum\limits_{w \in \mathcal{W}} \Phi_{w,t} = 1$.

$$\begin{aligned}
L &= \mathbb{E}_{\boldsymbol{\Theta} \sim q(\boldsymbol{\Theta}), \mathcal{Z} \sim q(\mathcal{Z})} \sum_{d=1}^{|\mathcal{D}|} \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} [z_{d,n} = t] \log \Phi_{w_{d,n},t} + \sum_{t=1}^{|\mathcal{Z}|} \lambda_t \left( \sum_{w=1}^{|\mathcal{W}|} \Phi_{w,t} - 1 \right) \\
&= \sum_{d=1}^{|\mathcal{D}|} \sum_{n=1}^{N_d} \sum_{t=1}^{|\mathcal{Z}|} \gamma_{d,n}^t \log \Phi_{w_{d,n},t} + \sum_{t=1}^{|\mathcal{Z}|} \lambda_t \left( \sum_{w=1}^{|\mathcal{W}|} \Phi_{w,t} - 1 \right).
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial \Phi_{\omega,\tau}} &= \sum_{d=1}^{|\mathcal{D}|} \sum_{n=1}^{N_d} \gamma_{d,n}^{\tau} \frac{1}{\Phi_{\omega,\tau}} [w_{d,n} = \omega] + \lambda_\tau = 0 \Rightarrow \\
\Phi_{\omega,\tau} &= \frac{\sum\limits_{d=1}^{|\mathcal{D}|} \sum\limits_{n=1}^{N_d} \gamma_{d,n}^{\tau} [w_{d,n} = \omega]}{-\lambda_\tau}.
\end{aligned}$$

Summing over $\omega' \in \mathcal{W}$, we will get

$$-\lambda_\tau = \sum_{\omega' \in \mathcal{W}} \sum_{d=1}^{|\mathcal{D}|} \sum_{n=1}^{N_d} \gamma_{d,n}^\tau [w_{d,n} = \omega'].$$

Thus

$$\Phi_{\omega,\tau} = \frac{\sum\limits_{d=1}^{|\mathcal{D}|} \sum\limits_{n=1}^{N_d} \gamma_{d,n}^\tau [w_{d,n} = \omega]}{\sum\limits_{\omega' \in \mathcal{W}} \sum\limits_{d=1}^{|\mathcal{D}|} \sum\limits_{n=1}^{N_d} \gamma_{d,n}^\tau [w_{d,n} = \omega']}.$$

And that means that we have formulas to update $\mathbf{\Phi}$ on the M-step.

It was the simplest version of LDA. The more complicated model assumes that the rows of $\mathbf{\Phi}$ are sampled from another Dirichlet distribution so that

$$\mathbf{\Phi}_{:,t} \sim Dir(\boldsymbol{\beta}).$$

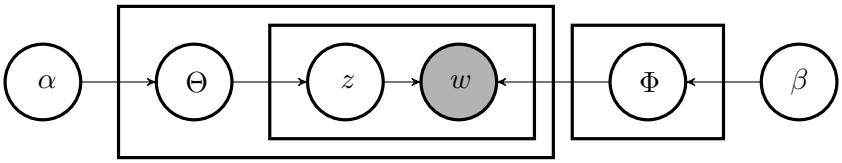



Figure 3.7: More complicated graphical model for LDA.

## 3.7 BigARTM

TODO

## 3.8 Seminar

- Practical application of topic models. Clusterization.
- Hands-on tutorial on LDA (gensim). Visualization.
- **Homework:** PLSA and LDA for topic modeling on Hillary Clinton emails.
- Hands-on tutorial on BigARTM.

# Bibliography

[1] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine learning research*, 3(Jan):993–1022.

# Chapter 4

# Prediction-based Distributional Models

**Keywords:** prediction-based models, neural-based models, neural networks, feed-forward network, hierarchical softmax, negative sampling, continuous bag-of-words, skip-gram, Word2Vec, dense vectors, word embeddings.

## 4.1 All the Buzz about Neural

So, as we have mentioned before, another big class of distributional semantics models are called prediction-based models. They are also called neural-based. This simply means that such models are implying on neural networks that predict word co-occurrence. What are neural networks and how do they predict? In this paragraph we will try to briefly explain this (and if you are familiar with this topic, feel free to skip it)!

You must live on another planet, if you have never heard about neural networks. Neural networks (or, more precisely, artificial neural networks) currently used almost in every aspect of software products: video recommendation, spam filtering, loan risk prediction, and so on. And distributional semantics is not an exception! But how do these networks work?

As the word "neural" suggests, these systems are inspired by our brain,—or, more precisely, biological systems that responsible of the way that we humans learn. Such systems are called **"neural networks"**, and computational models that replicate such systems are called **"artificial neural networks"**.

The very basic idea of these biological systems is that we try to process unstructured information, find patterns in this information and receive feedback of how successful was this finding. If we have found pattern correctly, we proceed. If not, we try to change something in our behavior, and then try again. And by changing behavior we mean tuning the parameters of our biological system. This process is called learning: we try to adapt our neurons to the situation, trying to get best feedback.

The same situation is in artificial neural networks. Such model consists of several layers, like a cake. Every layer has a list of neurons every of which is presented by some mathematical function (the same for each neuron in the layer) and a some scalar parameter (weight of neuron). Every network has an input layer (from which information comes to network) and an output layer (through which we receive feedback). Between input and output layer there could be a numerous amount of other layers. Their purpose is to receive information, put it into function, add their weight and receive modified piece of information. By doing

such operations the neural network tries to find patterns in the information by extracting different features until it can recognize what type of data it is processing.

Then, after iterations through all the layers, we receive some output. And we try to compare this output with desired. By comparing we mean finding degree of similarity of received output and desired one (in other word, we find a function of similarity, it is called **loss function**). If received output differs very much, we propagate the information back through the neural network (such process is called back-propagation), using the coefficient of similarity and updating weights of neurons on each layer. This allows networks to adjust their hidden layers of neurons in situations where the outcome doesn't match what the creator is hoping for—like a network designed to recognize dogs, which misidentifies a cat, for example.

For a basic idea of how a deep learning neural network learns, imagine a factory line. After the raw materials (the data set) are input, they are then passed down the conveyer belt, with each subsequent stop or layer extracting a different set of high-level features. If the network is intended to recognize an object, the first layer might analyze the brightness of its pixels. The next layer could then identify any edges in the image, based on lines of similar pixels. After this, another layer may recognize textures and shapes, and so on. By the time the fourth or fifth layer is reached, the deep learning net will have created complex feature detectors. It can figure out that certain image elements (such as a pair of eyes, a nose, and a mouth) are commonly found together. Once this is done, the researchers who have trained the network can give labels to the output, and then use backpropagation to correct any mistakes which have been made. After a while, the network can carry out its own classification tasks without needing humans to help every time.

So, this is simply how neural networks work. There is a big variety of different networks architectures – convolutional neural networks, recurrent neural networks, etc. All of them have more complicated features and tricks, but the basic idea is the same. The good thing is that all you need to know to understand how neural-based distributional semantic models work: the models considered in this paragraph are based on very simple three-layer neural networks. Despite this they are very effective, though! Let's continue and understand how can we connect neural networks and distributional semantics.

## 4.2 Doing the Prediction

So, imagine a neural network that does certain task. For example, predicts part-of-speech tags. The idea is that we have a set of possible tags (like "noun", "adjective", etc) and we need to associate tag to each word. We can try to put each word one-by-one in the network and predict tag for it. It will looks like we are putting the word, receive tag on the output, compare it with desired tag and update weights if it is not the same.

Of course, words and tags are unstuctured information, so we need to somehow present them in a computational form. In case with tags we can use numerical features. What should we do with words?

We can also try to use numerical features: representation of each word would be its index in vocabulary. Such approach also could take place, but indices are usually replaced with vectors where all components are zeros and one component is one. Such representations are called one-hot encodings.

Thus, we can represent words with one-hot encodings, and predict pos-tags, and update weights of neurons on hidden layers. Conventionally, supervised lexicalized NLP approaches take a word and convert it to a symbolic ID, which is then transformed into a feature vector

using a one-hot representation: the feature vector has the same length as the size of the vocabulary, and only one dimension is on. However, the **one-hot representation** of a word suffers from data sparsity: namely, for words that are rare in the labeled training data, their corresponding model parameters will be poorly estimated. Moreover, at test time, the model cannot handle words that do not appear in the labeled training data. These limitations of one-hot word representations have prompted researchers to investigate unsupervised methods for inducing word representations over large unlabeled corpora. Word features can be hand-designed, but our goal is to learn them.

One common approach to inducing unsupervised word representation is to use clustering, perhaps hierarchical. This leads to a one-hot representation over a smaller vocabulary size. Neural language models, on the other hand, induce dense real-valued low-dimensional word embeddings using unsupervised approaches.

We can try to use only one, input layer! We will be updating weights only on one layer. You remember, on the start of initialization we have network with random weights. The same would be with vocabulary. We can represent each word with a random vector and then tune its component to make it possible to predict POS-tag. And this is how neural network work! We have network that is doing some tasks, and word representation obtained by learning it are called **word embeddings**!

However, task of POS-tagging is not very suitable. Therefore, many models rely on word prediction task. The idea is to predict how it is likely to encounter word in a given context. This is called language modeling. Thus, a statistical model of language can be represented by the conditional probability of the next word given all the previous ones, since

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t \mid w_1^{t-1})$$

where $w_t$ is the $t$-th word, and writing sub-sequence $w_i^j = (w_i, w_{i+1}, \ldots, w_{j-1}, w_j)$.

Such statistical language models have already been found useful in many technological applications involving natural language, such as speech recognition, language translation, and information retrieval.

When building statistical models of natural language, one considerably reduces the difficulty of this modeling problem by taking advantage of word order, and the fact that temporally closer words in the word sequence are statistically more dependent. Thus, $n$-gram models construct tables of conditional probabilities for the next word, for each one of a large number of contexts, i. e. combinations of the last $n-1$ words:

$$\hat{P}(w_t \mid w_1^{t-1}) \approx \hat{P}(w_t \mid w_{t-n+1}^{t-1}).$$

The training objective follows the distributional hypothesis by trying to maximize the dot-product between the vectors of frequently occurring word-context pairs, and minimize it for random word-context pairs.

## 4.3 History

To understand how much today prediction-based model evolved we will briefly overview historical approaches.

Usually, the term word embeddings in previous literature was also used interchangeably with term "distributed representations" (Not to be confused with distributional representations). A distributed representation is dense, lowdimensional, and real-valued. Distributed

word representations are called word embeddings. Each dimension of the embedding represents a latent feature of the word, hopefully capturing useful syntactic and semantic properties. A distributed representation is compact, in the sense that it can represent an exponential number of clusters in the number of dimensions.

### 4.3.1 Language Modeling

TODO

### 4.3.2 Bengio Model

SOME WORDS ABOUT ITS MAIN IDEA: TO USE WORD EMBEDDINGS (INSTEAD OF STATISTICAL APPROACHES) IN LANGUAGE MODELING, AND HISTORY (2003).

The training set is a sequence $w_1, \ldots, w_T$ of words $w_t \in \mathcal{W}$, where the vocabulary $\mathcal{W}$ is a large but finite set. The objective is to learn a good model $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_1^{t-1})$, in the sense that it gives high out-of-sample likelihood. Below, we report the geometric average of $1/\hat{P}(w_t \mid w_1^{t-1})$, also known as perplexity, which is also the exponential of the average negative log-likelihood. The only constraint on the model is that for any choice of $w_1^{t-1}$, $\sum_{w \in \mathcal{W}} f(w, w_{t-1}, \ldots, w_{t-n+1}) = 1$, with $f > 0$. By the product of these conditional probabilities, one obtains a model of the joint probability of sequences of words.

We decompose the function $f(w_t, w_{t-1}, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_1^{t-1})$ in two parts:

1. A mapping $C$ from any element $w$ of $\mathcal{W}$ to a real vector $C(w) \in \mathbb{R}^d$. It represents the distributed feature vectors associated with each word in the vocabulary. In practice, $C$ is represented by a $|\mathcal{W}| \times d$ matrix of free parameters.

2. The probability function over words, expressed with $C$: a function $g$ maps an input sequence of feature vectors for words in context, $(C(w_{t-n+1}), \ldots, C(w_{t-1}))$, to a conditional probability distribution over words in $\mathcal{W}$ for the next word $w_t$. The output of $g$ is a vector whose $i$-th element estimates the probability $\hat{P}(w_t = w^{(i)} \mid w_1^{t-1})$:

$$f(w^{(i)}, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1})).$$

The function $f$ is a composition of these two mappings ($C$ and $g$), with $C$ being shared across all the words in the context. With each of these two parts are associated some parameters. The parameters of the mapping $C$ are simply the feature vectors themselves, represented by a $|\mathcal{W}| \times m$ matrix $\boldsymbol{C}$ whose row $i$ is the feature vector $C(w^{(i)})$ for word $w^{(i)}$. The function $g$ may be implemented by a feed-forward or recurrent neural network or another parametrized function, with parameters $\boldsymbol{\omega}$. The overall parameter set is $\boldsymbol{\theta} = (\boldsymbol{C}, \boldsymbol{\omega})$.

Training is achieved by looking for $\boldsymbol{\theta}$ that maximizes the training corpus penalized log-likelihood:

$$L = \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \ldots, w_{t-n+1} \mid \boldsymbol{\theta}) + R(\boldsymbol{\theta}),$$

where $R(\boldsymbol{\theta})$ is a regularization term.

The neural network computes the following function, with a softmax output layer, which guarantees positive probabilities summing to 1:

$$\hat{P}(w_t \mid w_1^{t-1}) = \frac{\exp y_{w_t}}{\sum_{w \in \mathcal{W}} \exp y_w}$$

The $y_w$ are the unnormalized log-probabilities for each output word $w$, computed as follows, with parameters $\boldsymbol{b}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{d}$ and $\boldsymbol{H}$:

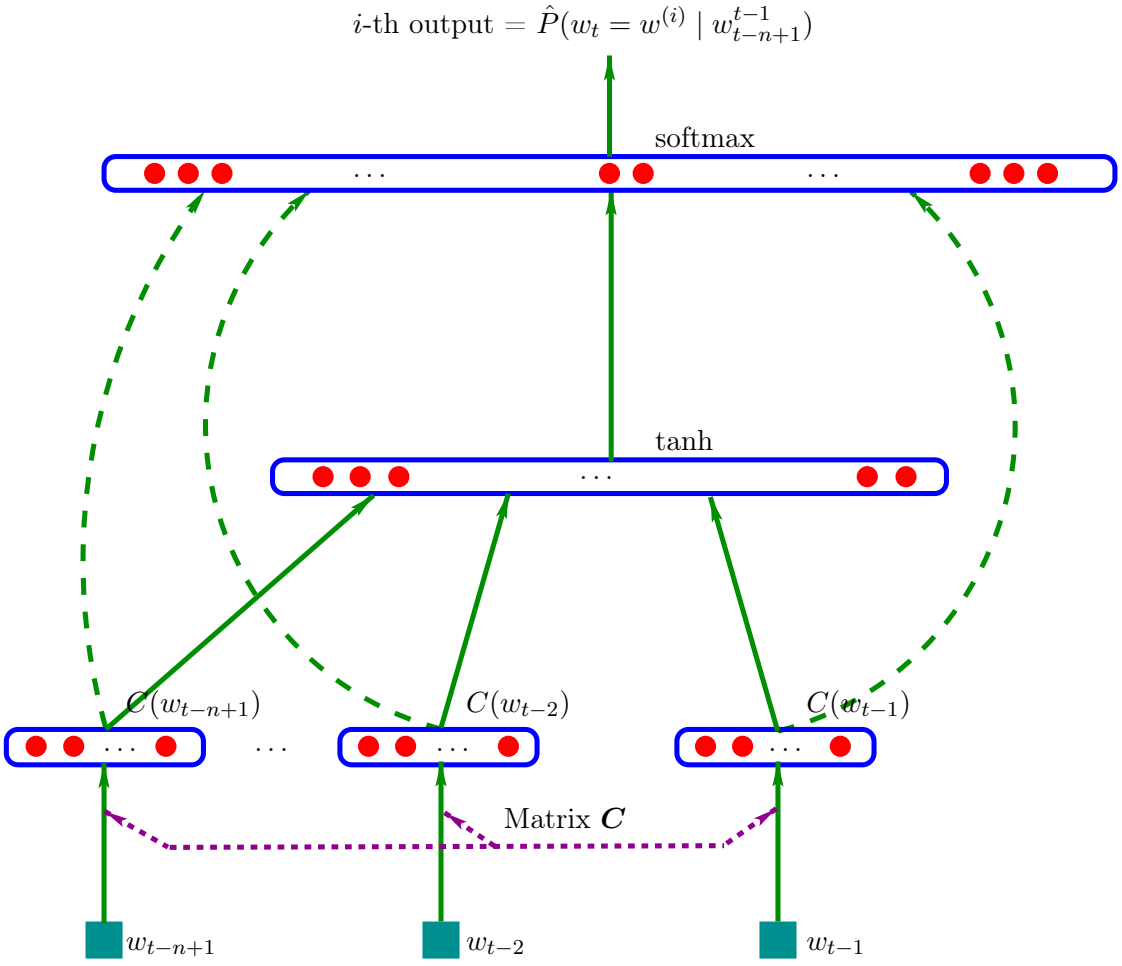
$$\boldsymbol{y} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{x} + \boldsymbol{U} \tanh(\boldsymbol{d} + \boldsymbol{H}\boldsymbol{x})$$

where the hyperbolic tangent tanh is applied element by element, $\boldsymbol{W}$ is optionally zero (no direct connections), and $\boldsymbol{x}$ is the word features layer activation vector, which is the concatenation of the input word features from the matrix $\boldsymbol{C}$:

$$\boldsymbol{x} = [C(w_{t-1}), C(w_{t-2}), \ldots, C(w_{t-n+1})].$$

In this model, $\boldsymbol{\theta} = (\boldsymbol{C}, \boldsymbol{b}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{d}, \boldsymbol{H})$.

Let $h$ be the number of hidden units, and $m$ the number of features associated with each word. How many trainable parameters are in $\boldsymbol{\theta}$?




Stochastic gradient ascent on the neural network consists in performing the following iterative update after presenting the $t$-th word of the training corpus:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \varepsilon \frac{\partial \log \hat{P}(w_t \mid w_1^{t-1})}{\partial \boldsymbol{\theta}},$$

where $\varepsilon$ is the "learning rate".

### 4.3.3 SENNA

Collobert and Weston in 2008 presented a neural language model that could be trained over billions of words, because the gradient of the loss was computed stochastically over a small sample of possible outputs, in a spirit similar to Bengio.

The model is discriminative and nonprobabilistic. For each training update, we read an $n$-gram $x = (w_1, \ldots, w_n)$ from the corpus. The model concatenates the learned embeddings of the $n$ words, giving $[C(w_1), \ldots, C(w_n)]$, where $C$ is the lookup table.

$C$ here is $e$!

MORE HARDCORE MATH!

TODO

### 4.3.4 HLBL

TODO

The log-bilinear model is a probabilistic and linear neural model. Given an n-gram, the model concatenates the embeddings of the $n-1$ first words, and learns a linear modelto predict the embedding of the last word. The similarity between the predicted embedding and the current actual embedding is transformed into a probability by exponentiating and then normalizing. Mnih and Hinton speed up model evaluation during training and testing by using a hierarchy to exponentially filter down the number of computations that are performed. This hierarchical evaluation technique was first proposed by Morin and Bengio. The model, combined with this optimization, is called the hierarchical log-bilinear (HLBL) model.

## 4.4 Word2Vec

TODO

Idea is very similar. We are now dealing with two architectures: Skip-Gram and Continuous Bag-of-Words.

The following example demonstrates multiple pairs of target and context words as training samples, generated by a 5-word window sliding along the sentence: *The man who passes the sentence should swing the sword.*

Each context-target pair is treated as a new observation in the data. For example, the target word "swing" in the above case produces four training samples: ("swing", "sentence"), ("swing", "should"), ("swing", "the"), and ("swing", "sword").

### 4.4.1 Skip-gram Model

TODO

### 4.4.2 Continuous Bag-of-Words Model

TODO

### 4.4.3 Negative Sampling

TODO

### 4.4.4 Hierarchical Softmax

TODO

### 4.4.5 Subsampling

TODO

The skip-gram model defines the embedding vector of every word by the matrix $w$ and the context vector by the output matrix $W'$. Given an input word $w_I$, let us label the corresponding row of $W$ as vector $v_{w_I}$ (embedding vector) and its corresponding column of $W'$ as $v'_{wI}$ (context vector). The final output layer applies softmax to compute the probability of predicting the output word $w_O$ given $w_I$, and therefore:

$$p(w_O|w_I) = \frac{exp({v'_{w_O}}^\top v_{w_I})}{\sum_{i=1}^{V} exp({v'_{w_i}}^\top v_{w_I})}$$

However, when $V$ is extremely large, calculating the denominator by going through all the words for every single sample is computationally impractical. The demand for more efficient conditional probability estimation leads to the new methods like hierarchical softmax.

Morin and Bengio (2005) proposed hierarchical softmax to make the sum calculation faster with the help of a binary tree structure. The hierarchical softmax encodes the language model's output softmax layer into a tree hierarchy, where each leaf is one word and each internal node stands for relative probabilities of the children nodes.

Each word $w_i$ has a unique path from the root down to its corresponding leaf. The probability of picking this word is equivalent to the probability of taking this path from the root down through the tree branches. Since we know the embedding vector $v_n$ of the internal node $n$, the probability of getting the word can be computed by the product of taking left or right turn at every internal node stop. The probability of one node is ($\delta$ is the sigmoid function):

$$p(turnright...w_I|n) = \delta({v'_{w_O}}^\top v_{w_i})$$

$$p(turnleft...w_I|n) = 1 - p(turnright...w_I|n) = \delta(-{v'_{w_O}}^\top v_{w_i}) =$$

The final probability of getting a context word $w_O$ given an input word $w_I$ is:

$$p(w_O|w_I) = \Pi_{k=1}^{L(W_O)} \delta I_{turn}(n(w_O,k), n(w_O,k+1)) {v'_{n(w_O,k)}}^\top v_{w_I})$$

,

where $L(w_O)$ is the depth of the path leading to the word $w_O$ and $I_{turn}$ is a specially indicator function which returns 1 if $n(w_O, k+1)$ is the left child of $n(w_O, k)$ otherwise -1. The internal nodes' embeddings are learned during the model training. The tree structure helps greatly reduce the complexity of the denominator estimation from $O(V)$ (vocabulary size) to $O(logV)$ (the depth of the tree) at the training time. However, at the prediction time, we still to compute the probability of every word and pick the best, as we don't know which leaf to reach for in advance.

A good tree structure is crucial to the model performance. Several handy principles are: group words by frequency like what is implemented by Huffman tree for simple speedup; group similar words into same or close branches (i.e. use predefined word clusters, WordNet).

The Negative Sampling (NEG) proposed by Mikolov et al. (2013) is a simplified variation of NCE loss. It is especially famous for training Google's word2vec project. Different from NCE Loss which attempts to approximately maximize the log probability of the softmax output, negative sampling did further simplification because it focuses on learning high-quality word embedding rather than modeling the word distribution in natural language.

NEG approximates the binary classifier's output with sigmoid functions as follows:

$$p(d = 1|w, w_I) = \delta({v'_w}^\top v_{w_I})$$

$$p(d = 0|w, w_I) = 1 - \delta({v'_w}^\top v_{w_I}) = \delta(-{v'_w}^\top v_{w_I})$$

The final NCE loss function looks like:

$$L\theta = -[log(v'_w \overset{\sum}{v} w_I) + \sum_{i=1}^{N} log\delta(-{v'_w}^\top v_{w_I})]$$

## 4.5 Loss functions

### 4.5.1 Cross-entropy

### 4.5.2 Noise Contrastive Estimation

## 4.6 Connection to Count-based Model

Levy casted SGNS's training method as weighted matrix factorization, and shown that its objective is implicitly factorizing a shifted PMI matrix—the well-known word-context PMI matrix from the word-similarity literature, shifted by a constant offset. In this subsection, we will prove that fact.

### 4.6.1 SGNS's Objective

Consider a word-context pair $(w, c)$. Did this pair come from the observed data? Let $P(D = 1 \mid w, c)$ be the probability that $(w, c)$ came from the data, and $P(D = 0 \mid w, c) = 1 - P(D = 1 \mid w, c)$ the probability that $(w, c)$ did not. The distribution is modeled as:

$$P(D = 1 \mid w, c) = \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) = \frac{1}{1 + \exp(-\boldsymbol{u}_w^\top \boldsymbol{v}_c)}$$

where $\boldsymbol{u}_w$ and $\boldsymbol{v}_c$ ($d$-dimensional vectors) are the model parameters to be learned.

The negative sampling objective tries to maximize $P(D = 1 \mid w, c)$ for observed $(w, c)$ pairs while maximizing $P(D = 0 \mid w, c_n)$ for randomly sampled "negative" examples (hence the name "negative sampling"), under the assumption that randomly selecting a context for a given word is likely to result in an unobserved $(w, c_n)$ pair. SGNS's objective for a single $(w, c)$ observation is then:

$$\log \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) + k \cdot \mathbb{E}_{c_n \sim P_n} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n})$$

Derive this from the expression $P(D = 1 \mid w, c) \prod_{j=1}^{k} P(D = 0 \mid w, c_{n,j})$.

where $k$ is the number of "negative" samples and $c_n$ is the sampled context, drawn according to the empirical unigram distribution

$$P_n(c) = \frac{\#(c)}{T} = \frac{\#(c)}{\sum_{c' \in \mathcal{C}} \#(c')}.$$

The objective is trained in an online fashion using stochastic gradient updates over the observed pairs in the corpus. The global objective then sums over the observed $(w, c)$ pairs in the corpus:

$$L = \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{C}} \#(w, c)(\log \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) + k \cdot \mathbb{E}_{c_n \sim P_n} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n})).$$

### 4.6.2 SGNS as Implicit Matrix Factorization

SGNS embeds both words and their contexts into a low-dimensional space $\mathbb{R}^d$, resulting in the word and context matrices $\boldsymbol{U}$ and $\boldsymbol{V}$. It is instructive to consider the product $\boldsymbol{U}\boldsymbol{V}^\top = \boldsymbol{M}$. Viewed this way, SGNS can be described as factorizing an implicit matrix $\boldsymbol{M}$ of dimensions $|\mathcal{W}| \times |\mathcal{C}|$ into two smaller matrices, and $M_{w,c} = \boldsymbol{u}_w^\top \boldsymbol{v}_c$. So, each cell of matrix $\boldsymbol{M}$ contains a quantity $f(w, c)$ reflecting the strength of association between that particular word-context pair. What can we say about the association function $f(w, c)$? In other words, which matrix is SGNS factorizing?

Consider the global objective (equation 2) above. For sufficiently large dimensionality $d$ (i.e. allowing for a perfect reconstruction of $\boldsymbol{M}$), each product $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ can assume a value independently of the others. Under these conditions, we can treat the objective $L$ as a function of independent $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ terms, and find the values of these terms that maximize it.

We begin by rewriting equation 2:

$$\begin{aligned} L &= \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{C}} \#(w, c) \log \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) + \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{C}} \#(w, c)(k \cdot \mathbb{E}_{c_n \sim P_n} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n})) = \\ &= \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{C}} \#(w, c) \log \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) + \sum_{w \in \mathcal{W}} \#(w)(k \cdot \mathbb{E}_{c_n \sim P_n} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n})) \end{aligned}$$

and explicitly expressing the expectation term:

$$\begin{aligned} \mathbb{E}_{c_n \sim P_n} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n}) &= \sum_{c_n \in \mathcal{C}} \frac{\#(c_n)}{T} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n}) = \\ &= \frac{\#(c)}{T} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_c) + \sum_{c_n \in \mathcal{C} \setminus \{c\}} \frac{\#(c_n)}{T} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_n}). \end{aligned}$$

Combining equations 3 and 4 reveals the local objective for a *specific* $(w, c)$ pair:

$$L(w, c) = \#(w, c) \log \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) + k \cdot \#(w) \cdot \frac{\#(c)}{T} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_c).$$

To optimize the objective, we define $x = \boldsymbol{u}_w^\top \boldsymbol{v}_c$ and find its partial derivative with respect to $x$:

$$\frac{\partial L}{\partial x} = \#(w, c) \cdot \sigma(-x) - k \cdot \#(w) \cdot \frac{\#(c)}{T} \cdot \sigma(x).$$

We compare the derivative to zero, and after some simplification, arrive at:

$$y^2 - (a-1)y - a = 0$$

where

$$y = e^x,\ a = \frac{\#(w,c) \cdot T}{\#(w)\#(c)} \cdot \frac{1}{k}.$$

This quadratic equation has two solutions: $y = -1$ (which is invalid given the definition of $y$) and $y = a$. Substituting $y$ with $e^x$ and $x$ with $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ reveals:

$$\boldsymbol{u}_w^\top \boldsymbol{v}_c = \log a = \log \left( \frac{\#(w,c) \cdot T}{\#(w)\#(c)} \right) - \log k.$$

Thus, $M_{w,c} = f(w,c) = \log 2 \cdot PMI_{w,c} - \log k$.

In the original paper, PMI is defined as a natural logarithm, so a multiplier $\log 2$ does not appear there. What is the geometrical difference between matrices $PMI$ and $const \cdot PMI$?

For a negative-sampling value of $k = 1$, the SGNS objective is factorizing a word-context matrix in which the association between a word and its context is measured by $f(w,c) = \log 2 \cdot PMI_{w,c}$.

## 4.7 Seminar

- Hands-on tutorial on Word2Vec (gensim).
- Arithmetic operations with word vectors.
- **Homework:** to implement skip-gram or CBOW.

# Bibliography


[1] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[2] Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185.

[3] Manning, C. D., Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

[4] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

# Chapter 5

# Lexical-level and Morphological-level Extensions of Word2Vec

**Keywords:** glove, lexvec, swivel, morpheme, inflectional morphology, derivational morphology, n-grams, character embeddings, fasttext.

## 5.1 Lexical-level extensions

This chapter will be briefly overviewing models came after Word2Vec that exploit the same idea of representing words with vectors.

### 5.1.1 Shifted PPMI

While the PMI matrix emerges from SGNS with $k = 1$, it was shown that different values of $k$ can substantially improve the resulting embedding. With $k > 1$, the association metric in the implicitly factorized matrix is $PMI_{w,c} - \log k$. This suggests the use of **Shifted PPMI (SPPMI)**, a novel association metric which, to the best of our knowledge, was not explored in the NLP and word-similarity communities:

$$SPPMI^k_{w,c} = \max(PMI_{w,c} - \log k, 0) = (PMI_{w,c} - \log k)^+.$$

As with SGNS, certain values of $k$ can improve the performance on different tasks.

### 5.1.2 GloVe

The **Global Vector (GloVe)** model proposed by Pennington et al. (2014) aims to combine the count-based matrix factorization and the context-based skip-gram model together.

We all know the counts and co-occurrences can reveal the meanings of words. We will define the co-occurrence probability as:

$$P(w_k \mid w_i) = \frac{\#(w_i, w_k)}{\#(w_i)}.$$

Say, we have two words, $w_i$ = "ice" and $w_j$ = "steam". The third word $w_k$ = "solid" is related to "ice" but not "steam", and thus we expect $P(w_k \mid w_i)$ to be much larger than $P(w_k \mid w_j)$ and therefore $P(w_k \mid w_i)/P(w_k \mid w_j)$ to be very large. If the third word $w_k$ = "water" is related to both or $w_k$ = "fashion" is unrelated to either of them, the equation above is expected to be close to one.

The intuition here is that the word meanings are captured by the ratios of co-occurrence probabilities rather than the probabilities themselves. The global vector models the relationship between two words regarding to the third context word as:

$$F(w_i, w_j, w_k) = \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)}.$$

Further, since the goal is to learn meaningful word vectors, $F$ is designed to be a function of the linear difference between two words $w_i$ and $w_j$. We denote vectors for "central" words as $\boldsymbol{u}$ (with indices), and vectors for context words as $\boldsymbol{v}$:

$$F(w_i, w_j, w_k) = F((\boldsymbol{u}_{w_i} - \boldsymbol{u}_{w_j})^\top \boldsymbol{v}_{w_k}).$$

With the consideration of $F$ being symmetric between target words and context words, the final solution is to model $F$ as an exponential function. And we obtain the following equations:

$$F(\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k}) = \exp(\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k}) = P(w_k \mid w_i),$$

$$F((\boldsymbol{u}_{w_i} - \boldsymbol{u}_{w_j})^\top \boldsymbol{v}_{w_k}) = \exp((\boldsymbol{u}_{w_i} - \boldsymbol{u}_{w_j})^\top \boldsymbol{v}_{w_k}) = \frac{\exp(\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k})}{\exp(\boldsymbol{u}_{w_j}^\top \boldsymbol{v}_{w_k})} = \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)}.$$

Finally,

$$\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k} = \log P(w_k \mid w_i) = \log \frac{\#(w_i, w_k)}{\#(w_i)} = \log \#(w_i, w_k) - \log \#(w_i).$$

Since the second term $-\log \#(w_i)$ is independent of $k$, we can add bias term $b_{w_i}^u$ for $w_i$ to capture $-\log \#(w_i)$. To keep the symmetric form, we also add in a bias $b_{w_k}^v$ for $w_k$. After that, we obtain

$$\log \#(w_i, w_k) = \boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k} + b_{w_i}^u + b_{w_k}^v.$$

The loss function for the GloVe model is designed to preserve the above formula by minimizing the sum of the squared errors

$$L = \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{C}} f\left(\#(w, c)\right) \left(\boldsymbol{u}_w^\top \boldsymbol{v}_c + b_w^u + b_c^v - \log \#(w, c)\right)^2.$$

The weighting schema $f(x)$ is a function of the co-occurrence of words $w$ (as a central word) and $c$ (as a context word) and it is an adjustable model configuration. It should be close to zero as $x \to 0$; should be non-decreasing as higher co-occurrence should have more impact; should saturate when $x$ become extremely large. The paper proposed the following weighting function:

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

with optimal values $\alpha = 0.75$ and $x_{\max} = 100$.

### 5.1.3 LexVec

Just like GloVe, **LexVec** (Alexandre Salle et al., 2016) also tries to factorize PPMI matrices, emerging characteristics of both count-based and prediction-based models. Unlike GloVe, it penalizes errors of frequent co-occurrences more heavily, while still treating negative co-occurrences.

Moreover, given that using PPMI results in better performance than PMI on semantic tasks, the authors propose keeping the SGNS weighting scheme by using window sampling and negative sampling, but explicitly factorizing the PPMI matrix rather than implicitly factorizing the shifted PMI matrix. The LexVec loss function has two terms:

$$L^{WC}(w,c) = \frac{1}{2}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PPMI^*_{w,c}\right)^2,$$
$$L^{W}(w) = \frac{1}{2}\sum_{j=1}^{k} \mathbb{E}_{c_{n,j}\sim P_n}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_{c_{n,j}} - PPMI^*_{w,c_{n,j}}\right)^2$$

where $PPMI^*$ is an improved (e.g. by using context-distribution smoothing (Levy et al., 2015) or subsampling the corpus (Mikolov et al., 2013b)) PPMI matrix.

These terms could be minimized using two alternative approaches. The first one is Mini-Batch. This variant executes gradient descent in exactly the same way as SGNS. Every time a pair $(w,c)$ is observed by window sampling and pairs $(w,c_{n,1}),\ldots,(w,c_{n,k})$ drawn by negative sampling, $\boldsymbol{u}_w$, $\boldsymbol{v}_c$ and $\boldsymbol{v}_{c_{n,1}},\ldots,\boldsymbol{v}_{c_{n,k}}$ are updated by gradient descent on the sum of two loss function terms.

Another approach for minimization is Stochastic. In this case every context window is extended with $k$ negative samples $c_{n,1},\ldots,c_{n,k}$. Iterative gradient descent of the first equation is then run on pairs $(w_t,c)$, for $c\in\mathcal{C}_t = \cup_{i=-win,i\neq 0}^{win} w_{t+i}$ and $(w_t,c_{n,j})$, $j=1,\ldots,k$ for each window.

### 5.1.4 Swivel

Noam Shazeer et al. in 2016 presented a **Submatrix-wise vector embedding learner (Swivel)**.

This model is based on applying SVD for PMI matrix, and the main idea is to use a loss function which penalties depend on whether the word-context pair co-occurs in the corpus or not, so the algorithm could be trained to not to over-estimate PMI of common values whose co-occurrence is unobserved. Notably, Word2Vec with a negative sampling is also capable of taking unobserved co-occurrences into account, but it is done indirectly.

The central claim of the authors of Swivel is that none of the mainstream word embeddings provide any special treatment to unobserved word-context co-occurrences, so the ability to capture unobserved word-context co-occurrences helped to outperform other embedding training algorithms in word similarity and word analogy tasks.

For co-occurrences that have been observed ($\#(w,c)>0$), we'd like $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ to accurately estimate $PMI_{w,c}$ subject to how confident we are in the observed count $\#(w,c)$ . Swivel computes the weighted squared error between the embedding dot product and the PMI of $w$ and $c$:

$$L^{+}(w,c) = \frac{1}{2} f(\#(w,c))\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PMI_{w,c}\right)^2 =$$
$$= \frac{1}{2} f(\#(w,c))\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - \log\#(w,c) - \log T + \log\#(w) + \log\#(c)\right)^2.$$

This encourages $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ to correctly estimate the observed PMI. The loss is modulated by a monotonically increasing confidence function $f(x)$: the more frequently a co-occurrence is observed, the more the model is required to accurately approximate $PMI_{w,c}$. The authors experimented with several different variants for $f(x)$, and discovered that a linear transformation of $x^{\frac{1}{2}}$ produced good results.

Unfortunately, if $w$ and $c$ are never observed together, $\#(w,c) = 0$, $PMI_{w,c} = -\infty$, and the squared error cannot be computed. What would we like the model to do in this case? Treating $\#(w,c)$ as a sample, we can ask: how significant is it that its observed value is zero? If the two words $w$ and $c$ are rare, their co-occurrence could plausibly have gone unobserved due to the fact that we simply haven't seen enough data. On the other hand, if words $w$ and $c$ are common, this is less likely: it becomes significant that a co-occurrence hasn't been observed, so perhaps we ought to consider that the features are truly anti-correlated. In either case, we certainly don't want the model to over-estimate the PMI between features, and so we can encourage the model to respect an upper bound on its PMI estimate $\boldsymbol{u}_w^\top \boldsymbol{v}_c$.

We address this by smoothing the PMI value as if a single co-occurrence had been observed (i.e., computing PMI as if $\#(w,c) = 1$), and using an asymmetric cost function that penalizes over-estimation of the smoothed PMI. The following "soft hinge" cost function accomplishes this:

$$\begin{aligned} L^0(w,c) &= \log\left(1 + \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PMI^*_{w,c}\right)\right) = \\ &= \log\left(1 + \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - \log T + \log \#(w) + \log \#(c)\right)\right). \end{aligned}$$

Here, $PMI^*$ refers to the smoothed PMI computation where $\#(w,c)$'s actual count of 0 is replaced with 1. This loss penalizes the model for over-estimating the objective value; however, it applies negligible penalty – i.e., is noncommittal—if the model under-estimates it.

## 5.2 Morphology. Inflectional and Derivational Morphology

Firstly, we will give several basic definitions.

**Morphology** is a study of internal structure of words.

**Morpheme** is the smallest linguistic unit which has a meaning or grammatical function. Words are composed of morphemes (one or more). There are some complications with this simple definition: *sing|er|s*, *moon|light*, *un|kind|ly*, *talk|s*, *ten|th*, *de|nation|al|iz|ation*. The order of morphemes matters: *talk|ed* $\neq$ **ed|talk*, *re|write* $\neq$ **write|re*.

**Morph**. The term morpheme is used both to refer to an abstract entity and its concrete realization(s) in speech or writing. When it is needed to maintain the signified and signifier distinction, the term morph is used to refer to the concrete entity, while the term morpheme is reserved for the abstract entity only.

**Allomorphs** are morphemes having the same function but different form. Unlike the synonyms they usually cannot be replaced one by the other.

1. (a) indefinite article: *an orange — a building*
   (b) plural morpheme: *cat/s[s] — dog/s[z] — judg/es[?s]*
2. (a) *matk/a (mother$_{nom}$) — mat/?ek/? (mothers$_{gen}$) — matc/e (mother$_{dat}$) — matč/in (mother's)*

### 5.2.1 Classification of Morphemes

**Bound and Free** .

- **Bound** morpheme cannot appear as a word by itself: *-s (dog/s), -ly (quick/ly), -ed (walk/ed).*
- **Free** morpheme can appear as a word by itself; often can combine with other morphemes too: *dog (dog/s), walk (walked), of, the, or.*

**Root and Affixes** .

- **Root** is a nucleus of the word that affixes attach too. In English, most of the roots are free. In some languages that is less common (Lithuanian: *Bill/as Clinton/as*). Compounds contain more than one root: *home/work.*
- **Affix** is a morpheme that is not a root; it is always bound. There are several kinds of affixes:
  - **suffix**: *talk — talk/ing, quick — quick/ly*;
  - **prefix**: *happy — un/happy, existing — pre/existing*;
  - **circumfix**: Dutch: *berg (mountain) — ge/berg/te (mountains), *geberg, *bergte.*
  - **infix**: Tagalog: *basa (read) — b/um/asa (read$_{past}$)*;
  - **transfix**: Arabic: *k-t-b (write) — k/a/t/a/b/a (he wrote), ya/kt/u/b/u (he is writing)*;
  - **interfix** TODO.

  Suffixes more common than prefixes which are more common than infixes/circumfixes

**Content and Functional** .

- **Content** morphemes carry some semantic content *car, -able-, un-, -ness.*
- **Functional** morphemes provide grammatical information *the, and, -s (plural), -s (3rd sg).*

**Inflection and Derivation** .

- **Inflection** is a process of creating various forms of the same word: *big — bigger, biggest.* **Lexeme** is an abstract entity; the set of all forms related by inflection (but not derivation). **Lemma** is a form from a lexeme chosen by convention (e.g., nom.sg. for nouns, infinitive for verbs) to represent that set. Also called the canonical / base / dictionary / citation form. E.g., *break, breaks, broke, broken, breaking* have the same lemma *break.* **Ending** is an inflectional suffix.
- **Derivation** is a process of creating new words: *slow — slowly, slowness.*

  Derivation tends to affects the meaning of the word, while inflection tends to affect only its syntactic function. Derivation tends to be more irregular — there are more gaps, the meaning is more idiosyncratic and less compositional. However, the boundary between derivation and inflection is often fuzzy and unclear.

For any word we can build a tree with a "history" of a word: *unbelievable = un + (believe + able)*, but *unbelievable ≠ *(un + believe) + able.* At the same time, some words can be ambiguous: *unlockable = (un + lock) + able, unlockable = un + (lock + able).*

### 5.2.2 Morphological processes

- **Concatenation** (adding continuous affixes) is the most common process. Often phonological changes on morpheme boundaries.
- **Reduplication** is a process when a part of the word or the entire word is doubled. Afrikaans: *amper (nearly) — amper/amper (very nearly)*; Tagalog: *basa (read) — ba/basa (will read)*; English: *humpty/dumpty.*
- **Templates** Both the roots and affixes are discontinuous. Only Semitic languages (Arabic, Hebrew). A root (3 or 4 consonants, e.g., *l-m-d (learn)*) is interleaved with a (mostly) vocalic pattern. Hebrew: *lomed (learn$_{masc}$) — lomad (learnt$_{masc.sg.3rd}$) — limed (taught$_{masc.sg.3rd}$) — lumad (was taught$_{masc.sg.3rd}$), shatak (be quiet$_{pres.masc}$) — shatak (was quiet$_{masc.sg.3rd}$) — shitek (made sb to be quiet$_{masc.sg.3rd}$) — shutak (was made to be quiet$_{masc.sg.3rd}$)*
- **Morpheme internal changes (apophony, ablaut)** are the processes when the word changes internally. English: *sing — sang — sung, man — men, goose — geese*; German: *Hund (dog) — Hünd/chen (small dog).*
- **Subtraction (Deletion)**: some material is deleted to create another form. Papago (a native American language in Arizona): *him (walking$_{imperf.}$) — hi (walking$_{perf.}$)*; French: *grande (big$_f$) — grand (big$_m$), fausse (false$_f$) — faux (false$_m$).*
- **Suppletion**: English: *be — am — is — was, go — went, good — better.*

### 5.2.3 Word formation

- **Affixation**. Words are formed by adding affixes: *write — writer, productive — unproductive.*
- **Compounding**. Words are formed by combining two or more words: *rain + bow — rainbow, over + do — overdo.*
- **Acronyms** are like abbreviations, but act as a normal words: *light amplification by simulated emission of radiation — laser.*
- **Blending** parts of two different words are combined: *breakfast + lunch — brunch, motor + hotel — motel.*
- **Clipping** longer words are shortened: *doctor — doc, laboratory — lab, advertisement — ad, examination — exam.*

### 5.2.4 Morphological Types of Languages

- **Analytic** languages have only free morphemes, sentences are sequences of single morpheme words.
- **Synthetic** languages have both free and bound morphemes.
  - **Agglutinating** — each morpheme has a single function, it is easy to separate them. E.g., Uralic lgs (Estonian, Finnish, Hungarian), Turkish, Basque, Dravidian languages (Tamil, Kannada, Telugu), Esperanto.

– **Fusional** — like agglutinating, but affixes tend to "fuse together", one affix has more than one function. Common homonymy of inflectional affixes. E.g., Slavic, Romance languages, Greek.

– **Polysynthetic** — extremely complex, many roots and affixes combine together, often one word corresponds to a whole sentence in other languages. E. g. Eskimo: *angyaghllangyugtuq (he wants to acquire a big boat).*

Usually languages combine several types in different proportions.

## 5.3 N-grams. FastText

In this section, we describe a model to learn word representations while taking into account morphology. It was proposed by Bojanowski et al. (2016). A morphology is modeled by considering subword units, and representing words by a sum of its character $n$-grams. We will begin by presenting the general framework that is used to train word vectors, then present a subword model and eventually describe how to handle the dictionary of character $n$-grams.

### 5.3.1 General Model

We start by briefly reviewing the continuous skipgram model introduced by Mikolov et al. (2013), from which this model is derived. Given a word vocabulary $\mathcal{W}$, where a word is associated with its index $w \in \{1, \ldots, |\mathcal{W}|\}$, the goal is to learn a vectorial representation for each word $w \in \mathcal{W}$. Inspired by the distributional hypothesis, word representations are trained to *predict well* words that appear in its context. More formally, given a large training corpus represented as a sequence of words $w_1, \ldots, w_T$, the objective of the skipgram model is to maximize the following log-likelihood:

$$\sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} \log P(c \mid w_t),$$

where the context $\mathcal{C}_t$ is the set of words surrounding word $w_t$. The probability of observing a context word $c$ given $w_t$ will be parameterized using the aforementioned word vectors. For now, let us consider that we are given a scoring function $s$ which maps pairs of (word, context) to scores in $\mathbb{R}$. One possible choice to define the probability of a context word is the softmax:

$$P(c \mid w_t) = \frac{\exp(s(w_t,\ c))}{\sum_{c' \in \mathcal{C}} \exp(s(w_t,\ c'))}.$$

However, such a model is not adapted to our case as it implies that, given a word $w_t$, we only predict one context word $c$.

The problem of predicting context words can instead be framed as a set of independent binary classification tasks. Then the goal is to independently predict the presence (or absence) of context words. For the word at position $t$ we consider all context words as positive examples and sample negatives at random from the dictionary. For a chosen context word $c$, using the binary logistic loss, we obtain the following negative log-likelihood:

$$\log\left(1 + e^{-s(w_t,\ c)}\right) + \sum_{c_n \in \mathcal{N}_{t,c}} \log\left(1 + e^{s(w_t,\ c_n)}\right),$$

where $\mathcal{N}_{t,c}$ is a set of negative examples sampled from the vocabulary. By using the sigmoid function, we can re-write the objective as:

$$\sum_{t=1}^{T} \left[ \sum_{c \in \mathcal{C}_t} \sigma(s(w_t,\ c)) + \sum_{c_n \in \mathcal{N}_{t,c}} \sigma(-s(w_t,\ c_n)) \right].$$

A natural parameterization for the scoring function $s$ between a word $w_t$ and a context word $c$ is to use word vectors.

Let us define for each word $w$ in the vocabulary two vectors $\boldsymbol{u}_w$ and $\boldsymbol{v}_w$ in $\mathbb{R}^d$. These two vectors are sometimes referred to as *input* and *output* vectors in the literature. In particular, we have vectors $\boldsymbol{u}_{w_t}$ and $\boldsymbol{v}_c$, corresponding, respectively, to words $w_t$ and $c$. Then the score can be computed as the scalar product between word and context vectors as $s(w_t, c) = \boldsymbol{u}_{w_t}^\top \boldsymbol{v}_c$.

### 5.3.2 FastText Model

By using a distinct vector representation for each word, the skipgram model ignores the internal structure of words. In this section, we propose a different scoring function $s$, in order to take into account this information.

Each word $w$ is represented as a bag of character $n$-grams. We add special boundary symbols `<` and `>` at the beginning and end of words, allowing to distinguish prefixes and suffixes from other character sequences. We also include the word $w$ itself in the set of its $n$-grams, to learn a representation for each word (in addition to character $n$-grams). Taking the word *where* and $n = 3$ as an example, it will be represented by the character $n$-grams:

```
<wh, whe, her, ere, re>
```

and the special sequence

```
<where>.
```

Note that the sequence `<her>`, corresponding to the word *her* is different from the tri-gram `her` from the word *where*. In practice, we extract all the $n$-grams for $3 \leq n \leq 6$. This is a very simple approach, and different sets of $n$-grams could be considered, for example taking all prefixes and suffixes.

Suppose that you are given a dictionary of $n$-grams. Given a word $w$, let us denote by $\mathcal{G}_w$ the set of $n$-grams appearing in $w$. We associate a vector representation $\boldsymbol{z}_g$ to each $n$-gram $g$. We represent a word by the sum of the vector representations of its $n$-grams. We thus obtain the scoring function:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \boldsymbol{z}_g^\top \boldsymbol{v}_c.$$

Often you can see a modified version of it:

$$s(w, c) = \frac{1}{|\mathcal{G}_w|} \sum_{g \in \mathcal{G}_w} \boldsymbol{z}_g^\top \boldsymbol{v}_c.$$

This simple model allows sharing the representations across words, thus allowing to learn reliable representation for rare words.

## 5.4 Subword LexVec

The LexVec model factorizes the PPMI-weighted word-context co-occurrence matrix using stochastic gradient descent. LexVec adjusts the PPMI matrix using *context distribution smoothing*.

With the PPMI matrix calculated, the sliding window process is repeated and the following loss functions are minimized for every observed $(w, c)$ pair and target word $w$:

$$L^{WC}(w,c) = \frac{1}{2}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PPMI^*_{w,c}\right)^2,$$
$$L^{W}(w) = \frac{1}{2}\sum_{j=1}^{k} \mathbb{E}_{c_{n,j}\sim P_n}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_{c_{n,j}} - PPMI^*_{w,c_{n,j}}\right)^2$$

where $\boldsymbol{u}_w$ and $\boldsymbol{v}_c$ are $d$-dimensional word and context vectors. The second loss function describes how, for each target word, $k$ *negative samples* are drawn from the smoothed context unigram distribution.

Given a set of subwords $\mathcal{S}_w$ for a word $w$, we follow fastText and replace $\boldsymbol{u}_w$ in loss function equations by $\boldsymbol{u}'_w$ such that:

$$\boldsymbol{u}'_w = \frac{1}{|\mathcal{S}_w| + 1}\left(\boldsymbol{u}_w + \sum_{s\in\mathcal{S}_w} \boldsymbol{q}_{hash(s)}\right)$$

such that a word is the sum of its word vector and its $d$-dimensional subword vectors $\boldsymbol{q}_x$. The number of possible subwords is very large so the function $hash(s)$[1] hashes a subword to the interval $[1, buckets]$. For OOV words,

$$\boldsymbol{u}'_w = \frac{1}{|\mathcal{S}_w|}\sum_{s\in\mathcal{S}_w} \boldsymbol{q}_{hash(s)}.$$

Salle and Villavicencio compared two types of subwords: simple n-grams (like fastText) and unsupervised morphemes. For example, given the word "cat", we mark beginning and end with angled brackets and use all n-grams of length 3 to 6 as subwords, yielding $\mathcal{S}_{\text{cat}} = \{\langle \text{ca}, \text{at}\rangle, \text{cat}\}$. Morfessor (**?** ) is used to probabilistically segment words into morphemes. The Morfessor model is trained using raw text so it is entirely unsupervised. For the word "subsequent", we get $\mathcal{S}_{\text{subsequent}} = \{\langle \text{sub}, \text{sequent}\rangle\}$.

## 5.5 Byte Pair Embeddings

### 5.5.1 Byte Pair Encoding

**Byte pair encoding (BPE)** is a variable-length encoding that views text as a sequence of symbols and iteratively merges the most frequent symbol pair into a new symbol. E.g., encoding an English text might consist of first merging the most frequent symbol pair *t h* into a new symbol *th*, then merging the pair *th e* into *the* in the next iteration, and so on. The number of merge operations $o$ determines if the resulting encoding mostly creates short character sequences (e.g. $o = 1000$) or if it includes symbols for many frequently occurring words, e.g. $o = 30000$ (cf. Table 5.1). Since the BPE algorithm works with any sequence of symbols, it requires no preprocessing and can be applied to untokenized text.

[1] http://www.isthe.com/chongo/tech/comp/fnv/

| Byte-pair encoded text |
|---|
| to y od a _station is _a _r ail way _station _on _the _ch ū ō _main _l ine |
| to y od a _station _is _a _railway _station _on _the _ch ū ō _main _line |
| toy oda _station _is _a _railway _station _on _the _ch ū ō _main _line |
| toy oda _station _is _a _railway _station _on _the _chū ō _main _line |
| toy oda _station _is _a _railway _station _on _the _chūō _main _line |
| toyoda station is a railway station on the chūō main line |

Table 5.1: Effect of the number of BPE merge operations on the beginning of the English Wikipedia article TOYODA_STATION. Since BPE is based on frequency, the resulting segmentation is often, but not always meaningful.

### 5.5.2 BPEmb

**BPEmb** is a collection of pre-trained subword unit embeddings in 275 languages, based on byte-pair encoding. In an evaluation using fine-grained entity typing as testbed, BPEmb performs competitively, and for some languages better than alternative subword approaches, while requiring vastly fewer resources and no tokenization. BPEmb is available at `https://github.com/bheinzerling/bpemb`.

The authors applied BPE[2] to all Wikipedias of sufficient size with various $o$ and pre-trained embeddings for the resulting BPE symbol using GloVe, resulting in byte-pair embeddings for 275 languages. To allow studying the effect the number of BPE merge operations and of the embedding dimensionality, the authors provide embeddings for 1000, 3000, 5000, 10000, 25000, 50000, 100000 and 200000 merge operations, with dimensions 25, 50, 100, 200, and 300.

## 5.6 Morphological Embeddings

### 5.6.1 Capturing Morphology in Existing Models

Why Word2Vec and GloVe aren't enough?

TODO

ME are theoretically grouded. ...

For some languages, FastText is ok.

### 5.6.2 Subword-level Embeddings for Korean

TODO

Trere are a number of approaches to use morphological information in learning word embeggings. We start with the simplest ones.

### 5.6.3 Morpheme Segmentation

Morpheme segmentation is a process of deriving words into morphemes. The model is the same as for FastText. The only difference is in using morphemes instead of n-grams. This may sound easy, but it is not so. ...

[2]They used the SentencePiece BPE implementation: `https://github.com/google/sentencepiece`.

For various languages segmentation algorithms are developed. Such algorithms in the most cases used statistical information, but in recent years neural network based model were proposed.

### 5.6.4 Compositionality on a Morphological Level

Lazaridou et al. (2013) were the first to predict distributional vectors for derived words using CDSMs and experimented with a range of established CDSMs. In their paper, all models are supervised, i.e., some word pairs for each pattern are used as training instances, and others serve for evaluation. Also, all models assume that the base word (input) $b$ and derived word (output) $d$ are represented as vectors in some underlying distributional space.

**Simple Additive Model** The simple additive model predicts the derived word from the base word as

$$\boldsymbol{v}_d = \boldsymbol{v}_b + \boldsymbol{v}_p$$

where $\boldsymbol{v}_p$ is a vector representing the semantic shift accompanying the derivation pattern $p$.

**Simple Multiplicative Model** The simple multiplicative model is very similar, but uses component-wise multiplication

$$\boldsymbol{v}_d = \boldsymbol{v}_b \odot \boldsymbol{v}_p$$

instead of addition to combine the base and pattern vectors (Mitchell and Lapata, 2010).

**Weighted Additive Model** The third model, the weighted additive model, enables a simple reweighting of the contributions of basis and pattern

$$\boldsymbol{v}_d = \alpha\boldsymbol{v}_b + \beta\boldsymbol{v}_p.$$

**Basic Additive Model** The Basic Additive model (introduced in (Mitchell and Lapata, 2008)) computes the distibutional semantics of a pair of words according to formula

$$\boldsymbol{v}_d = (1-\alpha)\boldsymbol{v}_b + \alpha\boldsymbol{v}_p.$$

**Lexical Function Model** The lexical function model (Baroni and Zamparelli, 2010) represents the pattern as a matrix $\boldsymbol{P}_p$ that is multiplied with the basis vector:

$$\boldsymbol{v}_d = \boldsymbol{P}_p\boldsymbol{v}_b,$$

essentially modelling derivation as linear mapping. This model is considerably more powerful than the others, however its number of parameters is quadratic in the number of dimensions of the underlying space, whereas the additive and multiplicative models only use a linear number of parameters.

The main advance of this model is that a commutative law generally does not hold for matrices, and it allows us to derive words taking into account the order of applying derivational patterns. E.g., $\boldsymbol{P}_{able}(\boldsymbol{P}_{un}(\boldsymbol{v}_{lock})) \neq \boldsymbol{P}_{un}(\boldsymbol{P}_{able}(\boldsymbol{v}_{lock}))$ .

Note that we can defeine pattern in a various ways. For instance, if we define them like $\boldsymbol{P}_{V+able}(\boldsymbol{P}_{un+V}(\boldsymbol{v}_{lock})) \neq \boldsymbol{P}_{un+Adj}(\boldsymbol{P}_{V+able}(\boldsymbol{v}_{lock}))$ then we will have different matrices (vectors in other models) for the same patterns applied to different parts of speech.

**Full Additive Model** The full additive model computes the compositional vector of a pair using two linear tranformations $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively applied to the vectors of the base and the pattern:

$$\boldsymbol{v}_d = \boldsymbol{A}\boldsymbol{v}_b + \boldsymbol{B}\boldsymbol{v}_p.$$

## 5.7 Character Embeddings (advanced)

TODO

# Bibliography


[1] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association of Computational Linguistics*, 5(1):135–146.

[2] Kisselew, M., Padó, S., Palmer, A., and Šnajder, J. (2015). Obtaining a better understanding of distributional models of german derivational morphology. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 58–63.

[3] Marelli, M. and Baroni, M. (2015). Affixation in semantic space: Modeling morpheme meanings with compositional distributional semantics. *Psychological review*, 122(3):485.

[4] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

# Chapter 6

# Evaluation of Distributional Semantic Models

**Keywords:** intrinsic evaluation, extrinsic evaluation, evaluation benchmark, word similarity, word analogy, word categorization, retrofitting.

## 6.1 Types of relations between words. Difference between word similarity and word relatedness

for instance, the cosine distance between words *"кошка"* (cat) and *"собака"* (dog) was 0.74

## 6.2 Impact of corpora and parameters on model performance

## 6.3 How model performance could be measured? Two paradigms of evaluation

## 6.4 Intrinsic evaluation benchmarks

## 6.5 Why evaluation is hard

## 6.6 Seminar

- Retrofitting to lexical relations.
- Hands-on tutorial on Vecto (word embeddings evaluation framework).

## Bibliography

[1] Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 238–247.

[2] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

[3] Rogers, A., Drozd, A., and Li, B. (2017). The (too many) problems of analogical reasoning with word vectors. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 135–148.

# Chapter 7

# From Words to Phrases, Sentences and Documents

**Keywords:** multiword expressions, collocations, pragmatics, semantic constituent, compositionality, bag-of-vectors, sentence embeddings, document embeddings.

## 7.1 Multi-word expressions, collocations, difference between MWE, words and idioms

TODO

## 7.2 Approaches to word vector composition

TODO

## 7.3 Introduction to syntax

TODO

### 7.3.1 Constituent grammar

TODO

### 7.3.2 Dependency grammar

TODO

## 7.4 Dependency-based contexts. Word2Vec-f

TODO

## 7.5 Syntax and pragmatics

TODO

## 7.6 Rhetorical structure theory

TODO

## 7.7 Sentence embeddings and document embeddings

TODO

### 7.7.1 Doc2Vec

TODO

### 7.7.2 Skip-Thought Vectors

TODO

## 7.8 Evaluation of sentence embeddings

TODO

# Bibliography

[1] Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. In *Proceedings of NAACL-HLT*, pages 1367–1377.

[2] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

[3] Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

[4] Taddy, M. (2015). Document classification by inversion of distributed language representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 45–49.

# Chapter 8

# Multi-sense Word Embeddings

**Keywords:** lexical ambiguity, polysemy, homonymy, word sense induction, word sense disambiguation, sense embeddings, multi-sense embeddings, automatic definition generation, adagram.

## 8.1 Lexical ambiguity

## 8.2 Dealing with ambiguity in word embeddings. Introduction to Bayesian probability theory

## 8.3 AdaGram and multi-sense embeddings

## 8.4 Word Sense Induction and Word Sense Disambiguation

## 8.5 Defenition Generation

## 8.6 Distributional Thesauri

## 8.7 CoVe

## 8.8 ELMo

# Chapter 9

# Cross-language Word Embeddings

**Keywords:** multilinguality, parallel corpora, comparable corpora, cross-language word embeddings, machine translation, unsupervised machine translation, muse, vecmap.

## 9.1 A brief introduction to machine translation

Machine translation, or MT, is the automatizing of translations by devices such as computers. Even if some processes or ideas about automatic translations have been described since the 17th century, the concept of it, as a scientific field that should be explored, started in 1949 with Warren Weaver, for the scientific world, and in 1951 for the first public experiment: the Georgetown–IBM experiment where sixty Russian sentences were translated into English.

At the beginning and mostly because resources were not allowing anything else, most of the translation work was based on different layers of semantic rules going from the word switch to some simple grammar modifications. Even if tools were rather simple, results were understandable.

Nowadays, computer's resources allow a larger range for technique used, such as neural interpretation, deep learning and so one, but even with its simplicity, the rule based model is still used and not always outperformed.

The main idea of machine translation can be described with the Vauquois' pyramid. Different layers of translations are possible and according to them different tasks can be achieve. The basement is a simple word to word translation whereas on the top, it is a meaningful translation.

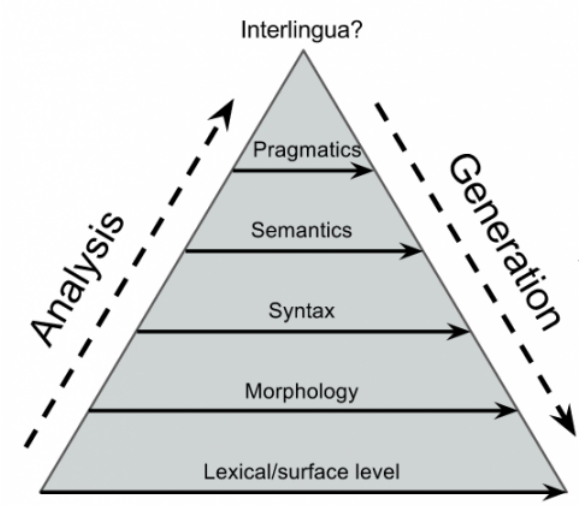



Figure 9.1: Vauquois pyramid

It has to be consider that according to the need, i.e. the text to translate, different layers can be reached. For some specific tasks such as specialist papers or scientific ones, the task does not require going as high and some word to word processes can be easily be used. On the contrary, translating books from literature is harder as words are used with a stronger care about their meanings. Hence, higher level in the Vauquois' pyramid have to be reached.

According to the layer where the translation occurs, resources used and time needed vary; the higher in the pyramid, the longer the process needs and bigger the amount of resources has to be.

## 9.2 Cross-language word embeddings based on parallel corpora

While learning a new language, people tend to write word in their target language and the equivalent in their mother tongue. This process can be seen as working with cross-language work based on comparable corpora. Indeed, behind this definition is the idea of comparing two languages linked by the fact they do have the same meaning, idea or concept.

This similitude may appear at different level such as word level, as it is the case in the given example higher, sentence level, as it may be seen with subtitles from movies and document level as it is the case with translated books for instance.

To each level, things possibly done are different and results vary. The followings sections give a brief overview of major advantages and drawbacks as well as some idea of main concepts used.

### 9.2.1 Word-aligned data

**Linear projection** Obtaining vectors for each word in a particular language is nowadays an easy task thanks to many models freely available. However when the wish is to combine models and vectors, let say to see similarities between one word from German and one from Russian, results are completely wrong. This is due to the fact that basis used to compute these vectors are not the same.

Studying relations between pairs of words English-Spanish), Mikolov et al. in (2) highlight that the geometric relation within words of different languages is conserved. Thus

they decided to encode this geometry through matrices $W_i$ and $W_j$ in order to find a linear mapping between them.

To learn weight of matrices, they used a gradient descent minimizing the distance monolingual representation $x_i$ of words $w_i$ and its translation $z_i$ using the following formula:

$$dist = min_W \sum_{i=1}^{n} |Wx_i - z_i|^2$$

**Canonical correlation analyse** The idea of the Canonical correlation analyse (CCA) is similar to the linear projection. The main difference is however in the fact that each language has its own matrix, which was not the case previously.

The matrices $\sum$ and $\Omega$ are word representations in two different languages and $V$ and $W$ are the two mapping projecting the embedded representation to a new common space.

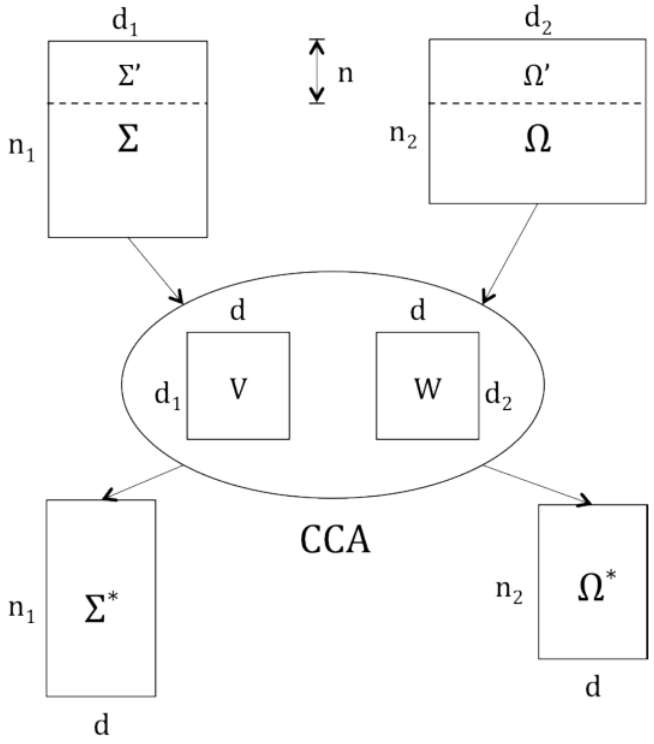



Figure 9.2: Canonical correlation analyse (Faruqui and Dyer, 2014)

An interesting result of this technique is the fact that antonyms and synonyms are separated as it can be seen in **??**.

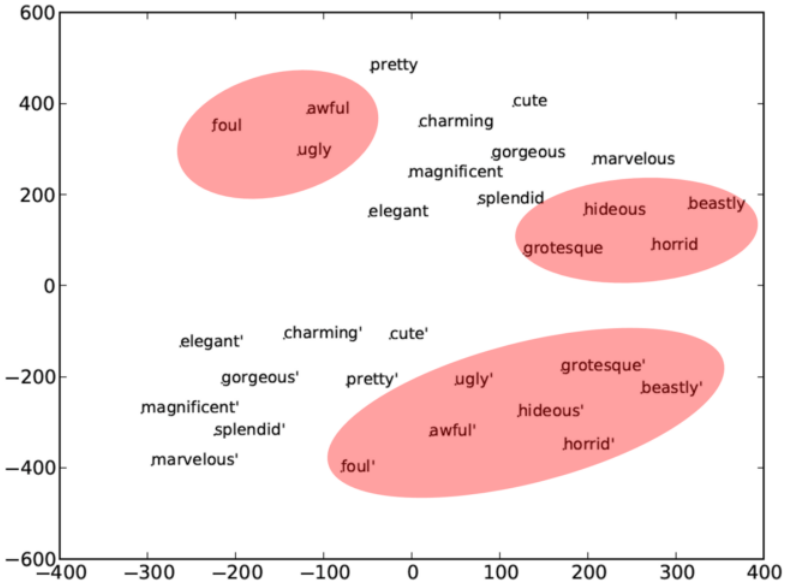



Figure 9.3: Projection of synonyms and antonyms (Faruqui and Dyer, 2014)

### 9.2.2 Sentence-aligned data

### 9.2.3 Document-aligned data

## 9.3 Cross-language word embeddings based on comparable corpora

## 9.4 MUSE. VecMap. Translation with minimal supervision

## Bibliography

[1] Conneau, A., Lample, G., Ranzato, M., Denoyer, L., and Jégou, H. (2017). Word translation without parallel data. *arXiv preprint arXiv:1710.04087*.

[2] Mikolov, T., Le, Q. V., and Sutskever, I. (2013a). Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168.

[3] Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

[4] Smith, S. L., Turban, D. H., Hamblin, S., and Hammerla, N. Y. (2017). Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *arXiv preprint arXiv:1702.03859*.

# Chapter 10

# Recent Trends in Distributional Semantics

**Keywords:** semantic shift, lexical change, diachronic word embeddings, computer vision, image embeddings, multimodal word embeddings, non-euclidian geometry, poincare embeddings, fairness in machine learning, model bias, fair word embeddings.

## 10.1 Diachronical word embeddings. The problem of semantic shifts

## 10.2 Multimodal word embeddings. A brief introduction to computer vision and convolutional neural networks

## 10.3 Gaussian word embeddings

## 10.4 Poincare word embeddings. A brief introduction to non-euclidian geometry

## 10.5 Fair word embeddings. A problem of ethics and fairness in machine learning

## Bibliography

[1] Hamilton, W. L., Leskovec, J., and Jurafsky, D. (2016). Diachronic word embeddings reveal statistical laws of semantic change. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1489–1501.

[2] Yao, Z., Sun, Y., Ding, W., Rao, N., and Xiong, H. (2018). Dynamic word embeddings for evolving semantic discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 673–681. ACM.

# Chapter 11

# Bridging Logics, Formal Semantics and Distributional Semantics

## Bibliography

[1] Asher, N., Van de Cruys, T., Bride, A., and Abrusán, M. (2016). Integrating type theory and distributional semantics: a case study on adjective–noun compositions. *Computational Linguistics*, 42(4):703–725.

[2] Beltagy, I., Roller, S., Cheng, P., Erk, K., and Mooney, R. J. (2016). Representing meaning with a combination of logical and distributional models. *Computational Linguistics*, 42(4):763–808.

[3] Boleda, G. (2019). Distributional semantics and the grammar-semantics interface. *Annual Review of Linguistics*.

[4] Yanaka, H., Mineshima, K., Pascual, M.-G., and Daisuke, B. (2018). Towards understanding bilingual textual entailment and similarity. *NLP in the Era of Big Data, Deep Learning, and Post Truth ESSLLI 2018 Workshop*.

# Chapter 12

# Algebraic Semantics and Quantum Semantics

**Keywords:** tensor algebra, higher-order tensors, convolution, tensor factorization, Frobenius algebras, quantum semantics, category theory.

## 12.1 TO DO

Update ref Change names Make pictures

## 12.2 Algebraic semantics

If having a vector for each word of the corpora, it does not mean that everything can be done. For instance, the compositionality of the relation adjective-noun or subject-verb-object is far from being accurate all the time with simple product between terms. Moreover, the composition of words can completely change the meaning of sentences, as it is the case with *Fake guns can't kill.* where the adjective $fake$ changes the main property of its subject *guns*.

In order to understand how it is possible to encode more accurately sentences, or how to combine vectors together leading to a meaningful composition, Marco Baroni and Roberto Zamparelli, in (**?** ), explore some alternatives where other mathematical objects are used instead of vectors only.

### 12.2.1 Find a title

#### Adjectives as linear maps

The main purpose to use adjectives is to modify properties of a noun. Hence, it has to be found a way to encode a kind a function for adjective where the input would be the noun and the output the modified noun. The assumption made within (**?** ) is that adjective in attribute position can be seen as linear functions from $n-dimension$ to $n-dimension$. In other words, adjectives are endomorphic linear maps in the noun space.

A mathematical representation can be given as follows:

$$p = B.v$$

where $p$ is the encoded relation "attribute-noun", $B$ is the weight matrix representing the adjective and $v$ the vector representation of the noun. In this case, the adjective matrices are evaluated and adjusted one by one.

### 12.2.2 Verbs

In sentences, verbs are the grammatical center. They express an act or a mode of a given object, considered as subject. Hence, the amount of information they carry is rather important and needs a more advance structure than a simple vector to store them.(Edward Grefenstette)

**Intransitive verbs** From an extensional view, the meaning of an intransitive verb (IN) is the set of objects which perform the action denoted by this one. (7) A formal definition is as follows :

$$[verb_{IN}] = \{x \mid x \in U \wedge x \ verb_{IN}\} \tag{12.1}$$

$U$ is the universe where the action takes place and $x$ denotes an object in the state of the one induced by the verb.

Considering a noun space $N$ created from orthogonal vectors $\{\overrightarrow{n_i}\}_i$ denoting individuals in a universe $U$, the set of object $S$ performing the action denoted by a verb $v_{IN}$ can be modeled as :

$$\overrightarrow{v_{IN}} = \sum_{n \in S} \overrightarrow{n} \tag{12.2}$$

However this representation does not follow the type induced by the Lambek category for intransitive verb (6). This one should be $N \otimes S$ instead of $N$ as it is now. A consideration to each element of the universe $U$ fixes this and changes the formal definition 12.1. Indeed, all are not in the state denoted by a verb $v$.

$$[verb_{IN}] = \{(x,t) \mid x \in U, t = \top if x \ verb_{IN} and \bot\} \tag{12.3}$$

Considering now a noun space $N$ and a sentence space $S$ spanned by the vector $\overrightarrow{1}$ for truth values and $\overrightarrow{0}$ for others, let set $A$ a set of objects denoted by the verb $v$. Mapping each pair $(x,t)$ gives a new representation of it :

$$\overrightarrow{v_{IN}} = \sum_{(n_i, \overrightarrow{s_i}) \in A} \overrightarrow{n_i} \otimes \overrightarrow{s_i}. \tag{12.4}$$

**Transitive verb** A formal definition of a transitive verb with 1 argument is as follows:

$$[verb_{Tr}] = \{(x,y) \mid x,y \in U \wedge x \ verb_{Tr} y\}. \tag{12.5}$$

where y corresponds to the object and x is the subject.

The category type for a transitive verb of degree 2 is $N \otimes S \otimes N$. Hence, let set $\overrightarrow{n_i}, \overrightarrow{n_j}$ the vectors denoting the pair of objects and $\overrightarrow{n_i} \otimes \overrightarrow{n_j}$ their semantic representation. (7) As for intransitive verbs, the tensor $s$ is composed of 0 excepted where the pair satisfies $n_i \ verb \ n_j$. The representation is as follows:

$$\overrightarrow{v_{Tr}} = \sum_{((n_i,n_j),\overrightarrow{s_{ij}}) \in U'} \overrightarrow{n_i} \otimes \overrightarrow{s_{ij}} \otimes \overrightarrow{n_j} \tag{12.6}$$

where $U'$ is the set of relation defined by the context of the verb.

From equation 12.16, an accurate distributional representation of the general sentence "subject verb object" can be computed with the formula :

$$\overrightarrow{sub\ verb\ obj} = (\overrightarrow{sub} \otimes \overrightarrow{obj}) \odot \overrightarrow{verb} \tag{12.7}$$

Similarly, the general sentence "subject verb" is computed thanks to the formula:

$$\overrightarrow{sub\ verb} = \overrightarrow{sub} \odot \overrightarrow{verb}. \tag{12.8}$$

In (5), E.Grefenstette and M.Sadrzadeh extend this computation to a more advanced sentence composed of an adjective and an adverb :

$$\overrightarrow{adj\ sub\ verb\ obj\ adv} = (\overrightarrow{adv} \odot \overrightarrow{verb}) \odot ((\overrightarrow{adj} \odot \overrightarrow{sub}) \otimes \overrightarrow{obj}) \tag{12.9}$$

## 12.3 Quantum Semantics

### 12.3.1 Pregroup grammar

In 1958, J.Lambek explained in (Lambek) a way to describe, in a general way, the grammar structure of language. The model he used is based on pregroup and captures the syntactic and grammatical reduction (Bankova).

His idea is to consider grammar from a mathematical approach : to each word, a type association is made. This leads to a possible and affective way of reducing sentences. According to the output, the kind of sentences is given (question, affirmation...).

A pregroup grammar $G = (P, \leq, \cdot, 1, (-)^l, (-)^r$ is a partially ordered algebra where $(P, \leq, 1, \cdot)$ is a partially ordered monoid.

A partially ordered monoid is a partially ordered set $(P, \leq)$ with an associative binary operation $\cdot$ such as $\cdot : P \times P \to P$ and an unit element 1 which remains the multiplication order preserving.

$(-)^l$ and $(-)^r$ are two unary relations called left and right adjoint such as $(-)^l : P \to P$ and $(-)^r : P \to P$. For $p^r, p^l \in P$,

$$p \cdot p^r \leq 1 \leq p^r \cdot p$$
$$p^l \cdot p \leq 1 \leq p \cdot p^l$$

## 12.4 Finite dimensional Hilbert spaces

If pregroup grammar gives access to the syntactic structure, the finite dimensional Hilbert spaces gives access to the distributive one.

According to this kind a definition, words may be seen as vectors expressed in a basis based on context. The set of vectors obtained this way can be grouped into a finite dimensional vector space bounded by linear maps called finite dimensional Hilbert spaces.

## 12.5 Frobenius algebra

In (12) and (13), an extension of the category theoretic framework is made through some elements of the Frobenius algebra. Developed by G.Frobenius in (4), a frobenius object $(O, \mu, \zeta, \Delta, \iota)$ in a monoid $(C, \otimes, 1)$ is an object $O$ of $C$ where :

$$\begin{aligned} \mu &: A \otimes A \to A \\ \zeta &: 1 \to A \\ \Delta &: A \to A \otimes A \\ \iota &: A \to 1 \end{aligned} \tag{12.10}$$

The structure $(O, \mu, \zeta)$ is called internal monoid and the structure $(O, \Delta, \iota)$ is called internal comonoid.

$$\begin{array}{ccc} A \otimes A & \xrightarrow{\delta \otimes A} & A \otimes A \otimes A \\ {\scriptstyle \mu}\downarrow & & \downarrow{\scriptstyle A \otimes \mu} \\ A & \xrightarrow[\delta]{} & A \otimes A \end{array}$$

$$\begin{array}{ccc} A \otimes A & \xrightarrow{A \otimes \delta} & A \otimes A \otimes A \\ {\scriptstyle \mu}\downarrow & & \downarrow{\scriptstyle \mu \otimes A} \\ A & \xrightarrow[\delta]{} & A \otimes A \end{array}$$

As shown, these two diagrams commute and are known as the Frobenius conditions :

$$(\mu \otimes 1_A) \circ (1_A \otimes \Delta) = \Delta \circ \mu = (1_A \otimes \mu) \circ (\Delta \otimes 1_A) \tag{12.11}$$

Let $V \in Ob(FHilb)$ be a finite dimensional Hilbert space with basis $\overrightarrow{v_i}_i$. A Frobenius algebra can be defined (Bankova) as :

$$\begin{aligned} \Delta &: V \to V \otimes V \\ \overrightarrow{v_i} &\longmapsto \overrightarrow{v_i} \otimes \overrightarrow{v_i} \end{aligned} \qquad \begin{aligned} \mu &: V \otimes V \to V \\ \overrightarrow{v_i} \otimes \overrightarrow{v_j} &\longmapsto \delta_{ij} \overrightarrow{v_i} \end{aligned}$$

$$\begin{aligned} \iota &: V \to 1 \\ \overrightarrow{v_i} &\longmapsto 1 \end{aligned} \qquad \begin{aligned} \zeta &: 1 \to V \\ 1 &\longmapsto \sum_i \overrightarrow{v_i} \end{aligned}$$

The comonoid comultiplication might be referred to by copying and the monoid multiplication by uncopying. The first one aim is to duplicate informations contained in one vector whereas the second one aim is to merge information in one vector (12).

Let $V$ be a two dimensional vector space and $\overrightarrow{v_1}, \overrightarrow{v_2}$ be a basis.

$$\Delta(a\overrightarrow{v_1} + b\overrightarrow{v_2}) = a\overrightarrow{v_1} \otimes \overrightarrow{v_2} + b\overrightarrow{v_1} \otimes \overrightarrow{v_2}$$

$$\mu(a\overrightarrow{v_1} \otimes \overrightarrow{v_1} + b\overrightarrow{v_1} \otimes \overrightarrow{v_2}) + c\overrightarrow{v_2} \otimes \overrightarrow{v_1} + d\overrightarrow{v_2} \otimes \overrightarrow{v_2} = a\overrightarrow{v_1} \otimes \overrightarrow{v_1} + d\overrightarrow{v_2} \otimes \overrightarrow{v_2}$$

## 12.6 Unification

In the last sections, mathematical concepts needed to build the model have been described. The two structures that capture the structure of the language, the distributional and syntactical one, are now going to be interpreted as compact closed category.

### 12.6.1 Pregroup

Let $G \in Preg$ be a pregroup which encodes the grammar structure of language : $G = \left\{P, \leq, \cdot, 1, (-)^l, (-)^r\right\}$ and $Preg$ a compact closed category. Objects of $P$ are grammatical types $\{s, n\}$ and morphisms are the grammatical reductions.

The sentence "John likes Mary" has for grammatical type $n$ for the nouns and $n \cdot s \cdot n$ for the verb as this one is transitive. As the pregroup is in $Preg$, a compact closed category, it is reliable to display connections through a graphic.

Figure 12.1: String diagram

Following string and reductions rules, the type of this combination of word or sentence is type.

### 12.6.2 Finite dimensional Hilbert space

One the same way as for pregroup, let define a space in which the information encoded is the distributional meaning. Let $FHilb$ be a compact close category. Objects are finite vectors and morphisms are linear maps. Let define also a left and right adjoint, $(-)^l$ and $(-)^r$ and $\eta$ and $\epsilon$ two structure preserving maps such as :

$$\epsilon^l = \epsilon^r : W \otimes W \to \Re :: \sum_{ij} c_{ij}(\overrightarrow{w_i} \otimes \overrightarrow{w_j}) \longmapsto \sum_{ij} c_{ij} \langle \overrightarrow{w_i} \mid \overrightarrow{w_j} \rangle$$

$$\eta^l = \eta^r : \Re \to W \otimes W :: 1 \longmapsto \sum_i \overrightarrow{w_i} \otimes \overrightarrow{w_i}$$

where $W \in Ob(FHilb)$ and $w_{ii}$ a basis for $W$.

Considering the morphism *state* which transforms an element of $\Re$ to an element of $FHilb$, and considering words from a sentence as vectors, it is possible to write that $\overrightarrow{word} \in W$. As $W$ is a compact closed category, a graphical representation is given :

Figure 12.2: Word representation

## 12.7 Quantizing the grammar

Now that both interpretations of the language are described through the same kind of structure, it is possible to link them together thanks to a strong monoidal functor.

This functor links the grammar part $Preg$ to the meaning part $FHilb$ :

$$F : Ob(Preg) \to Ob(FHilb) \tag{12.12}$$

This functor preserves the compact structure $F(x^l) = F(x)^l$ and $F(x^r) = F(x)^r$ for all objects $x \in Preg$. $F(x \otimes y) = F(x) \otimes F(y)$ and the maps $\eta$ and $\epsilon$ are preserved.

For basic types, it is defined as $F(s) = S$ and $F(n) = N$ leading to :

$$F(n \cdot s \cdot n) = F(n) \otimes F(s) \otimes F(n) = N \otimes S \otimes N$$

Considering the vector $w_i : I \to FHilb(p_i)$ encoding for a word $w_i$ with type $p_i$ in a sentence $w_1...w_n$ and given the type reduction $\alpha : p_1...p_n \to s$, the meaning of the sentence is given by : (10)

$$| \, w_1...w_n > := F(\alpha)(| \, w_1 > \otimes ... \otimes | \, w_n >) \tag{12.13}$$

**Example** Let consider the sentence "John likes big trees" where types are, in the order, $n$,$n^r s n^l$, $nn^l$ and $n$.

The type reduction $\alpha$ is given by :

$$n \cdot n^r s n^l \cdot nn^l \cdot n \xrightarrow{\epsilon_n^r \otimes 1_s \otimes 1_n \otimes \epsilon_n^l} s n^l \cdot n \xrightarrow{1_s \otimes \epsilon_n^l} s$$

Then, $\alpha$ can be written as :

$$\alpha = (1_s \otimes \epsilon_n^l) \circ (\epsilon_n^r \otimes 1_s \otimes 1_n \otimes \epsilon_n^l)$$

which, linked with the sentence gives :

$$\begin{gathered} F(\alpha)(\overrightarrow{John} \otimes \overline{likes} \otimes \overrightarrow{big} \otimes \overrightarrow{trees}) = \\ F((1_s \otimes \epsilon_n^l) \circ (\epsilon_n^r \otimes 1_s \otimes 1_n \otimes \epsilon_n^l))(\overrightarrow{John} \otimes \overline{likes} \otimes \overrightarrow{big} \otimes \overrightarrow{trees}) = \\ (1_s \otimes \epsilon_n^l) \circ (\epsilon_n^r \otimes 1_s \otimes 1_n \otimes \epsilon_n^l)(\overrightarrow{John} \otimes \overline{likes} \otimes \overrightarrow{big} \otimes \overrightarrow{trees}) \end{gathered} \tag{12.14}$$

Words in the sentence have the following representation according to their type :

$$\overrightarrow{John} = \sum_i a_i^{john} \overrightarrow{n_i} \qquad \overrightarrow{tree} = \sum_n a_n^{tree} \overrightarrow{n_n}$$

$$\overline{big} = \sum_j a_j^{big} \overrightarrow{n_j} \otimes \overrightarrow{n_j} \qquad \overline{likes} = \sum_{klm} a_{klm}^{like} \overrightarrow{n_k} \otimes s_l \otimes \overrightarrow{n_m}$$

Equation 12.14 can be rewritten :

$$
\begin{aligned}
&(1_s \otimes \epsilon_n^l) \circ (\epsilon_n^r \otimes 1_s \otimes 1_n \otimes \epsilon_n^l)(\overrightarrow{John} \otimes \overline{likes} \otimes \overrightarrow{big} \otimes \overrightarrow{trees}) = \\
&\alpha(\sum_i a_i^{john} \overrightarrow{n_i} \sum_{klm} a_{klm}^{like} \overrightarrow{n_k} \otimes s_l \otimes \overrightarrow{n_m} \sum_j a_j^{big} \overrightarrow{n_j} \otimes \overrightarrow{n_j} \sum_n a_n^{tree} \overrightarrow{n_n}) \\
&= (1_s \otimes \epsilon_n^l)(\sum_{ijklmn} a_{klm}^{like} a_i^{john} a_n^{tree} a_j^{big} \langle \overrightarrow{n_i} \mid \overrightarrow{n_k} \rangle \otimes s_l \otimes \overrightarrow{n_m} \otimes \overrightarrow{n_j} \langle \overrightarrow{n_j} \mid \overrightarrow{n_n} \rangle) \\
&= (1_s \otimes \epsilon_n^l)(\sum_{ijlm} a_{ilm}^{like} a_i^{john} a_j^{tree} a_j^{big} s_l \otimes \overrightarrow{n_m} \otimes \overrightarrow{n_j}) \\
&= \sum_{ijlm} a_{ilm}^{like} a_i^{john} a_j^{tree} a_j^{big} \langle \overrightarrow{n_m} \mid \overrightarrow{n_j} \rangle \otimes s_l \\
&= \sum_{ijl} a_{ilj}^{like} a_i^{john} a_j^{tree} a_j^{big} s_l
\end{aligned}
$$

## 12.8 Modeling concepts

The main force of this approach is the way of representing words. As they can be either a vector, a matrix or a high tensor, mixing them together through mathematical operations leads to more accurate outputs. The next parts will explain how to decide what kind of representation is needed and why.

### 12.8.1 Verb

Let consider the general sentence "subject verb object" and set $\overrightarrow{sub}$ the distributional representation of the subject, $\overrightarrow{obj}$ the object one and $\overline{s}$ the matrix representation of the verb. (5)

$$
\begin{aligned}
sub\ verb\ obj &= \sum_{ijk} \left\langle \overrightarrow{sub} \mid \overrightarrow{n_i} \right\rangle c_{ijk} s_k \left\langle \overrightarrow{obj} \mid \overrightarrow{n_k} \right\rangle \\
&= \sum_{ik} \left\langle \overrightarrow{sub} \mid \overrightarrow{n_i} \right\rangle c_{ijk} (\overrightarrow{n_i} \otimes \overrightarrow{n_k}) \left\langle \overrightarrow{obj} \mid \overrightarrow{n_k} \right\rangle \\
&= \sum_{ik} c_i^{sub} c_k^{obj} c_{ik} (\overrightarrow{n_i} \otimes \overrightarrow{n_k}) \qquad (12.15)
\end{aligned}
$$

The equation 12.15 can be seen as a point wise multiplication of two vectors where the left part is the Kronecker product of $\overrightarrow{sub}$ and $\overrightarrow{obj}$.

$$
(\sum_{ik} c_{ik}^{sub} (\overrightarrow{n_i} \otimes \overrightarrow{n_k})) \odot (\sum_{ik} c_{ik} (\overrightarrow{n_i} \otimes \overrightarrow{n_k}))
$$

This can be written under the following form:

$$
\overline{sub\ verb\ obj} = (\overrightarrow{sub} \otimes \overrightarrow{obj} \times \overline{verb})
$$

### 12.8.2 Adjectives

In English language, adjectives are linked to nouns in order to modify or add properties to their statement. According to it type $nn^l$ (6), it is usually followed by a nominal group and can not be used alone. From this idea, the adjective can be seen as an intransitive verb (Bolt Josef), the noun taking the place of the subject.

$$\overrightarrow{adjective} = \sum_i \overrightarrow{noun_i} \tag{12.16}$$

The iteration is made over all contexts where the adjective is contained. As output, there is a matrix containing all information about the adjective computed from a distributional point of view.

The general pattern for an adjective noun computation is displayed on figure 12.4. The associated formula is given by:

$$\begin{aligned} \overrightarrow{adjective\ noun} &= \mu(\overrightarrow{adjective} \otimes \overrightarrow{noun}) \\ &= \overrightarrow{adjective} \odot \overrightarrow{noun}. \end{aligned} \tag{12.17}$$

Figure 12.3: Representation of an adjective with the application of the $\Delta$ operation

Figure 12.4: Representation of an adjective with it noun

### 12.8.3 Adverbs

The adverbs can be seen as adjectives or intransitive verbs in a way they are linked with an object. This one can be a noun, an adjective, a verb or a preposition (6). For this reason, the type of an adverb can be multiple : $i^r i$, $i^r i o^l$, $ss^l$... In his work, Kartsaklis (7) considers only temporal and spatial adverbs as objects that modify the meaning of a verb or an adjective.

**Adverb after the verb** In the case where the adverb is after the verb, the representation is as in figure 12.5.

Figure 12.5: Representation of an adverbial sentence

As it might be induced with 12.5, the combination of the verb and the adverb can be reduced. The general reduction is as displayed on figure 12.7.

**Adverb before the verb** The case where the adverb is before the subject is harder than in the previous case. Here, the adverb cuts the direct link between the subject and the verb. Hence, to have the right to compute the sentence using the same algebra, a linear mapping has to be performed (11).

Figure 12.6: Verb - Adverb combination

The adverb is seen as an operator corresponding to the map $f : S \to S$ (7). The graphical representation is as follows :

Figure 12.7: Adverb - Verb combination

## Bibliography


[Bankova] Bankova, D. Comparing meaning in language and cognition.

[Bolt Josef] Bolt Josef, Coecke Bob, G. F. L. M. M. D. P. R. Interacting conceptual spaces.

[Edward Grefenstette] Edward Grefenstette, M. S. Experimenting with transitive verbs in a discocat.

[4] Frobenius, F. G. (1903). *Theorie der hyperkomplexen Großen.*

[5] Grefenstette, E. and Sadrzadeh, M. (2011). Experimental support for a categorical compositional distributional model of meaning. *CoRR*, abs/1106.4058.

[6] J.Lambek (2008). *From Word to Sentence.*

[7] Kartsaklis, D. (2015a). Compositional distributional semantics with compact closed categories and frobenius algebras. *CoRR*, abs/1505.00138.

[8] Kartsaklis, D. (2015b). Compositional distributional semantics with compact closed categories and frobenius algebras. *arXiv preprint arXiv:1505.00138.*

[Lambek] Lambek, J. The mathematics of sentence structure. *American mathematical monthly, 154-170.*

[10] Piedeleu, R., Kartsaklis, D., Coecke, B., and Sadrzadeh, M. (2015). Open system categorical quantum semantics in natural language processing. *CoRR*, abs/1502.00831.

[11] Preller, A. and Sadrzadeh, M. (2010). Bell states and negative sentences in the distributed model of meaning. In In P. Selinger B. Coecke, P. P., editor, *Electronic Notes in Theoretical Computer Science, Proceedings of the 6th QPL Workshop on Quantum Physics and Logic.*

[12] Sadrzadeh, M., Clark, S., and Coecke, B. (2014a). The frobenius anatomy of word meanings I: subject and object relative pronouns. *CoRR*, abs/1404.5278.

[13] Sadrzadeh, M., Clark, S., and Coecke, B. (2014b). The frobenius anatomy of word meanings II: possessive relative pronouns. *CoRR*, abs/1406.4690.

# Chapter 13

# Bridging Distributional Semantics and Neuroscience

**Keywords:** neuroscience, neurolinguistics, wernickes area, brockas area, brain scanning, neuroimaging, fmri, meg, eeg.

# Chapter 14

# Conclusion

## 14.1 A brief overview of the course. Conclusion

## 14.2 A survey of research labs in distributional semantics

## 14.3 A survey of useful resources, datasets and libraries