ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Compositionality through Projection

Master Thesis

Dimitrios Leventeas

October 26, 2013

Advisors: Prof. Dr. T. Hofmann, Dr. M. Ciaramita
Department of Computer Science, ETH Zürich

Abstract

We investigate ways of representing textual content in sparse vectorial representations to improve semantic similarity modelling for tasks such as word and phrase similarity prediction. In particular, we are interested in investigating compositional approaches to semantic similarity via randomization and hashing techniques. We also offer an experimental evaluation of the techniques that we used. The project is programming intensive (mostly C++) and it uses computational infrastructure and data from Google.

We focus on the following parts:

1. Carry out a thorough survey of randomization and hashing techniques for semantic similarity.

2. Implement baseline approaches to string (with focus on query) similarity, build an appropriate evaluation framework.

3. Design, implement and evaluate two variants of compositional models (vector addition, pointwise multiplication).

4. Implement/investigate use of randomization and hashing techniques.

5. Investigate features of varying linguistic complexity (term frequency, Positive Pointwise Mutual Information).

The described system achieves state of the art performance in the evaluation task of query similarity.

Contents

Chapter 1

Introduction

In this chapter, we explain briefly what *distributional semantics* are, we state their application in the domain of query similarity and, finally, we provide an overview of the rest of the thesis.

1.1 Background and Motivation

When we learn a new language, the advice that we can understand the meaning of a word by its context is often repeated. This notion was popularized in scientific literature, and especially in the field of linguistics by Harris and his work with title *Distributional Structure* [27]. Nevertheless, we can trace the idea further back to Wittgenstein "the meaning of a word is its use in language" [55] or even Frege "never ask for the meaning of a word in isolation but only in the context of a statement" [22].

Joos [30] stated more formally the principle of distributional semantics by defining the linguistic meaning of a morpheme as "the set of conditional probabilities of its occurrences in context with all other morphemes" which Harris later investigated in [27]. An important milestone has been [37] by Osgood, Suci & Tannenbaum who were the first to use a Euclidean vector space for semantic representation.

The most important use case for vector-based represetnation has been in the context of information retrieval, namely the vector space model first developed by Salton [46]. Other applications include modeling of word synonymy [40] and semantic relations [36, 52].

1.2 Problem Statement

We will apply the ideas of distributional semantics to the problem of extracting query similarities. Given pairs of queries, we want to assess how similar

they are. To evaluate our system, we use predefined sets of pairs of queries that are rated manually according to their degree of similarity.

Before being able to argue about query similarities, we have to define and specify some notions. First of all, in order to arrive query similarities, we need a representation for the queries and a notion of similarity. Because the distributional hypothesis is about words and not queries, we need a way to induce similarity from words to queries. Even before that, we need to specify what context is and how we can represent the meaning of a word depending on its context.

1.3 Outline

We start by reviewing some of the related literature in chapter 2 and by introducing the necessary notions that we use in our system. In some cases, we also discuss briefly some alternatives that could be used in some components of the system. The implemented system is described in chapter 3. The high level description of our system in chapter 3 explains the main design decisions and how we adjusted the system to work in a distributed environment. In the chapter 4, we evaluate the implemented system according to two datasets of golden standards and we compare it with other related systems in the literature. Finally, in chapter 5, we provide a brief summary of our work and suggest some possible next steps.

Chapter 2

Background and Related Work

In the context of the thesis, we focus on three main areas.

1. *Distributional semantics*: what is the context of a word and how it defines the meaning of a word.

2. *Vector representation*: reducing the dimensionality of the space that represents the meaning of a word according to the distributional hypothesis.

3. *Compositional process*: how we can combine the obtained word representations to infer the meaning of phrases and word sequences (queries, sentences).

We review some of the related literature in the corresponding sections in the rest of the current chapter and we also provide the fundamental ideas of our system.

2.1 Distributional Semantics

> You shall know a word by the company it keeps. (Firth 1957)

The main idea is that words with similar meanings occur in similar context. Therefore, word co-occurrence statistics can provide a natural basis for semantic representation. The notion was introduced several years ago [20, 27]. More recently, the idea underlies computational models such as Latent Semantic Analysis (LSA) or Hyperspace Analogue to Language (HAL) that we will define later. The distributional hypothesis is discussed in more detail in [44].

2.1.1 Computational model

We create a vector representation for each *word* in our corpus according to co-occurring words (context). In order to do so, we have to define the following:

- *context:*
  - *size*: number of neighbouring words.
  - *shape*: weight specification and distribution of the neighbouring words.
- *token*: We mentioned words before, which is an oversimplification. We are looking for the atomic units of meaning. Should we consider *white house* as one or two terms? What about United *States of America*?
- *corpus*: The collection of documents that we use for training.
- *vector representation:* How to arrive from co-occurrence statistics at a vector representation is related to the meaning of a word.

Examples of context window definitions

Examples of interesting variations of the shape of a window are:

- Restrict to one side only (left or right from the target word).
- Flat windows (all words within the context window count equally).
- Closest words have greater importance (for instance a triangular or Gaussian window).
- Offset window (exclude close words because they may be required only due to syntactic reasons).

Another variation of the context window could be captured based on the grammatical relations in which a target word occurs in a *parsed* corpus (syntactic parsing).

2.1.2 Limitations

In the context of the thesis, we built upon the distributional hypothesis. Nevertheless, we should at least mention briefly some limitations of the distributional semantics:

1. *Homographs*. It is *not* obvious how to disambiguate them.
2. *Quality of word representation*. The way we usually measure the quality of a word representation depends on the evaluation task.
3. *The Symbol Grounding Problem.* It is about how words obtain their meaning. See [26] for a description.

Simple variations of the model

We can slightly vary our model by apply as a preprocessing step to our corpus:

- *stemming* or *lemmatisation*
- *stop words removal*
- *truncate* very low frequency words (statistically unreliable)

Throughout our discussion, the streaming model for the input data is used. That is, we don't store the whole corpus in memory, but we can process it as it arrives. Moreover, we limit ourselves to a single pass over the data.

In the following sections, we will define models using different values of the above parameters. First, we will see how we can compute the co-occurrence statistics matrix that is common to all the approaches once the above parameters are fixed.

2.1.3 Parameter estimation

Each word t is represented by a vector. Each component $w_{i,t}$ of the vector corresponds to another word in our collection. For each component $w_{i,t}$, we increase the corresponding counter each time we find the word t in the context that includes word i and according to the weight distribution.

When we finish the process, we normalize the vectors so the sum of all components for a specific vector is 1. These are co-occurrence statistics vectors and they contain the frequencies of other words that we encounter according to our context definition.

2.1.4 Quality of vector representation

We need a way to evaluate how well the vector representations of a model represent the meaning of a word. To do so, there are some internal and external criteria. Usually, though, the choice of a metric depends heavily on the application.

Internal

- *Statistical reliability.*
- *Dimensionality.* Smaller/sparser vector allow for faster operations.
- *Compositional similarity.* Given a compositional function $\oplus$ (see section 2.3), how close are the representations of a phrase and the composition of the constituents of the phrase (for instance: $\overrightarrow{v,u}$ compared to $\overrightarrow{v \oplus u}$ where u, v are words and v, u is a phrase of v followed by u.)

External

- *Synonyms:* A word should be closer to the representation of its synonyms than to other words. In previous studies, the TOEFL test has been used as a benchmark.
- *Semantic Categorization:* Words should be closer to their semantic category (e.g. vegetables, building...) than others.
- *Phrasal Similarity:* Given a compositional function, similar phrases should be closer than non-similar phrases.
- *Dictionary definitions involving relative clauses* should be matched to the corresponding noun.
- *Word close to its definition*: A word should be closer to its definition than to the definition of alternative words or to its definition with some words replaced by irrelevant ones.

2.1.5 Distance (between semantic vectors)

We have to define a distance metric when we try to quantify how close some words are. In this respect, the distances on this section have been considered in related literature. See [10] for a related study that surveys the following metrics in a specific context.

We define two vectors $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$, where n is their dimension. Correspondingly, we have:

Minkowski distances

- City Block (Manhattan)

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i| \tag{2.1}$$

- Euclidean

$$d(a,b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{2.2}$$

Definition 2.1 *The Minkowski distance of order $p \geq 1$ between two points on Euclidean space is a metric defined as:*

$$d(a,b) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{\frac{1}{p}} \tag{2.3}$$

Cosine

$$d(a,b) = \frac{a \cdot b}{||a|| \cdot ||b||} \tag{2.4}$$

$$= \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.5}$$

Values close to 1 mean that a, b are related while values close to 0 mean that they are unrelated. Usually, and throughout this text, we assume that a vector can only have non-negative values on its components. Therefore, the cosine of two vectors in this case can take values only in the interval $[0, 1]$.

Information theoretic measures

On this subsection, we use the fact that the vectors are probabilities. That allows for information theoretic measures such as:

- Hellinger distance

$$d(a,b) = \frac{1}{\sqrt{2}} \sqrt{a} - \sqrt{b} \tag{2.6}$$

$$= \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{n} (\sqrt{a_i} - \sqrt{b_i})^2} \tag{2.7}$$

- Bhattacharya

$$d(a,b) = -\ln \left(\sum_{i=1}^{n} \sqrt{a_i \cdot b_i} \right) \tag{2.8}$$

- Kullback – Leibler

$$d(a,b) = \sum_{i=1}^{n} a_i \ln \frac{a_i}{b_i} \tag{2.9}$$

2.1.6 Alternative component weighting

It is not necessary to store probabilities in the vectors. In study [10], some alternatives are also explored.

- Positive Pointwise Mutual Information (Positive PMI)

$$pmi(a;b) = \log \frac{p(a,b)}{p(a) \cdot p(b)} \tag{2.10}$$

$$= \log \frac{p(a|b)}{p(a)} \tag{2.11}$$

  where negative values are replaced by 0. It can compare the conditional probabilities for each word in a specific context to the marginal probability of occurrence of the word.

- Odds ratio.

$$\frac{\frac{a}{1-a}}{\frac{b}{1-b}} = \frac{a(1-b)}{b(1-a)} \tag{2.12}$$

  Again, with similar semantics as the pointwise mutual information, that is, compare the actual conditional probabilities for each word in a specific context to the expected probability of occurrence of the word.

In study [10] conducted with using Positive PMI and window size of one, the authors got the best reported results. The same study concluded that removing very low frequency words or very high frequency words (stopwords) leads to worse performance in their experiments.

2.1.7 Distributional Similarity

We briefly mention some systems that utilize the ideas that we discuss in this chapter. These systems consist of essentially two steps:

1. Compute vector representation of a query.
2. Similarity of two queries is their cosine.

We focus on how to compute the vector representation of a query. After the description of the two systems, we compare them. In the next chapter, we explain our approach and the differences with these two systems.

A Web-based Kernel Function for Measuring the Similarity of Short Text Snippets (Sahami and Heilman)

We describe the algorithm used in [41].

Large-scale Computation of Distributional Similarities for Queries (Alfonseca, Hall and Hartmann)

We describe the algorithm used in [4].

Algorithm 1 Distributional Similarities for Queries

Require: Query x. Parameters n, m.
Ensure: Vector representation of x.

1: Submit x and let $R(x)$ be the set of n retrieved snippets $d_1, d_2, \ldots, d_n$.
2: Compute tf-idf term vector u_i for each snippet $d_i \in R(x)$.
3: Truncate each vector u_i to include its m highest weighted terms.
4: Let $C(x)$ be the centroid of the L_2 normalized vectors u_i:

$$C(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{u_i}{||u_i||_2}$$

5: Let $QE(x)$ be the L_2 normalization of the centroid $C(x)$:

$$QE(x) = \frac{C(x)}{||C(x)||_2}$$

Algorithm 2 Large-scale Computation of Distributional Similarities for Queries

Require: Query $x = [w_1, w_2, \ldots, w_n]$. Parameter m.
Ensure: Vector representation of x.

1: For each w_i collect all words that appear close to w_i in the web corpus (*not snippets*).
2: Compute frequencies of context words. (*not tf-idf*).
3: Truncate to its m highest weighted terms.
4: Compose using the geometric mean (*instead of arithmetic mean*).
5: Apply the χ^2 test as a weighting function to measure whether the query and the contextual feature are conditionally independent. (*new step*).

In parentheses, we have put the differences with the algorithm described in the previous part of 2.1.7 section (Distributional Similarities for Queries). We should note here that both approaches use vectors with $m = 50$ non-zero components.

2.2 Vector representation

The general idea behind word space models is that words are represented by context vectors as motivated by the distributional hypothesis.

The use of vector space model is a standard approach today. Some of the reasons that motivated towards this model are:

- There is extensive mathematical theory (linear algebra) on how the

vector spaces work.

- They make semantics computable.
- Vectors can be created automatically from a corpus.
- They allow the representation of gradations of meanings.
- The usefulness of the model has been validated experimentally.
- Their geometric metaphor seems plausible, even though they are a purely descriptive approach.

Two examples of these models are Latent Semantic Analysis (LSA) and Hyperspace Analogue to Language (HAL).

2.2.1 Latent Semantic Analysis (LSA)

LSA creates a term-by-document matrix. It is based on the distributional hypothesis that terms with similar meanings tend to occur in the same context. In the following paragraph, we describe LSA and how we can apply it to find how similar two words are.

We construct a matrix where each row represents a word and each column represents a document. Given the definition of a document (a specific sequence of words in our corpus), we construct a matrix containing the number of occurrences of each word in each document. We expect that the resulting matrix will be sparse (not every word is contained to every document). Then, we use Singular Value Decomposition (SVD) (see below) in order to reduce the dimensionality of the matrix (only number of columns, the number of rows remains the same). This step of dimensionality reduction is important because after the compression of the matrix using SVD, we get a better semantic matrix where words that are close in meaning are mapped together. Finally, words are compared by using the cosine similarity (see section 2.1.5). For details on LSA and some philosophical implications, see [33].

We note that the terms Latent Semantic Analysis (LSA) and Latent Semantic Indexing (LSI) are synonyms. LSI was the original term due to the application of the technique in document indexing and information retrieval. Then, LSA as term became popular at it was in a broader scope, for instance in recognizing synonyms.

Singular Value Decomposition

The Singular Value Decomposition (SVD) for a matrix A is defined as:

$$A = U\Sigma V^* \tag{2.13}$$

where U is a mxm (real or) complex unitary matrix (left-singular vectors), Σ is an mxn rectangular diagonal matrix with non-negative real numbers on the diagonal (singular values), and V^* is an nxn (real or) complex unitary matrix (right-singular vectors).

A practical application of SVD (for example in LSA and in HAL) is the low-rank matrix approximation, where we truncate the matrix Σ. It can be shown (Eckart - Young theorem) that the approximation is based on minimizing the Frobenius norm of the difference between the initial matrix A and the approximation $\hat{A}$. SVD can be thought as a technique for statistical dimensional reduction where the eigenvalues indicate the importance of the corresponding left and right singular vectors.

Definition 2.2 (Frobenius norm) *The Frobenius (or Hilbert-Schmidt) norm for a matrix $A^{m,n}$ can be defined as:*

$$\begin{aligned} ||A||_F &= \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{i,j}|} \\ &= \sqrt{trace(A^*A)} \\ &= \sqrt{\sum_{i=1}^{\min\{m,n\}}\sigma_i^2} \end{aligned}$$

where A^* denotes the conjugate transpose of A, σ_i are the singular values of A, and the trace function is used.

The computational complexity of performing SVD of a dense matrix is $O(mn^2)$ floating-point operations. The SVD is an expensive operation and it does not scale very well to large matrices. There is a version for sparse matrices [7], as the ones that we usually encounter in natural language processing, but it is still costly. Moreover, if we update our corpus, we have to follow incremental approximations [47]. Finally, it fails to avoid the initial co-occurrence matrix which may be very big for practical applications.

2.2.2 Hyperspace Analogue to Language

HAL creates a term-by-term matrix. The model is similar to LSA. We focus on their differences. In Hyperspace Analogue to Language (HAL) we define as document only words in the context of a target word. Moreover, the increase in a component in the matrix is inversely proportional to the distance between two words. We compute the similarities between two words using cosine, similarly to LSA.

2.2.3 Problems and solutions

There are some issues with the vector representation of words in the context of Natural Language Processing.

- *Scalability:* High dimensionality of the vectors.
- *Sparse data:* Only few of the components in a vector are non-zero.

In order to resolve these issue, LSA and HAL use SVD. Unfortunately, as we already mentioned in section 2.2.1, it is a costly operation.

2.2.4 Random Indexing

The basic idea of Random Indexing (RI) is to accumulate context vectors. It is backed up by the intuition of the Johnson-Lindenstrauss lemma [29] that we can project points of a vector space into a randomly selected subspace of sufficiently high dimensionality and approximately preserve the pairwise distances of the vectors in the initial space. To make the idea more concise, we can think of the initial matrix $A^{w,d}$ projected to a subspace $B^{w,k}$ by a random projection $R^{d,k}$, where d, k, w are dimensions and $k < d$. According to the Johnson-Lindenstrauss lemma, $k = O(\log w)$, independent of the initial d. In other words, the size of the vectors that we call *index vectors* depends only on the number of different words in our corpus.

A sketch of the Random Indexing algorithm is:

1. Each context word is assigned a vector called *index*.
2. We scan through the corpus and each time a word occurs in a context, we add its index to the target word's representation.

The *index* is generated randomly according to some distribution (see the analysis in [1]). It is sparse, high dimensional, nearly orthogonal and ternary (possible values for a component are $(+1, 0 - 1)$). It is nearly orthogonal because as it was proved in [39], there are many more nearly orthogonal vectors than truly orthogonal ones in a high dimensional space.

As we can notice, using Random Indexing we can avoid creating the big matrix that exists initially in methods like LSA or HAL since we accumulate the index vectors from the beginning.

Random Indexing hash the following nice properties:

- It is an incremental method.
- The exact dimensionality d is a design parameter.
- It is not required to construct the initial large co-occurrence matrix.
- It is independent of the definition of the context.

Random Indexing has been successfully used in practice in several studies. Examples of these studies are [31, 32, 43, 45].

2.2.5 Theory behind Random Projections

We are discussing the theory behind random projections. We split the section in three parts, the first two based on two publications and the last section about some recent work. All of them are based on the main idea behind Johnson-Lindenstraus (JL) lemma. We will skip the description of how the original idea of Johnson - Lindenstrauss works for the sake of brevity.

Lemma 2.3 (Johnson–Lindenstauss lemma) *Given* $0 < \varepsilon < 1$, *an arbitrary set* P *of* n *points in* $\mathbb{R}^d$, *and a number* $k > 8\ln(n)/\varepsilon^2$, *there is a linear map* $f : \mathbb{R}^d \to \mathbb{R}^k$ *such that*

$$(1-\varepsilon)||u-v||^2 \leq ||f(u)-f(v)||^2 \leq (1+\varepsilon)||u-v||^2 \tag{2.14}$$

for all $u, v \in P$.

Database-friendly random projections: Johnson - Lindenstrauss with binary coins

We already mentioned in section 2.2.4 that the main idea behind Random projections comes from the insight of the Johnson-Lindenstraus (JL) lemma. In 2003, Achlioptas in [1] formulated an improved version of the same idea.

The question is how we could embed a high dimensional set of points in a Euclidean space of a lower dimensionality without causing too much distortion in the pairwise distances of the vectors in the resulting space.

We know (see section: 2.2.1) that SVD guarantees:

$$||A - A_k||_F \leq ||A - D|| \tag{2.15}$$

where A is the initial matrix, A_k is the rank k approximation of A and D is any matrix of rank k. Unfortunately, this optimality (global property) does not guarantee a bound on the pairwise distances of the vectors (local property).

Before going to the theorem that Achlioptas proved, let us try a very simple embedding in a lower dimensional space. We can pick only the first k of the original components of the vectors. Unfortunately, if two vectors differ only in some of the dimensions that we discarded, then they could be very far apart in the original space while their distance is decreased significantly in the embedding space. That would not be an issue if we knew that all of the components in the vectors are roughly of the same importance. Therefore, we can simply resolve this issue by applying a random rotation to the original vectors.

Theorem 2.4 *Let P be an arbitrary set of n points in R^d, represented as an nxd matrix A. Given $\varepsilon, \beta > 0$ let*

$$k_0 = \frac{4 + 2\beta}{\varepsilon^2/2 - \varepsilon^3/3} \log n.$$

For integer $k \geq k_0$, let R be a dxk random matrix with $R(i,j) = r_{ij}$, where $\{r_{ij}\}$ are independent random variables from the following probability distribution:

$$r_{ij} = \sqrt{3} \times \begin{cases} +1 & \text{with probability } 1/6 \\ 0 & \text{with probability } 2/3 \\ -1 & \text{with probability } 1/6 \end{cases}$$

Let

$$E = \frac{1}{\sqrt{k}} AR$$

and let $f : \mathbb{R}^d \to \mathbb{R}^k$ map the ith row of A to the ith row of E. With probability at least $1 - n^\beta$, for all $u, v \in P$

$$(1 - \varepsilon)||u - v||^2 \leq ||f(u) - f(v)||^2 \leq (1 + \varepsilon)||u - v||^2.$$

Each column of the projection matrix R provides an estimation of the original vector length. We maximize the (mutual) information that we get by making these vectors (columns) independent with equal weight (orthonormal). Because having orthonormal vectors is expensive in terms of dimensionality, we use nearly orthogonal ones, as defined above.

As an open problem from this study was left the case where the initial vectors are guaranteed to live in some low-dimensional space.

The fast Johnson - Lindenstrauss transform and approximate nearest neighbors

The key idea is that we want to embed matrix A using a sparse random projection R. We may have a problem when A is also sparse. To solve it, we use Fourier transformation and randomization by exploiting the "Heisenberg principle". We see more precisely the intuition in the following paragraphs.

The Johnson-Lindenstrauss Lemma shows that for n points in Euclidean space, we need a subspace of $O(\log n) = c(\varepsilon) \log n$ dimensions with distortion ε. Alon [6] showed that $c(\varepsilon)$ can not be reduced by much. More precisely, Alon proved that $c(\varepsilon) = \Omega\left(\frac{1}{\varepsilon^2 \log(1/\varepsilon)}\right)$ even for a simplex.[1]

[1] A k-simplex has $k + 1$ affinely independent points, that is, and the differences of each one (apart from the first) minus the first one are linearly independent. In other words, it is a generalization of triangle in more than 2 dimensions.

Because of this lower bound, the research was focused on making the projection matrix more sparse. Already, as we saw in the previous section, Achlioptas provides a sparse matrix where the density has been reduced by a constant factor. Increasing the sparsity of the projection matrix even more could increase the distortion of a sparse matrix.

The study in [2] tried to overcome the problem that arises when we use sparse matrices by employing the *Heisenberg principle* from harmonic analysis: A signal and its spectrum cannot both be concentrated. With this idea in mind, the next step looks very intuitive. We apply (precondition) Fast Fourier Transformation (FFT) to the random projection matrix. If we had sparse vectors, according to Heisenberg's principle, we will enlarge their support. Because the reverse may happen (dense vectors with small support), we can randomize the Fast Fourier Transform.

We can now formalize the intuition in the previous paragraph by describing[2] how we can obtain a random embedding Φ such that:

Theorem 2.5 *For a fixed set A of n vectors in $\mathbb{R}^d$, $\varepsilon < 1$, and $p \in \{1,2\}$ with probability at least $2/3$, the following are true:*

1. *For all $x \in A$,*

$$(1-\varepsilon)\alpha_p||x||_2 \leq ||\Phi x||_p \leq (1+\varepsilon)\alpha_p||x||_2 \tag{2.16}$$

   where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$ where $k = c\varepsilon^{-2}\log n$ for some global c.

2. *The mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^k$ requires*

$$O(d\log d + \min\{d\varepsilon^{-2}\log n, \varepsilon^{p-4}\log^{p+1} n\}) \tag{2.17}$$

   operations.

In order to obtain the matrix Φ that the theorem 2.5 requires, we multiply three matrices $\Phi = PHD$.

- The elements of $P^{k\times d}$ are independently distributed:

$$P_{ij} = \begin{cases} 0, & \text{with probability } 1-q \\ \mathbb{N}(0, q^{-1}) & \text{otherwise} \end{cases}$$

  where $\mathbb{N}(0, q^{-1})$ is the normal distribution with mean value 0 and variance q^{-1}. The sparsity constant is $q = \min\left\{\Theta\left(\frac{e^{p-2}\log^p n}{d}, 1\right)\right\}$.

[2]The theorem 2.5 and its proof can be found in [2].

- The elements of H are given by:

$$H_{ij} = \frac{(-1)^{<i-1,j-1>}}{\sqrt{d}}$$

where $< i, j >$ is the dot product (modulo 2) of the m-bit vectors i, j expressed in binary. The matrix H can be characterized as normalized Walsh-Hadamard matrix. It is known for H the following holds on Euclidean space:

$$||Hx|| = ||x||_2$$

In other words, H is an isometry on l_2^d.

Moreover, we can show that the following recurrence relation with $H_1 = (1)$ holds for $H = H_d$.

$$H_d = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{pmatrix}$$

Thus, applying H takes $O(d \log d)$ time by divide-and-conquer.

- The matrix $D^{d \times d}$ is a diagonal matrix where:

$$D_{ii} = \begin{cases} -1, & \text{with probability } 1/2 \\ 1, & \text{with probability } 1/2 \end{cases}$$

Applying D to some vector x takes time $O(d)$.

Using Φ, the mapping Φx of any vector $x \in \mathbb{R}^d$ can be computed in time $O(d \log d + qd\varepsilon^{-2} \log n)$.

Recent improvements

Recently, Matousek [34] showed that we can combine this idea with Achlioptas approach using ± 1 matrices. More precisely, he showed that we can have a sparse matrix Φ such that the non-zero elements are ± 1 instead of normally distributed. In study [3], an improvement of the running time to $O(d \log k)$ for $k = O(d^{1/2-\delta})$ was shown.

Another important study is [17] where using hashing and local densification, they construct a sparse projection matrix with about $O(\frac{1}{\varepsilon})$ non-zero entries per column, or more precisely $O\left(\frac{1}{\varepsilon} \log^2(\frac{k}{\delta}) \log(\frac{1}{\delta})\right)$ non-zero entrie per column with $k = O(\frac{1}{\varepsilon^2} \log(\frac{1}{\delta}))$. One important insight from their paper is that having each dimension mapped to exactly one hash bucket and the lack of self-collisions leads to a reduction in the variance of the cross-product error.

We summarize their main result omitting the definition of the constants k, c that depend on δ, ε. The main result can be found on section 2 of [17].

Let $r = \{r_j\}_{j \in [cd]}$ be a set of independent and identically distributed (i.i.d) random variables such that for each $j \in [cd]$, $Pr[r_j = 1] = Pr[r_j = -1] = 1/2$. Let $\delta_{\alpha,\beta} = 1$ if and only if $\alpha = \beta$ and zero otherwise. Let $n_{nz}(x)$ denote the number of non-zero entries in vector x.

Let $h' : [cd] \to [k]$ be a hash function chosen uniformly at random and let $H' \in \{0, \pm 1\}^{k \times cd}$ be defined as $H'_{ij} = \delta_{ih'(j)r_j}$. We define $P \in \{0, \pm 1\}^{cd \times d}$ as

$$P_{ij} = \begin{cases} \frac{1}{\sqrt{c}} & \text{for } (j-1)c + 1 \leq i \leq jc, \\ 0, & \text{otherwise} \end{cases}$$

Now, we let $\Phi = H'P$.

Theorem 2.6 *For any given vector $x \in \mathbb{R}^d$, with probability $1 - 4\delta$, Φ satisfies the following property:*

$$(1 - \varepsilon)||x||_2^2 \leq ||\Phi x||_2^2 \leq (1 + \varepsilon)||x||_2^2 \tag{2.18}$$

The time required to compute Φx is $O(\frac{1}{\varepsilon} \log^2(\frac{k}{\delta}) \log(\frac{1}{\delta} \cdot n_{nz}(x))$.

The (very technical) proof of the above theorem is omitted.

2.2.6 Hashing kernels

Hashing can be used as a dimensionality reduction technique. *Hashing kernels* employ the *hashing-trick*.

Definition 2.7 (Hashing-trick) *Project (hash) a high dimensional (input) vector $x \in \mathbb{R}^n$ into a lower dimensional feature space $\mathbb{R}^m$.*

There are two main variants of the hash kernels: biased (example in [48]) and unbiased ones (see [53]). We start with the biased ones.

Definition 2.8 (Biased hash kernels) *We denote a hash function $h : \mathbb{N} \to \{1, \ldots, m\}$. Using that function, we define:*

$$\varphi_i(x) = \sum_{\forall j : h(j) = i} x_i \tag{2.19}$$

Then, we can compute the (approximate) inner product for vectors x_1, x_2 as:

$$< x_1, x_2 > \approx < \varphi(x_1), \varphi(x_2) > \tag{2.20}$$

In order to obtain unbiased hash kernels, sometimes instead of performing an addition, we do a subtraction. Somewhat more formally:

Definition 2.9 (Unbiased hash kernels) *We denote a hash function* $h : \mathbb{N} \rightarrow \{1, \ldots, m\}$. *We also need a hash function* $\xi : \mathbb{N} \rightarrow \{-1, 1\}$ *for the sign.*

$$\varphi_i(x) = \sum_{\forall j : h(j) = i} \xi(i) x_i \tag{2.21}$$

Then, we can compute the (approximate) inner product for vectors x_1, x_2 *as:*

$$< x_1, x_2 > \approx < \varphi(x_1), \varphi(x_2) > \tag{2.22}$$

In contrast to random projections, the hashing trick does not require to store the projection matrices (we only need to know the hash functions that we use) and it preserves sparsity in the initial matrix.

For proofs on the approximation bounds of the inner products, see [48] and [53]. It is also interesting how these ideas recur in Count-min sketch that we see in section 2.2.7.

A heuristic but very fast method for feature extraction using hashing on words is presented in [21].

2.2.7 Sketch algorithms

Creating a *sketch* of the input is popular in streaming models. A sketch can be thought as a "rough" description of the input data in limited space. A seminal paper [5] on the space complexity of the frequency moments by Alon, Matias and Szegedy won the Gödel prize in 2005. On the current section we will focus on count-min sketch [16], which can be thought of as a generalization of bloom filters. We begin with the idea of bloom filters as an introduction, proceed with the description of Count-Min sketch and at the end of the section, we explain what the pairwise independent hash functions are, which we need to implement the aforementioned probabilistic data structures.

Bloom filter

A bloom filter [8] is a bit array that has m bits initialized to 0. It supports only insertion (*increase*) of an element and *query*, that is, whether the element has been seen so far. It uses m universal hash functions to set/query the bits of the array according to the input element in the following way. Each of the m hash functions returns one position which should be set to 1. If we add an element to the Bloom filter, we set the corresponding positions to 1. If we query for an element, we require that all the corresponding positions were already set to 1.

That is all about the description of how a bloom filter works. Now, we can focus on why they work by making some important observations and

skipping many of the details. First of all, it is impossible to have a *false negative*, that is, to query for an element and get an answer of non existence in the set, even though the query exists. The reason is that if we set some bits because we read an element x, we are sure that all these bits are set in every subsequent query. On the other hand, if some bits were set, maybe that happened from a a combination of input elements and therefore we may get a *false positive* for an element that was not observed in the input stream but all of its corresponding bits in the array are set. That is the main drive behind the requirement of independent hash functions.

Because of the way bloom filters work, they are only suitable for representations of sets and not multisets because the number of times we observe an element makes no difference to the internal representation on the array.

In order for Bloom filters to support the *delete* operation, the counting filters were introduced [19]. Instead of storing just one bit (presence or absence), the counting filters store a counter of how many times we have observed an element. In case we want to delete one element, we decrease the number stored in the appropriate position. This idea, with some modifications lead us naturally to the idea of count-min sketch that we describe on the next section.

A survey on Bloom Filters can be found on [9].

Count-min sketch

We will use ideas from the previous section. Firstly, though, we define the problem that Count-min sketch tries to solve.

We assume that we have a large set U of identifiers (for instance words in a corpus). We are interested in the counts of the identifiers $u \in U$. We want to support operations such as: $increase(u, c)$, $decrease(u, c)$ and $query(u)$ assuming always that the counts never become negative.

The main idea is that we use m universal hash functions $h_i, 0 \leq i < m$ and m vectors $A_i, 0 \leq i < m$ of equal length. The function $increase(u, c)$ is implemented for every $0 \leq i < m$ by $A_i[h_i(u)] = A_i[h_i(u)] + c$ and *decrease* correspondingly by $A_i[h_i(u)] = A_i[h_i(u)] - c$. We answer $query(u)$ by $\min_i^k A_i[h_i(u)]$. The initial value of $A_i[j] = 0, \forall i, j < m$.

The name Count-min sketch comes from the way we answer a *query*. The rationale is simple. A hash function may map some elements to the same key, causing what we call a collision. Therefore, looking at a the value corresponding to a key from a table of a specific hash function, we can only overestimate the number of elements that we have observed so far. To alleviate this problem, from all the estimations that the hash functions provide, we take the minimum. We still have the guarantee that the estimation we

get is not less that the true value but we try to keep the value that is closer to the actual one. In our case, this value corresponds to the smallest one.[3]

An interesting application of Count-min sketch is to compute the inner product between two vectors. We can do that by multiplying the corresponding A_i of the two vectors together and then getting the minimum. For details, see the Theorem 3 in [16].

Count-min sketch with Conservative Update

We have already discussed that a count-min sketch only overestimates the counters of the elements. That gives the following idea. Whenever we use the $increase(u, c)$ function, submit a $query(u)$ and get the current estimation for u, which we call $\hat{c}(u)$. Then, we modify the procedure of the $increase(u, c)$ function to be $A_i[h_i(u)] = \max\{A_i[h_i(u)], \hat{c}(u) + c\}$. The idea is that since we always overestimate the counts, we don't have to increase them more than necessarily; more than their current estimation and the number that should be increased on current step.

We should note that if we use conservative update, the operation of $decrease(u, c)$ is no longer possible since we do not know if we should subtract or not c for a specific component. Moreover, the bias now depends on the order of the insertions, as it can be seen with a simple example.

Example 2.10 (The bias of Conservative Update depends on the order) *We consider as input data:*

wordA	3
wordB	5
wordC	4

Figure 2.1: Input data.

We use two hash functions with two values each one (codomains: [1,2], [a, b]). We assume that the word mapping is:

wordA	$\{1, a\}$
wordB	$\{2, a\}$
wordC	$\{1, b\}$

Table 2.1: The mapping of a word according to the hash value of the first and the second hash function, correspondingly.

[3]Recall our assumption that the counts never become negative, that is, we can not remove an element that we have not observed yet.

Assuming that the order of the insertion is: wordA, wordB, wordC, the results are depicted in table 2.2.

1	4
2	5
a	5
b	4

Table 2.2: Result if the insertion order is wordA, wordB, wordC.

Assuming that the order of the insertion is: wordB, wordC, wordA the results are depicted in table 2.3.

1	7
2	5
a	7
b	4

Table 2.3: Result if the insertion order is wordB, wordC, wordA.

The conservative update was introduced in [23].

Universal hash functions

For bloom filter and especially count-min sketch, we are interested in universal hash functions. We describe one simple idea of how we can obtain hash functions that fulfil the requirements for these probabilistic data structures.

We are specifically interested in 2-universal (also called pairwise independent) hash functions.

We define as U the universe with $|U| = m \geq n$, and $V = \{0, 1, \ldots, n-1\}$. We call a family of hash functions $\mathbb{H}$ that $h \in \mathbb{H} : U \to V$, 2-universal if for a uniformly selected function $h \in \mathbb{H}$ we have:

$$Pr(h(u_1) = h(u_2)) \leq \frac{1}{n}$$

In other words, if the collisions are as rare as possible.

We define a simple universal family. Let $p \geq m$ be a prime and $h_{a,b}(u) = ((ax + b) \mod p) \mod n)$. Then, the following family $\mathbb{H}$ is 2-universal:

$$\mathbb{H} = \{h_{a,b} | 1 \leq a \leq p-1,\ 0 \leq b \leq p\}$$

See study [12] where *Universal Hashing* was introduced in practice.

2.3 Compositional Process

> The meaning of a complex expression is determined by its structure and the meanings of its constituents. (the Principle of Compositionality)

The Principle of Compositionality (sometimes also called Frege's Principle) is the main idea that underlies this section. If we accept this principle in the context of our study, we are allowed to focus on smaller parts of a sentence (words) and then seek a compositional process that allows to derive the meaning of a sentence using as ingredients the words and their structure. More details on the Principle of Compositionality can be found in Stanford Encyclopedia of Philosophy [51].

2.3.1 Models of Compositionality

We mention in brief some methods used to model compositionality in the context of distributional semantics. All of them operate in the vector space model. We assume that we have already derived two vector u and v which correspond to two words and we want to extract their combined meaning c. We denote as $_i$ the i-th component of a vector.

- *Addition:* $c = u + v$. See an example in [42]. Usually, we take the centroid vector and not just the addition, that is, we also divide by the number of words in the composition the result.
- *Pointwise multiplication:* $c_i = u_i \cdot v_i$. Example in [4]. This method shares the simplicity of addition but better results are reported in literature. For instance [4] compared to [42] and the study in [35] which concludes that multiplicative models are superior. In both models the word order does not matter (bag of words model).
- *Tensor product:* $c = u \otimes v$. Example in [54]. Tensor product is the most general bilinear operation. Applied on the vectors u and v produces a matrix containing all the possible products $u_i \cdot v_i$. The main issue with tensor products is that the increase of dimensionality of the resulting space compared to the original vectors space, since we get a matrix instead of a vector.
- *Circular convolution:* $c_i = \sum_j u_j \cdot v_{i-j}$. Effectively it compresses the tensor product of two vectors back to the original space. See [38] for details.
- *Linear regression:* $c = Au + Bv$, where A, B are matrices estimated by a supervised learning algorithm (partial least squares regression). See [25] for details.

- *Recursive neural networks:* See [49, 50]. It uses the recursive structure that is commonly found in natural language. This model can learn the meaning of operators in natural language and outperforms many other models.
- *Through category theory:* Meanings of words with *functional types*, such as verbs and adjectives, should be represented with tensors. Because this idea differs significantly from the previous ones, we develop it in section 2.3.1. See [15, 24] for details.

A presentation and evaluation of most of the above models has been done also in the studies [25, 35].

So far, we have seen examples where the model of compositionality f is:

$$c = f(u, v) \tag{2.23}$$

A more general model is sometimes used (see [35] for a more detailed explanation) where f is defined as:

$$c = f(u, v, R, K) \tag{2.24}$$

where R is the syntactic relation of the words and K is any additional knowledge.

Use of functional semantics to model compositionality of distributional meaning.

The philosophy of formal compositional semantic models has its roots in [22] and the idea that a sentence is a function of the meaning of its parts. On the other hand, distributional models (meaning can be derived by its context) are more recent and can be attributed to Wittgenstein [55]. Even though Frege and Wittgenstein had different points of view as it is demonstrated by the correspondence between them [18], the approach of Compositional Distributional Model of meaning tries to marry the two ideas.

Two studies that focus on this approach [15, 24], with [15] laying out the mathematical foundations and [24] being an experimental study of a subset of the theory, argue that the use of pregroups is motivated by their common structure with vector spaces and tensor products. The main idea is as follows: The meaning of a word is a vector while the grammatical role is a type in Pregroup and in order to model the composition of *meaning, type* pairs the tensor product of the vector spaces paired with the Pregoup composition is used.

To provide an example, let us use the following symbols: n: noun s: declarative statement j: infinite of the verb σ: glueing type

We use the exponent x^l or x^r to denote, respectively, whether an x is required on the left, or on the right position.

We assume that if the juxtaposition of the types of the words within a sentence reduces to the basic type s, then the sentence is grammatical.

Example: Bob (n) likes $(n^r s n^l)$ Alice (n).

It is grammatical because: $n(n^r s n^l)n \to 1 s n^l n \to 1 s 1 \to s$

A second example: Bob (n) does $(n^r s j^l \sigma)$ not $(\sigma^r j j^l \sigma)$ like $(\sigma^r j n^l)$ Alice (n).

Again, someone can test that this sentence is also grammatical.

The procedure of assigning meaning to a string of words can be roughly described as follows:

1. Assign a grammatical type p_i to each word w_i of the string and then verify using the process that we described above that the sentence is grammatical.
2. According to the grammatical type, assign a vector space to each word of the sentence. Basic types like a noun can be assigned a vector while compound types like verbs are tensor spaces.
3. Take the tensor product of the words.

2.3.2 Compositionality in queries

Usually, queries do not contain the necessary syntactic information in order to apply the approaches that require some kind of syntactic information, either explicitly or implicitly (for instance Recursive Neural Networks in [49, 50] or [15, 24]. Moreover, if the word order appears arbitrary and which dictates towards a simple model of compositionality with the commutative and associative properties. We should note here that usually we find the reverse requirement in the majority of current bibliography, because the problem that is usually studied is composing the meaning of a sentence from the meaning of its words and not queries.

Chapter 3

System Description

We begin our description with some necessary definitions and then, we describe the architecture of our system. We describe in some more detail how we built the system in a distributed environment and resolved some scaling issues. Finally, at the end of this chapter, we give an overview of our system.

3.1 Basic concepts

We do not explain here concepts that were described in chapter 2. Instead, we provide some definitions and concepts that are necessary for the later discussion.

Definition 3.1 (Co-occurrence statistics vector of w) *The word frequencies of neighbouring words in relevant snippets.*

Definition 3.2 (Snippet) *By default, the snippet on standard web search result is a max of* 160 *characters.*

Definition 3.3 (Neighbouring to w) *Words that occur up to distance k ($k \leq 3$ so far) from word w in a snippet.*

Definition 3.4 (Relevant snippet to w) *The snippet is returned (top N queries) by a search engine when there is a query containing w that returns the snippet.*

We have a collection of snippets as input stored in a BigTable [14]. For the purposes of our system, we consider only snippets that are in the English language. To find these snippets, we filter out our collection according to the character set of a snippet by keeping only those in ASCII encoding and, then, using the output of a language classifier.

Moreover, we are only interested in a specific set of words W that exists also in our evaluation dataset. We do not build co-occurrence statistics vectors for the rest of the words. That decreases the scale of our system in terms of

how many vectors we produce. That part is parallelisable in a distributed environment because there are no dependencies between the word vectors of different words.

3.2 Input parameters

In order to explore different directions, in addition to our input corpus of snippets and the evaluation datasets, we consider as input to our system the following:

1. List of *stop words* (if it is empty, do not remove stopwords).
2. *Window size*.
3. *Sides* on which we expand the window (left or right or on both sides of the target word).
4. *Weight distribution*. It can be either uniform (each word within the context window has the same weight) or exponential (the weight of a word decreases exponentially according to its distance from the target word within the window).
5. Use of *stemming* on both keywords and context, only on context words or no use of stemming at all.
6. The weight of the components of the co-occurrence statistics vectors is using either *frequencies* or *positive pointwise mutual information* between the target word and the context word.
7. Use of *hash kernel* and what is the resulting size of the output vector.
8. Use of *Count-min sketch* and its parameters. The parameters include:
    a) *Width:* How many values each hash function can output.
    b) *Depth:* How many hash functions we use.
    c) Use of *Conservative Update*.

In the implementation of our system, we split the aforementioned parameters into two categories: parsing and compression parameters. In the compression parameters are the hash kernel and the count-min sketch while everything else is considered as a parsing parameter.

3.3 Constructing the Co-occurrence statistics vectors

The idea is to iterate over all the snippets in our collection, find the words of interest that are contained within a window of the target word according to

the previous definitions and using this information, create the corresponding distributional vectors. We skip the description of details in the code like determining language of the queries, stemming, removal stopwords and other technical details. We will visit them in a later section.

The first phase of the system is described in Algorithm 3.

Algorithm 3 Find co-occurring pairs.

Require: BigTable bt of queries q and associated snippets s. Set of words W.
Ensure: For each word in W, find neighbouring words according to previous definitions.

```
1: for ∀q ∈ bt do
2:     for ∀ normalized words t ∈ q do
3:         if t ∈ W then
4:             if t ∈ s where s is an associated to q snippet then
5:                 emit neighbouring words
6:             end if
7:         end if
8:     end for
9: end for
```

As we can see, we need a single pass over the collection of snippets in the BigTable, staying faithful to the streaming model. The algorithm 3 is executed in Flume [13], which is built on top of MapReduce. Flume is a set of libraries that provide an abstraction layer on top of MapReduce. They provide classes that represent immutable parallel collections and some operations on them. Flume instead of executing the described operations as listed, creates a plan of execution. Then, it examines which results are needed, optimizes that plan and then executes it.

The second phase of our system consists of collecting the co-occurrence pair that we found, and creating the vector representations that we need. The final vectors that we output are either frequencies for each component in the vector or their corresponding positive pointwise mutual information, depending on the set of parameters. A high level description of the second phase is provided in Algorithm 4.

We have omitted the discussion about how these pairs fit in memory and the necessary data structures in order to process them. Because the environment is distributed, we rely for that on Flume [13] data structures. We also take into consideration the (expected) sparsity of the resulting vectors.

The vectors for each word are finally stored in an SSTable. A description about what an SSTable is, can be found in [14].

Algorithm 4 Build Co-occurrences matrix

Require: Pairs of words (target word, context word). (I)
Ensure: Output Co-occurrence statistics vectors.

1: Create a map M of target word: (context word, occurrences)
2: **for** $\forall$pairs of words $(t, c) \in I$ **do**
3: $M[t][c] = M[t][c] + 1$.
4: **end for**
5: **if** we need PPMI **then**
6: Create a map G of context word: occurrences
7: **for** $\forall$pairs of words $(t, c) \in I$ **do**
8: $G[c] = G[c] + 1$.
9: **end for**
10: **end if**
11: Using M (and if necessary G) compute frequencies of the words (or PPMI).

3.4 From word representations to queries and their similarities

We outline the plan of how we create the query representations from the word vectors that we have due to the previous phase in diagram 3.1.

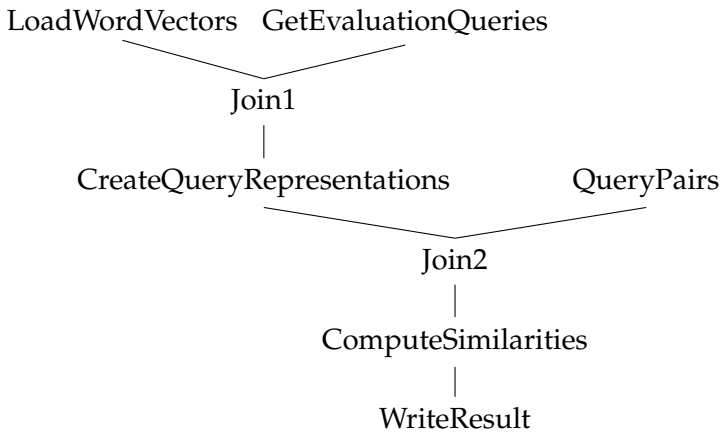


Figure 3.1: Outline of creating the query representation and getting the similarities between query pairs.

We proceed with the explanation of every function described in the diagram 3.1.

1. *GetEvaluationQueries:* Given the evaluation queries, we find which words

contained there. We also keep the information about the queries since we have to build their representation later.

2. *LoadWordVectors:* We simply load from the SSTable the necessary word representations.
3. *Join1:* For each word, we have its representation and a list of all the queries where it is contained.
4. *CreateQueryRepresentations:* For each query, we collect all the words that are necessary in order to create its representation and according to our compositional function, we create the query representation.
5. *QueryPairs:* We load the query pairs that indicate which query should be compared with which one.
6. *ComputeSimilarities:* We compute the similarities between the vectors that represent the two queries for each pair. The computation may vary slightly according to the vector representation (for instance, in the case of Count-min sketch).
7. *WriteResult:* We output the result of the query similarities.

We should not that we in our system we view queries as sets of words. That means, given that our compositionality functions are commutative, the word order does not matter. Moreover, repetition of words is ignored. For instance a query `q1 = "word1 word2 word1 word3"` is reduced to `"word1 word2 word3"`.

A very high level description of the current phase is described in Algorithm 5.

Algorithm 5 Query Representations and Similarities

Require: Query pairs QP, word representations WR
Ensure: Similarity measure for each query pair

1: Read all the queries.
2: For each query, find associated words and collect their vectors.
3: Build query representation from these associated words. (compose = addition)
4: For each query pair in evaluation dataset, extract similarity measure.

3.5 Differences with previous systems

We describe the main differences of our system with those describe in [41] and [4]. Someone can find a short description about these systems in the section 2.1.7.

Compared to [41], our main differences are:

1. Instead of truncating the vector representations of the words to $m = 50$ components, we are able to either use the full vector, a hash kernel or a count-min.
2. We don't consider fixed the definition of context. In [41], the window size has length three on both sides and a uniform weighting in all terms. Instead, we considered window size and sides as parameters. As for the weighting schema of each term, it can be either a uniform distribution or words closer to the target word count exponentially more.
3. We use either word frequencies or positive pointwise mutual information instead of tf-idf.

Compared to [4], our main differences are:

1. Flexibility on the vector representation (not just truncation) as before.
2. Similarly with the context definition.
3. As a compositional function we are able to use either the arithmetic mean of the geometric one.
4. We don't employ the χ^2 test.

3.6 Challenges of the distributed environment

The input corpus is large enough that needs more than few machines in order to be processed. We use Flume in order to run in a distributed manner the two phases that we described in section 3.3. During the first phase, every machine keeps internally a mapping between a target word t, a context word c and the number of occurrences t of c in the context of t. Each time we find a new word, we create the necessary entry, otherwise we increase the counter by one or the corresponding weight if we follow the exponential distribution. At the end of the processing on each computer, we emit key/-value pairs where as key we use the (t, c) and as value t. Then, we group these key/value pairs by key and we reduce them to a single key/value pair by adding the corresponding number of occurrences. Now, we regroup them using as key t. If we use frequencies, we normalize the counts by the aggregate value that corresponds to each t. If we use positive pointwise mutual information as component for the vectors, we also need a map with the frequencies of each word in the corpus collection. We can obtain this map following a similar process.

The parsing parameters are enforced when we read the collection of the snippets. This step is fully parallelisable since it does not have dependencies from other parts of the execution pipeline.

Similarly, the compression parameters are also enforced during the read of the corpus, apart from the calculation of the inner product according to count-min which can take place at the very end only. While we read the input, we apply the hash functions on the words that we read before inserting them to the corresponding maps. In order to make sure that the parsing takes place correctly, the compression can start only after we have applied on the text all the transformation that are indicated by the parsing parameters (stemming, removal of stopwords, etc). The rest of the pipeline remains similar whether we use compression or not.

3.7 Challenges of the reduced space

When we compress the words that we read using hash functions, we do not know any more what was the initial word. So, how can we compute the inner products of the co-occurrence statistics vectors or even the positive pointwise mutual information?

As long as we keep using the same hash functions, we know that a specific word gets mapped always to the same components. The theory that we mentioned briefly in the sections 2.2.6 and 2.2.7 provides us some guarantees about the approximation error of the inner product in case of collisions. We compute the positive pointwise mutual information under the assumption that the collisions will not cause to much distortion if we use the same hash functions.

3.8 Overview of the system

We already discussed about the details of the system. Let's look again now at the big picture of what is happening.

According to some input parameters, we read the snippets (input corpus) in parallel from many machines. The way we read the snippets depends on the parsing parameters. Then, we compress the words that we read according to the compression parameters. We store the compressed representation of the words in a mapping data structure in every machine along with necessary information in order to create the co-occurrence statistics vectors later. When the processing in every machine finishes, we regroup the data that we read in order to create the semantic vectors. That concludes the creation of the co-occurrence statistics vectors.

The creation of the query vectors presents no extra challenges. We have to gather the necessary co-occurrence statistics vectors, create the representations of the queries and then extract their similarities.

Chapter 4

Evaluation

In the current chapter, we present and discuss the results from the experiments on the different parameters of the system. We split the parameters into two parts: parsing and compression parameters where we dedicate the corresponding sections.

In order to evaluate our results, we use three correlation coefficients: *Pearson product-moment, Spearman* and *Kendall-tau*. We evaluate our system under different values of the input parameters and we also compare it with other systems on the same task. Our system outperforms most of these systems in both datasets that we used.

Before presenting our results, we explain the datasets that we used.

4.1 Description of evaluation datasets

We use two datasets. The evaluation dataset CC2000 was built from the results of a search engine under the assumption that two queries are related if a user clicks on same document. It contains 2000 pairs of queries. Each query pair was rated on the scale 1 - 4 (unrelated - same meaning) by 5 raters. The inter-rater agreement gave a Kappa value of 0.65. A more thorough description of CC2000 can be found in [28] in section 6.

The other evaluation dataset, called QS1500, can be found in [4]. It contains 57 source queries, each one paired with up to 20 target queries. The queries were rated in the 5-Likert scale. The inter-rater agreement gave a Kappa value of 0.711.

For more details on the datasets, see [4, 28, 42]. At the end of the section, we will present some aggregate results by comparing our system to several other systems. This is based primarily on the work in [28].

4.2 Parsing Parameters

As parsing parameters and in accordance to the discussion in chapter 3, we consider:

1. *Window size*.
2. *Window sides*.
3. *Weight distribution*.
4. *Stop words*.
5. *Stemming*.
6. *Frequencies* or *positive pointwise mutual information* as vector components.

We dedicate one section in the study of every parameter by conducting a series of experiments. After providing the data from the experiments, we discuss the results.

The experiments using the first three parameters (window size, sides, distribution), can be thought as our attempt to find experimentally evaluate different notions of the context. Experiments using stop words and stemming are about improving the quality of the context window by transforming the words that we observe within that window. Finally, the contrast between frequencies and positive pointwise mutual information serves the purpose of finding a good weighting scheme for the components of the co-occurrence statistics vectors.

4.2.1 Window sides

We start by studying the effect of defining the context of one word (the target word) in one side only or on both sides of a target word. As compositional process we consider either addition or pointwise multiplication of the vector components. The weights of the vector components are frequencies. We consider window sizes from one up to three.

In the case of using as compositionality function the vector addition, the results are very clear. We can observe in tables 4.1, 4.2, 4.3 and 4.4 that considering windows on both sides of the target word always outperforms one sided windows.

We omit the results for the component-wise multiplication because they are not competitive (the correlation is between $0.10 - 0.20$). Nevertheless, even in that case, looking both sides is in general a better approach than using only one side of the context window.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
left	0.32	0.30	0.26	0.26	0.23	0.21
right	0.30	0.35	0.24	0.30	0.21	0.23
both	**0.35**	**0.39**	**0.29**	**0.36**	**0.25**	**0.29**

Table 4.1: How choosing which sides to consider in the context window affects the correlation of the system prediction with the gold standard. The window size is one. As components of the vectors we use frequencies and the compositional function is addition.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
left	0.32	0.30	0.26	0.26	0.23	0.21
right	0.31	0.35	0.25	0.32	0.22	0.25
both	**0.35**	**0.38**	**0.29**	**0.38**	**0.25**	**0.30**

Table 4.2: Study of context sides. Window size is two. The other parameters are the same as in table 4.1.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
left	0.33	0.29	0.27	0.26	0.23	0.20
right	0.30	0.35	0.24	0.32	0.21	0.25
both	**0.34**	**0.38**	**0.28**	**0.37**	**0.24**	**0.29**

Table 4.3: Study of context sides. Window size is two and the distibution is exponential. The rest of the parameters are the same as in table 4.1.

In total, we can conclude that defining the context window on both sides of the target word outperforms the definitions that are limited to only one side (left or right).

4.2.2 Weight distribution

We fix the window size to two, and we measure the effect on the correlation with the gold standard if we double the weight of the words closer to the target word compared to the putting the same weight to every word. The results for vector addition as compositional method are listed on table 4.5.

The experiments presented here, as well as the experiments where the compositionality function was pointwise vector multiplication, did not show any significant differences (the uniform distribution looks slightly better) using different weights for context words according to their relative position.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
left	0.33	0.32	0.26	0.32	0.23	0.26
right	0.32	0.34	0.26	0.32	0.23	0.25
both	**0.35**	**0.38**	**0.29**	**0.38**	**0.25**	**0.30**

Table 4.4: Study of context sides. Window size is three. The rest of the parameters are the same as in table 4.1.

Correlation		Pearson		Spearman		Kendall tau	
Dataset		CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
l	u	**0.33**	**0.32**	**0.27**	**0.30**	**0.24**	**0.24**
l	e	**0.33**	0.29	**0.27**	0.26	0.23	0.20
r	u	**0.31**	**0.35**	**0.25**	**0.32**	**0.22**	**0.25**
r	e	0.30	**0.35**	0.24	**0.32**	0.21	**0.25**
b	u	**0.35**	**0.38**	**0.29**	**0.38**	**0.25**	**0.30**
b	e	0.34	**0.38**	0.28	0.37	0.24	0.29

Table 4.5: Study of weight distribution. The window size is two. We look at the left, right and both sides of the target word (correspondingly, every two rows). As components of the vectors we use frequencies and the compositional function is vector addition. A number is bold faced if under the same parameters, using the current distribution is at least as good as using the other one.

4.2.3 Window size

We now focus on what the window size should be in order to maximize the correlation with the gold standard. For brevity, we present only the uniform distribution of weights considering both sides in the context window and as compositional function we use the vector addition. Looking at the tables in the previous sections, someone could expand the presentation of the results to more cases. Still, the conclusions would remain the same.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
1	0.35	0.39	0.29	0.36	0.25	0.29
2	0.35	0.38	0.29	0.38	0.25	0.30
3	0.34	0.38	0.28	0.37	0.24	0.30

Table 4.6: Window size. We only look at both sides of the target word, assuming uniform distribution. As components of the vectors we use frequencies and the compositional function is vector addition.

Again, no significant differences increasing the window size. We arrive to the same conclusion even if we fix the other parameters (window sides, weighting, compositional function) to different values. We omit the presentation of these results but they can be deduced by the values that we have presented in the context of other experiments.

4.2.4 Do simple groups of words matter?

So far the top performance of our results is not very interesting. To compare with a reference system, the DistSim in [4] performs better by exhibiting Spearman Correlation of 0.322 and 0.438 in CC2000 and QS1500 correspondingly.

Using the conclusions so far (window size, side, weight distribution), we can define a somehow experimentally validated context window. We want proceed to the next round of experiments, where we try to improve the correlation of the results of our system with the gold standard. To do so, we want to test whether we can improve the correlation of the system prediction with the gold standard by treating some simple groups of words differently.

More specifically, we experimentally evaluate the effect of ignoring stop words within the context of a target word. Moreover, we also check whether we can group words together when we build their word co-occurrence vectors. A simple grouping can be based on their stems. We will test if the performance of the system increases if we group together words that have the same stem.

4.2.5 Stop words

We use a list of stop words which we ignore in our corpus. If a stop word happens to occur in a query, we consider in its place the identity element for our compositional function.

We present on table 4.7 the results of stop word removal under all the different parameters we have explored so far in one table. To facilitate the presentation, we create a smaller table (table 4.8) where we compare the results of stop word removal with the best performing results without stop word removal.

We can observe that stop words removal lead to increase in correlation with the gold standard. To verify it more easily, the table 4.8 presents the correlation of the system with and without stop word removal in the case of window size equal to three, both sides are considered and the weight distribution is uniform (one of the best performing cases so far).

Correlation			Pearson		Spearman		Kendall tau	
Dataset			CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
1	l	u	0.37	0.38	0.32	0.33	0.28	0.26
1	r	u	0.39	0.45	0.31	0.35	0.27	0.28
1	b	u	0.41	0.46	**0.35**	0.38	**0.31**	0.31
2	l	u	0.38	0.40	0.31	0.35	0.27	0.28
2	r	u	0.39	0.46	0.32	0.38	0.28	0.31
2	b	u	0.40	0.48	0.34	0.40	0.29	0.32
2	l	e	0.38	0.39	0.31	0.34	0.28	0.28
2	r	e	0.39	0.46	0.33	0.36	0.29	0.29
2	b	e	0.41	0.47	0.34	0.39	0.29	0.32
3	l	u	0.38	0.45	0.31	0.41	0.28	0.33
3	r	u	0.40	0.47	0.33	0.41	0.28	0.33
3	b	u	**0.42**	**0.49**	**0.35**	**0.42**	0.30	**0.34**

Table 4.7: The correlation of the system after stop word removal. The elements of the first column indicate window size, sides (left, right, both) and weight distribution (uniform, exponential) correspondingly.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
with	0.34	0.38	0.28	0.37	0.24	0.30
without	**0.42**	**0.49**	**0.35**	**0.42**	**0.30**	**0.34**

Table 4.8: We study the performance of the system with and without stopwords. We only look at both sides of the target word, assuming uniform distribution. As components of the vectors we use frequencies and the compositional function is vector addition.

Clearly, the stop word removal is beneficial for the system. Moreover, now we are very close to DistSim performance with slightly better results in the CC2000 dataset and slightly worse results in the QS1500 dataset.

In bibliography, it is not clear what the effect of the stop word removal is (for an example, see studies [10] and [11]).

4.2.6 Stemming

After the success of the stopword removal, we also tested the performance of our system after applying stemming. We tried two different approaches. Stem only the context of the target words or also apply stemming to the target words. If we apply stemming on the target words, we must apply it on the queries as well.

We present on the table 4.9 the results of the three approaches (no stemming, stemming only on context, stemming everywhere).

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
without	0.42	0.49	0.35	0.42	0.30	0.34
context	0.43	0.50	0.35	0.43	0.31	0.35
everywhere	**0.45**	**0.53**	**0.37**	**0.45**	**0.32**	**0.37**

Table 4.9: Stemming. We only look at both sides of the target word, assuming uniform distribution and stop word removal. As components of the vectors we use frequencies and the compositional function is vector addition.

The performance boost is maximized if we apply stemming everywhere (both context and target words). This is the first point where a specific set of values in our parameters results in better correlation in both datasets compared to DistSim. The main insight from the current section is that for queries, it is beneficial to merge the semantic representation of words with the same stem.

4.2.7 Positive Pointwise Mutual Information vs Frequencies

Lastly, we also tried using positive pointwise mutual information instead of frequencies. Because in order to compute the positive pointwise mutual information we need the probability of a word occurrence in the whole corpus, our system could not scale efficiently to that size due to memory constraints. Some back of the envelope calculations indicate that if we need 20 bytes for each word, frequency counter, alignment and other internal bookkeeping data (such as pointers if we use as a mapping data structure a tree), and assuming that the mapping data structure that we use is dense (which excludes many hash map implementations), then if using 1 GB memory exclusively for that dictionary, we can store up $5 \cdot 10^7$ different tokens. The problem is presented when we try to compute (efficiently) the pointwise mutual information which requires the global map to be in memory and load each one of the maps for the individual words. Unfortunately, the current implementation collects all the mapping data structures in the memory of one machine, before it starts processing them, that is, compute the pointwise mutual information.

To make it more easy to understand why that happens we have to dive into the internals of the system. Flume is based on Map Reduce. The specific high level operation that we use, during the shuffle phase (redistribution of the key/value pairs according to their keys), sends all mapping data structures to one machine and forces that machine to load them. That explodes

the memory needs. To circumvent this issue, we have to change the way we perform the computation in a similar way to how we perform the merge operation on the mapping data structures. Unfortunately, that would require a lot of effort, and it would result to a only slight increase in the correlation with the gold standard (as we will see, the compression methods work surprisingly well) and it would slow down the system in general.

Therefore, we were forced to use compression (we analyze the the performance under compression in the section 4.3). Nevertheless, despite the use of compression, we get the best performance using positive pointwise mutual information instead of frequencies as we can see in the table 4.10.

Correlation	Pearson		Spearman		Kendall tau	
Dataset	CC2000	QS1500	CC2000	QS1500	CC2000	QS1500
frequencies	0.42	0.51	0.35	0.45	0.31	0.36
ppmi	0.46	0.61	0.40	0.50	0.36	0.41

Table 4.10: We examine the effect of using positive pointwise mutula information instead of frequencies as a weighting scheme for the vector components. We only look at both sides of the target word, assuming uniform distribution , stop word removal and stemming both on context and the keywords. The compositional function is vector addition. We use compression by having a hash function with output of 2^{11} different values.

4.2.8 Comparison with other methods

For this comparison, we use the data from [28] as well as their original scripts in order to produce a table similar to the Table 4 of [28] including our system. We call our system as *Our System*.

The parameters that we used are:

- Window: 3 words before and after the target word.
- Weight distribution: uniform.
- Components: positive pointwise mutual information.
- Stop words: removal.
- Stemming: both on target words and context.
- Number of components: 2^{11} using a single hash function.

From the tables 4.11 and 4.12 can be seen that the system performs relatively well outperforming significantly the other similar system DistSim [4]. For details on the different metrics displayed on the tables 4.11 and 4.12, see [28].

#	Feature	Spearman	mAP	Prec@1	Prec@3	Prec@5	Sig.
1	NN(All)	0.500	0.806	0.836	0.741	0.637	9
2	*Our System*	0.496	0.786	0.829	0.713	0.653	9
3	PCA	0.494	0.793	0.800	0.722	0.652	9
4	Oommen-Kashyap	0.470	0.747	0.782	0.698	0.637	12
5	DistSim (Alfonseca et al.)	0.438	0.745	0.767	0.676	0.628	15
6	Mean all	0.435	0.772	0.818	0.691	0.633	16
7	SortedSessionEdit(S)	0.429	0.772	0.837	0.710	0.640	17
8	SortedSessionEdit(G)	0.428	0.776	0.835	0.719	0.648	17
9	WebPMI(S)	0.417	0.713	0.764	0.630	0.589	17
10	WebPMI(J)	0.409	0.730	0.782	0.679	0.595	17
11	SortedSessionEdit(J)	0.408	0.769	0.828	0.709	0.641	17
12	SortedWebEdit(J)	0.386	0.740	0.764	0.676	0.624	19
13	SessionEdit(S)	0.382	0.745	0.796	0.693	0.617	20
14	SessionEdit(G)	0.380	0.742	0.797	0.696	0.624	20
15	SessionEdit(J)	0.365	0.737	0.792	0.692	0.613	22
16	SortedWebEdit(S)	0.357	0.701	0.757	0.663	0.589	23
17	SortedEdit2	0.320	0.714	0.756	0.669	0.631	25
18	SortedEdit1	0.314	0.695	0.761	0.668	0.590	25
19	SortedWebEdit(G)	0.306	0.702	0.753	0.660	0.596	25
20	WebEdit(S)	0.302	0.639	0.701	0.612	0.561	25
21	WebEdit(J)	0.299	0.682	0.691	0.666	0.598	25
22	WebPMI(G)	0.283	0.669	0.608	0.548	0.570	26
23	WordEdit2	0.270	0.648	0.721	0.615	0.571	26
24	WordEdit1	0.252	0.636	0.697	0.620	0.555	26
25	WebEdit(G)	0.220	0.620	0.655	0.593	0.530	26
26	Len2(Char)	0.139	0.519	0.437	0.454	0.436	28
27	Len2(Term)	0.112	0.505	0.454	0.429	0.411	28
28	lp2	-0.161	0.452	0.309	0.309	0.341	-
29	lp1	NaN	0.467	0.379	0.382	0.369	-
30	Len1(Term)	NaN	0.465	0.391	0.376	0.371	-
31	Len1(Char)	NaN	0.467	0.375	0.373	0.365	-

Table 4.11: QS1500. Comparing of many different systems using data from [28]. The last column gives the index of the model with the highest Spearman correlation that the corresponding model is *significantly* higher with $p < 0.05$.

4.3 Compression Parameters

In order to have a system that scales well, we require it to be able to handle growing corpus of data without requiring too much memory. That is not true when we use the full vectors as we did in most of the previous cases. Moreover, we already encountered a scaling problem when we tried to use as vector components the positive pointwise mutual information. To solve these issues, we try to compress the vector representations of the word and we study the performance drop due to this compression.

As compression parameters and in accordance to the discussion in chapter

#	Feature	Spearman	mAP	Prec@1	Prec@2	Prec@3	Sig.
1	NN(All)	0.432	0.739	0.583	0.511	0.451	12
2	SessionEdit(G)	0.424	0.716	0.544	0.486	0.446	13
3	SortedSessionEdit(G)	0.419	0.711	0.550	0.478	0.446	13
4	SessionEdit(S)	0.414	0.713	0.543	0.486	0.446	13
5	SortedSessionEdit(S)	0.407	0.709	0.548	0.477	0.445	13
6	SessionEdit(J)	0.402	0.713	0.540	0.483	0.448	13
7	*Our System*	0.402	0.713	0.525	0.492	0.442	13
8	PCA	0.392	0.708	0.531	0.481	0.449	13
9	Oommen-Kashyap	0.391	0.705	0.516	0.484	0.451	13
10	SortedSessionEdit(J)	0.391	0.707	0.537	0.473	0.445	13
11	Mean all	0.386	0.711	0.531	0.485	0.448	14
12	WebPMI(G)	0.369	0.699	0.507	0.473	0.449	17
13	WebPMI(J)	0.330	0.692	0.486	0.474	0.444	23
14	SortedWebEdit(J)	0.323	0.696	0.513	0.466	0.441	25
15	DistSim (Alfonseca et al.)	0.322	0.707	0.532	0.492	0.427	25
16	WebEdit(J)	0.322	0.678	0.481	0.451	0.442	25
17	WordEdit1	0.299	0.659	0.444	0.435	0.418	26
18	SortedEdit1	0.298	0.661	0.463	0.434	0.418	26
19	SortedWebEdit(G)	0.295	0.690	0.509	0.457	0.431	26
20	WordEdit2	0.292	0.675	0.473	0.457	0.432	26
21	SortedEdit2	0.288	0.687	0.486	0.462	0.431	26
22	WebEdit(G)	0.287	0.669	0.458	0.457	0.430	26
23	WebPMI(S)	0.264	0.682	0.477	0.461	0.437	26
24	SortedWebEdit(S)	0.264	0.678	0.488	0.458	0.426	26
25	WebEdit(S)	0.258	0.667	0.472	0.438	0.428	26
26	lp2	0.114	0.626	0.382	0.416	0.404	27
27	Len2(Char)	-0.036	0.603	0.363	0.390	0.391	-
28	Len2(Term)	-0.077	0.604	0.364	0.388	0.385	-
29	lp1	NaN	0.609	0.388	0.388	0.387	-
30	Len1(Term)	NaN	0.614	0.391	0.388	0.388	-
31	Len1(Char)	NaN	0.608	0.391	0.385	0.387	-

Table 4.12: CC2000. Comparing of many different systems using data from [28]. The last column gives the index of the model with the highest Spearman correlation that the corresponding model is *significantly* higher with $p < 0.05$.

3, we consider:

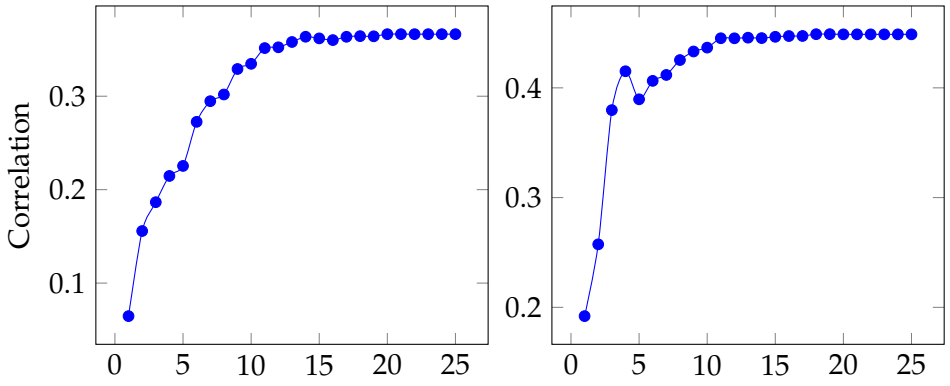
1. Hash kernels
    a) Number of output values in a hash function.
2. Count-min sketch
    a) Number of output values in a hash function.
    b) Number of hash functions.
    c) Use of conservative update.

4.3.1 Hash kernels

As we increase the number of hash values that a hash function is allowed to output, the closer we get to the performance without using hash functions. The impressive fact here is how fast we approach the accuracy of the non-compressed representation (see figure 4.1).

For all the plots that we present, the following are true: We only look at both sides of the target word, within a window of size three, assuming uniform distribution, stop word removal and the compositional function is vector addition. Moreover, we present only the Spearman correlation in order to make our results directly comparable with other studies. The Pearson correlation is typically sightly higher and the Kendall tau is sightly lower. The x axis is logarithmic, that is, for a value i, the co-domain of the hash function has cardinality 2^i.

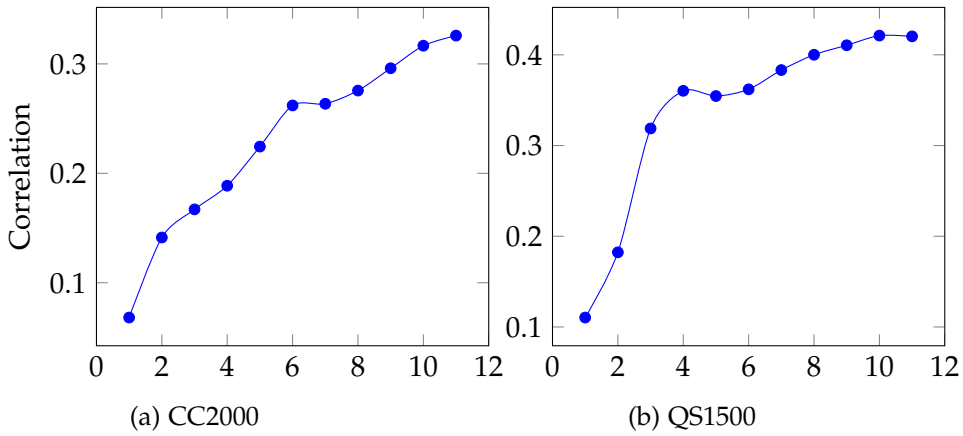
We present first the plot on figure 4.1 that demonstrates that it is possible to aggressively reduce the dimensionality of the data, without worrying too much about a performance drop. We observe that about 2^{11} output values of the hash function are enough to achieve accuracy close to that of the non-compressed representation.



(a) CC2000 (b) QS1500

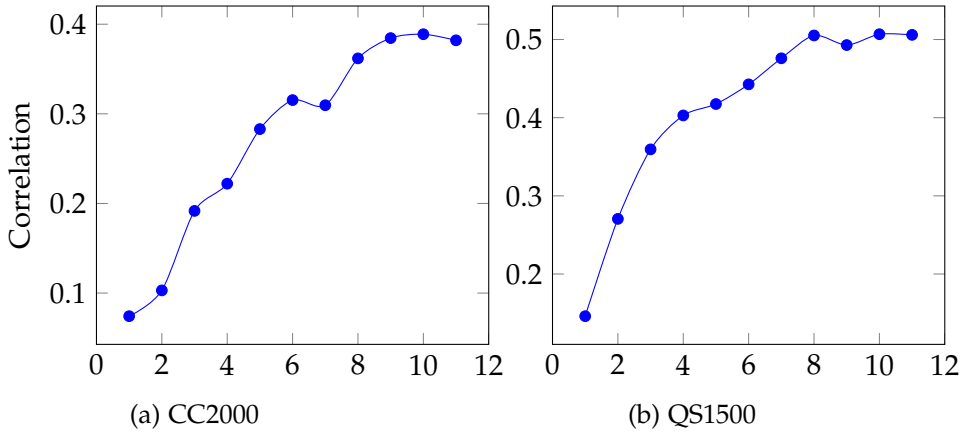
Figure 4.1: We increase the possible number of output values for the hash function. The horizontal axis is the logarithm of the number of values that a hash function may output. The value x_i corresponds to 2^{x_i} different possible output values from the hash function. For the other parameters, we use stemming on everything with frequencies as component weight.

For completeness, we present some more plots with different set of parameters in our system. The relevant plots for stemming only the context are: figure 4.2 (frequencies) and figure 4.3 (ppmi). Correspondingly for stemming everything: figure 4.1 (frequencies) and figure 4.4 (pmi).



(a) CC2000 (b) QS1500

Figure 4.2: Similar to figure 4.1 but we perform stemming only on context.



(a) CC2000 (b) QS1500

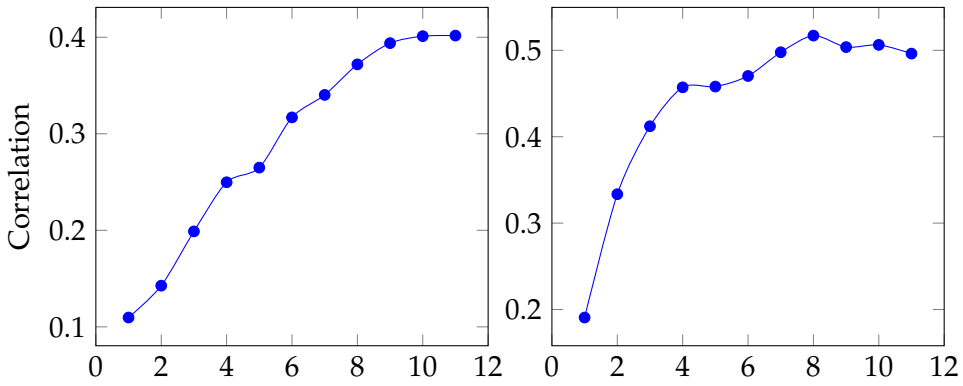
Figure 4.3: Similar to figure 4.1 but we perform stemming only on context and we use positive pointwise mutual information as component weights.

The application of a hash kernel is very compelling in our case. It is very simple and the benefits can be shown even having a relatively small co-domain. In the next section, we are motivated to study the count-min sketch. A count-min sketch using only one hash-function is the same as the case that we examined in this section.

4.3.2 Count-min sketch

In the case of Count-min sketch, we have three parameters.

1. Number of output values in a hash function. (width)
2. Number of hash functions. (depth)
3. Use of conservative update.



(a) CC2000 (b) QS1500

Figure 4.4: Similar to figure 4.3 but we perform stemming on everything.

We will present results in the following two combinations:

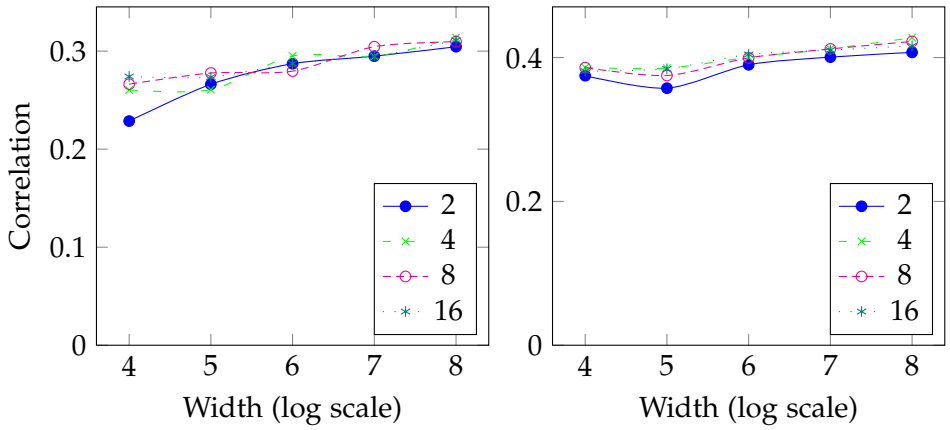
1. For stemming everywhere, we study what happens when we increase the depth as we increase the width without conservative update. We present two different figures for positive pointwise mutual information and frequencies.
2. For stemming everywhere, we study the effect of conservative update by fixing the width to 256 values and using as an x axis the depth.

Width vs depth in count-min sketch

We observe that increasing the depth is more beneficial than increasing the width for small co-domains of the hash function. On the other hand, for larger values, it is not clear that increasing the depth helps any more. Someone, could also argue that we have already reached about the maximum (uncompressed) performance of the system and therefore we should not expect any benefits from a better lossy compression method.

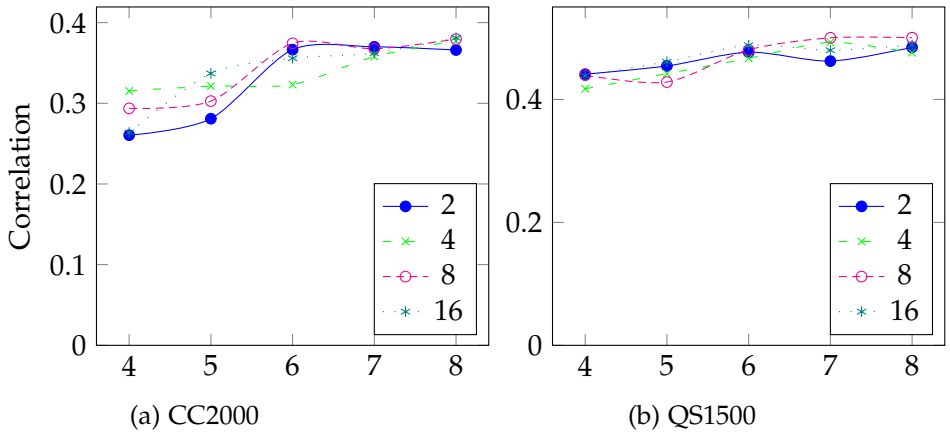
Conservative update and its relation to the number of hash functions

The usage of conservative update looks important only if we use few hash functions, otherwise the benefits are diminishing.



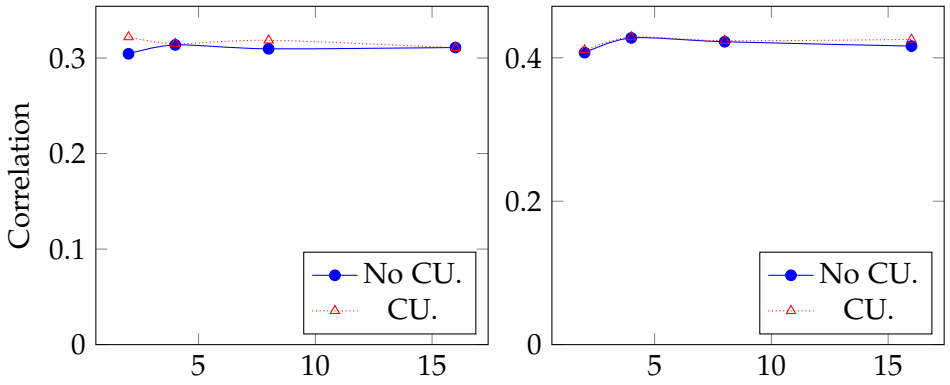
(a) CC2000 (b) QS1500

Figure 4.5: We study the effect of using more hash functions while increasing the width (the number of output values of each hash function). We perform stemming on everything, and remove stop words. The components are weighted by frequencies.



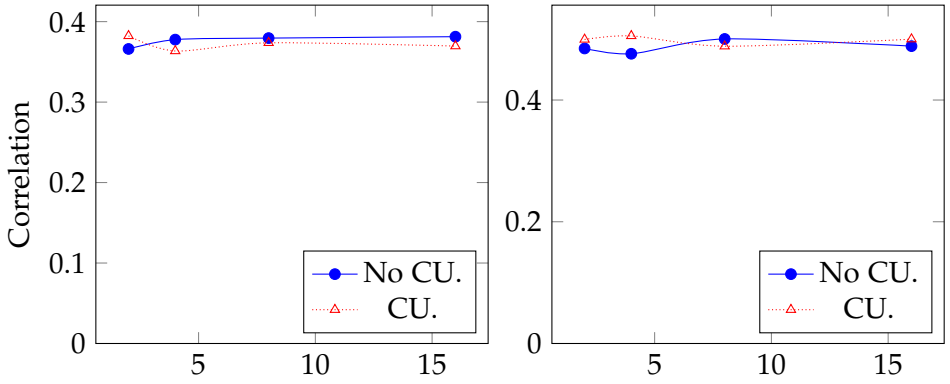
(a) CC2000 (b) QS1500

Figure 4.6: Width/dedpth study. The components are weighted by positive pointwise mutual information. The other parameters are the same as in figure 4.5.



(a) CC2000 (b) QS1500

Figure 4.7: We study the effect of conservative update with stemming on everything, increasing the depth of count-min sketch. The components are weighted by frequencies and the width is fixed to 256 values.



(a) CC2000 (b) QS1500

Figure 4.8: We study the effect of conservative update with stemming on everything, increasing the depth of count-min sketch. The components are weighted by positive pointwise mutual information and the width is fixed to 256 values.

Chapter 5

Conclusions and Further Work

We studied some concepts of distributional semantics in the context of assessing query similarities. During the implementation of the system, where we were exploring slightly different definitions of context and semantic representation, we needed to compress the vectors that we were constructing. We managed to achieve close to state of the art performance using a relatively simple system in conceptual terms which allowed us to focus on the engineering side, and more specifically, the scalability of the system. We managed to scale the system to process a very large collection of snippets in a distributed environment and in a single pass. The vectors that were created during that phase were used by another program that performed the evaluation task (query similarity). Our main insights as result of this work is that in an experimentally driven definition of the necessary notions for distributional semantics, we can derive better performance but taking into consideration simple linguistic facts (such as stop words and stems). Moreover, we can apply aggressive dimensionality reduction for the vector representations using either hash kernels or the count-min sketch. In the later case, it would be interesting to investigate what are the best parameters when the vocabulary size increases significantly (when, for instance, we consider bi-grams as tokens). Our limited results point that count-min sketch would be more important in that case. Especially, the trade-off between width and depth while keeping the space needs constant is interesting. Similarly, for the effect of the conservative update if we can tolerate spending some time more in order to generate our word vectors.

Bibliography

[1] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 274–281, New York, NY, USA, 2001. ACM.

[2] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput*, pages 302–322, 2009.

[3] Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual bch codes. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 1–9, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[4] Enrique Alfonseca, Keith Hall, and Silvana Hartmann. Large-scale computation of distributional similarities for queries. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL-Short '09, pages 29–32, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[5] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[6] Noga Alon and Dedicated To Miki Simonovits. Problems and results in extremal combinatorics–i. *Discrete Mathematics*, 2003.

[7] Michael W. Berry. Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, 6:13–49, 1992.

[8] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[9] Andrei Broder, Michael Mitzenmacher, and Andrei Broder I Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[10] J.A. Bullinaria and J.P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, (3):510, 2007.

[11] John A Bullinaria and Joseph P Levy. Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and svd. *Behavior research methods*, 44(3):890–907, 2012.

[12] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 106–112, New York, NY, USA, 1977. ACM.

[13] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. Flumejava: easy, efficient data-parallel pipelines. *SIGPLAN Not.*, 45(6):363–375, June 2010.

[14] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[15] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. 2010.

[16] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, April 2005.

[17] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlos. A sparse johnson: Lindenstrauss transform. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 341–350, New York, NY, USA, 2010. ACM.

[18] Enzo (ed.) De Pellegrin. *Interactive Wittgenstein. Essays in memory of Georg Henrik von Wright.* Synthese Library 349. Berlin: Springer., 2011.

[19] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.

[20] John R. Firth. A Synopsis of Linguistic Theory, 1930-1955. *Studies in Linguistic Analysis*, pages 1–32, 1957.

[21] George Forman and Evan Kirshenbaum. Extremely fast text feature extraction for classification and indexing. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1221–1230, New York, NY, USA, 2008. ACM.

[22] Gottlob Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 100:25–50, 1892.

[23] Amit Goyal, Hal Daumé, III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1093–1103, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[24] Edward Grefenstette and Mehrnoosh Sadrzadeh. Experimental support for a categorical compositional distributional model of meaning. 2011.

[25] Emiliano Guevara. Computing semantic compositionality in distributional semantics. In *Proceedings of the Ninth International Conference on Computational Semantics*, IWCS '11, pages 135–144, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[26] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42:335–346, 1990.

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

[28] Amac Herdagdelen, Massimiliano Ciaramita, Daniel Mahler, Maria Holmqvist, Keith Hall, Stefan Riezler, and Enrique Alfonseca. Generalized syntactic and semantic models of query reformulation. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 283–290, New York, NY, USA, 2010. ACM.

[29] William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

[30] Martin Joos. Description of language design. *The Journal of the Acoustical Society of America*, 22:701, 1950.

[31] Pentti Kanerva, Jan Kristoferson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *In Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, pages 103–6. Erlbaum, 2000.

[32] J. Karlgren and M. Sahlgren. From words to understanding. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, *Foundations of real-world understanding*, pages 294–308. 2001.

[33] Thomas K Landauer and Susan T. Dutnais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, pages 211–240, 1997.

[34] Jirí Matousek. On variants of the johnson-lindenstrauss lemma. *Random Struct. Algorithms*, 33(2):142–156, 2008.

[35] Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *In Proceedings of ACL-08: HLT*, pages 236–244, 2008.

[36] Preslav Nakov and Marti A Hearst. Solving relational similarity problems using the web as a corpus. In *ACL*, pages 452–460, 2008.

[37] Charles E. Osgood, George J. Suci, and Percy H. Tannenbaum. *The Measurement of Meaning*. University of Illinois Press, 1957.

[38] T. A. Plate. Holographic reduced representations. *Neural Networks, IEEE Transactions on*, 6(3):623–641, 1995.

[39] Hecht-Nielsen R. Context vectors: general purpose approximate meaning representations self-organized from raw data. *Computational Intelligence: Imitating Life, IEEE Press*, pages 43–56, 1994.

[40] Reinhard Rapp. Word sense discovery based on sense descriptor dissimilarity. In *Proceedings of the Ninth Machine Translation Summit*, pages 315–322, 2003.

[41] Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 377–386, New York, NY, USA, 2006. ACM.

[42] Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the*

15th international conference on World Wide Web, WWW '06, pages 377–386, New York, NY, USA, 2006. ACM.

[43] M. Sahlgren and J. Karlgren. Automatic bilingual lexicon acquisition using random indexing of parallel corpora. *Nat. Lang. Eng.*, 11(3):327–341, September 2005.

[44] Magnus Sahlgren. The Distributional Hypothesis. *Special issue of the Italian Journal of Linguistics*, 20, 2008.

[45] Magnus Sahlgren and Rickard Cöster. Using bag-of-concepts to improve the performance of support vector machines in text categorization. In *COLING*, 2004.

[46] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[47] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems, 2002.

[48] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, December 2009.

[49] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2012.

[50] Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2011.

[51] Zoltán Gendler Szabó. Compositionality. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2013 edition, 2013.

[52] Peter D Turney, Patrick Pantel, et al. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

[53] K.Q. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[54] Dominic Widdows. Semantic vector products: Some initial investigations. In *Proceedings of the Second AAAI Symposium on Quantum Interaction*, 2008.

[55] L. Wittgenstein. *Philosophical Investigations*. Basil Blackwell, Oxford, 1953.