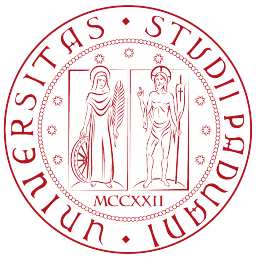


Università degli Studi di Padova

Corso di Laurea Magistrale in Informatica

Dipartimento di Matematica
"Tullio Levi-Civita"

A reproducibility study of a neural IR system

Student
Emanuele Carraro

Supervisor
Dr. Ing. Gianmaria Silvello

Co-Supervisor
Dott. Alberto Purpura

July 2019

dedica

Credits

Ringraziamenti

Padova, July 2019 Emanuele Carraro

Abstract

As the availability of large datasets and computing power grows, artificial neural networks gain interest from the scientific community and their range of application gets wider.

In recent years they have been applied to Information Retrieval, leading to the birth of an "hybrid" discipline called *Neural IR*.

Neural IR models have shown some improvements over traditional IR baselines on the task of document ranking.

By the end of 2016, Deep Relevance Matching Model (DRMM) developed by Jiafeng Guo, Yixing Fan, Qingyao Ai, W. Bruce Croft was one of the first Neural IR models to show improvements over traditional IR baseline models (e.g. Bm25 and Query Likelihood).

Since then, Neural IR has been an emerging trend, leading to the possibility of advancing the state-of-the-art, which makes even more important to verify published results, to build future directions on a solid foundation.

The aim of this work is to repeat, reproduce (from scratch) DRMM and test it on the collection Robust04 ([31]), a dataset of "difficult topics" where the state-of-the-art has reached a maximum of $\sim 30.2\%$ MAP (approximately).

Sommario

Negli ultimi anni la disponibilità di grandi moli di dati e di potenza di calcolo ha fatto sì che le reti neurali artificiali riscontrassero interesse da parte della comunità scientifica.

È stato solo di recente che sono state applicate al reperimento dell'informazione, portando alla nascita di una disciplina ibrida chiamata *Neural IR*.

I modelli di Neural IR hanno mostrato miglioramenti rispetto alle baseline date da modelli tradizionali di IR - e uno di questi è DRMM ([10]).

Alla fine del 2016, il modello "Deep Relevance Matching Model" (DRMM) sviluppato da Jiafeng Guo, Yixing Fan, Qingyao Ai, W. Bruce Croft è stato uno dei primi a battere le baselines (ad esempio, i modelli Bm25 e Query Likelihood).

Da allora il Neural IR è stato un trend in crescita e ha contribuito a far avanzare lo stato dell'arte, fatto che rende ancora più importante verificare risultati pubblicati, in modo tale da far procedere la ricerca su una base solida.

Lo scopo di questo lavoro è di ripetere, riprodurre (da zero) DRMM e testarlo sulla collezione Robust04 ([31]), un dataset di topic su cui è difficile riuscire a ottenere delle buone performance. Lo stato dell'arte ha raggiunto un massimo di 30.2% MAP.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In recent years, there have been dramatic improvements in performance in computer vision, speech recognition, and machine translation tasks.

These improvements were possible thanks to the combination of three factors: (i) advancement in the study of neural network models; (ii) the availability of large dataset and (iii) increased computing power.

In this context, Information Retrieval (IR) community has just begun to explore neural networks models with the purpose of verifying if they can be beneficial also in some popular IR tasks (e.g. document ad-hoc retrieval, as discussed in this thesis - all of the concepts introduced here will be explained in the next chapters).

Starting from 2013, a new field was born from the intersection of IR and Machine Learning: Neural IR, appealing researchers and students.

Although some Neural IR models have indeed produced some improvements over the baselines (w.r.t. document ad-hoc retrieval), the consequence of their application to IR have not yet been understood.

There is a strong discussion going on whether IR needs "Neural IR" or not. In fact, there are two main problems with these models: one regards efficiency (especially when a large collection of documents is involved), the other regards their ability to learn in a way that can address the complexity of IR tasks (where the concept of *relevance* plays an important role).

The first problem arises from the long time required by a Neural IR model to compute a similarity score between an (appropriate) learnt representation of a document and a query. In case of a large corpus, this time becomes prohibitive.

The second problem is especially linked to the difficulty to learn from queries and documents when no large-scale supervised data is available.

Unfortunately, this is often the case, in fact it is very expensive for a human to label a document "relevant" or "not relevant" with respect to a certain topic (most of it because relevance is a multifaceted concept).

A couple of strategies have been applied to cope with these problem: pseudo relevance feedback and a re-ranking approach.

Both of them involve the use of a traditional retrieval model in order to obtain supervised data, contributing to the debate on the need of Neural IR.

This work consists of two parts: a bibliographic section and an experimental section.

In the first few chapters I conducted an analysis of the field(s) of study: IR, artificial neural networks and Neural IR.

Then I reviewed several state-of-the-art works in Neural IR and compared them, pointing out their common characteristics and differences.

In the final chapters I replicated the experiment of Guo et al. in [10] and critically evaluated it, with a discussion of the results.

Thus, my contributions were: a critical analysis of an emerging (hybrid) field of study (Neural IR), the replication (redoing from scratch) and reproduction of DRMM, a deep relevance matching model, contributing to the IR community's growing interests in reproducibility and related issues; and making the source code publicly available so that others can inspect it.

CHAPTER 2

Background

2.1 Information Retrieval

The following definition of Information Retrieval (IR) was given by *Gerard Salton* in 1968 and is still accurate nowadays [4]:

> "Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information".

This definition is very general, in fact IR refers to a wide range of applications related to search.

The primary focus of the field since the 1950s has been on textual data (e.g. web pages, email, scholarly papers, books, and news). This type of data is mostly unstructured, thus matching between two pieces of text is not that easy: understanding and modeling how people compare texts, and designing algorithms to accurately perform this matching, is at the core of IR.

One typical **task** involves a user submitting a query to a **search engine** (a practical application of IR techniques to large-scale text collections) and receiving a list of documents in **ranked order**.

Such order depends on **relevance**, a key and complex concept in IR. Although there is not a formal definition, it can be described as the likelihood of an information to satisfy the user need.

There are many factors that influence whether a user considers relevant or not a document (e.g. contextual factors, time, location, language ...).

Sometimes a document which is **topical relevant** to a query (i.e. that talks about the same topic) may not be **relevant to the user** (i.e. that satisfy the user information need).

To address the issue of relevance, researchers propose various retrieval models and test how well they work through experimental evaluation (later described).

A **retrieval model** is a formal representation of the process of matching a query and a document. It is part of the ranking algorithm that is used in a search engine to produce the ranked list of documents.

A good retrieval model will find documents that are likely to be considered relevant by the person who submitted the query.

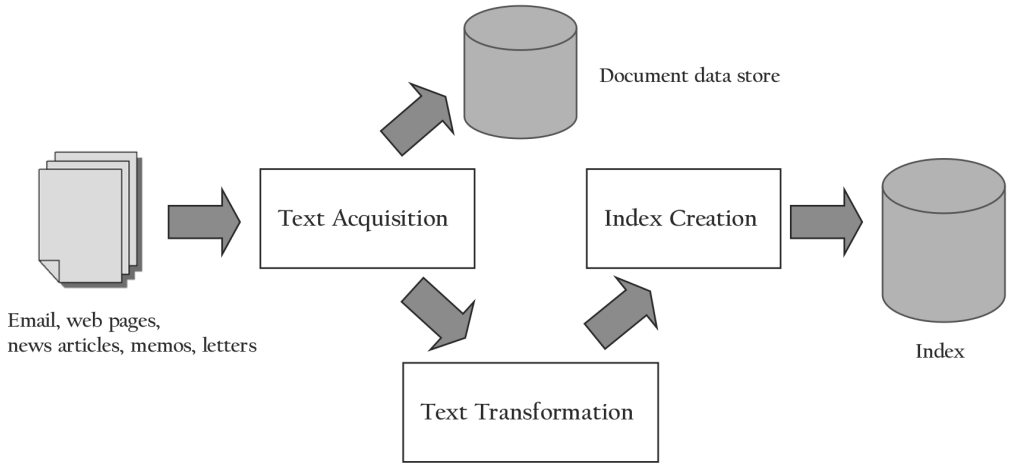


Figure 2.1: The indexing process (picture taken from [4])

2.2 Principal components of a retrieval systems

Search engine components support two major functions: the indexing process and the query process [4].

The indexing process builds the structures that enable searching and the query process uses those structures and a query submitted by a user to produce a ranked list of documents.

2.2.1 Indexing

The major components of indexing are text acquisition, text transformation, and index creation (picture 2.3).

Text acquisition is the process of acquiring the documents that will be searched.

Although in some cases this will involve simply using an existing collection, text acquisition will more often require building a collection by scanning the Web or other sources of information.

The **text transformation** component transforms documents into *index terms* or *features* by processing them through the application of several techniques - e.g., split documents into units terms (tokenization), removing common words (stopwords), clearing from punctuation, reduce similar words to their root (stemming).

In order to compare a query and a document, text transformation should be applied to both of them in the same way.

In the following, a description of sub-steps of text transformation component is given.

2.2.1.1 Parsing

A parser usually processes text and can recognize its structure, whenever it is composed of tags, links, images, title or headings.

Tokenization happens during parsing and determines how the text should be

broken down to terms - often this comes down to split text into words separated by a space.

However, this simple rules does not specify how to behave with languages like Chinese where there is no obvious word separator or with special characters such as capital letters, hyphens, and apostrophes.

2.2.1.2 Stopwords removal

In language, some words occur more frequently than others. For instance, in English, "and" or "the" are the two most frequent words, accounting for about 10% of all word occurrences ([4]).

This was observed by Luhn in 1958: he proposed that the significance of a word depended on its frequency in the document.

By looking at the distribution of word frequencies in a large sample of text, it can be noticed that it is very skewed. Typically, about one half of all the unique words in that sample occur only once.

This distribution is described by Zipf's law, which states that the probability of occurrence of the r-th most common word P_r is inversely proportional to its rank r:

$$r \cdot P_r = c \qquad \text{(Zipf's law)}$$

For English, this constant is approximately equal to 0.1. The graph for Zipf's law is shown in the following figure (2.2):

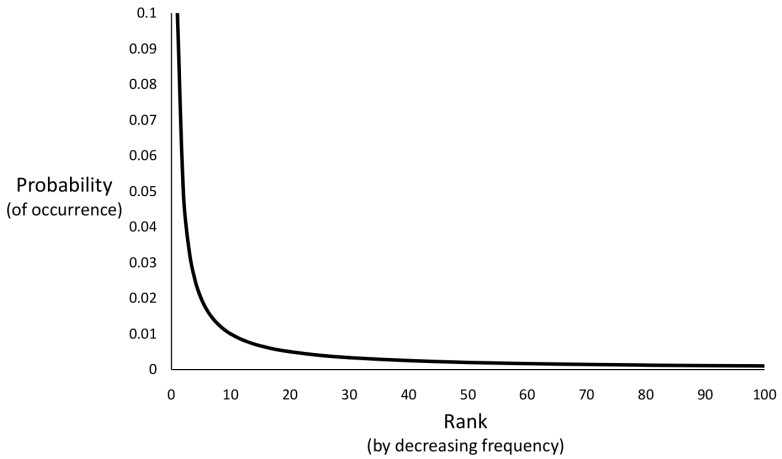


Figure 2.2: Rank versus probability of occurrence for words assuming Zipf's law (rank · probability = 0.1)

This shows that a small number of words account for a very significant fraction of all text's size.

These terms make very poor index terms because of their low discriminative power. The job of a stopping component is to prevent them to appear in the index which size, in turn, results much smaller.

Depending on the retrieval model that is used as the basis of the ranking, removing these words usually has no impact on the search engine's effectiveness, and may even improve it.

However, the choice of words in *stopwords list* may not be so easy: removing all common words makes impossible for the user to formulate queries like "to be or not to be", in which frequent words are put together to create meaningful content.

Some stoplists can be found at https://github.com/igorbrigadir/stopwords. They are taken from various sources such as Galago, Lucene, Terrier, NLTK and Gensim.

2.2.1.3 Stemming

The task of the stemming component (or stemmer) is to group words that are derived from a common stem (for instance, "fish" and "fishing") and replace them with that stem ("fish").

Stemming generally produces improvements in ranking effectiveness because it increases the likelihood that words used in queries and documents will match.

There are two types of stemmers: rule-based and dictionary-based. A rule-based stemmer is a logic procedure that decide whether two words are related, usually based on knowledge of word suffixes, whereas a dictionary-based stemmer uses pre-created dictionaries of related terms to transform text.

The most popular rule-based stemmer to english is the Porter stemmer [1].

An example of an hybrid approach (rule-based and dictionary-based) is the Krovetz stemmer [2].

Another open source project called "Snowball" [3] was launched in 2001: it is a small string-handling language that can be used for several languages other than english.

Both stemming and stopwords removal do not follow a general rule and can be done aggressively, conservatively, or even be ignored - it all comes down to the type of text to process.

2.2.1.4 Index creation

Index terms are the parts of a document that are stored in the index and used in searching.

The simplest index term is a word, but not every word may be used for searching.

"Feature" is a term often used in the field of machine learning to refer to a part of a text document that is used to represent its content, which also describes an index term.

All the indexed terms contribute to create the *index vocabulary*.

The **index creation** component takes the output of the text transformation and creates the indexes or data structures that enable fast searching.

Inverted indexes are by far the most common form of index used by search engines. For each term indexed, an inverted index contains a list of documents called *posting list*.

Elements of this list are called *posting* and typically they are the documents in which the index term appears.

[1] https://tartarus.org/martin/PorterStemmer
[2] https://sourceforge.net/p/lemur/wiki/KrovetzStemmer
[3] https://snowballstem.org

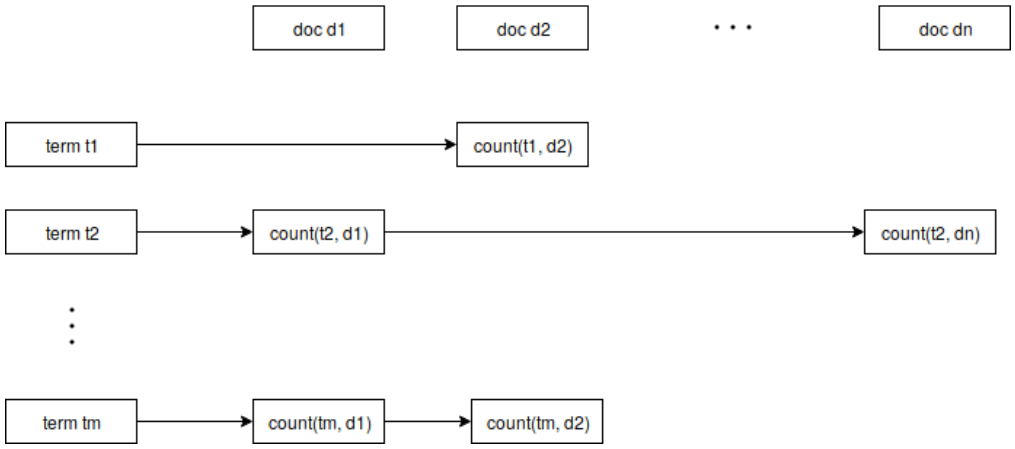


Figure 2.3: An example of inverted index where postings are based on counting term occurrences in the document list

Given the large number of documents in many search applications, index creation must be efficient, both in terms of time and space.

During index creation, other information may be collected such as **document statistics**, which may vary upon what is needed by the ranking model used by the search engine.

Generally, some useful information for traditional IR ranking models are the counts of index term occurrences (both words and more complex features) in individual documents, the positions in the documents where the index terms occurred, the lengths of documents in terms of the number of tokens and possibly many more.

Index term may also be associated to **weights** to reflect their relative importance in documents (see tf-idf model).

2.2.2 Query process

The major components of query process are user interaction, ranking, and evaluation (picture 2.4).

The **user interaction** component provides the interface between the person doing the searching and the search engine.

Some tasks for this component include accepting the user's query and transforming it into index terms, display the ranked results and refining the query submitted.

The **ranking component** is the core of the search engine. It takes the transformed query from the user interaction component and generates a ranked list of documents using scores based on a retrieval model and the index created during the indexing process.

As happens with indexing, ranking have the desirable requirements of efficiency, which depends on indexes, and effectiveness, which depends on the retrieval model.

The **evaluation** is another important part of a search engine and, more generally, a fundamental concept in IR. Since the quality of a document ranking depends on how well it matches a user's expectations, it was necessary early on to develop

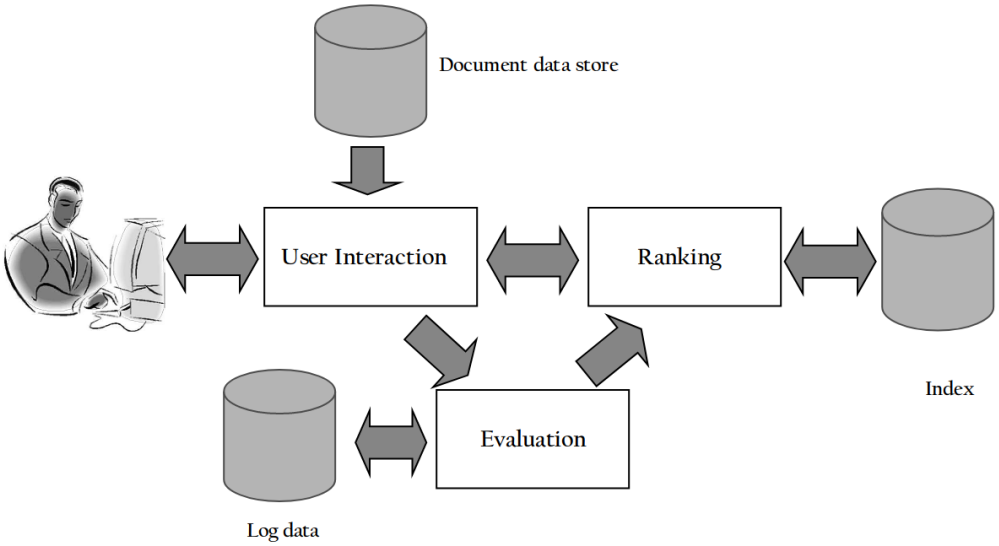


Figure 2.4: The query process (picture taken from [4])

evaluation measures and experimental procedures for evaluating retrieval systems and compare different ones.

2.2.3 The ranking task

Suppose that there are a set of queries $\mathcal{Q} = \{q_1, \dots q_M\}$ and a set of documents $\mathcal{D} = \{d_1, \dots d_N\}$. Given a query $q \in \mathcal{Q}$, and a document $d \in \mathcal{D}$, ranking is performed through a ranking function F(q, d): $\mathcal{Q} \times \mathcal{D} \to \mathbb{R}$:

$$s_{q,d} = F(q, d) \qquad \text{(Ranking model)}$$

which gives a score $s_{q,d}$ for every pair (query q, document d) representing the relevance of d w.r.t. q. Then, in order to obtain a ranking list π, elements in $\mathcal{D}$ are sorted by their score $s_{q,d}$.

$$\pi = sort_{S_{q,d}}(\mathcal{D}) \qquad \text{(Ranking list)}$$

2.2.4 Retrieval

In this section, two traditional probabilistic retrieval models are described and will be used later for comparison purposes against a neural retrieval model.

2.2.4.1 Vector space model

In this model, documents and queries are assumed to be part of a t-dimensional vector space, where t is the number of index terms.

A document D_i is represented by a vector $(d_{i1}, d_{i2}, \dots d_{in})$, where d_{ij} represents the weight of the jth term.

A document collection containing n documents can be represented as a matrix of term weights (see 4.1).

Given this representation, documents can be ranked by computing the distance between the points representing the documents and the query.

In fact, in this model, it is implicitly assumed that relevance is related to the similarity of query and document vectors.

Several similarity measures can be used to compare terms. A common one is cosine similarity, that determines how similar two vectors $(\vec{v}, \vec{w})$ are:

$$\frac{\vec{v}^t \cdot \vec{w}}{\|\vec{v}\|_2 \cdot \|\vec{w}\|_2} \qquad \text{(Cosine similarity)}$$

where $\|\vec{v}\|_2$ is the SRSS (euclidean norm - square root of the sum of squares).

The cosine correlation measures the cosine of the angle between the query and the document vectors.

A popular weighting scheme applied in the vector space model is *tf-idf* [4]: tf stands for *term frequency*, the frequency of index term occurrences in a document while idf stands for *inverse document frequency*, the reciprocal of the frequency of index term occurrences over the entire collection of documents.

citare qualcosa per tf-idf

Idf weights are useful because they can give information about which terms are frequent or rare over the entire collection of documents.

If a term occurs in many times in the collection, then it cannot discriminate between documents and, consequently, is not useful for retrieval.

A typical formula for idf is $log\left(\frac{N}{n}\right)$ where N is the number of documents indexed and n is the number of documents that contain the term.

The effects of these two weights are combined by multiplying them (due to empirical results).

2.2.4.2 Binary Independence model

One early theoretical statement known as the **probability ranking principle** [24], encouraged the development of probabilistic retrieval models, which are the dominant paradigm today [4].

The Probability Ranking Principle can be summed up as follow: "if a retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance, the effectiveness of the system will be the best that is obtainable on the basis of the data available".

Given some assumptions, such as that the relevance of a document to a query $(p(R|d,q))$ is independent of other documents and it is binary (relevant / non relevant), it is possible to show that this statement is true.

A document is relevant if $p(R|d,q) > p(NR|d,q)$, i.e. if its probability of being relevant is greater than its probability of being non relevant.

Thanks to Bayes rule the computation can unfold as follow:

$$P(R|d,q) = \frac{P(d,q|R) \cdot P(R)}{P(d,q)} \qquad \text{(Bayes rule)}$$

$$P(d,q|R) \cdot P(R) > P(d,q|NR) \cdot P(NR) \implies \frac{P(d,q|R)}{P(d,q|NR)} > \frac{P(NR)}{P(R)} \qquad (2.1)$$

The left-hand side of equation 2.1 is known as the **likelihood ratio**.

Thus, the highly ranked documents will be those that have a high likelihood of belonging to the relevant set.

To compute $P(d, q|R)$ and $P(d, q|NR)$ the following assumptions are made:

1. documents are a combination of words, the relevant and non-relevant sets are probabilities distributions over word;

2. each term is independent of one another (term independence, also known as the Naive Bayes assumption) and

3. documents are represented as a vector of binary features, $D = (d_1, d_2, \ldots d_t)$, where $d_i = 1$ if term i is present in the document, and 0 otherwise.

So, $P(d, q|R)$ is the product of individual term probabilities (p_i for a term $t_i \in d$) where d is a relevant document while $P(d', q|NR)$ is the product of individual term probabilities (s_i for a term $t_i \in d'$) where d' is a non relevant document.

If there are no other information about the relevant set, then it can be assumed that p_i is a constant and s_i can be estimated by using the term occurrences in the whole collection as an approximation.

This shows that, in the absence of information about the relevant documents, the term weight derived from the binary independence model is very similar to an idf weight. There is no tf component, because the documents were assumed to have binary features.

This model is generally known as the **binary independence model** (BIM).

The absence of a tf component makes a significant difference to the effectiveness of the ranking, and most effectiveness measures will drop by about 50% if the ranking ignores this information.

It turns out, however, that the BIM is the basis for one of the most effective and popular ranking algorithms, known as **BM25**.

BM25 extends BIM scoring function to include document and query term weights.

The extension is based on probabilistic arguments and experimental validation, but it is not a formal model.

2.2.4.3 Probabilistic language modelling

The simplest form of language model, known as unigram language model, is a probability distribution over the words in the language. This means that the language model associates a probability of occurrence with every word in the index vocabulary for a collection.

An n-gram model predicts a word based on the previous n - 1 words. The most common n-gram models are bigram and trigram models.

In a language modelling based approach, documents are ranked by the posterior probability $P(d|q)$ that a document d is relevant w.r.t. a query q.

In a **query likelihood retrieval model** documents are ranked by the probability that the query text could be generated by the document language model.

This is a model of topical relevance, in the sense that the probability of query generation is the measure of how likely it is that a document is about the same topic as the query.

An obvious estimate of such distributions is just the frequency of a term in the query over a document.

The major problem with this estimate is that if any of the query q words are missing from a document d, the score given by the query likelihood model for $p(d|q)$ will be zero. This is clearly not appropriate for long queries.

Smoothing is a technique for avoiding this estimation problem and overcoming data sparsity.

The general approach to smoothing is to lower the probability estimates for words that are seen by multiplying them by a certain quantity $1-\alpha$ in the document text, and give the probability left over α to the estimates for the words that are not seen in the text.

The estimates for unseen words are usually based on the frequency of occurrence of words in the whole document collection.

Both tf-idf and language modelling based approaches estimate document relevance based on the count of only the query terms in the document; the position of these occurrences and the relationship with other terms in the document are ignored.

2.3 Experimental evaluation

As previously introduced, evaluation is a core issue in IR. It is an expensive activity both in terms of time and energies needed to perform it, in fact early experiments conducted by Cleverdon took years to be concluded (the first, Cranfield-I, runned from 1957 to 1961 and the second, Cranfield-II, runned from 1963 to 1966).

2.3.1 Cranfield paradigm

The first person to set the stage for IR experimental evaluation was Cyril Cleverdon, who developed the Cranfield paradigm ([19]).

After WW2, there has been a huge increase in the volume of scientific papers and so, an efficient way to index and retrieve them was required, based on the information needs of researchers.

2.3.1.1 Cranfield-I

Back in 1955 the first experiment (called *Cranfield-I*) began: 18000 papers and reports from the field of aerodynamics were manually indexed on the basis of 4 different indexing systems.

It took 2 years of work before the indexing was completed. After that, Cleverdon asked the authors of the indexed documents to select some of them and frame a question that could be answered by that document (thus, avoiding the need of explicit relevance judgements).

The searching phase of the experiment required using each of the 4 indexes to manually search for the documents, recording the search time and the success (or failure) of the search.

The search failed an average of 35% of the time, with no significant differences among the indexing systems.

Cleverdon was able to discover the real problem simply because of the huge amount of data that was examined in the failure analysis.

The issue was not the specific indexing system used, but rather the actual content descriptors that were used to index each document.

2.3.1.2 Cranfield-II

The problem of how to select content descriptors for indexing, and an increasing interest in evaluation issues, led Cleverdon to continue his investigations in a second experiment (called *Cranfield-II*).

That time, the experiment was conducted with 1200 documents and 300 questions. Cleverdon thought that it was critical to first build the test collection (documents, questions, and relevance judgments), and then do the experiments on the indexing and searching.

The documents needed to be ones that researchers would naturally be search, the questions needed to reflect ones they might ask and the relevance judgments needed to mirror the type of judgments they would make for documents they examined in the search process.

The papers along with their references were sent to their authors who were asked to formulate the basic problem in the form of a search question, and then assess the relevance of each paper w.r.t. that question.

There were 173 useful forms returned, with an average of 3.5 questions per form. The document collection was built by merging the 173 source documents with their cited documents (those that had been previously sent to the authors for judgments), and 209 additional similar documents, for a total of 1400 documents.

Five graduate students spent the summer of 1963 making preliminary (and liberal) judgments for these 361 questions against all 1400 documents.

The goal of Cranfield 2 was to examine in more depth the various properties of index methodologies.

	Relevant	**Non-Relevant**	
Retrieved	a	b	a + b
Not retrieved	c	d	c + d
	a + c	b + d	

Table 2.1: Categories of document in searching

The metrics used referred to the well-known categories shown in table 2.1; at that time they were called **"Recall Ratio"** defined as $\frac{a}{a+c}$, and the **"Precision Ratio"** defined as $\frac{a}{a+b}$. Cleverdon liked both the simplicity of recall and precision and the fact that they directly described a user's experience.

An important outcome of these experiments was the Cranfield paradigm for evaluation.

Today, for evaluation purposes, it is common the use of a static test collection of documents, questions, and relevance judgments, often with standard recall and precision metrics.

The collection of scientific documents, the selection of users that would heavily use this collection, and the careful definition of relevance based on how these particular users might judge documents were critical pieces of the Cranfield paradigm.

Other important components of the experiment were the careful modeling of the task being tested and the strict separation of the building of the test collection from the experimentation itself.

2.3.2 International IR evaluation campaigns

Nowadays there are public, large-scale evaluation campaigns at international level that produce large experimental collections and compare state-of-the-art systems and algorithms [4].

Relevant and long-lived examples are the Text REtrieval Conference (TREC) [32] in the United States, the Conference and Labs of Evaluation Forum (CLEF) initiative in Europe, and the NII Testbeds and Community for Information access Research (NTCIR) in Japan and Asia.

In this context, it is worth to mention some important conferences in IR such as SIGIR [4], considered the most important in the field of IR (internationally) and ECIR [5] (in Europe).

2.3.2.1 TREC

TREC (co-sponsored by the National Institute of Standards and Technology (NIST) and U.S. Department of Defense) was started in 1992 as part of the TIPSTER Text program.

Its purpose is to support research within the IR community by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies.

In particular, the TREC workshop series has the following goals:

* to encourage research in IR based on large test collections;
* to increase communication among industry, academia, and government by creating an open forum for the exchange of research ideas;
* to speed the transfer of technology from research labs into commercial products;
* to increase the availability of appropriate evaluation techniques for use by industry and academia.

For each TREC, NIST provides a test set of documents and questions.

Participants run their own retrieval systems on the data, and return to NIST a list of the retrieved top-ranked documents. NIST pools the individual results, judges the retrieved documents for correctness, and evaluates the results.

The TREC cycle ends with a workshop that is a forum for participants to share their experiences.

[4] acronym for Special Interest Group on Information Retrieval

[5] European Conference on Information Retrieval

As already stated, relevance is a complex concept. TREC uses the following definition:

> **Working definition of relevance**: if was to write a report on the subject of the topic and would use the information contained in the document in the report, then the document is relevant.

Only binary judgments (relevant or not relevant) are made, and a document is judged relevant if any piece, however small, of it is relevant. This follows the **scope hypothesis** which states that relevance matching could happen in any part of a document, as opposed to the **verbosity hypothesis** where the whole document is required to be relevant to a query (global relevance).

Judging is done using a **pooling technique** on the set of documents used for the task that year: only a subset of documents returned by different retrieval systems are manually judged.

2.3.3 Formal framework

In this section I will present a detailed explanation of experimental evaluation and the metrics typically applied to measure the performance of an IR search engine, based on the formalization given in [2].

A test collections is a triple: $(\mathcal{D}, \mathcal{T}, \mathcal{GT})$, where $\mathcal{D}$ is a set of documents, also called collection of documents, $\mathcal{T}$ is a set of topics, which express user information needs and $\mathcal{GT}$ is the ground truth, defined below.

Let REL be a set of relevance degrees and let $\succ$ be a total order relation on REL so that $(REL, \succ)$ is a total ordered set. $min(REL)$ is what defines a non-relevant degree.

The ground truth can be formalized in the following way:

Definition 2.3.1. Ground truth
The ground truth is a function:

$$\begin{aligned} \mathcal{GT}\colon \mathcal{T} \times \mathcal{D} &\to REL \\ (t, d) &\mapsto rel. \end{aligned}$$

The recall base is linked to the definition of ground truth and it is the total number of relevant documents for a given topic t:

Definition 2.3.2. Recall base
The recall base is a function:

$$\begin{aligned} \mathcal{RB}\colon \mathcal{T} &\to \mathbb{N} \\ t &\mapsto \mathcal{RB}_t = |\{d \in \mathcal{D} | \mathcal{GT}(t, d) \succ min(REL)\}|. \end{aligned}$$

An experiment in IR, also called **run**, is the output of an IR system which usually is composed by a set of ranked lists of documents – one for each topic.

Definition 2.3.3. Run

A run is a function:

$$\begin{aligned} \mathcal{R} \times \mathcal{T} &\to \mathcal{D}^n \\ t &\mapsto r_t = (d_1, d_2, \dots d_n). \end{aligned}$$

where r_t documents $d_1, d_2, \dots d_n$ are all different.

2.3.3.1 Metrics

The most used metrics in IR are based on a rank-based comparisons of the retrieved result set $\mathcal{R}$ ($\mathcal{R}_q$ is a run w.r.t. a query q) to an ideal ranking of documents $\mathcal{I}$, determined by manual judgments or implicit feedback from user behavior data (ground truth).

These metrics are typically computed at a rank position (k) and then averaged over all queries in the test set.

Before presenting them, a useful definition is the **relevance score** of a run:

Definition 2.3.4. Relevance score

Given a run $\mathcal{R}(t) = r_t$, its relevance score is a function:

$$\begin{aligned} \hat{\mathcal{R}} \colon \mathcal{T} \times \mathcal{D}^n &\to REL^n \\ (t, r_t) &\mapsto \hat{r_t} = (rel_1, rel_2, \dots rel_n). \end{aligned}$$

where $\hat{r_t}[j] = \mathcal{GT}(t, r_t[j])$

For simplicity, from now on, I assume that REL is simply a binary set ({rel, non rel }) which can be weighted as $\{0, 1\}$, so $\hat{r_t}$ is a binary vector.

In the following, formal definitions of the metrics used in Cranfield experiment (precision and recall) are given.

Precision indicates the ratio of relevant documents retrieved by a system over the total number of retrieved documents:

Definition 2.3.5. Precision

Let $\mathcal{R}(t) = r_t$ be a run with length $N \in \mathbb{N}^+$ where $t \in \mathcal{T}$ is a given topic. Precision is defined as follow:

$$P_{r_t} = \frac{1}{N} \sum_{j=1}^{N} \hat{r_t}[j]$$

It's worth notice that precision differs from **accuracy** (the rate of correct predictions), often used for classification task in machine learning. In fact, the task considered here is document ranking which is different from document classification.

Recall indicates the ratio of relevant documents retrieved by a system over the total number of relevant documents for a given topic (i.e. its recall base):

Definition 2.3.6. Recall

Let $\mathcal{R}(t) = r_t$ be a run with length $N \in \mathbb{N}^+$ where $t \in \mathcal{T}$ is a given topic. Recall is defined as follow:

$$R_{r_t} = \frac{1}{\mathcal{RB}_t} \sum_{j=1}^{N} \hat{r}_t[j]$$

By looking at the context, one measure may be more important than the other (e.g. in web search it is more important to get a high precision on the first page than a high recall considering all pages retrieved).

These two quantities trade off against one another: it is possible to get a high recall but very low precision by retrieving all documents for all queries, in fact recall increases as the number of documents retrieved increase.

Precision and recall can be defined at different levels of cut-off: this means that a run is partially evaluated up to a certain threshold / cutoff k. The notation used may vary, for instance to indicate precision at k one may use either $P@k$ or $P[k]$.

Average precision is a measure that combines precision and recall.

It is **top-heavy**: it means that it assign more weight to a relevant document placed at high rank than a relevant document placed at low rank.

Definition 2.3.7. Average Precision

Let $\mathcal{R}(t) = r_t$ be a run with length $N \in \mathbb{N}^+$, where $t \in \mathcal{T}$ is a given topic and RB_t its recall base. Average Precision is defined as follow:

$$AP_{r_t} = \frac{1}{RB_t} \sum_{j=1}^{N} \hat{r}_t[j] \cdot \frac{\sum_{h=1}^{j} \hat{r}_t[h]}{j}$$

The MAP value for a test collection is the arithmetic mean of average precision values for each topic.

Definition 2.3.8. Mean Average Precision

Let $\mathcal{S}$ by an IR search engine, $\mathcal{T}$ a set of topics and r_t the run generated by $\mathcal{S}$ associated to the topic $t \forall t \in \mathcal{T}$. Mean Average Precision is defined as follow:

$$MAP_{\mathcal{T}} = \frac{\sum_{t \in \mathcal{T}} AP_{r_t}}{|\mathcal{T}|}$$

This metric has the drawback of weighing each topic equally in the final reported number, even if many documents are relevant to some queries whereas very few are relevant to other queries - as happens with the results reported in experimental section (7.4).

Precision, Recall, Average Precision and MAP can only handle binary judgments (relevant/not relevant), whereas cumulative gain family of measures in IR can handle graded relevance degrees.

The premise of these measures is that highly relevant documents appearing lower in a search result list should be penalized.

Definition 2.3.9. Cumulative gain (CG)

Let $\mathcal{R}(t) = r_t$ be a run with length $N \in \mathbb{N}^+$, where $t \in \mathcal{T}$ is a given topic, $\mathcal{RB}_t$ its recall base and $1 \leq j \leq N$ the cutoff up to which the run should be evaluated. $CG[j]$ is defined as follow:

$$CG[j] = cg_{r_t} = \sum_{k=1}^{j} \hat{r}_t[k]$$

The normalized version of CG (nCG) definition uses the concept of "ideal run" which is a run that, given a topic $t \in \mathcal{T}$, has retrieved all relevant documents and place them in the best possible order. Formally:

Definition 2.3.10. Ideal run I(t)

Given a topic $t \in \mathcal{T}$, the ideal run $\mathcal{I}(t) = i_t$ is a run which satisfies the following constraints:

1. recall base: $\forall t \in \mathcal{T}, |\{1 \leq i \leq N \quad \mathcal{GT}(t, i_t[j]) \succ min(REL)\}| = RB_t$
2. ordering: $\forall t \in \mathcal{T} \quad \forall j, k \quad j < k \implies i_t[j] \succeq i_t[k]$

Definition 2.3.11. Normalized cumulative gain (nCG)

nCG at position j is defined as the ratio between the CG of the considered run $R(t) = r_t$ and the CG of the ideal run $\mathcal{I}(t) = i_t$:

$$nCG[j] = \frac{cg_{r_t}[j]}{cg_{i_t}[j]}$$

The discounted cumulative versions of these metrics give another view of the run by weighting down the gain received through documents found later in the ranked results, thus giving more importance to the early positions in ranking. Both discounted cumulative gain (DCG) and normalized DCG (nDCG) are top-heavy.

DCG and nDCG uses the following definition of discounted gain (dg):

Definition 2.3.12. Discounted gain (dg)

Let $\mathcal{R}(t) = r_t$ be a run with length $N \in \mathbb{N}^+$, where $t \in \mathcal{T}$ is a given topic and a log base $b \in \mathbb{N}^+$; for all $1 \leq k \leq N$, the discounted gain is defined as follow:

$$dg^b_{r_t} = \begin{cases} \hat{r}_t[k] & \text{if k < b} \\ \frac{\hat{r}_t[k]}{log_b k} & \text{otherwise} \end{cases}$$

So, DCG can be defined as follow:

Definition 2.3.13. Discounted cumulative gain (DCG)

Let $\mathcal{R}(t) = r_t$ be a run with length $N \in \mathbb{N}^+$, where $t \in \mathcal{T}$ is a given topic and a log base $b \in \mathbb{N}^+$. The discounted gain at some cutoff j is:

$$DCG[j]^b_{r_t} = \sum_{k=1}^{j} dg^b_{r_t}[k]$$

The respective normalized version uses the ideal run, just like nCG:

Definition 2.3.14. Normalized discounted cumulative gain (nDCG)

nDCG at position j is defined as the ratio between the DCG of the considered run $R(t) = r_t$ and the DCG of the ideal run $\mathcal{I}(t) = i_t$:

$$nCG[j] = \frac{DCG[j]^b_{r_t}}{DCG[j]^b_{i_t}}$$

2.4 Advanced topics in IR

Recently, different approaches to IR have been explored by researchers.

Here, two new areas are presented: one, called **learning to rank** (LTR) emerged from the intersection of machine learning, IR and natural language, and the other, called **Neural IR**, focuses on the application of neural network to IR tasks.

Unlike classical LTR models and non-neural approaches to IR, Neural IR techniques are data-hungry, requiring large scale training data before they can be deployed.

Those two areas are not strictly separated: in fact neural networks have been employed over the years as LTR models. However, Neural IR exploits more in depth neural networks and can use them even without the need of labelled data (i.e. ground truth).

2.4.1 Learning to Rank

Learning to rank refers to machine learning techniques applied in a ranking task.

Liu in [16] give the following general definition:

> "(...) LTR is the task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance (...)"

LTR models take as input a numerical vector $\vec{x} \in \mathbb{R}^n$ which represents a rankable item (e.g. a document) given some context (e.g. a user-issued query).

The ranking model $F : \vec{x} \to \mathbb{R}^n$ is then trained to map the vector to a real-valued score such that relevant items are scored higher.

2.4.1.1 Training and testing

Let $\mathcal{Q} = \{q_1, \dots q_M\}$ and $\mathcal{D} = \{d_1, \dots d_N\}$ be the queries and documents set respectively.

Suppose that $\mathcal{Y} = \{1, \dots l\}$ is a set of grades for which exists a total order $l \succ l-1 \succ \cdots \succ 1$.

Let S be the training set: $S = \{(q_i, d_j), y_{i,j}\}, 0 \leq i \leq m, 0 \leq j \leq n$ where each pair (query q_i, document d_i) is associated to a relevance score $y_{i,j}$.

The aim is to train a model $F(q, d)$ based on the training set S, that can assign a score to any pair (q, d).

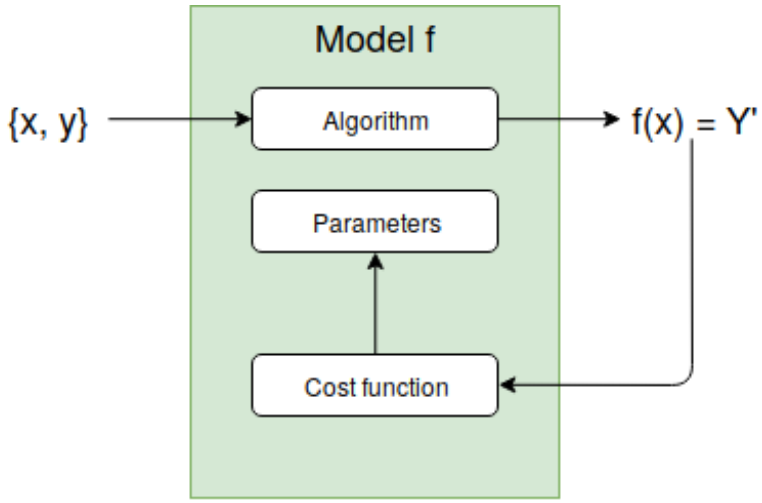


Figure 2.5: Learning to rank abstract model

Ranking uses the output of F to sort documents in $\mathcal{D}$ by the scores obtained.

2.4.1.2 Learning to rank approaches

Liu in [16] categorizes 3 different LTR approaches on the basis of training objectives (i.e. loss function, explained in next chapter):

* **Pointwise approach**: Typically, a regression or classification model is trained to predict a relevance label $y_{q,d}$ given $\vec{x}_{q,d}$. During training each document is considered separately;
* **Pairwise approach**: Binary classification model trained to predict which document in a given pair is the most relevant with respect to a certain query. Pairwise loss minimizes the average number of inversions in ranking (i.e. $y_{i,h} \succ y_{i_k}$ but document d_k is ranked higher than d_h w.r.t. query q_i). This approach is sensible to noise as a mis-labeled document can results in a huge number of mislabeled pairs.
* **Listwise approach** This approach considers the entire list of documents and try to come up with an ordering that optimizes an IR measure (i.e. nDCG). The resulting optimization problem is more challenging than those in the pointwise or pairwise approach, due to the non-continuous and non-differentiable loss function.

The pointwise and pairwise approaches transform the ranking problem into classification, regression, and ordinal classification problem. The listwise approach addresses the learning-to-rank problem in the most natural way, by taking document lists as instances in the learning process.

The main differences among the approaches actually lies in the loss functions employed.

Similarly to LTR, in Neural IR the ranking objective function adopted can be categorized in the three categories mentioned in the list above.

2.4.2 Neural IR

Since the turn of the decade, there have been dramatic improvements in performance in computer vision, speech recognition, and machine translation task. These breakthroughs were largely fuelled by recent advances in neural network models, which have captured the attention of IR community.

Compared to traditional LTR models that employ machine learning techniques over hand-crafted IR features (i.e. features manually computed from input text which are not learned), neural models can learn representations of language from raw text.

Such representations can bridge the gap between query and document vocabulary and directly address the **vocabulary mismatch problem** (when different words are used to express the same concept) - which often occurs when comparing pairs of short-texts.

This turns out to be particularly useful, for instance, in search task where there is a lexical gap between search queries and retrieved documents (e.g. in bug and feature location, where queries are expressed in natural language and documents are expressed in code / some programming language).

Many Neural IR models depend on learning good low-dimensional vector representations - or **embeddings** - of query and document text, and using them within traditional IR models or in conjunction with simple similarity metrics (e.g., cosine similarity).

Unlike classical IR models, these new machine learning based approaches are data-hungry and require large scale training data before they can be deployed.

Motivations for Neural IR include increased availability of big data, more powerful computing resources, and better neural network models and parameter estimation techniques [36].

IR community interest for neural approaches begun when Mikolov and Dean et al. [20] proposed **word2vec**, a model and estimation procedure for word embeddings (also known as distributed term representations).

The availability of word2vec code and existing pre-trained embeddings has provided one of the key avenues for early work on Neural IR, especially for extending traditional IR models.

2.5 Resources

There are several open source resources for IR, some of which are used in this work. Among them, the most known are the search engines Terrier ([30]), Lucene and Galago and the evaluation tool trec eval.

2.5.1 Terrier

Terrier IR Platform is a modular open source software written in Java for the rapid development of large-scale IR applications. Terrier was developed by members of the IR Research Group, Department of Computing Science, at the University of Glasgow.

A core version of Terrier is available as open source software under the Mozilla Public License (MPL), with the aim to facilitate experimentation and research in the wider IR community.

2.5.2 Trec eval

Trec eval is the standard tool used by the TREC community for evaluating an ad hoc retrieval run, given the results file and a standard set of judged results.

It contains several implementations of the most common metrics cited in the evaluation section.

Other open-source toolkits have also been made available to enable and ease repeatable IR search experiments: Galago, Lemur and Lucene [4].

CHAPTER 3

Artificial Neural Networks

Artificial neural network models take their inspiration from the brain.

The brain is an information processing device (very different from a computer) that surpasses current engineering products in many domains (e.g., vision, speech recognition, learning and many others).

It is composed of a very large number of processing units, the neurons, (10^{11}) operating in parallel and largely connected. Neuron connections are called synapses, and each neuron connects to around 10^4 other neurons, all operating in parallel.

In a computer, the processor is the active component while the memory is mainly separate and passive, but it is believed that in the brain, both the processing and memory are distributed together over a network.

The structure of the brain gave inspiration to a theoretical approximation [1], where a collection of processors with a small amount of local memory implements a fixed function and executes the same instructions but each processor load different values so that they can do "different things" (e.g. attributing more importance to certain patterns of data). In this way, the main operation - whatever it is - can be distributed over such processors.

This theoretical approximation is called **Neural Instruction Multiple Data** (NIMD) machine, where each processor corresponds to a neuron, local parameters correspond to its synaptic weights, and the whole structure is a **neural network**.

The process through which local parameters are updated in order to achieve the task is called **learning**.

There are two types of learning: supervised and unsupervised. Both in supervised and unsupervised approach, learning is guided by samples of data.

In **supervised learning**, samples are associated to some labels that determine their class/score/desired feature; the networks should learn a mapping between them and reflects it on unknown data.

Regression and classification are two types of problems of supervised learning.

In classification problems, the aim is to find a mapping from the input to a discrete or categorical output, while in regression problems, the input has to be mapped toward a numerical/continuous output.

In **unsupervised learning**, labels are not provided and the network should discover the patterns and regularities within the samples to learn.

3.1 Feed Forward Neural Networks

There are many types of neural networks, but in this thesis I will cover in detail just feed forward neural networks due to the fact that they are used in DRMM [10] and, in the context of that experiment (a task of ad hoc retrieval with a small data collection) have been proved to be effective compared to other approaches (e.g. convolutional in [29]).

3.1.1 The perceptron

The perceptron is the basic processing element of an artificial neural network. It takes as input a vector $\vec{x} \in \mathbb{R}^d$ from $\mathcal{X}$ and another vector of *connection weights* $\vec{w} \in \mathbb{R}^d$.

Suppose that there is a dataset that contains some data samples $\mathcal{X}$ and their labels $\mathcal{Y}$.

The output y, in the simplest case, is a linear combination of the two:

$$y = \sum_{j=1}^{d} (w_j \cdot x_j) + w_0 \qquad \text{(linear combiner)}$$

where w_0 is the *intercept value*, independent from data and it's useful to give preference to a class over another (e. g., in case of unbalanced datasets).

The perceptron (in equation linear combiner) defines a hyperplane (a generalization of the two-dimensional plane in a space with n dimensions) that in a binary classification problem can split the input space into two parts: the half-space with positive samples and the half-space with negative samples.

So, the perceptron can separate two classes by checking the sign of the output (defined as *threshold function*).

Under supervised learning, it is necessary to compare the prediction y to the label (desired output) $y' \in \mathcal{Y}$ associated to x in the dataset and define a cost function (loss) $\mathcal{L}$ that indicates how well the perceptron performed.

Such loss function is used to define the total error, given by computing the loss function over all instances in $\mathcal{X}$ received by the perceptron:

$$E(\mathcal{W}|\mathcal{X}) = \sum_{i=1}^{|\mathcal{X}|} \mathcal{L}(y'_i, y_i) \tag{3.1}$$

where $\mathcal{W}$ are the connection weights.

For instance, one pretty common loss function is the **mean squared error** function:

$$\mathcal{L}(y, y') = (y - y')^2 \tag{3.2}$$

If the error function is differentiable, a technique generally used to minimize it is **gradient descent**. The aim is to find $\mathcal{W}^*$ such that $\mathcal{W}^* = \arg\min_{\mathcal{W}} E(\mathcal{W}|\mathcal{X})$.

The gradient descent algorithm iteratively (over a fixed number of *training epochs*) updates weights using the following update equation:

$$w_{j,t+1} = w_{j,t} - \eta_t \cdot \frac{\partial \mathcal{L}}{\partial w_j} \qquad \forall j = 0 \dots d \qquad \text{(gradient descent)}$$

where t is the epoch number and η_t is a learning rate that typically declines with t.

When samples from the training set are selected randomly, a variant of gradient descent is used, called *stochastic* (or "on-line") gradient descent.

In case of large dataset, the estimation of the true gradient at each sample would be too expensive to compute. A common solution consists in the use of **mini batches** of data: the gradient is computed over small random batches of data samples.

3.1.2 Multi Layer Perceptron (MLP)

A perceptron with a single layer of weights, as described in the previous section, can only approximate linear functions; however this limitation does not apply to networks with intermediate or hidden layers between the input or output.

Such multilayer perceptrons (MLP) can approximate nonlinear functions of the input.

Input x is fed to the input layer, then the "activation" propagates in the forward direction, and the outputs z_h of the hidden units are computed. This type of networks are called "feed forward" neural networks.

For instance, the feed forward network represented in figure 3.1 uses the *sigmoid* as non linear activation function in each hidden unit:

$$z_h = \frac{1}{1 + exp(-\sum_{j=1}^{d} w_{hj} x_j + w_{h0})} \quad h = 1, \dots, H \qquad \text{(Sigmoid function)}$$

The hidden units make a nonlinear transformation from the m-dimensional input space to the H-dimensional space spanned by the hidden units, and then, in this space, the third (and final) output layer implements a linear function:

$$y_i = v_i^T \cdot z = \sum_{h=1}^{H} v_{ih} \cdot z_h + v_{i0} \tag{3.3}$$

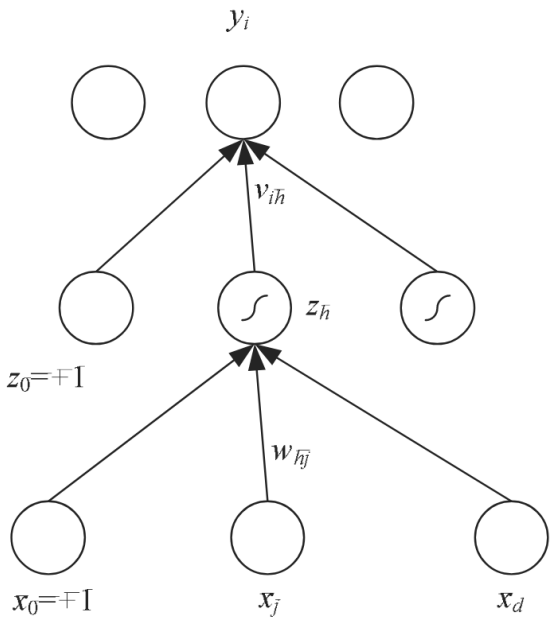


Figure 3.1: The structure of a MLP with d input units $(x_0, \ldots, x_d)$ and H hidden units $(z_0, \ldots, z_H)$ where x_0 and z_0 are the bias units. w and v are the weights for the first and the second layer respectively (image source: [1]).

Sigmoid is the continuous, differentiable version of thresholding and thus, gradient descent is suitable for learning process in MLP.

Another nonlinear function that can be used is the hyperbolic tangent function, *tanh*, which ranges from -1 to $+1$, instead of 0 to $+1$ (used in 5).

3.1.3 Complexity

A MLP with d input units, H hidden units, and K outputs has $H \cdot (d+1)$ weights in the first layer and $K \cdot (H+1)$ weights in the last layer (including the bias term).

Let e be the number of training epochs for the MLP. Then, H and e are two free parameters (in this simplified setting).

The tuning of these parameters may lead to overcomplex model, which tend to memorizes the noise in the training set and do not generalize to the test set.

This could happen when the number of hidden units is large and the generalization accuracy deteriorates.

A similar behavior occurs when training is continued too long: the error on the training set decreases, but the error on the validation set starts to increase beyond a certain point.

This problem is known as *overfitting* and can be alleviated by *stopping early* the learning process.

Because of the nonlinearity, the error function has many minima and gradient descent converges to the nearest minimum. To be able to assess expected error, the same network is trained a number of times starting from different initial weight values, and the average of the validation error is computed.

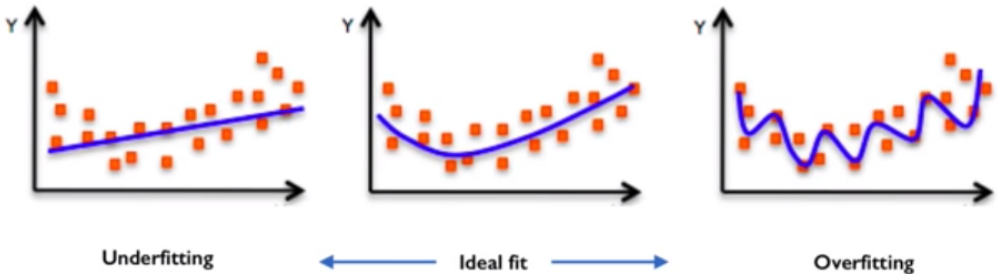


Figure 3.2: Underfitting, ideal fit and overfitting, source pingax.com/wp-content/uploads/2014/05/underfitting-overfitting.png

3.1.4 Overfitting / Underfitting problem and regularization

Underfitting happens when a model does not have the capacity to fully learn the data while **overfitting** occurs when a model is too complex and does not generalize well (or even just memorize the training data).

The ideal fit should be the purpose of a neural network, so that it does not underfit neither overfit (see figure 3.2).

In order to achieve this result, *regularization* is often used. It is a set of techniques that constraint the optimization problem to discourage complex models. Some examples:

* **Dropout**: randomly set some activation to zero. This forces the network to not rely on any particular node, and encourages it to explore different patterns on input data.

* **Early stopping**: stops the training before it overfits the data by monitoring a given metric (e.g. validation loss).

To monitor a neural network performance, the dataset available is usually split into two parts: training and testing (e.g. 50-50%). A fraction of the training part is then used as validation set to evaluate the network without updating its parameters. This is useful in order to avoid overfitting.

When the dataset available is too small, it can be exploited through k-fold cross validation so that each example in it is used both for training and validation and exactly once for validation.

With K-fold cross-validation, the dataset is partitioned into k parts. Then, k-1 parts are used as training set and the remaining part is used as validation set. This process is repeated k times so that each part is used exactly once as validation set. The k results can then be averaged to produce a single estimation.

It is often used for parameters tuning by doing it for several possible values of a set of parameters.

3.1.5 Universal approximation theorem

In 1989 Hornik, Stinchcombe and White proved an important result: a MLP with one hidden layer can learn any nonlinear function of the input.

There are some caveats in this statement to consider: the number of hidden units may be infeasibly large; the resulting model may not generalize and there are no instructions on how to build/recognize such network.

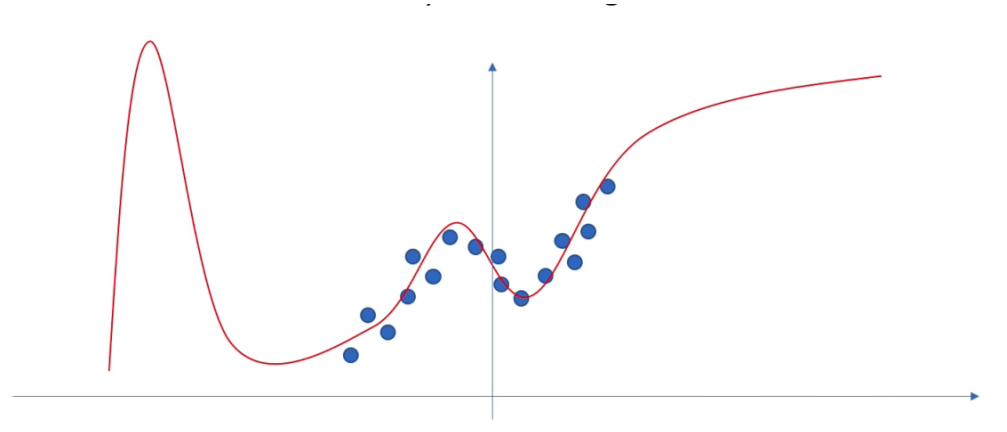


Figure 3.3: Training and test data problem

Neural networks are excellent function approximations, but there is a huge problem: as long as they have training data, they may be perform well, but there is no guarantee on their performance outside that data.

3.2 Neural networks limitation

The main advantage of neural network lies in their ability to outperform nearly every other machine learning algorithm, but this goes along with some disadvantages. They are:

* very data hungry;
* computationally intensive;
* easily fooled by adversarial examples;
* poor at representing uncertainty;
* (sometimes) uninterpretable black boxes;
* difficult to optimize: there could be many factors to consider, depending on the task;
* difficult to model: often they requires expert knowledge to design, fine tune architectures.

CHAPTER 4

Neural IR

4.1 Overview

As previously stated, IR community has recently been very interested in neural network approaches, given the fact that they deliver state-of-the-art performance in many machine learning tasks (e.g., speech recognition, computer vision, and natural language processing), they may also be beneficial in IR tasks.

Relevant researches in the last period of time (2009 - 2016) has mainly been focused on the long standing vocabulary mismatch problem in textual IR [22].

Guo et al. distinguished two categories of Neural IR models [9] based on whether their target is learning words representation (to alleviate the first of the previous problems) or a ranking function (to alleviate the second problem): representation-focused models and interaction-focused models.

Representation-focused models assume that the relevance between a query and a document depends on their compositional meaning and, therefore, they focus on learning complex, high level representations of the input text (e.g. distributed representations, explained below).

Interaction-focused models assume that the relevance between a query and a document depends on relations between them and learning is thus achieved by looking at their interactions rather than finding complex representations.

These two categories are reflected in two architectures, reported in figure 4.1. In the first part (A), queries and documents are separately passed through mirror (siamese) neural networks with shared parameters and scores are generated based on the combination of the respective outputs. By contrast, the second part (B) defines an approach in which a joint representation for queries and documents is constructed and then run through a neural network.

Alongside with these two approaches, Onal, Zhang et al. in [22] described two categories in which neural techniques and distributed representations can be used together to build a model for an IR task: *aggregate* and *learn*.

Qui ho ampliato e spostato una parte che prima era alla fine

In the first category, *aggregate*, the models rely on pre-trained word representations as external resources in order to build or extend relevance matching functions.

In the second category, *learn*, the models are designed and trained to learn "good" word representations and semantic compositionality functions for building distributed representations from scratch, given only raw text.

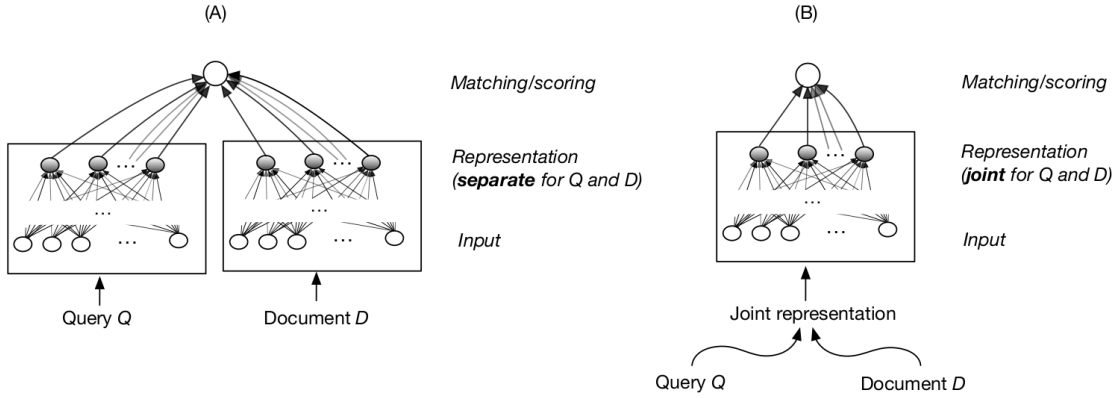


Figure 4.1: Two basic neural architectures for scoring the relevance of queries to documents: (A) representation-focused model and (B) interaction-focused model (picture taken from [22])

The approaches of the second category may vary upon the training objectives defined:

* *learn to autoencode*, where the training objective is to restore the input based on the distributed representation learned by a neural network;
* *learn to match* (under supervised learning), where the training objective is to minimize a loss function, given a neural network that computes a degree of similarity between a query and a document;
* *learn to predict*, which refer to neural language models used for learning representations of larger textual units (e.g. sentences, paragraphs, documents) from unlabelled data - and
* *learn to generate*, where a synthetic textual unit is generated based on the distributed representation of an input text.

The work studied and reproduced in this thesis belongs to the second category, more specifically, to *learn to match*.

Typically, interaction-focused and learn to match approaches are conducted under supervised learning while all the other are fall into unsupervised learning.

The proportion of labelled and unlabelled data that is available influences the level of supervision that can be employed for training a deep Neural IR model.

Since it is difficult to obtain large amounts of supervised data, when click information in click-through logs are available, they are exploited to derive similarity assessments for query-document pairs.

Moreover, if few labelled data are available, then they can be leveraged to train a retrieval model with few parameters that in turn uses text representations that is pretrained on larger unlabelled corpus (semi-supervised approach).

Much of the explorations in Neural IR models have focused on learning good representations of text [3]. However, these representation learning models tend to perform poorly when dealing with rare terms.

Based on similar motivations, the authors of the model reproduced in this thesis have recently emphasized the importance of modelling lexical matches using deep neural networks.

The majority of neural ranking models are implemented as multistage rankers: since most of them rely on semantic matching that can be achieved using distributed dense representations, computing the retrieval score for all the documents in a large-scale collection is generally infeasible.

Zamani et al. states in [35] that the representations of documents and queries are what makes the traditional term based ranking models fast, and the neural ranking models slow.

This strategy, where different IR models are chained to re-rank a smaller number of candidate documents is called **telescoping evaluation** in [3].

Telescoping evaluations are common in the Neural IR literature, where a neural model only re-ranks the top ranked documents retrieved by a first-stage efficient ranker in response to a given query.

The reliance on a first stage ranker creates a dual problem: first, the interaction and combination effects are not well understood. Second, the first stage ranker serves as a "gate-keeper" or filter, blocking the potential of neural models to uncover new relevant documents.

Moreover, Mitra et al. in [21] demonstrate that good performances on re-ranking tasks may not be indicative how a neural retrieval model would perform if the retrieval involves larger document collections. They found that, in that case, their embeddings-based approach (DESM) was prone to false positives, retrieving documents that are only loosely related to the query.

To counteract this fact, they defined a "mixture" model - a linear combination of a neural retrieval model and a traditional retrieval model (DESM + Bm25) - and evaluated it in a non-telescoping setting. The mixture model provided improvements over the performance of DESM alone. fino a qui

4.1.1 A brief history of Neural IR

The first successful Neural IR model was the Deep Structured Semantic Model (DSSM) [13], introduced in 2013, which is a supervised neural ranking model that directly tackles the ad-hoc retrieval task. In the same year, Lu and Li proposed DeepMatch [18], which is a deep matching method applied to the Community-based Question Answering (CQA) and micro-blog matching tasks.

At the same time, there was a number of studies focused on learning low-dimensional representations of text with and unsupervised neural approach; the most known model of them is *Word2Vec* [20].

Later, between 2014 and 2015, work on neural ranking models began to grow, bringing new variants of DSSM [29], ARC-I [12], ARC-II and MatchPyramid. Methods to compose word embeddings over long textual units were also proposed (e.g. Paragraph Vector [14]).

Moreover, in 2014, the first two Neural IR papers also appeared at SIGIR.

In 2016, work on Neural IR continued to accelerate in quantity of work, sophistication of methods, and practical effectiveness (e.g. [10] - the work studied and

reproduced in this thesis). SIGIR also featured its first workshop on the subject [1].

aggiunto modello SNRM

With respect to the test collection used in the experimental section (Robust04 [31]) (7.2), I explored a neural model that demonstrate state-of-the-art performances, published after DRMM.

In 2018, Zamani et al. proposed in [35] a *standalone* neural ranking model (SNRM) that addresses the inefficiency problems of neural models by enforcing and rewarding sparsity in the representation learning, creating a latent representation that aims to capture meaningful semantic relations while still matching documents.

SNRM aim is learning highly sparse representations for documents and queries that result in better matching compared to the dense term vectors and exact matching models. It can also take advantage of pseudo-relevance feedback for further improvements.

Limited training data has been a perennial problem in IR, and many machine learning-related domains. This has motivated researchers to explore building models using pseudo-labels. For example, pseudo-relevance feedback (PRF) assumes that the top retrieved documents of a previous search result in response to a given query are relevant.

Although this assumption does not necessarily hold, PRF has been proven to be effective in many retrieval settings.

Onal et al. in [22] reviewed a huge amount of works and drew some interesting conclusions about Neural IR, along with some criticisms.

While neural networks methods have worked quite well on short texts, effectiveness on longer texts typical of ad-hoc search has been problematic with only very recent evidence to the contrary [10].

Side by side comparisons of lexical versus neural methods often show at least as many losses as gains for neural methods, with at best an advantage “on average”.

In addition, deep structures with a very large number of hidden layers have typically been less effective in IR compared to their shallow counterpart.

When Neural IR has led to improvements in ad-hoc search results, improvements appear relatively modest when compared to traditional query expansion techniques for addressing vocabulary mismatch problem.

Query expansion (QE) is a process in IR which consists of selecting and adding terms to the user’s query with the goal of minimizing query-document mismatch and thereby improving retrieval performance; however it is not relevant w.r.t. the work in this thesis.

In [9] Guo et al. argue that there is still a long way to go for neural ranking models: 1) they have not had the level of breakthroughs achieved by neural methods in speech recognition or computer vision; 2) there is little understanding and few guidelines on the design principles of neural ranking models; 3) special capabilities of neural ranking models that go beyond traditional IR models have not been yet identified.

[1] https://www.microsoft.com/en-us/research/event/neuir2016

4.2 Text representations

Neural models have shown their importance in learning "good", dense and small text representations.

The way queries and documents are represented is very important in IR, for their representations are the building blocks of the entire IR process.

In fact, different representations of a text raise different notions of similarity.

Two popular ways to represent textual data are the following:

* **Local representation** (*one-hot* encoding/representation): given a dictionary $\mathcal{V}$, a word is represented with a vector $\vec{v}$ with length $|\mathcal{V}|$. All positions are 0 except for the position corresponding to that word (which is set to 1); $\vec{v} \in \{0,1\}^{|V|}$

* **Distributed representation**: given some features of text, a word is represented with a vector $\vec{v}$ of |features| positions with real values; $\vec{v} \in \mathcal{R}^{|features|}$

When the vectors are high-dimensional, sparse, and based on distributional features they are referred to as *explicit vector representations*.

On the other hand, when the vectors are dense, small ($k << |V|$), and learned from data then they are commonly referred to as **embeddings**.

In a distributed representation the feature space is very important because it influences the notion of similarity captured.

Popular features used are distributional properties of terms (e.g., document term frequency, neighboring terms with or without distances, etc.) and different weighting schemes (e.g., TF-IDF) applied over the raw counts.

4.2.1 Distributed representation

The idea of distributed representations originated from the work of John Rupert Firth who was an English linguist and a leading figure in British linguistics during the 1950s.

Firth studied the context-dependent nature of meaning and described it as "context of situation". He is known for the famous quotation:

> "You shall know a word by the company it keeps" [7]

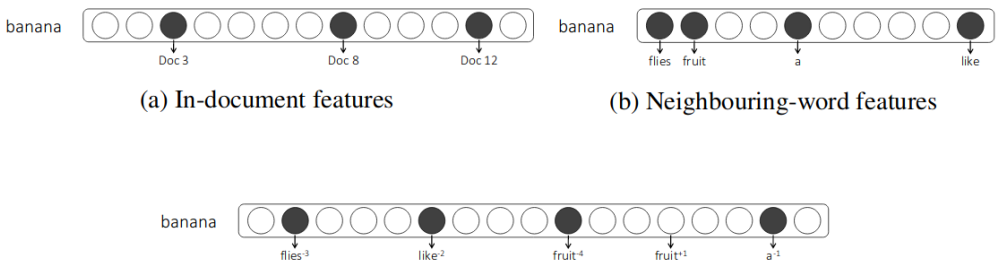
This theory implies that a desirable property is that the representation of a word should be similar to the representations of the words surrounding it.

Both distribution and semantics by themselves are not well-defined and under different context may mean very different things.

In fact, they are very dependent by the distributional properties chosen as features, which in turn influence the notion of similarity.

For instance, let's consider three different features space based on the following properties (respectively): (a) presence or absence of the term in a document; (b) neighbouring words in a window and (c) neighbouring words with distance.

If one considers the sentence "Time flies like an arrow; fruit flies like a banana" then, the distributed representations of the term "banana" are shown in figure 4.2.



(a) In-document features

(b) Neighbouring-word features

(c) Neighbouring-word w/ distance features

Figure 4.2: Examples of different feature-based distributed representations of the term "banana" (picture taken from [3])

In [3], Mitra and Craswell refer to two notions of similarity: **typical** (two words that belong to the same class/type - e.g. capital cities names) and **topical** (two words that are related to the same topic - e.g. a writer and its publications).

They state that representations like (a) yield a notion of topical similarity while representations like (c) yield a notion of typical similarity. (b)-like representations yield a mixture of topical and typical similarity.

In the context of neural models, distributed representations generally refer to learnt embeddings.

Common approaches for learning embeddings include either factorizing one of the matrix presented in the following section (e.g. with LSA) or using gradient descent based methods that try to predict some features given the term (e.g. *Global Vectors* GloVe [8] - GloVe is an unsupervised learning algorithm for obtaining vector representations for words https://nlp.stanford.edu/projects/glove/ - or Word2Vec).

The query and the document embeddings themselves can be compared using a variety of similarity metrics, such as cosine similarity or dot-product.

A study [15] shows that, similarly to the neural embedding space, the explicit vector space also encodes a vast amount of relational similarity which can be recovered in a similar fashion (although it is more expensive in terms of size).

What's interesting about this work is that it implies that the neural embedding process is not discovering novel patterns, but rather is doing a remarkable job at preserving the patterns.

4.3 Models for semantic matching

The terms distributional, corpus-based or statistical all refer to a family of approaches to semantics that share the assumption that the statistical distribution of words in context plays a key role in characterizing their semantic behavior.

> Statistical patterns of human word usage can be used to figure out what people mean.
>
> - **Statistical semantics hypothesis** (SSH)

This general hypothesis underlies several more specific hypotheses, such as the

distributional hypothesis, the extended distributional hypothesis and the latent relation hypothesis, discussed below.

4.3.1 Similarity of documents: the term-document matrix

Given a corpus C, the term-document matrix A represents the relation (based on a feature (function) f) between unique terms and documents in C.

Let m be the number of unique terms and let n be the number of documents in C. Then, A has size $m \cdot n$.

Let the feature considered be the frequency. Then, each row $a_{i,}$ contains the frequencies of a term t_i over the corpus C (*signature of* t_i), and each column $a_{,j}$ contains the frequencies of all m terms over the document d_j (*signature of* d_j).

$$A = \begin{bmatrix} & d_1 & d_2 & \dots & d_n \\ t_1 & f_{11} & f_{12} & \dots & f_{1n} \\ t_2 & f_{21} & f_{22} & \dots & f_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_m & f_{m1} & f_{m2} & \dots & f_{mn} \end{bmatrix} \tag{4.1}$$

If two documents have similar topics, then the two corresponding column vectors will tend to have similar patterns of numbers (SSH).

A popular representation that implements this notion of similarity is called **Bag Of Words** (BOW) representation. It is called *bag* of words because any information about the order or structure of terms in the documents is discarded.

4.3.2 Similarity of words: the term-context matrix

> The distributional hypothesis in linguistics is that words that occur in similar contexts tend to have similar meanings.
>
> - **Distributional semantics hypothesis** [11]

Therefore, according to the DSH, at least certain aspects of the meaning of lexical expressions depend on the distributional properties of such expressions, i.e. on the linguistic contexts in which they are observed.

The term-context matrix B is similar to the term-document matrix, with the distinction that columns are context c of words. Row and column vectors now have both length equal to the size of the vocabulary.

$$B = \begin{bmatrix} & c_1 & c_2 & \dots & c_m \\ t_1 & f_{11} & f_{12} & \dots & f_{1m} \\ t_2 & f_{21} & f_{22} & \dots & f_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_m & f_{m1} & f_{m2} & \dots & f_{mm} \end{bmatrix}$$

The size of the context window determines the count in each position of the matrix as well as its sparseness.

Whit a short context window the representation of words captures a syntactic notion of similarity, viceversa with a long context window the representation of words is likely to capture their semantic meaning.

The output of a distributional model is a distributed representation.

4.3.3 Similarity of relations: the pair-pattern matrix

In a pair-pattern matrix, the columns represent the patterns (e.g. X works with Y) and the rows represent the pairs (e.g. blacksmith:iron).

The **extended distributional hypothesis** states that patterns that co-occurs with similar pairs tends to have similar meaning (e.g. carpenter:wood co-occurs in the pattern X works with Y and X cuts Y. Both patterns could be represented with the relationship artisan:material).

The **latent relation hypothesis** states that pairs that co-occurs with the same patterns have similar semantic relationship. This is the inverse of the previous hypothesis (e.g. carpenter:wood and blacksmith:iron co-occur in X works with Y, so carpenter's relationship with wood is similar to blacksmith's relationship with iron).

To sum up, pair-pattern matrices are suited to measuring the similarity of semantic relations between pairs of words; that is, relational similarity.

In contrast, word-context matrices are suited to measuring attributional similarity.

4.4 Word embeddings

An important consideration here is the choice of the term embeddings that is appropriate for the retrieval scenario.

It is important to understand how the notion of inter-term similarity modelled by a specific vector space may influence its performance on a retrieval task

Models that consider term-document pairs generally capture topical similarities in the learnt vector space.

On the other hand, models like Word2Vec [20] and GloVe [8] capture a mixture of topical and typical notions of relatedness.

Embedding based models often perform poorly when the retrieval is performed over the full document collection; however the errors made by embedding based models and exact matching models may be different—and the combination of the two is often preferred.

For example, an exact matching model may be tricked if some query term are in a non-relevant document and an embedding based model may be tricked if a non relevant document is topically (or typically) close to a query term.

4.4.1 Word2Vec: unsupervised approach to learn term representations (exploiting semantic models)

Mikolov et al. proposed in [20] two main Word2Vec models:

* Continuous Bag of Words (CBOW), that predicts a word given the context;

∗ Skip-Gram, which predicts the context given a word.

Word2Vec is part of the **"Neural Probabilisitic Language Models (NPLMs)"** family, also known as *context-predicting* models.

A NPLM represents each word in the vocabulary as a vector $\vec{x} \in \mathbb{R}^n$ and defines the scoring function s_θ in terms of vectors of the context words and the next word.

In these models, a neural network tries to predict co-occurrence likelihood of context-word pairs. Such probability $P(w_t|w_c)$, given a target word w_t corresponding to a context w_c is computed by the *softmax function*:

$$P(w_t|w_c) = softmax(w_t, w_c) = \frac{exp(s_\theta(w_t, w_c))}{\sum_{w' \in \mathcal{V}} exp(s_\theta(w', w_c))} \tag{4.2}$$

where $\mathcal{V}$ is the vocabulary and s_θ is the scoring function (the neural network), with θ as parameters, that quantifies the compatibility of word w_t with context w_c.

The softmax function is another non-linear activation function which output is obtained by considering information about all set of input elements, not just one (e.g. sigmoid). The output values of softmax are in the range $[0, 1]$ and sum up to 1.

It is often used to turn an arbitrarily large or small numbers into a probability distribution.

4.4.1.1 Word2Vec: CBOW model

CBOW aim to predict a word w_t from the context w_c

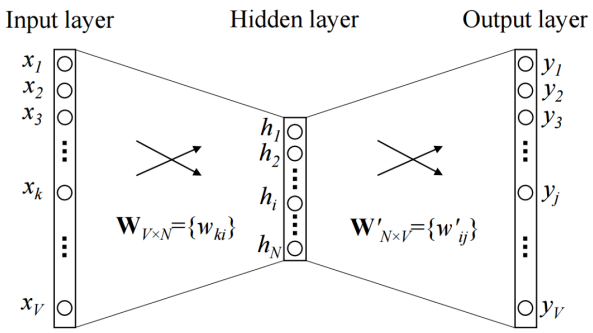


Figure 1: A simple CBOW model with only one word in the context

Figure 4.3: CBOW simple model (context = 1 word) (reference image [25])

Let vocabulary size be $|\mathcal{V}| = V$ and embeddings size be N. The context w_c surrounding the target word w_t is a hot vector $x_1, ..., x_n$.

This vector is compressed in the hidden layer $h = x^T \cdot W$, then it is decompressed to obtain a score $u = W'^t \cdot h$ that is a measure of the match between the context and the target word.

Finally the posterior probability of each word is calculated, i.e. the probability that the output word is equal to the target word (softmax function is applied to u):

$$p(w_t|w_c) = y_t = softmax(u_t) = \frac{e^{u_t}}{\sum_i^V e^{u_i}} \tag{4.3}$$

The aim of this model is to maximize such probability (the loss function needs to evaluate the output vector u at the ideal index t):

$$\begin{aligned} \max p(w_t|w_c) &= \max y_t \\ &= \max log(y_t) \\ &= \max log(softmax(u_t)) \\ &= \max log\left(\frac{e^{u_t}}{\sum_i^V e^{u_i}}\right) \\ &= -u_t + \sum_i^V e^{u_i} \end{aligned} \tag{4.4}$$

So, the loss function is:

$$\mathcal{L} = -u_t + \sum_i^V e^{u_i} \tag{4.5}$$

aggiunto negative sampling

The softmax function requires that the probability e^{u_t} is normalized by summing over all the vocabulary, which is an expensive computation to do in case of large corpus. Thus, to make the model scalable, the authors proposed a sightly altered **negative sampling** objective:

$$\mathcal{L} = -u_t + \sum_i^N e^{u_i} \tag{4.6}$$

where N is the number of negative sample words drawn either from the uniform or empirical distribution over the vocabulary.

This model can be generalized with multiple context words; in fact the general form is used for the creation of word embeddings in later experiments of this thesis.

4.4.1.2 Word2Vec: Skip-Gram

Skip-Gram is the opposite of CBOW: in Skip-Gram the context w_c is predicted given an input word w_t.

Basically the training objective of the Skip-Gram model is to learn word vector representations that are good at predicting nearby words in the associated context.

At the output layer, instead of outputting one multinomial distribution, the output consists of $|w_c|$ multinomial distributions. Each output is computed using the same hidden layer.

The output of the model is the probability that the prediction of the j-th word on the c-th panel, $w_{c,j}$, equals the actual word, conditioned on the input word w_t.

$$p(w_{c,j}|w_t) = \frac{exp(u_{c,j})}{\sum_{j'=1}^V exp(u_{c,j'})} \tag{4.7}$$

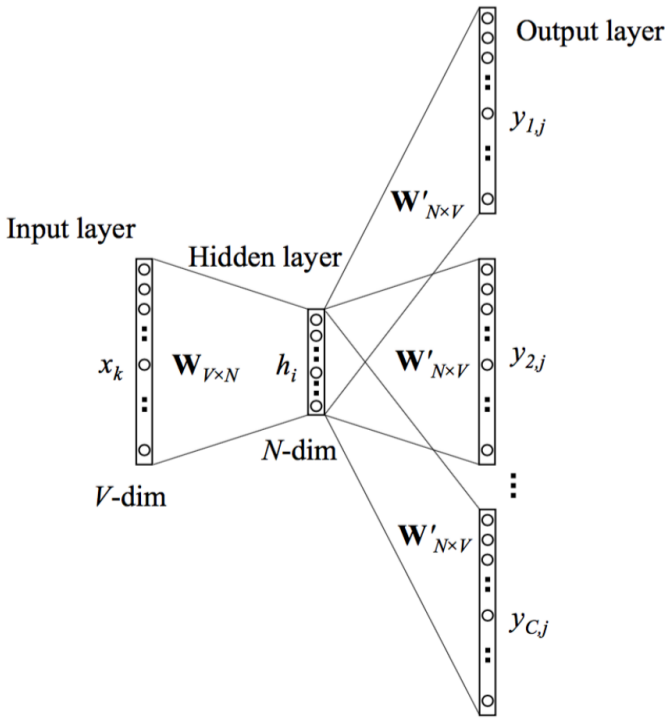


Figure 4.4: Skip-gram model (reference image [25])

Word2Vec model's output consists of two different weight matrices: W (IN-embeddings) and W' (OUT-embeddings). Generally, only W is used and W' is discarded after training.

aggiunto

4.4.2 On the importance of word embeddings

The usage of word embeddings comes with its own challenges and decision problems.

aggiunta sezione

For instance, some decisions to be considered are the choice of the possible algorithms: Word2Vec (cbow or skip-gram) [20] or GloVe the choice of hyper-parameters (e.g., dimensionality of embedding space, window size, etc.) and the choice of training data/corpora.

Does performance vary much if pre trained embeddings are used vs. re-training embeddings for a target domain, either by fine-tuning pre trained embeddings or re-training from scratch? How should one deal with out-of-vocabulary (OOV) query terms not found in the word embedding training data for query-document matching?

Some studies gave empirical answers to these questions. In [36] it is reported a number of studies that found no significant difference between Word2Vec or GloVe, skip-gram of cbow.

In the same review study, there are discording results on the impact of training corpus for word embeddings: some found that when the background training corpus matched the target corpus from a topical point of view, the performance improved; other found that the choice of corpus used to construct word embeddings had little effect on retrieval results.

Regarding the handling of OOV terms, some solutions proposed were to ignore

the OOV terms or to randomly initialize their embeddings.

Mitra et al. in [21] explored the possibility to combine both IN-embeddings and OUT-embeddings of Word2Vec (usually discharged after training) in order to explore different topic-based relationship between a query and its relevant documents.

For instance, table 4.1 shows that IN-IN and OUT-OUT cosine similarities are high for words that are similar by function or type (typical), and the IN-OUT cosine similarities are high between words that often co-occur in the same query or document (topical).

yale			**eminem**		
IN-IN	OUT-OUT	IN-OUT	IN-IN	OUT-OUT	IN-OUT
yale	yale	yale	eminem	eminem	eminem
harvard	uconn	faculty	rihanna	rihanna	rap
nyu	harvard	alumni	ludacris	dre	featuring
cornell	tulane	orientation	kanye	kanye	tracklist

Table 4.1: Tables with data reported by Mitra et al. in [21]. They shows nearest neighbours for the words "yale" and "eminem" according to the cosine similarity based on the IN-IN, OUT-OUT and IN-OUT embeddings. Their Word2Vec model was trained on a query corpus with a vocabulary of 2748230 words

They proposed a "Dual Embedding Space Model", with one embedding for query words and a separate embedding for document words, learned jointly based on an unlabelled text corpus.

Recently, Zamani and Croft in [34], point out that embedding vectors used in Neural IR are typically learned based on term proximity in a large corpus. In Word2Vec, for instance, the objective is to predict adjacent word/s for a given word or context. They argue that such objectives (based on term proximity, syntactic, or even semantic similarity) is not is not necessarily equivalent to capture relevance.

The following example shows this fact: let "dangerous vehicles" be a query submitted by a user. One of the most similar terms to this query based on the typical word embedding algorithms (e.g., Word2Vec and GloVe) is "safe", and thus it would get a high weight in the expanded query model. The reason is that the words "dangerous" and "safe" often share similar contexts. However, expanding the query with the word "safe" could lead to poor retrieval performance, since it changes the meaning and the intent of the query. This can be seen in a later example (7.3), where "crime" is similar to "justice". For the same reason, Word2Vec fails on the notion of antonymy: words with opposite meanings are actually very likely to occur in similar contexts,

In [34], Zamani and Croft developed two learning models with different objective functions; one that learns a relevance distribution over the vocabulary set for each query and the other that classifies each word as belonging to the relevant or non-relevant class for each query.

They used a technique similar to PRF to train their models. Contrary to PRF, they used an offline approach (without the need of an extra retrieval run) in which they considered a fixed-length dense vector for each vocabulary term and estimate

these vector based on the information extracted from the top retrieved documents for large numbers of training queries.

They reported that the relevance-based word embedding models outperform state-of-the-art word embedding algorithms, and showed that the expansion terms chosen by their models are related to the whole query, while those chosen by typical word embedding models are related to individual query term.

CHAPTER 5

Deep relevance matching model

The main paper studied in this work is [10], a deep relevance matching model (DRMM) to solve ad-hoc retrieval task developed by Jiafeng Guo et al.

Their approach is a mix of an interaction-focused model and a representation-focused one.

5.1 Architecture

The DRMM architecture (see figure 5.1) takes as input a representation of the local interactions between each query-document term pair (called matching histograms).

These histograms inputs are then processed by a feed forward neural network and aggregated with a term gating network, which in turn contributes to weigh each query term with its IDF or its embedding.

Finally, the score aggregation (the dot product in this implementation) produces the matching score of the initial pair (query, document) which is the value that will be used for ranking.

Each component serves to achieve one of the following three goals:

* matching histogram mapping is useful to obtain a signature of both lexical and semantic signals between a query and a document;
* term gating network takes into account query term importance by weighing them;
* forward matching network handles diverse matching requirements (i.e. global and local matching).

In the next paragraphs each component will be described more in detail.

5.1.1 Matching Histogram Mapping

The inputs of DRMM are the local interactions between each pair of terms from a query q and a document d: $w^{(q)} \otimes w^{(d)} \qquad \forall w^{(q)} \in q, w^{(d)} \in d$ - where $\otimes$ is the matching function (e.g. cosine similarity).

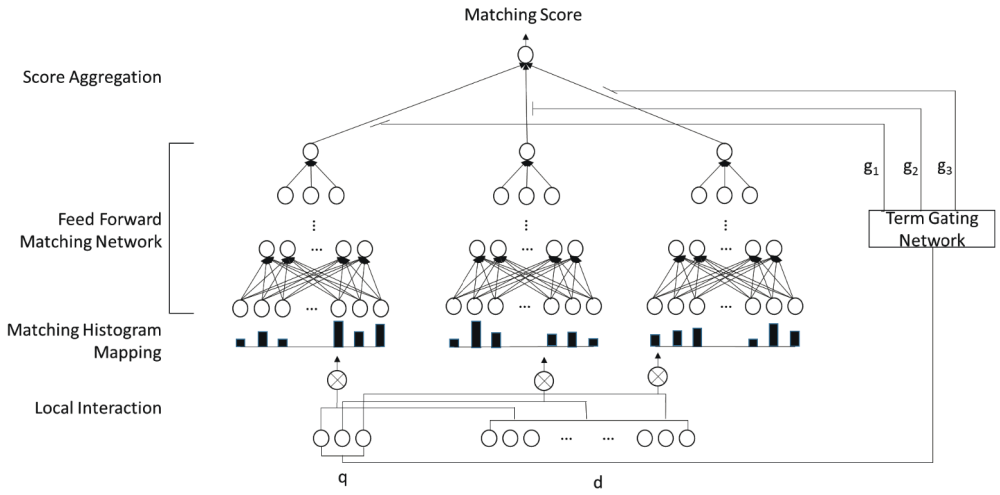


Figure 5.1: DRMM architecture (reference: [10])

The local interactions represent the matching signals between a (query, document) pair. It is stored into a structure denominated *matching histogram*.

An example: let the similarity function be the cosine similarity (so the resulting matching values are in the range [-1, 1]) and let the bin size be 0.5.

Thus, the bins of the histograms are five: $\{[-1, -0.5), [-0.5, 0), [0, 0.5), [0.5, 1), [1, 1]\}$.

Consider the query term "car" and the following document sample: ("car", "rent", "truck", "bump", "injunction", "runway").

Assume the corresponding cosine similarities: $(1, 0.2, 0.7, 0.3, 0.1)$, then there are three matches in $[0, 0.5)$, one match in $[0.5, 1)$ and one exact match in the last bin.

Thus, the resulting matching histogram is $[0, 0, 3, 1, 1]$.

This approach is a strength preserving representation that groups local interactions according to different levels of signal strengths rather than their positions.

The main disadvantage of this approach is that all information of word positions in documents is lost.

Guo et al. experimented with three different matching histogram mappings:

* **Count-based Histogram (CH)**: the count of local interactions in each bin is the histogram value, computed as in the previous example (standard version);
* **Normalized Histogram (NH)**: the count value in each bin is normalized by the total count;
* **LogCount-based Histogram (LCH)**: logarithm is applied over the count value in each bin.

5.1.2 Semantic matching vs relevance matching

Matching histogram mapping address both semantic and lexical matching.

Traditional IR models estimate relevance using **lexical matches** of query terms in document. However, this is not the only choice: representation based models gain evidence of relevance from all document terms based on **semantic matches** with a query.

Both lexical and semantic matching are important and can be modelled with neural networks.

Guo et al. in [10] claim that exact matching of terms is still the most important signal in ad-hoc retrieval, although others (e.g. Huang et al. in [13]) consider semantic matching more accurate than lexical matching.

The general intuition is that a good IR model should consider both matches in term space (exact/lexical matches) and semantic matching (matching in the semantic space).

If a term is rare it should be easy to estimate the relevance of a document by individuate patterns of exact matches. On the other hand, if a term is frequent, semantic matching may be more suitable.

5.2 Feed forward Matching Network

Suppose that a query and a document are represented as a set of term vectors denoted by $q = (w_1^{(q)}, w_2^{(q)}, \ldots, w_M^{(q)})$ and $d = (w_1^{(d)}, w_2^{(d)}, \ldots, w_N^{(d)})$.

Let h be the mapping function from local interactions to matching histograms (it can be CH, NH or LCH), $\otimes$ the cosine similarity function and $z_i^{(l)}$ the input of the l layer for the i-th query term. The input of the DRMM is denoted by $z_i^{(0)}$:

$$z_i^{(0)} = h(w_i^{(q)} \otimes d) \qquad \forall i = 1 \ldots M \qquad \text{(Input to the first layer of ffnn)}$$

The input to the l-th layer of the feed forward network is:

$$z_i^{(l)} = tanh(W^{(l)} \cdot z_i^{(l-1)} + b^{(l)}) \qquad \text{(Input to the l-th layer of ffmn)}$$

where $W^{(l)}$ denotes the weights matrix and $b^{(l)}$ denotes the bias term of the l-th layer.

There are three hidden layers of this type for the histogram input and only one layer for the query input, respectively.

Each of them uses the *tanh* as activation function.

5.3 Term gating network

One difference between DRMM and existing interaction-focused models is that it employs a joint deep architecture at the query term level.

In this way, the model can explicitly model query term importance.

This is achieved by using a term gating network (TGN), which produces an aggregation weight for each query term controlling how much the relevance score on that query term contributes to the final relevance value.

The softmax function is employed as the gating function:

$$g_i = \frac{exp(w_g \cdot x_i^{(q)})}{\sum_{j=1}^{M} exp(w_g \cdot x_j^{(q)})} \qquad \forall i = 1 \dots M \qquad \text{(Aggregation weight for TGN)}$$

where w_g denotes the weight vector of the term gating network and $x_i^{(q)}, \quad \forall i = 1 \dots M$ denotes the i-th query term input.

The final relevance score for a pair (query, document) is given by the dot product between the output of the term gating network and the feed forward network:

$$s = \sum_{i=1}^{M} g_i \cdot z_i^{(L)} \qquad \text{(Relevance score for a pair (q, d))}$$

There are two variants of the term gating network, depending on its input:

* **TV** - the input consists of a term vector. In this case, $x_i^{(q)}$ (in Aggregation weight for TGN) corresponds to the i-th query term vector embedding and w_g is a weight vector with $|w_g| = |x_i^{(q)}|$.
* **IDF** - the input consists of the inverse document frequency of the i-th query term and w_g reduces to a single parameter.

5.4 Model training

Since the ad-hoc retrieval task is a ranking problem, Guo et al. [10] employ the **pairwise hinge loss function** to train their deep relevance matching model.

Given a triple (q, d^+, d^-) where document d^+ is ranked higher above document d^- with respect to query q, the loss function is defined as follows:

$$\mathcal{L}(q, d^+, d^-; \Theta) = max(0, 1 - s(q, d^+) + s(q, d^-)) \qquad \text{(Hinge loss function)}$$

where $s(q, d)$ denotes the predicted score for (q, d) and Θ denotes the parameters of the whole model. The equality holds iff $label(q, d) = \pm 1$.

This function aims to assign an higher score to relevant documents w.r.t. non-relevant documents so that the ranking list produced will have high precision at lower cutoffs.

The training is performed using mini-batches stochastic gradient descent.

Guo et al. [10] applied stochastic gradient descent algorithm *Adagrad* with mini-batches (20 in size) but they don't report the initial learning rate (which has probably been determined empirically) nor the number of epochs.

Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data.

For regularization, Guo et al. [10] found that the early stopping strategy worked well for the model.

5.5 Experimental setup

To conduct the experiments, the authors used the following settings:

Data and platform	Name
Text collections	Robust04 [31]
Search engine	Galago [1]
Tokenization	White-space tokenization
Stemmer	Krovetz stemmer
Stopword list	INQUERY stop list [2]
Term Embeddings	300-dimensional term vectors trained with Word2Vec [20] with the CBOW model (context window size = 10; negative sample = 10; subsampling of frequent words with sampling threshold of 10^{-4}). Each corpus was pre-processed by removing HTML tags and stemming, all the terms that occur less than 10 times in the corpus were removed
Network configuration	4-layer architecture: one histogram input layer (30 nodes), 2 hidden layers in the feed forward matching network (5 and 1 nodes respectively) and one output layer (1 node)

Table 5.1: Data and platform of the experiment

The authors performed 5-fold cross-validation to minimize overfitting on the small experimental collection: topics for each collection were randomly divided into five folds; the parameters for each model were tuned on four of five folds and the overall process was repeated five times.

Each evaluation statistic reported by the authors was the average of the five fold-level evaluation values.

The metric to be optimized during training process was MAP.

A **re-ranking strategy** for efficient computation was adopted: an initial retrieval was performed using Dirichelet Language Model and Bm25 to obtain the top two thousand ranked documents for each topic.

Finally, the top-ranked one thousand documents were compared w.r.t. MAP, normalized discounted cumulative gain at rank twenty (nDCG@20), and precision at rank twenty (P@20).

The aim is to bring most (hopefully all) relevant documents in the pre-ranked results as high as possible and keep non relevant documents towards the bottom of the run.

The network should be able to generalize to histograms (query/document pair) never seen before (i.e., documents not in the training set).

[1] https://sourceforge.net/p/lemur/wiki/Galago

[2] https://github.com/igorbrigadir/stopwords/blob/master/en/indri.txt

5.6 Observations on the model

The quality of the relevance judgments is very important: since the moment that documents not presented in the ground truth are considered to be non relevant, if the relevance judgments are incomplete the network may fail to generalize.

Training and validation labels needs to be chosen in a way that prevents overfitting and underfitting, but in the paper there is no mention on how many relevant/non relevant documents should be considered for each topic during a training epoch.

Another question that needs to be addressed is the case of equal input to the model (matching histograms) generated from different pairs (query, document).

To answer this question it is necessary to study the behavior of the function that maps a query/document pair to a histogram.

Such function (h - see equation Input to the first layer of ffnn) needs to be injective in order to generate always different histograms representations: if two different pairs (query, document) have the same matching signals distribution over bins for each query term then they will have the same histogram too.

It turns out that such function is *not* injective, because it only considers matching signals strength.

A trivial case to show this fact is that in which two documents are permutation of each other (since the moment that matching histograms representation does not preserve information about position of words): when they are matched against the same query they produce the same histograms.

CHAPTER 6

Reproducibility

Science advances on a foundation of trusted discoveries. "Trust" depends on the reproducibility of an experiment, which leads scientists to gain confidence in their conclusions.

The American Physical Society (APS), the world's second largest organization of physicist, emphasizes reproducibility in the following definition of science [1]: "(...) the success and credibility of science are anchored in the willingness of scientists to expose their ideas and results to independent testing and replication by others. This requires the open exchange of data, procedures and materials.".

So, reproducibility is key for reliable, referenceable and extensible research. Experimental papers are therefore most useful when their results can be tested and generalized by people not involved in the original work.

In later years, there is growing alarm about results that cannot be reproduced. The causing factors may vary, including increased levels of scrutiny, complexity of experiments and statistics, fraudulent results, pressures on researchers, poor experimental design, execution and analysis.

6.1 Reproducibility in computer science

Reproducibility assumes differents meanings in different disciplines.

In computer science, reproducibility often refers to the ability to reproduce computations alone (it relates exclusively to sharing documented data and code), while replication describes the redoing of whole experiments.

In this sense, the experiment studied in this thesis has been both reproduced and replicated.

Reproducing (complex) computational research poses some challenges ([28]). The problem arises when, for example, in a traditional article, the author simply outlines the relevant computations without complete documentation, which would ideally include experimental data, parameter values, and the author's programs. If someone wants to use and verify the work must reimplement it, which is often a painful process.

[1] https://www.aps.org/policy/statements/99_6.cfm

Even when the author's source files are available, they can be undocumented or difficult to understand, and can only recompute results by invoking the various programs exactly as the author invoked them.

A good practice of reproducibility is necessary in order to allow previously developed methodology to be effectively applied on new data, to allow reuse of code and results for new projects and, more generally, to save time.

6.1.1 Reproducibility in IR

Reproducibility of empirical results is a fundamental requirement of disciplines such as IR, a primarily empirical discipline where advances are built on experimental validation of proposed methods.

Experimentation in systems-oriented IR research typically involves two major aspects: an IR system and one or more test collections.

Reproducibility is partly ensured by the widespread use of standardised test collections, experimental protocols and evaluation measures, such as those provided/defined by TREC, although with a neural approach they are no longer sufficient.

IR community has started only recently to consider the topic of reproducibility as part of the review process of major conferences. Starting from 2015, ECIR announced a Reproducibility Track [2] and SIGIR started the Open-Source IR Reproducibility Challenge [3].

6.2 PRIMAD

PRIMAD is a theoretical model general enough for computer science (and related fields) ([6]) and acts as a framework to distinguish the major components of an experiment:

* **R: Research goal** - what is the purpose of the study?
* **M: Method** - which approach was considered by the researcher?
* **I: Implementation** - how was the method implemented (which programming language was used)?
* **P: Platform** - which software and hardware resources were used?
* **D: Data** - what data were used as input? And how were set the parameters (if any exist) of the method?
* **A: Actor** - who was the researcher?

Possible changes (often some of them happens simultaneously):

* $R \rightarrow R'$: one or more components of the experiment are *re-purposed* to answer another research question.

[2] http://ecir2019.org/reproducibility_track/
[3] https://sigir.org/sigir2019/program/workshops/osirrc

* $M \rightarrow M'$: an alternative method is chosen to conduct the experiment (therefore the implementation change accordingly).
* $I \rightarrow I'$: the researcher uses another implementation or makes its own.
* $P \rightarrow P'$: the platform on which the experiment was run changes (there may be subtle effects on the outcomes of the experiment).
* $D \rightarrow D'$: the input data for the experiment may change in order to test the generality of the method. The parameters of the method may change as well (robustness).
* $A \rightarrow A'$: the experiment is reproduce by another actor.

Reproducibilty increases as the number of PRIMAD's **shared** components increases. There are two elements that are not considered in the PRIMAD model:

* **Transparency**: it's the ability to look into all necessary components to verify that the experiment does what it claims;
* **Consistency**: two experiments conduct with the same criteria must lead to equal outcomes.

6.2.1 PRIMAD for System-oriented Evaluation

A system-oriented evaluation is concerned with the ability of a retrieval system to find answers in a test collection.

Evaluation in IR has already been discussed in the background chapter. It can be noticed that Cranfield paradigm enhance reproducibility since the moment that the elements in the triple $(\mathcal{D}, \mathcal{T}, \mathcal{GT})$ are fixed.

Runs with a given system on the same data triples should produce identical results, and more generally if the retrieval methodology is sufficiently well described then a fresh implementation should be identical.

Here, I present an instance of the PRIMAD model applied to system-oriented evaluation.

* Research goal: produce a high-quality ranking of answers, on average across a set of queries.
* Method: mapping from the query to an ordering of the documents (using a similarity function).
* Implementation and platform: the retrieval system used and the environment in which the study was carried out.
* Actor: the experimenter.
* Data: test collection used in the experiment.

6.2.2 Obstacles to reproducibility

Even though a shared collection provides sufficient basis for reproducibile research, there are still some possible obstacles to discuss.

Retrieval systems are huge complex pieces of software that often depend on other resources such as dictionaries, stemmers, stopwords, statistical and machine learning tools. So, it is not always possible to encapsulate the complete environment of the experiment, and behaviour of the system will change as the environment changes.

Moreover there are others obstacles regarding the availability of data collections: privacy or limitations of anonymization, confidentiality, volatility of data, size of data (and the lack of adequate computing resources).

6.3 Instance of PRIMAD for this study

Instance for DRMM:

* **R** - Study a deep relevance matching model (DRMM) for ad-hoc retrieval
* **M** - Data preprocessing (as illustrated in the first chapter, 2.2), building the model DRMM, training, testing and evaluation (using a standard TREC collection)
* **I** - Two implementations were found [4]
* **P** - Galago for initial retrieval, the authors made no additional assumptions
* **D** - Robust04 TREC collection, initial retrieval model parameters, preprocessing parameters, DRMM parameters
* **A** - Jiafeng Guo, Yixing Fan, Qingyao Ai, W. Bruce Croft

Changes applied:

* $R \rightarrow R'$: Same research goal
* $M \rightarrow M'$: Same method
* $I \rightarrow I'$: I reproduce the experiment firstly with one implementation already given and then I made my own implementation in Python
* $P \rightarrow P'$: I used Terrier for the initial retrieval, builded the model with Tensorflow, (Evaluation) Trec eval
* $D \rightarrow D'$: Same input data
* $A \rightarrow A'$: Emanuele Carraro

[4] https://github.com/NTMC-Community/MatchZooMatchZoo [5] and https://github.com/faneshion/DRMM

Because most retrieval engines are relatively complex, multi-component, highly configurable systems, precisely reproducing a set of experimental results can be challenging in the absence of a detailed description of the retrieval engine used and its settings.

Such a description should preferably cover at least the following components of the IR system:

* tokenisation/parsing method used: which parts of a text are tokenised, what characters are regarded as token delimiters, the nature of the tokens themselves (e.g., words, n-grams), which tokens are discarded, if any (e.g., strings consisting of numerals only), and so on.
* Stopword list used, if any.
* Stemming algorithm used, if any.
* Additional indexing units (e.g., phrases and named entities) identified, if any, and details of the identification method used.
* Details of retrieval model used (e.g. in language modeling the smoothing method used and values of parameters).
* Information about other techniques (e.g., reranking, query expansion) that are used on top of a basic, keyword-matching approach to ranked retrieval.

In practice, however, many of the above details are often missing from the system descriptions provided in scholarly articles.

In most cases, authors mention only the retrieval model and the engine that was used (particularly when it is an open-source engine such as Indri, Lucene, or Terrier) along with some additional information, such as the stopword list used, and the stemming method employed.

Missing details lead to potential reproducibility issues, since the various components of an IR system may, in general, have a significant effect on the overall performance of the system.

6.4 How to improve reproducibility

To ensure reproducibility, one should provide the following items: research question and variables used in the experiment, experimental design, participant characteristics, experimental protocol, environmental conditions, data collections, retrieval systems, baseline results, methods and assumptions for data analysis, degree of control on the system.

6.4.1 Ten Simple Rules for Reproducible Computational Research

The following "rules" define a set of helpful information to be put on a project documentation for reproducibility purposes ([27]).

1. The process (sequence of steps) that originate a result should be documented (e.g. shell scripts, makefiles).

2. The execution of programs should be preferred over manual procedures to modify data. Such manual procedures are not only inefficient and error-prone, they are also difficult to reproduce, and might not be documented.

3. The programs exact versions used originally should be documented.

4. When a continually developed piece of code (typically a small script) has been used to generate a certain result, it is necessary to keep track of all of its versions (e.g. using a version control system, such as Subversion, Git, or Mercurial).

5. Having easily accessible intermediate results may be of great value.

6. Many analyses and predictions include some elements of randomness, thus, it should be used an initial seed for all such processes and report it in documentation.

7. When plotting, it can be useful to store both the underlying data and the processed values that are directly visualized.

8. When working with summarized results, the underlying data should be generated and validated at least once.

9. Connect a given textual statement (interpretation, claim, conclusion) to the precise results underlying the statement, already by the first time they are formulated.

10. All input data, scripts, versions, parameters, and intermediate results should be made publicly and easily accessible.

6.5 Reproducibility in Neural IR

When IR employs neural networks to develop a retrieval model, the reproducibility task gets even more complex.

Neural IR is quite a new topic of interest that started to appear both in SIGIR and ECIR from 2016 and only in later years papers from these conferences have been taken as subject of replicability and reproducibility (see reports from SIGIR workshops on NeuIR [5].

Some papers that tackle the issue of reproducibility in neural IR are [17] and [33], in which the authors claim that some works compare themselves against "weak" baselines - e.g. bag-of-words methods such as Bm25 and QL often with parameters poorly tuned, in order to appear higher in IR leaderboards.

[5] https://staff.fnwi.uva.nl/m.derijke/wp-content/papercite-data/pdf/craswell-report-2016.pdf and http://sigir.org/wp-content/uploads/2018/01/p152.pdf

These works show relative improvements, which are not additive. In some cases they can lead to the false belief that newest neural approaches are actually more efficient than IR baselines and as a consequence, a stagnation of results.

[33] proposes an open-source IR toolkit built on Lucene [6] that address the problem of reproducible baselines. [17] instead propose various solutions, among which the adoption of a execution-centric leaderboard and a cultural shift. Other works like [23] and [26] instead focuses on replicating and reproducing (and extending - e.g. with regularization techniques) a neural IR model that usually show some significant improvement over traditional IR methods.

Some major issues in reproducing a neural IR model are the usage of off-the-shelf embeddings, maybe built with a private text collection, the insufficient documentation of the hyper-parameter choices and the introduction of randomness through different sources.

Some randomness sources are the following:

* shuffling of dataset before training, if any;
* k-fold cross validation, or more specifically, how to choose the folds;
* random initialization of layer weights - neural networks often initialized the weights of their layer with values sampled from a particular distribution;
* noisy hidden layers (e.g. dropout layers set to zero the output of random nodes);
* changes in ML frameworks: e.g. switching Keras backend from Thean to Tensorflow.

Typically, to ensure reproducibility, it's a good practice to run the network many times and use statistics to summarize the performance of the model and use them for comparison. Unfortunately this may require too much time so another best practice is to use a fixed seed whenever there is randomness introduced.

[6] https://github.com/castorini/anserini

CHAPTER 7

An implementation of DRMM

The implementation and evaluation of this Neural IR system were divided in 4 macro tasks:

* preprocessing and analysis of dataset (Robust04 [31]);
* implementation of the model;
* evaluation of results;
* discussion.

The rest of this chapter describes and analyzes each task in detail.

7.1 Preliminar Analysis

Before (and while) implementing DRMM I considered different options available regarding all the four previous steps.

Collection preprocessing:

* the corpus and queries can be stemmed/not stemmed, processed with stopwords/without stopwords, and different fields (or combination of them) from topics can be considered (i.e. title and description);
* the preranked results can be obtained with a wide range of retrieval models. For instance, the authors of the original paper use Bm25 and QL models as first-stage rankers.

DRMM system:

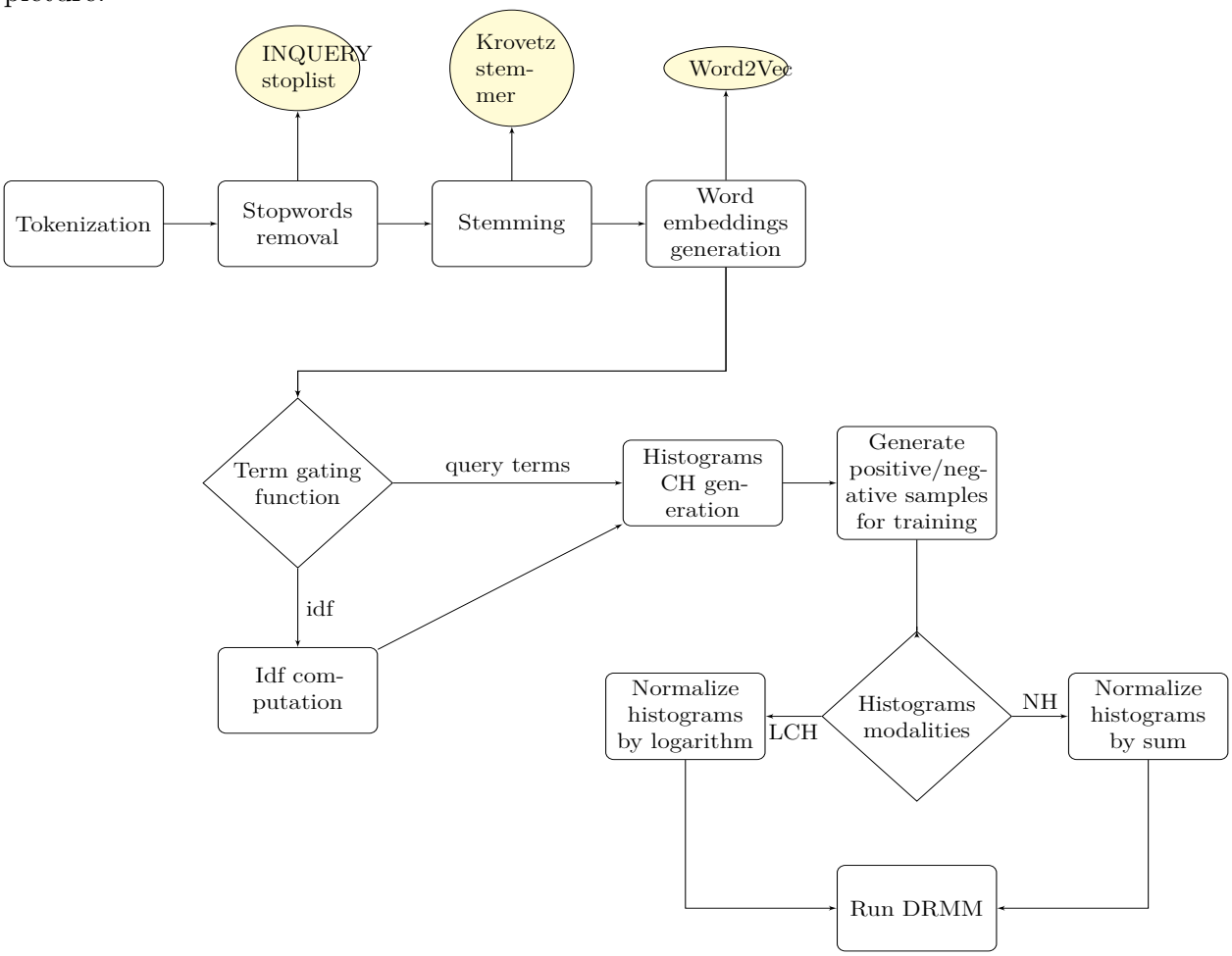
* the size of the word embeddings may vary;
* the query term gating can take as input the query embeddings or the query terms IDF;

* the hyperparameters of the model need to be tuned (i.e. number of epochs, learning rate, early stopping, batch size).

After considering the above options, I chose the following configuration for my tests: both corpus text / query titles stemmed and without stopwords; the top 2000 preranked results with both QL and Bm25 retrieval algorithms, 300 embeddings size, and training/testing on all histograms mode/term gating network.

Despite a few hyperparameters of the model were given in [10], I had to (empirically) try all the others (e.g. number of epochs, number of documents to sample from the training set, initial learning rate and early stopping values).

The workflow that I followed for my implementation is reported in the following picture:



7.1.1 Experimental collection

The goal of the TREC Robust track ([31]) is to improve the consistency of retrieval technology by focusing on the most difficult topics.

The document collection for the Robust track is the set of documents from both TREC Disks 4 and 5 minus the the Congressional Record on disk 4.

Trec format

There are 2 formats, depending on the type of document (pure text or HTML).

<DOC>
<DOCNO> document_number **</DOCNO>**
Here there could be tags like Title, Author, Profile, Headline, Date, Page...
<TEXT> (plain text or HTML format)
Document text
</TEXT>
</DOC>

Topics

TREC calls a natural language statement representing an information need a "topic" to distinguish it from a "query", which is the portion of text actually presented to the retrieval system.

The topics are formatted using a very simple SGML (Standard Generalized Markup Language) style tagging (different than XML format).

An example taken from Robust04:

<top>
<num> Number: 301
<title> International Organized Crime
<desc> Description: Identify organizations that participate in international criminal activity, the activity, and, if possible, collaborating organizations and the countries involved.
<narr> Narrative: A relevant document must as a minimum identify the organization and the type of illegal activity (e.g., Columbian cartel exporting cocaine).
Vague references to international drug trade without identification of the organization(s) involved would not be relevant.
</top>

Relevance Judgements

The format of a relevance judgment (qrels) file is the following:

* TOPIC: topic number;
* ITERATION: feedback iteration (almost always zero and not used);
* DOCUMENT#: official document number that corresponds to the docno field in the documents;
* RELEVANCY: binary code of 0 for not relevant docs and 1 for relevant docs.

Qrel row example: "301 0 FBIS3-10082 1"

Documents not occurring in the relevance judgements file were not judged by the human assessor and are **assumed to be non relevant** in the evaluations used in TREC.

In a qrels file the human assessors are told to judge a document relevant if any piece of the document is relevant (regardless of how small the piece is in relation to the rest of the document). Thus, judgements have been given under the **scope hypothesis**.

7.2 Ad-Hoc Information Retrieval on TREC Robust04

DRMM is not the first (Neural) IR model to evaluate its performance on TREC Robust 2004 track ([31]). In fact, during past years, others retrieval models have been tested against this collection.

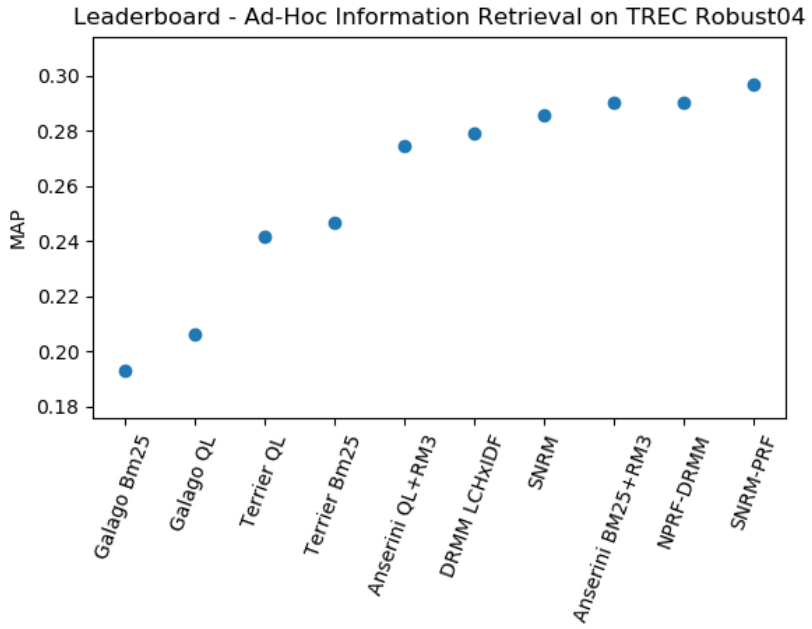


Figure 7.1: Ad-Hoc Information Retrieval on TREC Robust04

The above picture reports the state-of-the-art leaderboard of ad-hoc retrieval on the TREC Robust04 collection [1]. QL and Bm25 are two traditional IR models and RM3 is a relevance-based model, that provide a solid baseline while all the others are recent Neural IR models (described in 4.1.1).

[1] https://paperswithcode.com/sota/ad-hoc-information-retrieval-on-trec-robust04

Method	MAP	P@20	nDCG@20	Year
Anserini BM25+RM3	0.290	*Not available*	*Not available*	2018
Anserini QL+RM3	0.274	*Not available*	*Not available*	2018
SNRM	0.285	0.376	0.431	2018
SNRM-PRF	0.297	0.394	0.439	2018
NPRF-DRMM	0.290	0.406	0.45	2018
DRMM	0.279	0.382	0.431	2017

Table 7.1: State of the art of ad-hoc retrieval on TREC Robust04

7.3 Dataset analysis

Preprocessing steps:

* Parsing of collection and topics;
* Indexing of parsed collection and queries with Terrier Dirichlet QL and Bm25 algorithms (to obtain preranked data);
* Stemming and stopwords removal;
* Word-embeddings preparation both for collection and queries with Word2Vec (although queries-based embeddings were ignored) - this is where originates the out-of-vocabulary problem;
* Pre-computed IDF values for each query term (input to query term gating);
* Subdivision of ground truth labels (based on topics) for k-fold cross validation.

7.3.1 Parsing of documents and topics

For documents, the following attributes were considered: docno, headline and text, whereas for topics only the title attribute was considered.

Both documents and queries were lower-cased, punctuation and any non-alphabetic characters (including numbers and words that contain numbers) were removed along with every tag and HTML entities such as &lg, &gt etc.

Example of sentence cleaning:

"BOOK REVIEW; A SURVIVOR'S ACCOUNT OF BRAIN SURGERY A Bomb in the Brain: A Heroic Tale of Science, Surgery and Survival" becomes "book review a survivors account of brain surgery a bomb in the brain a heroic tale of science surgery and survival".

7.3.2 Indexing of parsed collection and queries

Indexing was performed both with Galago and Terrier search engines using the krovetz stemmer and the INQUERY stoplist. The original paper performed stopwords removal only on queries, but I performed it also on the entire text collection, in order to remove noisy words.

7.3.3 Data preliminar analysis

The following tables and plots report statistics on unique (vocabulary) words per documents and per queries.

	Stemmed	Stopwords removal	Min	Max	Mean	Median	Mode	Std
Doc. vocabulary length			1	13779	211.05	170	89	168.81
Queries vocabulary length			1	5	2.7	3	3	0.704
Doc. vocabulary length		x	1	13725	158.44	122	56	141.19
Queries vocabulary length	x	x	1	4	2.62	3	3	0.65
Doc. vocabulary length	x	x	1	13409	148.42	116	54	127.23
Queries vocabulary length	x	x	1	4	2.62	3	3	0.65

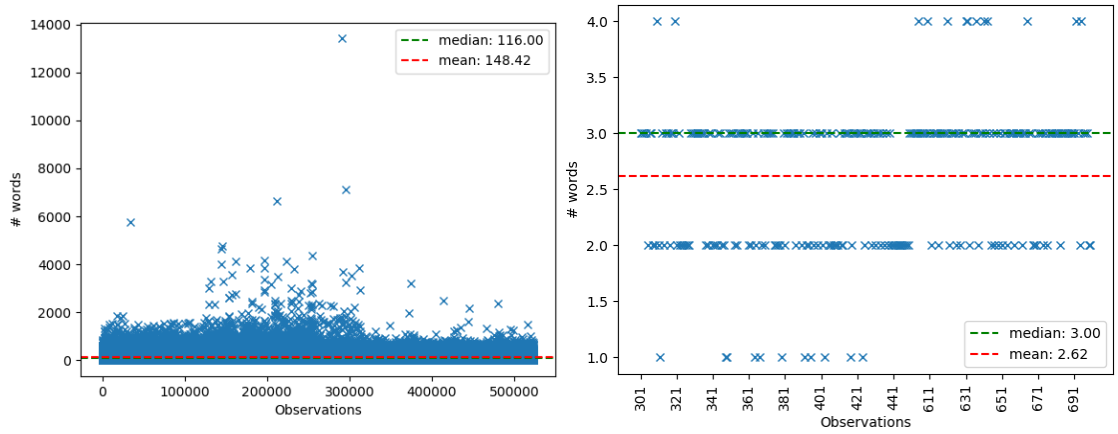
Table 7.2: Data statistics

As it can be understand from the statistics, the length of documents vocabulary is not uniform throughout the corpus. Most of them are, in fact, far from the average-length. Queries, instead, have a vocabulary length between one and four words.

These statistics shows that the stopwords and the stemming components are reducing the length of both documents and queries vocabularies.

	Stemmed	Stopwords removal	Count
Docs words			110563253
Queries words			3624
Docs words		x	83003929
Queries words		x	2131
Docs words	x	x	77750991
Queries words	x	x	2063

Table 7.3: Data words count



(a) Length of vocabularies per document (corpus without stopwords and stemmed)

(b) Length of vocabularies per queries (without stopwords and stemmed

Figure 7.2: Histograms count-based

Guo et al. reported their statistics on the original paper for the test collection, after preprocessing:

	Theirs	Mine
Vocabulary	0.6M	509731
Document Count	0.5M	523857
Query Count words	250	250

Table 7.4: Data words count

My vocabulary has less words than theirs because I performed stopwords removal on the full corpus of documents, not just on queries.

7.3.4 Word-embeddings training with Word2Vec

I used the Gensim implementation of Word2Vec [2] with the following parameters:

Word2Vec

Parameters	Value
Algorithm	CBOW
Dimension of term embeddings	300
Context window size	10
Negative samples per word	10
Sampling threshold (subsampling of frequent words)	10^{-4}
Minimum word count (words with frequency $<$ 10 are not considered)	10

Table 7.5: Parameters configuration for Word2Vec

[2] https://radimrehurek.com/gensim/models/word2vec.html

If a word is present only in a query and not in the corpus, that word is considered out-of-vocabulary.

7.3.5 A quick inspection of word embeddings

The vocabulary of Word2Vec model contains less words than the vocabulary of the collection, because all words that occurs less than 10 times are removed.

At this point, in the pipeline that builds the input to DRMM, the "words out-of-vocabulary" problem originates. That is, some words present in the collection have not a corresponding embedding.

stemming	stopwords removal	Original corpus vocabulary size	Word2Vec model vocabulary size
		629164	145226
	x	628751	144834
x	x	509731	110158

Table 7.6: Word counts

I report a simple look up of words similar to the word "night", with the respective cosine similarity, for all cases (Unstemmed + no stopwords removal; unstemmed + stopwords removal; stemmed + stopwords removal)

Similar words	Cosine similarity
nights	0.745
evening	0.736
morning	0.657
afternoon	0.619
afternoons	0.584
weekend	0.574

Table 7.7: Lookup word: "night"; unstemmed corpus without stopwords removal

Similar words	Cosine similarity
evening	0.777
nights	0.770
morning	0.699
afternoon	0.642
midnight	0.620
sunday	0.597

Table 7.8: Lookup word: "night"; unstemmed corpus with stopwords removal

Similar words	Cosine similarity
evening	0.779
nights	0.755
morning	0.719
afternoon	0.651
midnight	0.631
sunday	0.607

Table 7.9: Lookup word: "night"; stemmed corpus with stopwords removal

I also tried to inspect the IN-OUT and OUT-IN embeddings (cited in [21]) of the trained Word2Vec model. The results were as follows:

night	
OUT-IN	IN-OUT
gatoil	sleepless
usepaoppt	night
olefine	monday
irlo	saturday
chokepoint	sunday

Table 7.10: Lookup word: "night"; stemmed corpus with stopwords removal

7.3.6 Word embeddings matching signals analysis

Histograms in DRMM are generated based on the cosine similarity between word embeddings of a query and a document (see section 5.1.1).

Figure 7.3 shows the cosine similarity results between the query for "international organized crime" and a portion of a document in the collection.

A bin i,j in the matrix correspond to the cosine similarity between the embedding vector for the query term i and the document term j.

It can be noticed that the query term embedding "crime" has an high cosine similarity with terms embeddings of "evidence", "drug", "traffick", "mafia" and "justice".

This happens because Robust04 [31] is a news collection and "crime" co-occurs with all of these terms.

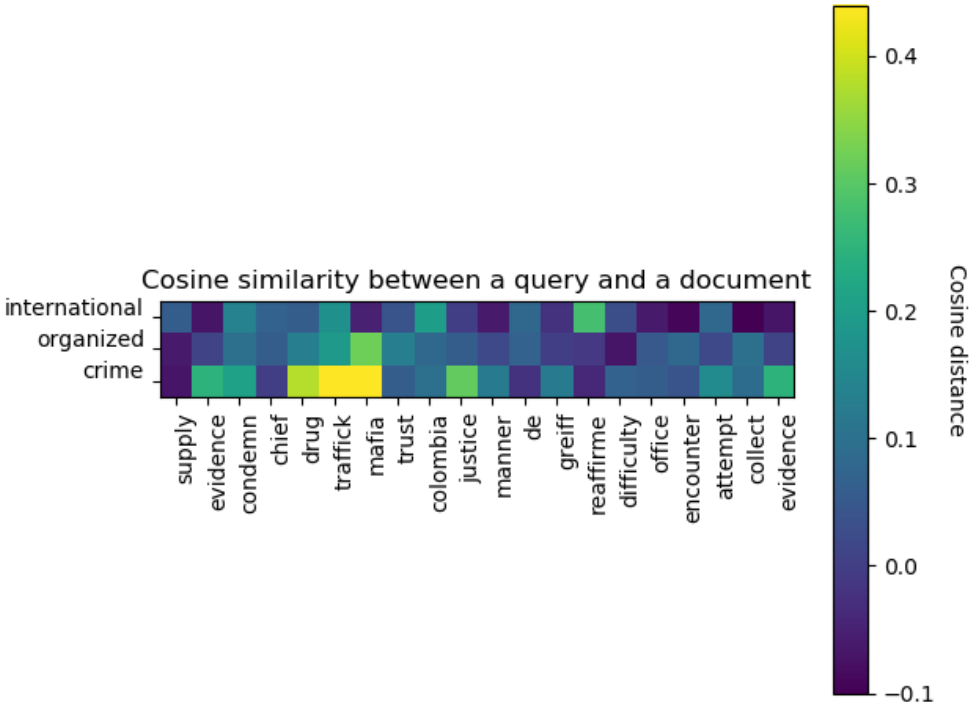
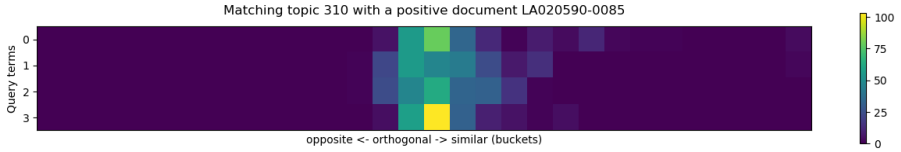


Figure 7.3: Cosine similarity between query terms and a slice of document

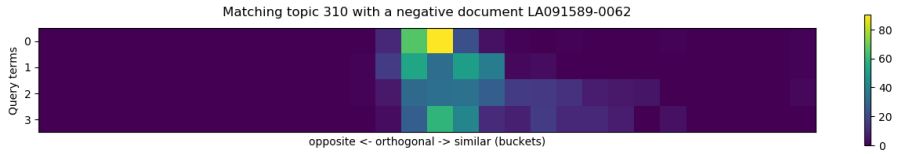
Next, I give some example of histograms, with respect to the title of topic 301 and two documents, relevant and non relevant. I chose two documents where it can be clearly seen that the matching signals distribution between the relevant and the non relevant is different.

Both document and query were stemmed and without stopwords. For each query term, there is a row of 30 bins (colored different, according to their frequency). Each bin contains a specific similarity range, for example the ones at the middle (0-(∼) 0.067) indicate how many words in the document are orthogonal to the query term; the ones on the right (left) indicate how many words are closer ("opposite") to the query term.

Note that word2vec does not capture antonymy, so opposite vector embeddings are **not** semantically opposite, which is counter-intuitive.



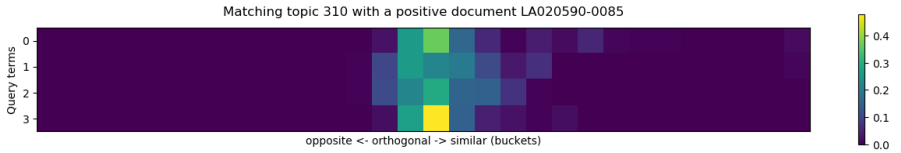
(a) Positive document histograms



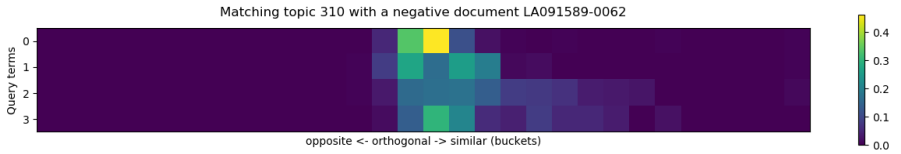
(b) Negative document histograms

Figure 7.4: Count-based istograms

In the following example, each histogram has been normalized by sum, in fact the range of values is smaller than the previous example.



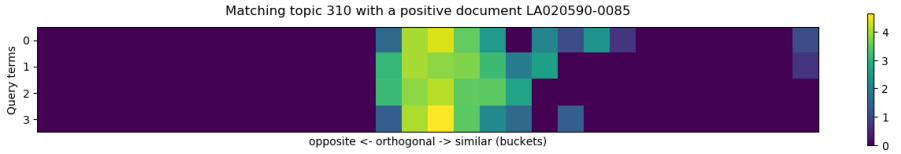
(a) Positive document histograms



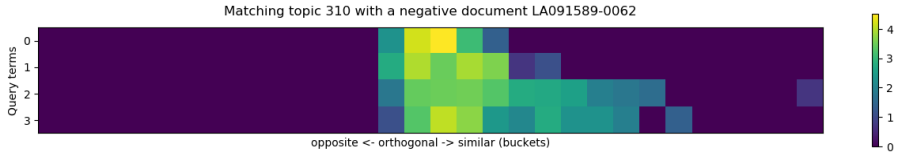
(b) Negative document histograms

Figure 7.5: Histograms normalized by sum

The last case is logarithm-based histograms. Matching signals stand out more in this case.



(a) Positive document histograms



(b) Negative document histograms

Figure 7.6: Histograms normalized by logarithm

7.3.7 Generation of histograms pair

All pair query/document from preranked results were considered and their histograms were taken as input to the feed forward neural network.

Histograms generation is a computationally expensive operation, which can be done offline (before training DRMM). Since the moment that this can save a lot of time, I chose to do it in my implementation (although it came with a disk space cost: saving the histograms for the whole corpus (Robust04 [31]) required up to 0.5GB).

7.4 Evaluation and metrics

For all the deep matching models Guo et al. ([10]) adopted a re-ranking strategy for efficient computation.

An initial retrieval was performed using two open-source search engines, Terrier and Galago, to obtain the top 2000 ranked documents (pre-ranked documents) for each topic.

The following table reports the evaluation results for the pre-ranked runs used in [10] by Guo et al. and the evaluation results for mine:

Method	MAP	nDCG20	P20
Galago DirichletLM($\mu = 1500$)	0.206	0.390	0.326
Galago Bm25($k_1 = 1.2, k_3 = 1.0, b = 0.75$)	0.193	0.377	0.319
Terrier DirichletLM($\mu = 2500$)	0.241	0.404	0.343
Terrier Bm25($k_1 = 1.2, k_3 = 8d, b = 0.75d$)	0.247	0.417	0.359
Original QL(*unknown parameters*)	0.253	0.415	0.369
Original Bm25(*unknown parameters*)	0.255	0.418	0.370

Table 7.11: Evaluation of preranked results

I chose to use the retrieval results which performance were closer to the original (Terrier runs).

As Guo pointed out in an issue in MatchZoo [3], on Robust04 dataset there is a data imbalance problem, in fact the number of labelled documents for each topic is significantly different from one another. In this way, the model could be dominated by a specific topic. Figure 7.7 shows the (logarithmically scaled) distribution of documents retrieved by Bm25 per topic. It can be noticed that there is also a different distribution of positive and negative examples. For instance, QL algorithm, on average, retrieves only 3.68% of positive samples for each topic (with some topics with 0 positive samples - e.g. topic 672) while Bm25 retrieves on average 3.70% positive samples per topic. Since the moment that the distributions relative to the runs obtained with these two algorithms are very similar, just the one relative to Bm25 is shown.

[3] https://github.com/NTMC-Community/MatchZoo/issues/604

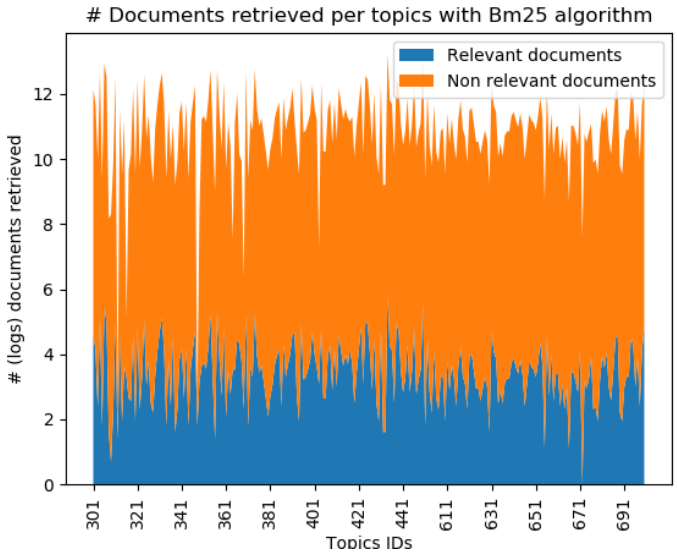


Figure 7.7: Queries imbalance problem - Bm25 algorithm

7.5 5-fold cross validation and parameters tuning

Guo et al. conducted 5 fold cross validation, to minimize overfitting on topics.

The 250 topics were divided into 5 folds. Each training phase was conducted with documents sampled from 4 folds and the fold left out was used as test set.

Each test fold contains approximately (due to the query imbalance problem, explained in the previous section) $2000 \cdot 50 = 100000$ documents.

At the end of each training phase, the model was re-initialized.

7.6 DRMM model configuration

Picture 7.8 shows the graph of DRMM. It consists of:

* a feed forward "matching" neural network for histograms (see equation Input to the l-th layer of ffmn);
* a term gating network for each query term (with query term embedding or idf as weight), which use the softmax as "gating" function (see equation Aggregation weight for TGN);
* the final output, computed as stated in equation "Relevance score for a pair (q, d)";
* the computation of the loss function (see equation Hinge loss function).

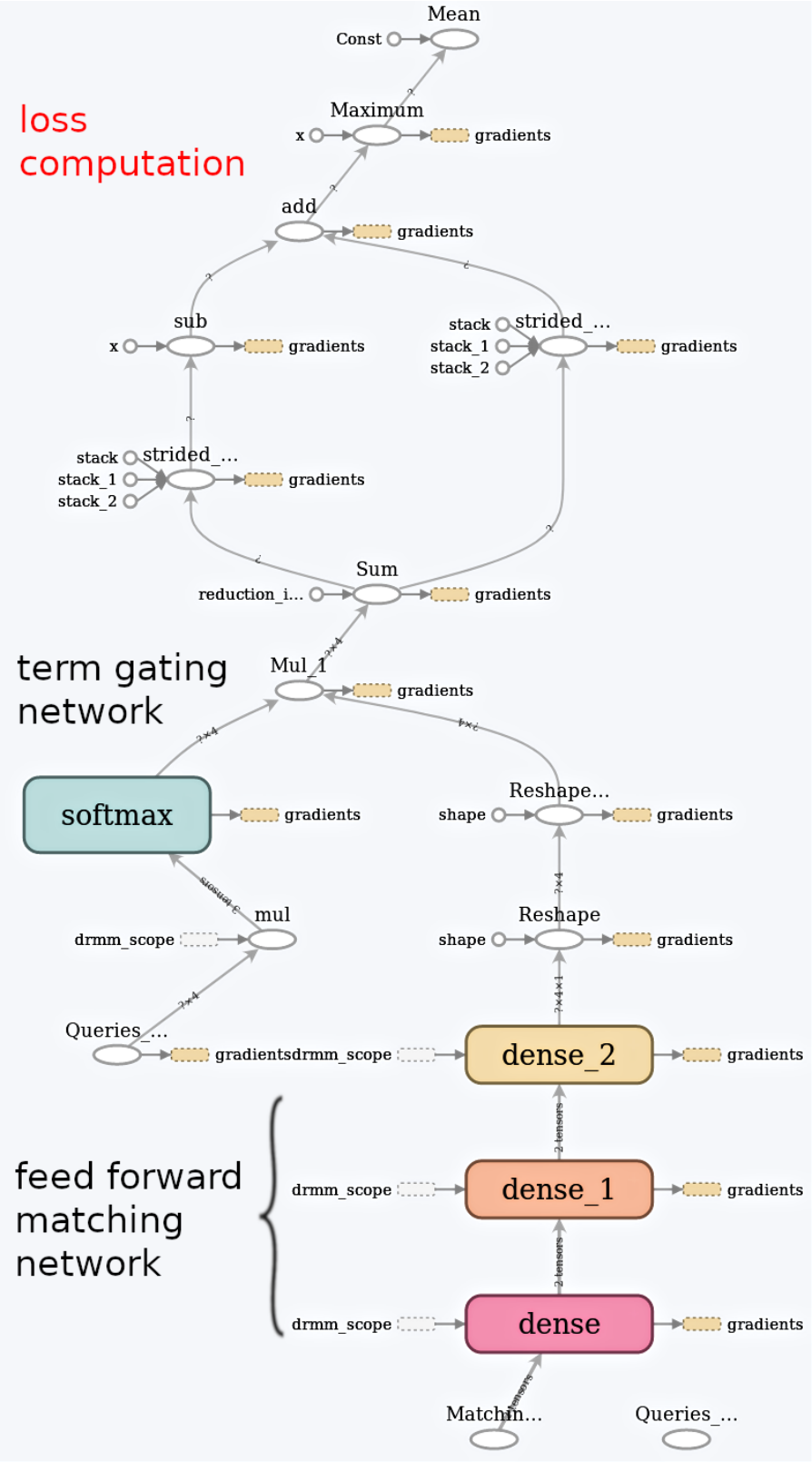


Figure 7.8: DRMM model (tensorflow graph)

7.6.1 Neural network configuration for experiment

For each of the experiments, DRMM feed forward neural network used the following parameters (some of them were empirically chosen):

DRMM configuration

Parameters	Value
Number of epochs	20
Mini batches size	20
Initial weights	glorot uniform initializer [4]
Initial learning rate	0.01
Positive/negative documents sampled	$\{30-30, 50-50, 100-100\}$
Callbacks	Early stopping (restoring best weights when stopping)
Early stopping (min_delta)	0.01
Early stopping (patience)	5
Seed (e.g. weights initialization, shuffling topics for cross validation etc. ...)	42

Table 7.12: Parameters configuration for DRMM

In order to evaluate the model under different configurations (to obtain possible improvements), some variations to these parameters were also applied.

7.6.2 Term gating with IDF

The following tables report the results of my implementation of DRMM. Each set of tables was obtained w.r.t. one of the three histograms modalities (ch, nh, lch). Then, each table in a set show results for different number of (balanced) positive (relevant)/negative (non relevant) samples for each query used for training.

The run to re-rank is given by the pre-ranked results from Terrier Bm25.

Table 7.13: DRMM runs (count-based histograms), IDF weighting, stemmed with stopwords removal

30-30 pos/neg				50-50 pos/neg			100-100 pos/neg		
fold	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20
1	0.12	0.202	0.222	0.147	0.265	0.293	0.164	0.261	0.298
2	0.109	0.173	0.216	0.119	0.206	0.241	0.136	0.232	0.264
3	0.11	0.172	0.186	0.126	0.193	0.22	0.157	0.233	0.27
4	0.098	0.175	0.198	0.122	0.199	0.235	0.138	0.238	0.264
5	0.106	0.152	0.2	0.133	0.187	0.233	0.171	0.25	0.311
avg.	**0.108**	**0.174**	**0.204**	**0.129**	**0.21**	**0.244**	**0.153**	**0.242**	**0.281**

Table 7.14: DRMM runs (sum normalized histograms), IDF weighting, stemmed with stopwords removal

30-30 pos/neg				50-50 pos/neg			100-100 pos/neg		
fold	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20
1	0.106	0.155	0.174	0.14	0.21	0.225	0.157	0.24	0.254
2	0.096	0.143	0.171	0.122	0.2	0.236	0.137	0.231	0.26
3	0.11	0.165	0.166	0.128	0.198	0.203	0.15	0.22	0.234
4	0.09	0.133	0.156	0.114	0.166	0.194	0.134	0.198	0.228
5	0.103	0.149	0.165	0.14	0.206	0.243	0.175	0.245	0.302
avg.	**0.101**	**0.149**	**0.166**	**0.128**	**0.196**	**0.22**	**0.15**	**0.226**	**0.255**

Table 7.15: DRMM runs (logarithm normalized histograms), IDF weighting, stemmed with stopwords removal

30-30 pos/neg				50-50 pos/neg			100-100 pos/neg		
fold	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20
1	0.198	0.331	0.373	0.217	0.347	0.392	0.224	0.354	0.402
2	0.209	0.309	0.388	0.21	0.309	0.389	0.211	0.301	0.381
3	0.203	0.276	0.32	0.206	0.287	0.329	0.206	0.281	0.324
4	0.172	0.306	0.341	0.177	0.304	0.349	0.191	0.333	0.373
5	0.237	0.316	0.407	0.236	0.321	0.405	0.239	0.324	0.411
avg.	**0.203**	**0.307**	**0.365**	**0.209**	**0.313**	**0.372**	**0.214**	**0.318**	**0.378**

A trivial consideration on the previous tests is that (in all case) increasing the number of (balanced) samples (i.e. increasing the training set size), leads to better performances. However, when using more than 100 samples of positive/negative documents per query, the performance of my implementation started to decrease, so I stopped the experiments at that number. I also tried unbalanced number of positive/negative documents per query (using repetitions of the smaller fraction of sample), but the results were not as good as the ones reported.

7.6.3 Term gating with term vector (TV)

Due to the memory constraint of the GPU I used, I lowered the mini batch size to 16; thus the comparison with the original results (20 mini batches) may be slightly unfair.

Table 7.16: DRMM runs, 100-100 pos/neg TV weighting, stemmed with stopwords removal

CH histograms				NH histograms			LCH histograms		
fold	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20
1	0.133	0.236	0.265	0.105	0.183	0.2	0.167	0.288	0.336
2	0.114	0.197	0.242	0.096	0.17	0.187	0.173	0.248	0.33
3	0.127	0.191	0.225	0.116	0.176	0.189	0.163	0.244	0.281
4	0.115	0.184	0.213	0.095	0.134	0.161	0.169	0.278	0.331
5	0.123	0.203	0.239	0.124	0.191	0.219	0.164	0.252	0.31
avg.	**0.122**	**0.202**	**0.236**	**0.107**	**0.17**	**0.19**	**0.167**	**0.262**	**0.317**

Term gating with query terms embeddings performed worse than its counterpart.

7.6.4 Different word embeddings

To evaluate the impact of the word embeddings used, I build the histograms using GloVe off-shelf word embeddings [5] (300-dimensional vectors), trained on Wikipedia2014 corpus and Gigawords 5 [6].

Table 7.17: DRMM runs, 100-100 pos/neg IDF weighting, stemmed with stopwords removal, word embeddings used: GloVe.6B.300D

	CH histograms			NH histograms			LCH histograms		
fold	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20
1	0.134	0.204	0.239	0.134	0.194	0.212	0.199	0.305	0.367
2	0.129	0.129	0.222	0.115	0.163	0.204	0.202	0.294	0.368
3	0.159	0.244	0.27	0.134	0.206	0.225	0.198	0.283	0.328
4	0.127	0.229	0.254	0.112	0.162	0.183	0.152	0.271	0.28
5	0.145	0.197	0.264	0.129	0.195	0.244	0.197	0.306	0.361
avg.	**0.1388**	**0.21**	**0.249**	**0.124**	**0.184**	**0.213**	**0.189**	**0.286**	**0.346**

These results appear to confirm that, in this case (100-100 pos/neg + IDF weighting), using word embeddings trained on the test collection (see the third table of 7.6.2, 7.6.2 and 7.6.2) is preferable over using off-shelf embeddings (7.6.4).

7.6.5 Different pre-ranking results

I repeated the experiments of section 7.6.2, using another test set, and using the best sample configuration (100-100 pos/neg) for each histograms mode mapping.

The test set for the following results refer to the top 2000 pre-ranked results from Terrier DirichletLM($\mu = 2500$):

Table 7.18: DRMM runs, 100-100 pos/neg IDF weighting, stemmed with stopwords removal, run to re-rank generated with "DiricheletLM" algorithm

	CH histograms			NH histograms			LCH histograms		
fold	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20	MAP	P@20	nDCG@20
1	0.163	0.261	0.285	0.171	0.278	0.291	0.213	0.345	0.387
2	0.133	0.222	0.264	0.147	0.248	0.279	0.192	0.271	0.344
3	0.13	0.2	0.224	0.162	0.228	0.248	0.199	0.279	0.314
4	0.139	0.239	0.284	0.141	0.214	0.243	0.181	0.315	0.354
5	0.154	0.219	0.282	0.184	0.26	0.31	0.235	0.299	0.382
avg	**0.1438**	**0.228**	**0.267**	**0.16**	**0.245**	**0.274**	**0.204**	**0.301**	**0.3562**

Since the moment that the Terrier Dirichlet LM run had a lower MAP and P@20 than the Bm25 one, these results are lower than the previous ones. This confirm that a Neural IR system that operate in a re-ranking strategy is very dependent on the previous retrieval system(s).

[5] downloaded from https://nlp.stanford.edu/projects/glove

[6] an archive of newswire text data https://catalog.ldc.upenn.edu/LDC2011T07

7.7 Final results

DRMM configuration	Average MAP	Average nDCG20	Average P20
DRMM CHxTV	0.122	0.202	0.236
DRMM NHxTV	0.107	0.17	0.19
DRMM LCHxTV	0.167	0.262	0.317
DRMM CHxIDF	0.153	0.242	0.281
DRMM NHxIDF	0.15	0.226	0.255
DRMM LCHxIDF	0.214	0.318	0.378

Table 7.19: Summary of my results

Compared to the original results:

DRMM configuration	Average MAP	Average nDCG20	Average P20
DRMM CHxTV	0.253	0.407	0.357
DRMM NHxTV	0.160	0.293	0.258
DRMM LCHxTV	0.268	0.423	0.381
DRMM CHxIDF	0.259	0.412	0.362
DRMM NHxIDF	0.187	0.312	0.282
DRMM LCHxIDF	**0.279**	**0.431**	**0.382**

Table 7.20: Summary of original results

7.8 Software used

All of the code was written in Python (v3.6). The libraries used are reported in the following list:

* Tensorflow-gpu v1.12.0: an open source software library for high performance numerical computation, with strong support for machine learning and deep learning.
* Sci-kit learn v0.20.3: a free machine learning library for Python;
* Gensim v3.4.0: a library for topic modelling, document indexing and similarity retrieval with large corpora;
* Numpy v1.13.3: the core library for scientific computing in Python;
* Matplotlib v3.0.2: a plotting library.;
* Numba v0.42.0: an open source just-in-time compiler that translates a subset of Python and NumPy code into fast machine code;
* BeautifulSoup v4.7.1: a library for pulling data out of HTML and XML files;

* Krovetz Stemmer [7];
* Terrier v4.2 ([30]);
* Trec eval 9.0.4.

[7] https://pypi.org/project/KrovetzStemmer, credits to the author: Ruey-Cheng Chen

CHAPTER 8

Discussion and conclusions

This thesis had two main objectives: the conduction of an analysis of Neural IR and the reproduction of the work of Guo et al. (DRMM, [10]).

Firstly, I introduced the field of study, described a search engine at a high level, some traditional retrieval algorithms and the evaluation methodology in IR.

Then, I reviewed some of the most relevant and recent works of Neural IR, an hybrid field that put together machine learning techniques and IR.

Although it is not clear yet what are the potential benefits of Neural IR to IR, in recent years - with the increasing data availability and computing power - it has been a trending topic within IR community.

Some traditional models based on a probabilistic approach (e.g. Bm25) or pseudo-relevance feedback (e.g. RM3) seems to be hard to beat for neural models based on raw text (at least in a relatively small collection of news like Robust04 [31]).

Craswell et al. in [3] argue that this may be caused by inadequate neural architectures for tasks of ad-hoc retrieval: the problem that this type of task faces is tied with the ability of a system to find out relevant documents with respect to a query.

Being a multi-faceted and abstract concept, relevancy is very difficult to model through a neural architecture.

In an ad-hoc retrieval setting, the documents in a test collection are typically heterogeneous: they are written by different authors and their length is variable. This leads to the critical vocabulary mismatch problem (between queries and documents) and diverse relevance pattern requirements (e.g. scope hypothesis vs verbose hypothesis).

Given this setting, it is not surprising that most of the recent Neural IR models rely on finding good and dense representations for queries and documents, and extract valuable information from their interactions. On the other hand, traditional systems rely on statistical properties of text.

As discussed in chapter 4 (4), the former aim towards semantic matching, while the latter aim towards syntactic matching. These two properties may be exploited or combined on the basis of the task considered.

For instance, documents that contains query words are retrieved by statistical-based/lexical models, while documents that do not contain query terms, but are

semantically similar to the query, are more likely to be retrieved by semantic models. However, there is no guarantee on their relevancy. DRMM tackles the problem by combining these properties and using supervised learning.

DRMM is a deep relevance matching model proposed by Guo et al. in 2016 ([10]) that tries to merge some peculiar aspects of neural architectures with properties of traditional retrieval algorithms. It employs word embeddings and allows the use of IDF and exact matching (for out of vocabulary words) to compute a similarity score between a query and a document.

Most of Neural IR models (including DRMM) have two problems (linked together): they are time-inefficient for document retrieval and most of them work only in re-ranking mode.

In fact, DRMM rely on a first-stage ranker that returns the top 2000 documents for each topic in the collection, thus, limiting the recall of its results.

For this reason, it is very important to rely on high quality pre ranked results (they should have an equal number of retrieved documents for each topic and a "sufficient" number of relevant documents).

As the authors did in the original paper, I run two retrieval algorithms for baseline comparison: DirichletLM and Bm25. Then, I parsed the test collection and applied a very soft pre-processing: the text was white-space tokenized, lower-cased, and stemmed with the Krovetz stemmer.

Stopwords removal was performed both on documents and query words with the INQUERY stoplist. Small changes at this stage can have a big impact in later stages: using different tokenization, stemming, apply lemmatization etc. . . can lead to significant different behaviors of neural models, often difficult to track.

Another important passage was the generation of word embeddings with Word2Vec (its hyper parameters were set according to the original paper).

My results indicated that when generating word embeddings on the test collection, DRMM performance improved. When using off-shelf embeddings, trained on external texts, the system performance got worse.

To sum up, the performance of my implementation of DRMM were influenced by three critical factors: the run to re-rank, the preprocessing steps applied and the word embeddings used.

Additionaly, I observed that, even with the same pre-ranked documents, the neural IR models that Guo et al. used as comparison and DRMM returned very different results.

For instance, DSSM achieved a MAP of 0.095, while its convolutional variant achieves 0.067. ARCI and ARCII performances were poor as well: 0.041 and 0.067 values of MAP respectively. All of these models started with a run that achieved a MAP of 0.253 with the QL algorithm.

This means that a serious problem of adopting a re-ranking strategy with neural IR models is that they can actually worsen the performance of the first-stage ranker.

I wondered what differentiates DRMM from other models I reviewed in order to justify such a gap in the results. I have found 3 possible answers:

* its *asymmetric architecture* based on *query split*;
* the usage of a query gate in the network to weigh query terms individually (and, particularly, the usage of IDF, as my results justify);

* the matching histograms mapping representation, which is suitable for strength-related signals.

The third property points out that DRMM does not takes into consideration position-related signals. This is somehow counter-intuitive, as many neural models that deal with text (in NLP particularly) take into account words order (e.g. LSTM, RNN). Comparisons between such models and DRMM were not mentioned, as IR and NLP have different objectives. In fact, in the original paper, DRMM was tested with a different input representation - one that considered positional signals - and its performance worsened. That indicates that ad hoc retrieval is a strength-related task rather than position-related.

Although the authors succeded in beating the baseline models with DRMM, I have not find the same results. My implementation of DRMM was able in some cases to get closer to the baseline, but was not able to improve them.

Some of the problems that I faced during the implementation of DRMM were systematic mistakes that I made during the implementation, others may be caused by an incorrect estimation of unknown parameters or missing unknown/additional procedures applied by the authors (for instance, they did not explain how to deal with the "query imbalance" problem).

Although I encountered some difficulties while trying to reproduce the work, it was a good (and challenging) opportunity for me to explore this new area of research.

Bibliography

[1] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2010 (cit. on pp. xiv, 23, 26).

[2] Marco Angelini et al. "VIRTUE: A Visual Tool for Information Retrieval Performance Evaluation and Failure Analysis". In: *J. Vis. Lang. Comput.* (Aug. 2014), pp. 394–413 (cit. on p. 14).

[3] Nick Craswell et al. "Neural information retrieval: introduction to the special issue". In: *Information Retrieval Journal* 21.2 (2018) (cit. on pp. xiv, 30, 31, 34, 77).

[4] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Morgan & Claypool Publishers, 2015 (cit. on pp. xiv, 3–5, 8, 9, 13, 21).

[5] Yixing Fan et al. "Matchzoo: A toolkit for deep text matching". In: *CoRR* abs/1707.07270 (2017) (cit. on p. 52).

[6] Nicola Ferro. "Reproducibility Challenges in Information Retrieval Evaluation". In: *J. Data and Information Quality* 2 (2017) (cit. on p. 50).

[7] J. R. Firth. "A synopsis of linguistic theory 1930-55." In: (1957), pp. 1–32 (cit. on p. 33).

[8] *GloVe: Global Vectors for Word Representation - website*. URL: https://nlp.stanford.edu/projects/glove (visited on 06/01/2019) (cit. on pp. 34, 36).

[9] Jiafeng Guo et al. "A Deep Look into Neural Ranking Models for Information Retrieval". In: *CoRR* abs/1903.06902 (2019) (cit. on pp. 29, 32).

[10] Jiafeng Guo et al. "A Deep Relevance Matching Model for Ad-hoc Retrieval". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. ACM, 2016 (cit. on pp. ix, xiv, 2, 24, 31, 32, 43–46, 58, 68, 77, 78).

[11] Zellig Harris. "Distributional structure". In: *Word* 10 (1954), pp. 146–162 (cit. on p. 35).

[12] Baotian Hu et al. "Convolutional Neural Network Architectures for Matching Natural Language Sentences". In: NIPS'14 (2014) (cit. on p. 31).

[13] Po-Sen Huang et al. "Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data". In: *Proceedings of the 22Nd ACM Inter-*

national Conference on Information & Knowledge Management. CIKM '13. ACM, 2013 (cit. on pp. 31, 45).

[14] Quoc V. Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *CoRR* (2014) (cit. on p. 31).

[15] Omer Levy and Yoav Goldberg. "Linguistic Regularities in Sparse and Explicit Word Representations". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, June 2014 (cit. on p. 34).

[16] Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers, 2011 (cit. on pp. 18, 19).

[17] Jimmy Lin. "The Neural Hype and Comparisons Against Weak Baselines". In: *SIGIR Forum* 52.2 (2019) (cit. on pp. 54, 55).

[18] Zhengdong Lu and Hang Li. "A Deep Architecture for Matching Short Texts". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 1367–1375. URL: http://papers.nips.cc/paper/5019-a-deep-architecture-for-matching-short-texts.pdf (cit. on p. 31).

[19] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008 (cit. on p. 11).

[20] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Curran Associates Inc., 2013 (cit. on pp. 20, 31, 36, 39, 47).

[21] Bhaskar Mitra et al. "A Dual Embedding Space Model for Document Ranking". In: *CoRR* (2016) (cit. on pp. xv, 31, 40, 65).

[22] Kezban Dilek Onal et al. "Neural Information Retrieval: At the End of the Early Years". In: *Inf. Retr.* 21.2-3 (2018) (cit. on pp. xiv, 29, 30, 32).

[23] Jinfeng Rao, Hua He, and Jimmy Lin. "Experiments with Convolutional Neural Network Models for Answer Selection". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. ACM, 2017 (cit. on p. 55).

[24] Stephen Robertson. "The Probability Ranking Principle in IR". In: *Journal of Documentation* 33 (1977), pp. 294–304 (cit. on p. 9).

[25] Xin Rong. "word2vec parameter learning explained". In: *arXiv preprint arXiv:1411.2738* (2014) (cit. on pp. xiv, 37, 39).

[26] Corby Rosset et al. "An Axiomatic Approach to Regularizing Neural Ranking Models". In: *CoRR* abs/1904.06808 (2019) (cit. on p. 55).

[27] Geir Kjetil Sandve et al. “Ten Simple Rules for Reproducible Computational Research.” In: *PLoS Computational Biology* 9.10 (2013) (cit. on p. 53).

[28] Matthias Schwab, Martin Karrenbach, and Jon Claerbout. “Making Scientific Computations Reproducible”. In: *Computing in Science and Engg.* 2.6 (2000) (cit. on p. 49).

[29] Yelong Shen et al. “Learning Semantic Representations Using Convolutional Neural Networks for Web Search”. In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW ’14 Companion. ACM, 2014 (cit. on pp. 24, 31).

[30] *Terrier website*. URL: http://terrier.org (visited on 06/01/2019) (cit. on pp. 21, 75).

[31] *TREC 2004 Robust Track Guidelines*. URL: https://trec.nist.gov/data/robust/04.guidelines.html (visited on 06/01/2019) (cit. on pp. vii, ix, 32, 47, 57, 58, 60, 65, 68, 77).

[32] *TREC website*. URL: https://trec.nist.gov (visited on 06/01/2019) (cit. on p. 13).

[33] Peilin Yang, Hui Fang, and Jimmy Lin. “Anserini: Reproducible Ranking Baselines Using Lucene”. In: *J. Data and Information Quality* 10.4 (2018) (cit. on pp. 54, 55).

[34] Hamed Zamani and W. Bruce Croft. “Relevance-based Word Embedding”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’17. ACM, 2017 (cit. on p. 40).

[35] Hamed Zamani et al. “From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing”. In: 2018 (cit. on pp. 31, 32).

[36] Ye Zhang et al. “Neural Information Retrieval: A Literature Review”. In: *CoRR* abs/1611.06792 (2016) (cit. on pp. 20, 39).