Relevance Ranking for Web Search

João Pedro Gonçalves Lages

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. João Paulo Baptista de Carvalho
Doctor Ramón Fernandez Astudillo

Examination Committee
Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Supervisor: Prof. João Paulo Baptista de Carvalho
Member of the Committee: Prof. Bruno Emanuel Da Graça Martins

May 2018
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
Acknowledgments

I would like to start by thanking Prof. João Paulo Carvalho for supervising this thesis and having embraced its topic.

For the technical part, I have to express my gratitude to Ramón Astudillo for the support and guidance provided. I would like to acknowledge as well some people who had a relevant impact on my work and helped me. Some of them are Kai Hui and Andrew Yates that, besides being the authors of the baseline model analyzed in this thesis, also were eager to answer all my questions regarding implementation details and that ended up being crucial in the most early stages of my work.

Moreover, I also have to mention Jeanne Jirges for her assistance and the Diffbot company, specially its CEO, Mike Tung, that did not hesitate in letting me use their awesome article extraction API, to retrieve texts from HTML documents. I also have to thank the Carnegie Mellon University, for providing the datasets used in this project.

Last, but not least, I would like to thank my family and friends for the never ending support.
Resumo

Ordenar documentos de acordo com a sua relevância é um problema fulcral na área de Recuperação de Informação. Para uma certa query e um conjunto de documentos, um algoritmo é treinado para determinar o grau de relevância de cada documento em relação à query. Uma lista é depois construída com os documentos todos ordenados por ordem descendente, consuante os seus graus de relevância.


Experiências usando data do conhecido TREC Web Track mostram como é possível obter pequenos ganhos em relação ao modelo original, e apontam para várias limitações da abordagem ao problema.

**Palavras-chave:** Recuperação de Informação, Ordenação por Rellevância, Deep Learning, Attention Mechanisms, Rede Neural Convolucional
Abstract

Relevance ranking is a core problem of Information Retrieval which plays a fundamental role in various real world applications, such as search engines. Given a query and a set of candidate documents, relevance ranking algorithms determine how relevant each document is for the given query. This degree of relevance allows them to rank the documents and perform actions such as returning the best matching documents for the query.

As in other machine learning disciplines, deep learning techniques have recently achieved state of the art results by successfully capturing relevance matching signals between query-document pairs. This dissertation focuses on the recently introduced Position-Aware Convolutional-Recurrent Relevance Matching approach. On a first phase, it reimplements the original work, reproduces the published results and performs a number of additional experiments that identify potential model limitations. On a second phase, it explores possible model improvements based on deep learning techniques such as self-attention and deep transfer learning.

Experiments on the well known TREC Web Track data show that it is possible to obtain small improvements over the original model and point to a number of limitations of the general approach due to the information bottlenecks involved.

Keywords: Information Retrieval, Relevance Ranking, Deep Learning, Attention Mechanisms, Convolutional Neural Networks
Contents

Acknowledgments .......................................................... v
Resumo ................................................................. vii
Abstract ................................................................. ix
List of Figures .......................................................... xiii
List of Tables ............................................................ xv
List of Acronyms ........................................................ xvii

1 Introduction 1
  1.1 Motivation ....................................................... 1
  1.2 Contributions .................................................. 2
  1.3 Thesis Outline .................................................. 2

2 Background 5
  2.1 Information Retrieval .......................................... 5
     2.1.1 Ad-hoc Retrieval .......................................... 6
     2.1.2 Search Engines ........................................... 6
     2.1.3 Note on Relevance ....................................... 6
  2.2 Article Text Extraction from HTML ........................... 7
  2.3 Term Representations .......................................... 8
  2.4 Traditional IR Models ......................................... 10
     2.4.1 BM25 ...................................................... 10
     2.4.2 Language Model ......................................... 11
     2.4.3 Dependence Model ...................................... 11
  2.5 Learning to rank ............................................... 12
     2.5.1 Pointwise approach ...................................... 13
     2.5.2 Pairwise approach ....................................... 13
     2.5.3 Listwise approach ...................................... 14
  2.6 Deep Learning .................................................. 14
     2.6.1 Introduction .............................................. 14
     2.6.2 Training procedure ...................................... 15
     2.6.3 Challenges in neural network optimization .......... 16
3 Relevance Matching for Adhoc Retrieval using Deep Neural Networks 29

3.1 Baseline Framework 29

3.1.1 Similarity matrix and query IDF vector 30

3.1.2 Convolution and Pooling Layers 31

3.1.3 Producing the final score 32

3.1.4 Training Phase 32

3.2 Proposed changes 34

3.2.1 Lack of regularization 34

3.2.2 Parameterless pooling operations 35

3.2.3 Static similarity matrix 38

3.2.4 Loss function 40

3.3 Final proposed model 41

4 Experiments 43

4.1 Description of the task 43

4.2 Implementation details 45

4.2.1 Framework 45

4.2.2 Input Pre-processing 45

4.2.3 Weight initialization 46

4.2.4 Masking CNNs 47

4.2.5 Gradient norm clipping 47

4.3 Objective Evaluation 48

4.3.1 Baseline model reimplementation 49

4.3.2 Matrices dimensionality reduction 53

4.3.3 Changing the input 54

4.3.4 Best performing model 54

4.4 Discussion 57

5 Conclusions 61

5.1 Achievements 61

5.2 Future Work 62

Bibliography 63
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Overview of the retrieval system.</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Simplified architecture of a search engine.</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Two types of term representations. Extracted from [15].</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>High-level abstraction diagram of LTR models.</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>GoogleNet architecture, used in Image Recognition. Extracted from [29].</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Example of a non-convex function.</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>Example of a saddle point over a three-dimensional surface. Extracted from [16].</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>Sample architecture of a Multilayer Perceptron with several hidden layers and units.</td>
<td>18</td>
</tr>
<tr>
<td>2.8</td>
<td>The skip-gram model. Extracted from [36].</td>
<td>20</td>
</tr>
<tr>
<td>2.9</td>
<td>Continuous bag-of-words model. Extracted from [36].</td>
<td>21</td>
</tr>
<tr>
<td>2.10</td>
<td>Representation of a common soft attention mechanism.</td>
<td>23</td>
</tr>
<tr>
<td>2.11</td>
<td>Representation of a common hard attention mechanism.</td>
<td>24</td>
</tr>
<tr>
<td>2.12</td>
<td>High-level abstraction diagram comparing representation and interaction focused models.</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the baseline model.</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of a k-max pooling operation.</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>Loss computation during training phase.</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Effects of dropout on the network topology. Extracted from [58].</td>
<td>35</td>
</tr>
<tr>
<td>3.5</td>
<td>Simplified baseline model: pooling layers marked in red.</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>Using an attention mechanism as an alternative to k-max pooling.</td>
<td>37</td>
</tr>
<tr>
<td>3.7</td>
<td>Simplified baseline model: similarity matrix construction marked in red.</td>
<td>38</td>
</tr>
<tr>
<td>3.8</td>
<td>Construction overview of the proposed extra similarity matrix using 1D convolutions.</td>
<td>39</td>
</tr>
<tr>
<td>3.9</td>
<td>Construction overview of the proposed 3D similarity matrix.</td>
<td>40</td>
</tr>
<tr>
<td>3.10</td>
<td>Overview of the proposed model.</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of the baseline architecture with and without dropout.</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Simplified final model.</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Query example, from the 2014 TREC Web Track [63].</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>Text markup illustrating the input to different k-max pooling layers.</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Query example, from the 2009 TREC Web Track [72].</td>
<td>57</td>
</tr>
<tr>
<td>4.6</td>
<td>Examples of similar documents with different relevance grades.</td>
<td>57</td>
</tr>
</tbody>
</table>
4.7 Illustrations of different parts of documents that were not extracted from the HTML code.
List of Tables

2.1 Evaluation results of the comparison runned by Diffbot .......................... 8

4.1 Relevance judgments used in the TREC Web Track ............................... 44

4.2 ERR@20 and nDCG@20 values comparison between PACRR [50] and Query Likelihood. 48

4.3 ERR@20 and nDCG@20 values comparison between RPACRR trained on the self-made
or original similarity matrices. ................................................................. 49

4.4 ERR@20 and nDCG@20 values comparison between RPACRR with and without query
IDF vector. ....................................................................................................... 52

4.5 ERR@20 and nDCG@20 values comparison between RPACRR with binary cross entropy
loss and custom loss. .......................................................... 52

4.6 ERR@20 and nDCG@20 values comparison between RPACRR methods for dimension-
ality reduction. .......................................................... 53

4.7 ERR@20 and nDCG@20 values comparison between RPACRR methods for changing
the model’s input. ...................................................................................... 54

4.8 ERR@20 and nDCG@20 values comparison between RPACRRF and PACRR. .......... 55
List of Acronyms

ANN    Artificial Neural Network
CNN    Convolutional Neural Network
CBOV   Continuous Bag of Words
ERR    Expected Reciprocal Rate
HTML   Hypertext Markup Language
IR     Information Retrieval
IDF    Inverse Document Frequency
LTR    Learning to Rank
MLP    Multilayer Perceptron
NLP    Natural Language Processing
nDCG   Normalized Discounted Cumulative Gain
OOV    out-of-vocabulary
RBF    Radial Basis Function
ReLU   Rectified Linear Unit
SVM    Support Vector Machine
TF-IDF Term Frequency-Inverse Document Frequency
TREC   Text Retrieval Conference
Chapter 1

Introduction

1.1 Motivation

Document relevance ranking is a core problem in Information Retrieval (IR) and performed by popular search engines such as Google, Bing, Baidu, or Yandex. In traditional Web search, the query consists of only few terms but the body text of the documents can range vastly in length. A ranking model aims at evaluating the interactions between different documents and a query, assigning higher scores to documents that better match the query. In the absence of click information, such as for newly-published or infrequently-visited documents, the raw body text can be a useful signal to determine the relevance between a query-document pair.

Traditional machine learning algorithms for document relevance ranking relied on handcrafted features to encode interactions between queries and documents. This feature engineering work was usually time-consuming, incomplete and over-specified, which largely hindered further development of these approaches. In recent years, new algorithms have been reported in the area of Information Retrieval [1] [2], that apply deep neural networks for this purpose.

As it happened with other machine learning disciplines such as vision or speech processing, the potential of deep learning methods to advance state of the art retrieval quality has attracted a lot of attention.

The newly introduced neural IR models differ from previous approaches in that they model the interactions between query and document directly based on the raw text, without the need for manual feature engineering. However, unlike many classical IR models, these new deep learning based approaches are data-hungry, requiring large scale training data before they can be employed.

This thesis is motivated by this recent success of neural IR to relevance ranking, as well as the large improvements in performance attained in other areas of Natural Language Processing by employing deep learning techniques applied directly to raw text.
1.2 Contributions

With the objective of improving the performance of prior neural relevance ranking models, this dissertation considers only data in the form of text and no other information about either the query or document.

Figure 1.1 depicts a common Web retrieval system. The work in this dissertation concerns the relevance ranking components depicted in blue. This model is used to rerank search results, reordering the previously extracted texts by their descending relevance score to the given query. This ranking model is trained using a supervised machine learning approach, using a set of query-document pairs labeled by human annotators.

In short, this thesis intends to contribute with the following aspects:

1. Design of a relevance ranking pipeline, given a text query and HTML documents as inputs to the system.
2. Implementation of a baseline model along with proposed improvements over it for relevance matching between query-document pairs.

1.3 Thesis Outline

This thesis is organized in five chapters. Chapter 2 explains different topics related with the work that is developed in this thesis, as well as detailing techniques that have been used throughout the field of Information Retrieval.

Chapter 3 describes the chosen state-of-the-art baseline approach for this work, firstly explaining different building blocks used on it, and presents various proposed changes to that architecture.

In Chapter 4 all the experiments made with the baseline and proposed models are described and discussed, based on the results of the objective evaluation metrics used, while some conclusions based
on the architectures and experimental results will also be presented.

Finally, Chapter 5 briefly discusses the main outcomes from this thesis and some possible future work directions.
Chapter 2

Background

This chapter provides an overview of the general concepts related with the developed work, focusing on making a superficial literature review of the state of the art of Information Retrieval. It is assumed that the reader has basic knowledge of supervised Machine Learning concepts.

This chapter has the following structure: Section 2.1 introduces in detail the problem at hand; Section 2.2 briefly indicates algorithms used in article text extraction; Section 2.3 distinguishes the most common word representations used in the area; Sections 2.4 and 2.5 describe traditional and Learning to Rank models, respectively, that have been used in the area, while more focus is given to deep neural models in Section 2.6; finally, Section 2.7 explains how those models are evaluated.

2.1 Information Retrieval

The main problem in Information Retrieval (IR) is finding relevant data from a collection of resources, according to a searcher’s query, where the query is typically expressed as a string of words. Retrieval can involve ranking previous existing information in the collection or composing new responses with it. These kind of systems may consider user history, physical location or temporal changes in information that are supplementary to the query provided. They may also help users formulate their intent, for instance, via query auto-completion or query suggestion, and/or extract succinct summaries of results for easier inspection.

IR systems should learn patterns in query and document text pairs that indicate relevance between them. Although IR shares some common aspects with the area of Natural Language Processing (NLP), it also comes with its particularities. Some of them being that short queries can contain previously unseen vocabulary and that documents vary in length and may have large sections of irrelevant text. Nonetheless, NLP techniques can still be applied to IR when dealing with raw text data although these specific task-related characteristics have to be taken into account.
2.1.1 Ad-hoc Retrieval

This project is focused on ad-hoc retrieval in IR, where the user enters a text query and the retrieval system returns a list of search results, ordered by their relevance to the query, where the relevance represents the satisfaction of the user’s information need. Search results may be excerpts or full text documents. This problem is a central one in the IR literature, with well understood challenges and solutions.

Ranked document retrieval is performed by popular search engines. Although more information such as clickthrough data and other document fields [3] can be used, in this thesis it is assumed that only the document’s body text is available, as is the case of newly created documents present in Web pages. We also argue that not every document is popular enough to have past links and clicks, but it still may be the best search result for a user’s rare or tail query. In such cases, when text metadata is unavailable, it is crucial to estimate the document’s relevance primarily based on its text content.

2.1.2 Search Engines

Web search engines have, broadly speaking, three basic phases: crawling, indexing and searching. They work according to a user’s query, taking some keywords as input and returning the URLs that satisfy the query. This can be achieved by reading the content of all Web pages and search for matching patterns with the words specified by the user. The problem of this technique is that it requires a lot of computational effort at query time (linear time in relation with the length of the documents stored), which makes it infeasible. The solution is to index the content of the Web pages by building an inverted index prior to the searches.

Search engines have to deal with the size and rate of change of the Web, with no search engine indexing more than one third of the publicly available Web. When we use search engines to search information about a particular topic, we are not searching on the Web on-line, but actually in the search engine’s database (off-line) of Web pages information (the inverted index that it built previously).

When ordering results, some ranking algorithms are employed. A famous one, used by Google Search, is PageRank [4]. We will further analyze these ranking models in the next sections. Figure 2.1 presents how a machine-learned ranking model can be applied in search engines to rank the results after them being retrieved by the inverted index.

2.1.3 Note on Relevance

Although the relevance of a document to a query has long been the primary criterion according to which IR systems have been designed and judged, there is an equally long history of dissatisfaction, or disquiet, with this criterion [5] [6]. The problem appears to be that relevance, as most traditionally construed, deals with topical relationships between document and query, while the judgments that people make about the usefulness of a document to their information problems seem often to be biased on their personal opinions, making the problem subjective. Additionally, topical relevance itself may be a
complex phenomenon, with judgments being based upon a variety of aspects of the aboutness of the
document in relationship to the information problem.

Having that said, ad-hoc retrieval is a complicated problem that tries to predict the relevance of a
query-document pair that was, most of the times, accorded by a group of human annotators. Due to the
intrinsic subjective characteristic of relevance (or relatedness) the data used to train or evaluate these
systems is particularly noisy, i.e., can have contradictory or questionable relevance grades.

2.2 Article Text Extraction from HTML

One of the more general ways of getting information is accessing many websites on the Internet. How-
ever, when examining a Web page, humans are more or less able to distinguish the main content from
navigational text, advertisements, related articles and other text portions. Extracting the same informa-
tion from HTML code is not so trivial though, since a website can be structured in different ways, so that
the article text does not always appear where it was predicted to be.

This problem was baffling researchers since the early 2000s. Major methods presented in research
papers from that epoch are nowadays useless due to strong assumptions and heuristics that do not
apply on today’s web development practices.

Analyzing how Article Extraction is performed is out of the scope of this thesis. Nonetheless, it is a
crucial step in ad-hoc retrieval when applied in the Web, as the raw HTMLs have to be converted to texts
first in order to apply all the techniques that we have seen above.

As in various fields of study, precision $P$ and recall $R$ are used in the text retrieval context to evaluate
systems. Given a set of words retrieved from HTML documents, $W_{\text{ret}}$, and its subset of relevant words
$W_{rel}$, these metrics are defined as:

$$P = \frac{|W_{rel} \cap W_{ret}|}{|W_{ret}|}$$ (2.1)

$$R = \frac{|W_{rel} \cap W_{ret}|}{|W_{rel}|}$$ (2.2)

Another metric, the F1 score [7], which is more robust to class imbalance is also used. The F1 can be seen as the geometric mean between the precision and recall and is computed using the following equation:

$$F1 = 2 \frac{P \times R}{P + R}$$ (2.3)

Applying these metrics, Diffbot, a commercial API [8], released a comparison of its Article API against similar text extractors [9]. These other extractors were Boilerpipe [10], Goose [11], Readability [12] and AlchemyAPI [13]. The HTML pages used were from a random assortment of archived Google News-indexed articles, in multiple languages and manually human-labelled. The results of this evaluation can be viewed in Table 2.1. Note that these different metrics will not be used for the remaining of this document and are only here explained to have a deeper understanding of these results.

<table>
<thead>
<tr>
<th>Software</th>
<th>$P$</th>
<th>$R$</th>
<th>$F1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffbot</td>
<td>0.968</td>
<td>0.978</td>
<td>0.971</td>
</tr>
<tr>
<td>Boilerpipe</td>
<td>0.893</td>
<td>0.924</td>
<td>0.893</td>
</tr>
<tr>
<td>Goose</td>
<td>0.498</td>
<td>0.815</td>
<td>0.608</td>
</tr>
<tr>
<td>Readability</td>
<td>0.819</td>
<td>0.911</td>
<td>0.854</td>
</tr>
<tr>
<td>AlchemyAPI</td>
<td>0.876</td>
<td>0.892</td>
<td>0.850</td>
</tr>
</tbody>
</table>

Table 2.1: Evaluation results of the comparison runned by Diffbot.

These results are similar to the first broad evaluation of web page text-extraction engines, performed by Kovacic [14].

2.3 Term Representations

At a high level, document ranking in ad-hoc retrieval comprises of performing three steps:

1. Generate a representation of the query;
2. Generate a representation of the document;
3. Match both representations to estimate their mutual relevance.
Transforming text in vector representations is elementary in the fields of information retrieval and machine learning. In IR, words are typically the smallest unit of representation for indexing and retrieval, making many models to focus on learning good vector representations of those terms. Different vector constructions exhibit distinct levels of generalization and notions of similarity from the definition of the corresponding vector spaces. Some consider every term as distinct entities (e.g., bag-of-words approach) while others learn to identify common attributes (e.g., vector space model). Some representations operate over fixed-size vocabularies, while the design of others obviate such constraints. They also differ on the properties of compositionality that defines how representations for larger units of information, such as passages and documents, can be derived from individual term vectors. Having that said, we can classify text vector representations as local and distributed (Figure 2.2).

For local (or one-hot) representations of words, every term is represented by a binary vector, which has the size of the whole vocabulary, where only one of the values in the vector is one and all the others are zeros. Each position in this vector corresponds to a particular word.

On the other hand, distributed representations express a word in a vector that can be either sparse or dense, that can have handmade features or learnt representations in which singular dimensions are not interpretable in isolation, like they were in one-hot vectors. The main motivation for using any distributed representation scheme is that relationships between terms can be defined this way, based on the chosen properties, which is not possible with local representations.

Depending on whether the distributed representation is high-dimensional and sparse or small and dense, they are referred to as explicit vector representations or embeddings, respectively. Word embeddings are typically preferred over explicit vectors due to their compact representation.

An embedding is a representation of items in a new space such that the properties of, and the relationships between, the items are preserved. Goodfellow et al. [16] articulate that the goal of an embedding is to generate a simpler representation — where simplification may mean a reduction in the
number of dimensions, an increase in the sparseness of the representation, disentangling the principle components of the vector space, or a combination of these. In the context of term embeddings, the explicit feature vectors constitute the original representation. An embedding trained from these features assimilates the properties of the terms and the inter-term relationships observable in the original feature space. The most popular approaches for learning embeddings include either factorizing the term-feature matrix (e.g. Latent Semantic Analysis [17]) or using gradient descent based methods that try to predict the features given the term. Baroni et al. [18] empirically demonstrate that these feature-predicting models that learn lower dimensional representations, in fact, also perform better than explicit counting based models on different tasks.

When applied in deep learning, embeddings can be used as the first layer of a neural network and they can also be trained end-to-end along with the weights and biases of the model.

2.4 Traditional IR Models

Years of insights in the area of Information Retrieval are mandatory to highlight diverse features that models should retain. They also serve as important baselines to compare with.

2.4.1 BM25

Most statistical functions in IR consider the number of occurrences (term-frequency \( tf \)) of each query term \( t_q \) in the document \( d \), \( tf(t_q, d) \), and the corresponding inverse document frequency, \( idf \), of the same terms in the whole corpora. One theoretical basis for such formulations is the probabilistic model of IR that yielded the popular bag-of-words retrieval system, the BM25 ranking function. Its formula is given by

\[
BM25(q,d) = \sum_{t_q \in q} \frac{idf(t_q) \cdot tf(t_q, d) (k_1 + 1)}{tf(t_q, d) + k_1 (1 - b + b |D| / \text{avgdl})}
\]  

where, \( \text{avgdl} \) is the average length of documents in the collection \( D \), and \( k_1 \) and \( b \) are parameters usually tuned on a validation dataset. In practice, \( k_1 \) is sometimes set to some default value in the range \([1.2, 2.0]\) and \( b \) as 0.75. The \( idf(t) \) is popularly computed as,

\[
idf(t) = \log \frac{|D| - df(t) + 0.5}{df(t) + 0.5}
\]

BM25 assembles contributions from individual terms but ignores any positional information or semantic relations between different query and document terms. A variant of BM25 was later published by Robertson, BM25-F [19], that could take document structure into account.
2.4.2 Language Model

In the language modeling based approach, the relevance of a document \( d \) to a query \( q \) is ranked according to its posterior probability \( p(d | q) \).

\[
p(d | q) = \frac{p(q | d) \cdot p(d)}{\sum_{\bar{d} \in D} p(q | \bar{d}) \cdot p(\bar{d})} \propto p(q | d) \cdot p(d) \tag{2.6a}
\]

Assuming \( p(d) \) is uniform,

\[
p(q | d) \cdot p(d) = p(q | d) \tag{2.6b}
\]

\[
= \prod_{t_q \in q} p(t_q | d) \tag{2.6c}
\]

\[
= \prod_{t_q \in q} (\lambda \hat{p}(t_q | d) + (1 - \lambda)\hat{p}(t_q | D)) \tag{2.6d}
\]

\[
= \prod_{t_q \in q} \left( \frac{\lambda f(t_q, d)}{|d|} + (1 - \lambda) \frac{\sum_{\bar{d} \in D} f(t_q, \bar{d})}{\sum_{\bar{d} \in D} |\bar{d}|} \right) \tag{2.6e}
\]

where, \( \hat{p}(\epsilon) \) is the maximum likelihood estimate of the probability of event \( \epsilon \). \( p(q | d) \) indicates the probability of generating query \( q \) by randomly sampling terms from document \( d \). For smoothing, terms are sampled from both the document \( d \) and the full collection \( D \) — the two events are treated as mutually exclusive, and their probability is given by \( \lambda \) and \( (1 - \lambda) \), respectively.

There are other methods to calculate the posterior probability \( p(d | q) \) that differ from the one described above. Another particular method important to refer is the query likelihood, that constructs a language model \( M_d \) for every document and ranks them according to the probability that the query analyzed is observed as a random sample from the document model, \( p(q | M_d) \). The most common way to calculate this probability is using the multinomial unigram language model, which is equivalent to a multinomial Naive Bayes, where the documents are the classes.

\[
p(q | M_d) = K_q \prod_{t \in V} p(t | M_d)^{tf(t,d)} \tag{2.7}
\]

where \( V \) represents the whole vocabulary and \( K_q \) is the multinomial coefficient for query \( q \), defined as

\[
K_q = \frac{|d|!}{\prod_{t_q \in q} tf(t_q,d)!} \tag{2.8}
\]

2.4.3 Dependence Model

None of the IR models previously described so far consider proximity between query terms. To address this, Metzler and Croft [20] proposed a linear model over proximity-based features.
\[ DM(q, d) = (1 - \lambda_{ow} - \lambda_{uw}) \sum_{t_q \in q} \log \left( 1 - \alpha_d \frac{tf(t_q, d)}{|d|} + \alpha_d \frac{\sum_{d \in D} tf(t_q, d)}{|d|} \right) \]
\[ + \lambda_{ow} \sum_{c_q \in ow(q)} \log \left( 1 - \alpha_d \frac{tf_{ow}(c_q, d)}{|d|} + \alpha_d \frac{\sum_{d \in D} tf_{ow}(c_q, d)}{|d|} \right) \]
\[ + \lambda_{uw} \sum_{c_q \in uw(q)} \log \left( 1 - \alpha_d \frac{tf_{uw}(c_q, d)}{|d|} + \alpha_d \frac{\sum_{d \in D} tf_{uw}(c_q, d)}{|d|} \right) \]  

(2.9)

where, \( ow(q) \) and \( uw(q) \) are the set of all contiguous n-grams (or phrases) and the set of all bags of terms that can be generated from query \( q \), respectively. \( tf_{ow} \) and \( tf_{uw} \) are the ordered-window and unordered-window operators from the Indri search engine [21]. Finally, \( \lambda_{ow} \) and \( \lambda_{uw} \) are the tunable parameters of the model.

### 2.5 Learning to rank

In Learning to Rank (LTR) systems, a query-document pair is represented by a vector of numerical features and a machine learning model is trained for mapping the feature vectors to a real-valued relevance score between the query and document, \( rel(q, d) \). Depending on the flavor of LTR, in addition to the feature vector, each query-document pair in the training data can be augmented with some relevance information.

![High-level abstraction diagram of LTR models.](image)

The input features for LTR models typically belong to one of three categories.

- **Query-independent or static features** (e.g., PageRank [4] or spam score of the document)
- **Query-dependent or dynamic features** (e.g., BM25 or TF-IDF)
- **Query-level features** (e.g., number of words in query)

According to Liu [22], Learning to Rank algorithms can be grouped into three approaches: the pointwise approach, the pairwise approach and the listwise approach.
2.5.1 Pointwise approach

In the pointwise approach, the input space contains a feature vector of each document. The output space contains the relevance degree of each single query-document pair. Algorithms that take this feature vector and predict the relevance degree of the document are here employed. Documents are ranked at the end according to their predicted score. It is important to note that the pointwise approach does not consider the inter-dependency between documents, and thus the position of a document in the final ranked list is not known at train time. Furthermore, the approach does not make use of the fact that some documents are actually associated with the same query. Considering that most evaluation measures for information retrieval are query level and position based, the pointwise approach has its limitations. It is assumed that the exact relevance degree of each document is what we are going to predict, although this may not be necessary since the target is to produce a ranked list of the documents.

The method proposed by Shashua and Levin [23] utilizes a number of parallel hyperplanes as a ranking model. Their method, referred to as OC SVM in this article, learns the parallel hyperplanes by the large margin principle. In one implementation, the method tries to maximize a fixed margin for all the adjacent relevance degrees, while the other implementation maximizes the sum of all margins.

2.5.2 Pairwise approach

The pairwise approach takes two documents, represented by its feature vectors, at a time, considering they are associated with the same query. Models are then trained to predict the relative order between the input pair. In many pairwise algorithms ranking is modeled as a binary classification problem, meaning that the objective of the trained model is to detect what document is more related to the query. This training procedure still has its constraints as it only considers the relative order between two documents. The position of the documents in the final ranked list can hardly be derived.

In 2000, Ranking SVM was introduced by Herbrich et al. [24]. Documents associated with the same query are grouped in the feature space. A linear function is then used to project the objects into vectors and sort the objects according to the vectors’ projections. Later, in 2005, RankNet [25] was originally developed by Burges et al. using neural nets, but the underlying model can be different and is not constrained to just neural nets. RankNet aimed to minimize the number of inversions in ranking, i.e., when lower rated results are rated above a higher rated result in a ranked list.

Burges et al. [26] found that during the RankNet training procedure, the costs are not needed, but only the gradients of the cost with respect to the model score. The gradient associated to each document theoretically points to the direction that the document should move on the reranked list. Further, they discovered that scaling the gradients by the change in nDCG (a metric used to evaluate models that will be covered in section 2.7), found by swapping each pair of documents, yielded good results. Their new proposed model, LambdaRank, had the core idea to use this new cost function for training a RankNet, which leads to speed and accuracy improvements.

This model was still improved by Wu et al. [27] that created LambdaMART, a combination of LambdaRank and Multiple Additive Regression Trees (MART). While MART uses gradient boosted decision
trees for prediction tasks, LambdaMART uses gradient boosted decision trees with a cost function derived from LambdaRank for solving a ranking task.

### 2.5.3 Listwise approach

The listwise approach differs from the pairwise for having a set of documents, in contrast with only two. In here, the output space while training is exactly the same as the output space of the task. In this regard, its theoretical analysis can have a more direct value to understanding the real ranking problem than the other approaches.

Although algorithms are evaluated with a ranking function (as it is presented in section 2.7) on the whole set of documents, cost functions are many times pointwise or pairwise to facilitate the learning process, as the documents are not indexed. In most cases, the pairwise cost function can be extended to a listwise approach, using more than two query-document pairs to calculate its value, as is the case of the previously described models LambdaRank [26] and LambdaMART [27]. Cost functions are also exhaustively used in deep learning and they will be explained in detail in Section 2.6.2.

### 2.6 Deep Learning

#### 2.6.1 Introduction

With the success of neural networks in the field of machine learning, many researchers tried to apply some of its techniques to the ad-hoc retrieval problem.

Modern deep learning provides a very powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector $x$ to an output vector $y$ can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples.

Deep feedforward networks are the basis of these models, that have the objective of approximating some function $f$. This type of network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters $\theta$ with the objective of getting a suitable approximation function, $f^*$.

These models are called feedforward because information flows through the function being evaluated from $x$, through the intermediate computations used to define $f$, and finally to the output $y$. Nonetheless, there are networks that have feedback connections in which outputs of the model are fed back into itself. These kind of models are called Recurrent Neural Networks. We refer the curious reader to the published work of Lipton et al. [28] for a better understanding of these particular methods.

Feedforward neural networks are called networks because they are typically represented by composing together many different functions, where each function is designated as a layer. The model is associated with a directed acyclic graph describing how these functions are composed together. The length of this chain structure is designated by depth, hence the ‘deep’ nomenclature. In Figure 2.4 a deep neural network is shown to exemplify how large these models can be. Training data points $(x, y)$ provide a way of constructing the function $f^*$ in such a way that $y \approx f^*(x)$. Although these training
examples specify what needs to be in the final (output) layer of the model, the same does not apply for other layers’ behavior. Therefore, these other layers are called hidden layers and the learning algorithm must decide how to use them to produce the desired output.

![GoogleNet architecture](image)

Figure 2.4: GoogleNet architecture, used in Image Recognition. Extracted from [29].

### 2.6.2 Training procedure

Deep learning algorithms learn their parameters $\theta$ (weight vectors and biases) by directly maximizing or minimizing an objective function. Most of the times, minimization is employed and the function can also be referred to as loss, cost or error, $L$. The loss function is therefore designed to be lower when $y \approx f^*(x)$, i.e., when the output of the model is closer to the ground truth label $y$.

Considering that the error function is a fairly complex composition of many non-convex functions contained throughout the neural network, finding its global minimum is a challenging task. Thus, an algorithm is applied that finds the gradient of a function at a particular value and then updates that value by moving in the direction of the negative of the gradient. This method is called gradient descent. More specifically, for a given learning rate $\eta$, the parameters $\theta$ of the network will be updated according to the following formula,

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta) \quad (2.10)$$

This process repeats until a local minimum is found, or the gradient sufficiently converges (i.e., becomes smaller than some threshold). In practice, computing the gradient over all the training samples is computationally expensive and stochastic gradient descent is used instead. The insight of stochastic gradient descent is that the gradient is an expectation and that it may be approximately estimated using only a small portion of the dataset at a time, called mini-batch. Moreover, the use of mini-batch also accelerates the convergence of the model [30] [31].

Since the step direction is determined by the gradient at a single point, gradient descent is considered a local method, being only able to find local minima.

In order to calculate the gradient $\nabla_\theta$ with respect to the parameters $\theta$ of the network, backpropagation is used. Backpropagation is an algorithm for supervised learning of artificial neural networks using gradient descent. The “backwards” part of the name stems from the fact that calculation of the gradient
proceeds backwards through the network, with the gradient of the final layer being calculated first and the gradient of the first layer being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer, using the chain rule of calculus.

The chain rule states that, for a given \( x \in \mathbb{R}^m, y \in \mathbb{R}^n, g : \mathbb{R}^m \rightarrow \mathbb{R}^n \) and \( f : \mathbb{R}^n \rightarrow \mathbb{R} \), if \( y = g(x) \) and \( z = f(y) \), then

\[
\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_j}
\] (2.11)

that can be translated to vector notation and rewritten as

\[
\nabla_x z = (\frac{\partial y}{\partial x})^\top \nabla_y z
\] (2.12)

where \( \frac{\partial y}{\partial x} \) is the \( n \times m \) Jacobian matrix of \( g \). These functions \( f \) and \( g \) can be seen as different layers in the neural network architecture and the backpropagation algorithm, thus, consists of performing such Jacobian-gradient products for each operation in the network.

### 2.6.3 Challenges in neural network optimization

Despite knowing what data is fed to a particular neural network, what operations are applied to it afterwards and how the model is trained to minimize a certain cost function, it is usually hard to analyze why the algorithm worked or why it did not. The lack of ability of neural networks for reason on an abstract level makes it difficult to optimize high-level cognitive systems, i.e., find the parameters \( \theta \) of the network that significantly reduce the loss. In this section, an overview of prominent challenges encountered while optimizing deep learning algorithms will be given.

Traditionally, machine learning methods avoid the difficulty of general optimization by carefully designing the objective function and constraints to ensure that the optimization problem is convex. However, that is not the case of deep learning, where the function that is being minimized is most of the times non-convex. Therefore, even if the algorithm finds a local minimum, it is not guaranteed to be a global minimum, as seen in Figure 2.5, and the optimization process can be stuck in this non-optimal solution. Thus, optimizers have been developed to avoid convergence to local minima, proposing new gradient descent formulations and/or having adaptive learning rates [32], that has a direct impact on how much
the values of $\theta$ change at each step (equation 2.10). Nonetheless, there are no guarantees that this
global minimum will be found. Other common difficulties include:

- **Flat regions** - common areas that represent both a local minimum for a sub-region and a local maximum for another.

- **Saddle points** - a point of a function which is stationary (partial derivatives equal to zero) but is neither a maximum nor a minimum value.

- **Misleading gradients** - local regions of the cost function do not correspond with its global structure.

- **Vanishing and exploding gradients** - while backpropagating the error, the gradient that is being calculated can either get too close to zero or grow exponentially, respectively.

![Figure 2.6: Example of a saddle point over a three-dimensional surface. Extracted from [16].](image)

Another key concept to bare in mind is that, most of the times, the metric used to evaluate a system’s performance is not the one that is being directly minimized, as it is typically intractable, and a surrogate loss function is used. Furthermore, the neural network is trying to model an unknown function that is being approximated by training on some data points, which tend to be noisy, as is the case when applied to ad-hoc retrieval, as explained in Section 2.1.3.

### 2.6.4 Basic concepts

During the following section, an explanation will be given about some of the most important building blocks that are going to be posteriorly referred throughout this thesis.

**Multilayer Perceptron**

Inspired on the real biological neural networks of the brain, Artificial Neural Networks (ANN) are a kind of feedforward networks (Section 2.6.1) that consist of interconnected units, also known as artificial neurons, that communicate with each other to process information. An ANN learns by first doing a feedforward sweep and, when it reaches the output, it checks the error between the prediction from the feedforward sweep and the actual output it should return. It then performs a backpropagation sweep, optimizing the weights and bias of each unit.
A Multilayer Perceptron (MLP) is a class of these ANNs and has been frequently used in machine learning due to its simplicity and ability to distinguish data that is non-linearly separable. An MLP uses hidden layers of many single perceptrons with non-linear activation functions, such as the sigmoid, $f(x) = (1 + e^{-x})^{-1}$ or the ReLU, $f(x) = \max(0, x)$, among others. In an MLP, the output of a perceptron is given by,

$$y(x) = f(\sum_i w_i \times x_i + b_i)$$

(2.13)

where $f$ is the non-linear activation function, $w$ are the weights and $b$ the biases of the neurons. The MLP consists of an input layer, one or more hidden layers and one output layer. Each node is connected to all nodes in the previous layer. Networks with only one hidden layer are typically designated as shallow in contrast with deep networks that have more than one. In practice, deeper networks may require exponentially fewer neurons than shallower networks to express the same function. A multilayer perceptron (with non-linear activation) can theoretically represent any function provided it has sufficient capacity, i.e., is sufficiently deep and has enough perceptrons to represent that function.

![Figure 2.7: Sample architecture of a Multilayer Perceptron with several hidden layers and units.](image)

**Convolutional Neural Networks**

Being modified Artificial Neural Networks, used mostly in computer vision, Convolutional Neural Networks (CNNs) are able to directly classify raw pixels into high-level concepts without actually using hand-engineered feature extraction algorithms, which results in less human interaction and better performance. CNNs have hidden layers to extract features from the raw input signals without any pre-processing. They consist of at least two main altering kinds of layers: convolutional layers and pooling layers. For the most part, these layers generate features that will serve as input of other, succeeding, hidden layers (like the MLP seen previously) providing a much more accurate model for a specific task.
Each convolutional layer performs a discrete 2D spatial convolution operation with a filter kernel, adds the biases and applies a non-linear transfer function $f$. The mathematical expression of the convolution between a three-dimensional input $I$ and the kernel $H$, with dimension $m \times n \times d$, can be visualized in equation 2.14. The convoluted output is constructed by overlapping the kernel on top of the input in all possible ways, and recording the sum of element-wise products between both.

$$I \ast H = f(b + \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{d} H_{ijk} \cdot I_{x+i-1,y+j-1,k})$$ (2.14)

Kernels may be learned from a given training dataset via gradient descent, exactly as the weights of an MLP. In fact, an MLP is perfectly capable of reproducing a convolutional layer, but it would require a lot more training time (and data) to learn to approximate that mode of operation. It is also important to mention the concept of stride. In equation 2.14, the kernel was convoluted with the input at every possible spatial location, which corresponds to a stride $s$ of 1. However, if $s > 1$, every movement of the kernel skips $s - 1$ matrix entrances. Furthermore, convolution by any kernel larger than $1 \times 1$ will decrease the output size. In this case, zero padding is usually applied to keep the original dimensions.

As already hinted, convolutions are not typically meant to be the sole operation in a CNN but rather used to extract useful features of an input prior to downsampling it sufficiently to be manageable by an MLP. A very popular approach to downsampling is a pooling layer, that uses some statistic that summarizes neurons from a small spatial neighborhood and aggregates them into a single value. Pooling is used to achieve invariance to image transformations, more compact representations, and better robustness to noise and clutter. In general terms, the objective of pooling is to transform the joint feature representation into a new, more usable one that preserves important information while discarding irrelevant details. The pooling operation is typically a sum, an average, a max, or more rarely some other commutative (i.e., independent of the order of the contributing features) combination rule. The type of pooling performed is commonly referred with its pooling operation in the name (e.g., max pooling).

**Word2Vec Embeddings**

As previously mentioned in Section 2.3, embeddings are the most conventional way of modeling word interactions by representing terms in fixed size vectors. This section will dig deeper into one of the most popular word embedding models: Word2Vec [33] [34].

Despite of being a key building block when applied in deep learning, Word2Vec is not considered to be part of it because its architecture is neither deep nor uses non-linearities. Mikolov et al. [34] proposed two architectures for learning word embeddings that are computationally less expensive than previous language models [35] and enable them to take additional context into account. One of the methodologies, called skip-gram [33] [34], takes an input word and attempts to estimate the probability of other words appearing close to that word. The other approach, Continuous Bag of Words (CBOW) [34], does the opposite - it takes some context words as input and tries to find the single word that has the highest probability of fitting that context.

The skip-gram approach tries to maximize the objective function $J_\theta$, that consists on a sum of log
probabilities of the surrounding $C$ words to the left and to the right of the target word $w_t$, as seen in equation 2.15.

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-C \leq j \leq C, j \neq 0} \log p(w_{t+j} \mid w_t)$$ (2.15)

where $C$ represents the window dimension of words around the target word $w_t$ at each time step $t$.

The input of the skip-gram model is a single word and the output is the words around it within the context window size. All of these terms are one-hot encoded and the training examples are created from plain text. A representation of the model is given in Figure 2.8, where $x$ is the one-hot encoded vector corresponding to the input word in the training instance and $(y_1, ..., y_n)$ are the one-hot encoded vectors corresponding to the output words in the training instance. The matrix $W$ is the weight matrix between the input layer and hidden layer with one row for each word in the vocabulary of size $V$ and $N$ is a hyperparameter, that is typically set to 300 in Word2Vec. $W$ is what the model is trying to ultimately learn, since it contains vector encodings of all of the words in the vocabulary. Each output word vector also has an associated $N \times V$ output matrix $W'$. Between both matrices there is a hidden layer $h$ with size $N$.

![Figure 2.8: The skip-gram model. Extracted from [36].](image)

As described in Section 2.6.4, the input to a unit in the hidden layer is simply the weighted sum of its inputs. Since the input vector $x$ is one-hot encoded, the weights coming from the nonzero element will be the only ones contributing to the hidden layer. Therefore, for the input $x$ with $x_i = 1$ and $x_{i'} = 0$, the outputs of the hidden layer will be equivalent to the $i^{th}$ row of $W$.

CBOW, as the name suggests, uses continuous representations whose order of the words is of no importance and tries to maximize $J_\theta$,

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \log p(w_t \mid w_{t-C}, ..., w_{t-1}, w_{t+1}, ..., w_{t+C})$$ (2.16)
Looking like the skip-gram model but reversed, the inputs of the CBOW model are the one-hot encoded context words \((x_1, ..., x_C)\) while the output is a word \(y\), also one-hot encoded.

Traditionally, predictive models are trained using the maximum likelihood principle to maximize the probability of the next words given the previous words in terms of a softmax function (equation 2.18) over all the vocabulary words, whose result is further used to backpropagate the error and update the weights. However, this training procedure is quite computationally expensive given a large vocabulary set, because all words need to be normalized at each training step. Therefore, two main approaches are used in Word2Vec for reducing the computation overhead of softmax:

- Hierarchical softmax [37] - To approximate the conditional log-likelihood a model seeks to maximize, this method uses a Huffman tree that represents an efficient way of computing an approximation of the softmax function.

- Noise Constrative Estimation (NCE) [38] - Also known as negative sampling, this method reduces computation by sampling just some negative instances along with the target word instead of sampling the whole vocabulary. This way, NCE ignores most of the zeros in the one-hot label word vector, and only propagates and updates the weights for the target and a few negative classes which were randomly sampled, minimizing the log-likelihood of the negative instances and maximizing the log-likelihood of the target word.

For the more experienced reader, we refer to the work of Rong [36] for a more detailed explanation on how Word2Vec embeddings are trained.

**Transfer learning**

The use of pre-trained word embeddings is recurrent in the area of Natural Language Processing and can be classified as a specific application of a common design methodology in deep learning, transfer
The general idea behind transfer learning is to use knowledge learned from tasks for which a lot of labeled data is available in settings where only little labeled data is available. Creating labeled data is expensive, so optimally leveraging existing datasets is key. Transfer learning breaks down the traditional supervised learning paradigm and solves the lack of training samples by re-using parts of another model, that was trained with more data.

Using Word2Vec embeddings in a new network architecture is a clear example of transfer learning. Word representations are learned a priori, in an unsupervised fashion, by training either a CBOW or skip-gram model with another dataset [39], consisting of billions of words. A model can then use these learned representations as lookup tables in its embedding layers and, considering the training data available, fine-tune them along with the other parameters of the network by employing stochastic gradient descent (Section 2.6.2).

Word2Vec pre-trained embeddings allows us to build some powerful models without the need for a large labeled corpora, since word representations were learned previously. Nonetheless, it is important to note that, if used as a non-trainable layer, the model will not adapt these embeddings to the specific task at hand.

**Attention Mechanism**

Motivated by the recent developments in areas of NLP, such as Natural Language Inference [40] and Machine Translation [41], and even when applied to images [42], the use of attention mechanisms will be explored in this thesis.

The most common attention model is a method that takes \( n \) arguments \((y_i, i \in [1, ..., n])\) of the same size and a context \( c \). It returns a vector \( z \) which has the same dimension as \( y_i \) and represents a more compact representation of the inputs, with focus on information provided by \( c \). More formally, it returns a weighted arithmetic mean of the inputs \( y \), and the weights are chosen according to the relevance of each \( y_i \), given the context \( c \).
The network computes $m_i$ by multiplying $y_i$ and $c$ with their respective trainable weight vectors, $W_{ym}$ and $W_{cm}$. The results of both operations are then summed and a non-linear function $\tanh$ is then applied. It is relevant to notice that the same vector of weights is always applied to the inputs $y_i$ and $c$ and each $m_i$ is computed independently.

\[ m_i = \tanh(W_{ym} \cdot y_i + W_{cm} \cdot c) \] (2.17)

Subsequently, the $m_i$ vectors are passed to probability vectors $s_i$ by using a softmax layer (2.18). These $s_i$ vectors now represent the weight that its input $y_i$ will have on the weighted average. $z$ is finally constructed by multiplying element-wise each $s_i$ with the corresponding input $y_i$ and summing all the end as illustrated in Figure 2.10.

\[ \text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \] (2.18)

The attention mechanism previously described is denominated as soft attention because it is fully differentiable and deterministic where the gradients are propagated through the model. On the other hand we have hard attention (Figure 2.11), which is a stochastic process. Instead of performing a weighted average, the system samples a hidden state $y_i$ with probabilities $s_i$. When backpropagating the error, the gradient has to be estimated via Monte Carlo sampling in this approach.
Neural IR refers to the employment of deep learning algorithms to retrieval tasks. Unlike LTR approaches, previously seen in Section 2.5, that take as inputs a set of handcrafted features, these models typically accept pure query and document texts, making them less dependent on manual feature engineering and better at automatically detecting relevance signals that would not be captured otherwise.

A search query may usually contain a few terms, while the document length can range vastly. Neural IR models use vector representations of text, and usually contain a large number of parameters that need to be tuned. Learning suitable representations of text demands large-scale datasets for training. Therefore, unlike traditional IR models, these neural approaches tend to improve their performance with more training data.

Text representations, usually embeddings as seen in Section 2.3, can be learned in an unsupervised or supervised manner. The former approach uses IR data, such as query-document pairs labeled with a graded relevance score, to learn a representation that is optimized end-to-end for the task at hand. When sufficient training data is not available, the unsupervised approach can be applied to learn a representation using just the queries and/or documents. In this case, different unsupervised learning setups may lead to different vector representations, that differ in the notion of similarity that they capture between represented items. When applying such representations, the choice of unsupervised learning setup should be carefully considered, to yield a notion of text similarity that is suitable for the target task.

Deep learning applied to ad-hoc retrieval provides a new way of thinking about the problem as a general text matching task, i.e., query matches document. According to Guo et al. [43], these approaches can be mainly categorized as representation focused models and interaction focused models (Figure 2.12). The representation focused models, commonly referred to as semantic matching models, try to embed both queries and documents in a low-dimensional space with a neural network, and then conduct matching between the two vectors. Interaction focused models first build the local interactions between
two texts and use deep learning to learn the more complicated interaction patterns for matching. This type of algorithms can be also referred to as relevance matching models.

Salakhutdinov and Hinton [44] introduced one of the first deep neural models for ad-hoc retrieval, the Semantic Hashing model. The algorithm is a deep auto-encoder, trained on unlabeled document corpus, that represents words in one-hot vectors and uses binary hidden units to encrypt those documents so that they can be posteriorly quickly retrieved using a hash function. The model is first pre-trained layer-by-layer and then trained further end-to-end for additional tuning. Given a search query, a corresponding hash is generated and the relevant candidate documents that match the same hash vector are retrieved. A standard IR model (e.g., BM25 addressed in 2.4.1) can then be employed to rank between the selected documents.

More recently, Huang et al. [45] applied siamese networks to ad-hoc retrieval. Their Deep Semantic Similarity Model (DSSM) trains on query and document title pairs where both the pieces of texts are represented as bags-of-character-trigraphs. The DSSM architecture consists of two deep models - one for the query and the other for the document - with all fully-connected layers and cosine distance as the choice of similarity function in the middle. This architecture was later improved by Shen et al. [46] by making use of convolutional layers.

This use of Convolutional Neural Networks (CNNs) encouraged posterior interaction-focused models, such as MatchPyramid [47] introduced by Pang et al., that applied convolutions over a query-document matching matrix, although with still worse results than traditional IR approaches. Still, Guo et al. developed the Deep Relevance Matching Model (DRMM) [43] that was able to significantly outperform them by modeling query-document interactions using matching histograms. Mitra et al. [48] introduced a state-of-the-art model that makes use of both local interactions and distributed representations, arguing that a combination of the two approaches is preferred.

Subsequently, Xiong et al. [49] and Hui et al. [50] published results on interaction-focused models that would take as inputs only the similarity matrices between the query and document terms, \( t_q \) and \( t_d \), respectively. While the former method applied RBF kernels to that matrix, the latter used multiple convolutions with different kernel sizes along with pooling layers. This last model, the Position-Aware Convolutional-Recurrence Relevance Matching model (PACRR), will be analyzed in detail in chapter 3.
2.7 Evaluation metrics

In the field of Information Retrieval, metrics considering only the top-ranked documents for each query are more popular, since users of retrieval systems tend to pay attention mostly to the first results. Therefore, evaluations are conducted to focus on rank-based comparisons of the retrieved results with ideal best documents, determined by manual judgments or implicit feedback from user behavior data. These metrics are typically computed at a rank position, \( @K \), and then averaged over all queries in the test set. Next, a few popular metrics used in IR evaluations are described.

Mean Reciprocal Rank

Mean reciprocal rank (MRR) is computed over binary relevance judgments. It is given as the multiplicative inverse rank of the first relevant document, \( \text{rank}(q) \), for each query \( q \), averaged over all queries \( Q \).

\[
MRR = \frac{1}{Q} \sum_{q} \frac{1}{\text{rank}(q)}
\]  

(2.19)

Normalized Discounted Cumulative Gain

There are few different variants of the discounted cumulative gain (DCG) [51] metric which can be used when graded relevance judgments are available for a query \( q \). A popular implementation of this metric is as follows.

\[
\text{DCG}@K = \frac{1}{Q} \sum_{q} \sum_{k=1}^{K} \frac{2^{\text{rel}(q,d_k)} - 1}{\log_2(k + 1)}
\]  

(2.20)

Where \( \text{rel}(q,d_k) \) is the graded relevance between a document at position \( k \) with a particular query \( q \).

The ideal DCG (IDCG) is computed the same way but by assuming an ideal rank order for the documents up to rank \( K \). The normalized DCG (nDCG) is then given by,

\[
\text{nDCG}@K = \frac{\text{DCG}@K}{\text{IDCG}@K}
\]  

(2.21)

Expected Reciprocal Rank

Another metric for non-binary relevance judgments, the expected reciprocal rank (ERR) [52] introduced by Chapelle et al., is based on the cascade model of search which assumes a user scans through ranked search results in order and, for each document, evaluates whether the document satisfies the query, and if it does, stops the search. Therefore, this metric presupposes that a single document can satisfy a search.

Stopping at rank \( k \) involves being satisfied with document \( k \), the probability of which is \( p(q,d_k) \). It also involves not having been content with any of the previous documents with lower rank, the probability of which is \( \prod_{i=1}^{k-1} (1 - p(q,d_i)) \). This is all multiplied by \( 1/k \), i.e., being satisfied with a document at rank \( k \)
adds at most $1/k$ to the final ERR score, as you can see in equation 2.22. It is important to refer that these probabilities $p(q, d_k)$ are assigned beforehand to each query-document pair, proportionally to its ground truth relevance label.

$$ERR@K = \sum_{k} \frac{1}{k} p(q, d_k) \prod_{i=1}^{k-1} (1 - p(q, d_i))$$

(2.22)
Chapter 3

Relevance Matching for Adhoc Retrieval using Deep Neural Networks

Ad-hoc retrieval is a topic that still has a lot to be explored. In this thesis, we chose to follow a supervised deep learning approach, because it is a fast growing research area and it shows great potential from what has been seen during the last years of related work in this area. The objective of the Artificial Neural Network will be to attribute a relevance score, \( rel(q,d) \), to any query-document pair so that the top documents for each query can be obtained.

In this chapter, the baseline model is firstly presented and an extended explanation will be given about how query and document texts are converted into a single relevance score. Subsequently, different information bottlenecks and strong assumptions of the model will be discussed, along with proposed modifications to it, that are implemented in this work.

Throughout this chapter, we admit that the reader is familiar with basic supervised machine learning concepts, as well as other concepts that were previously explained in Section 2.6.

3.1 Baseline Framework

The baseline framework adopted in this thesis is based on the PACRR model [50] introduced by Hui et al.. An overview of the architecture, excluding the pre-processing step, can be seen in Figure 3.1. Being a relevance matching model, the core concept of this architecture is to construct first a word similarity matrix between the query and the document words and use convolutional neural networks to account for "soft" pattern matches involving one or more adjacent words (n-grams). These signals are then non-linearly combined to produce a single query-document relevance score. The result is a non-linear function mapping a query document pair to a relevance scalar value. Following the usual neural network approach, this model can be trained end-to-end using gradient descent techniques.
3.1.1 Similarity matrix and query IDF vector

As mentioned previously, each query $q$ and document $d$ are formerly converted into a query-document similarity matrix $\text{sim}_{|q| \times |d|}$. Stopword removal is first applied to both document and query, i.e., removing
common English words like ‘and’, ‘or’ and ‘the’. Thenceforth, query and document terms ($t_q$ and $t_d$) are converted into 300-dimensional vectors using the pre-trained Word2Vec CBOW embeddings and the cosine similarity is computed between every $t_q$ and $t_d$, measuring the cosine of the angle $\theta$ between both vectors, as detailed in equation 3.1. In principle, word embeddings can be trained using the available document query data as any other parameters of the model. In practice, this demands large amounts of data to prevent overfitting. For this reason, pre-trained embeddings are utilized for each word and fixed during training. Pre-trained embeddings can be seen as a simple form of transfer learning where word representations are trained from another task. It is also important to notice that, by using pre-trained embeddings, there can be unknown words that were not present in the vocabulary when those term representations were generated. These words are designated by out-of-vocabulary (OOV) words. The original authors [50] solve this problem by re-training the word embeddings \textit{a priori} with the whole text corpus. Negative sampling is used for model training.

$$\cos(\theta) = \frac{A \cdot B}{||A|| ||B||} \tag{3.1}$$

In order to apply CNNs, the similarity matrices need to have the same dimensionality. Given that the lengths of queries and documents vary, the similarity matrices $sim_{|q| \times |d|}$ are transformed into $sim_{l_q \times l_d}$ matrices with uniformed $l_q$ rows and $l_d$ columns. This is achieved by selecting the first $l_q$ or $l_d$ terms for query and document, respectively, when either $|q|$ or $|d|$ exceed the maximum fixed dimension. If the actual dimension of the query or document is lower than the desired, then zero padding is applied.

Additionally, a vector containing the IDF score of each query term $t_q$ is also computed, $IDF_{l_q \times 1}$. This vector has length $l_q$ and will be passed directly to the last layers of the model as we can see in Figure 3.1. This IDF array is computed in an unsupervised fashion, so that all documents and queries are gathered previously and the equation 2.5 is applied for each query term.

### 3.1.2 Convolution and Pooling Layers

Having the similarity matrices $sim_{l_q \times l_d}$, they are reshaped to have a 3D shape ($l_q \times l_d \times 1$) and two two-dimensional convolutional layers are applied to them with different kernel sizes, $2 \times 2$ and $3 \times 3$, corresponding to bi-gram and tri-gram matching, respectively. The original matrix accounts for unigram matching.

Each convolutional layer applies $n_f$ different filters to its input, where $n_f$ is a hyperparameter. A stride of size $(1, 1)$ is used, meaning that the convolutional kernel advances one step at a time in both the query and document dimensions. At the end of this step we end up with two 3D matrices, $C^2_{l_q \times l_d \times n_f}$ and $C^3_{l_q \times l_d \times n_f}$, as well as the initial 2D similarity matrix $sim_{l_q \times l_d}$.

In order to capture the strongest similarity signals for each query term, max pooling is performed over the filter dimension $n_f$ to keep only the strongest signal from the $n_f$ different filters, assuming that there only exists one particular true matching pattern in a given $n \times n$ window, as is commonly done in computer vision. This pooling layer transforms the 3D $C^2_{l_q \times l_d \times n_f}$ and $C^3_{l_q \times l_d \times n_f}$ matrices into 2D, $C^2_{l_q \times l_d}$ and $C^3_{l_q \times l_d}$, respectively, leaving them with the same size as the original similarity matrix, $l_q \times l_d$.  

Our data at this point is still particularly vast, having $3 \times l_d$ signals for each query term $t_q$. To suppress this curse of dimensionality, the model captures the top $k$ matching signals, for each query term and for each of the 3 matrices, by applying k-max pooling layers. This layer is implemented in such a way that, for each row, the higher $k$ values are retained, keeping the same number of matching signals for each query term $t_q$. An illustration of the formerly described operation can be found in Figure 3.2.

![Figure 3.2: Example of a k-max pooling operation.](image)

$k$ is another hyperparameter that needs to be tuned on validation data. After concatenating, we are left with a 2D matrix $P_{l_q \times 3k}$ that contains information from both unigram, bi-gram and tri-gram matches.

### 3.1.3 Producing the final score

After extracting the most important query term similarity signals, there is the need to produce a single relevance score $\text{rel}(q,d)$. This is accomplished with the use of a multilayer perceptron with non-linear activations, that can theoretically model any function.

Previous to inputing the matching signals to the MLP, the query IDF vector, $IDF_{l_q \times 1}$, is passed through a softmax layer for normalization, such that a vector of probabilities (that all sum to 1) is joined with the matrix $P$ thereafter. The authors of this baseline architecture argue that information about the IDF of each query term acts as a regularizer for the final score, giving more or less importance to some query terms.

The matrix $P$ along with $IDF_{l_q \times 1}$ are then flattened into a single vector of size $l_q \times (3k + 1)$ and passed to a feed-forward network with 2 hidden layers (similar to Figure 2.7) and a final single neuron that produces the final relevance score, $\text{rel}(q,d)$.

### 3.1.4 Training Phase

To train a neural network in a supervised manner, it is necessary to repetitively feed it train data pairs $(x,y)$ in order to minimize a certain loss function. In our case, since the final objective is to rerank documents, we are not interested in predicting a $\hat{y}$ label, as that does not allow proper ranking. The neural model needs to distinguish relevant documents from less relevant (but likely not completely non-relevant).

The most common approach for neural ad-hoc retrieval is to train pairwise with a set of documents $D^+$ and a set of documents $D^-$ for the same query $q$, where $D^+$ are more relevant than $D^-$. This allows the model to distinguish, and therefore to rank, relevances between documents. In this architecture only
a single relevant document $d^+$ is used and the number of $D^-$ documents is a parameter that was fixed to 6. So this means that, for each training sample the model described in Figure 3.1 is actually repeated seven times, with a loss function gathering all the outputs as will be seen in equation 3.2.

Figure 3.3: Loss computation during training phase.

**Loss function**

The optimal model parameters are those which best approximate the dataset distribution. Formulating this setting as an optimization problem is possible through a so called cost/loss function, which measures the difference between the reference distribution, $y$, and the predicted distribution $p_\theta(\hat{y}|x)$ as a function of the model parameters, $\theta$.

Considering we have multiple final scores as inputs to the loss function (one from $d^+$ and six from $D^-$), some kind of prior normalization needs to be done. Hence, the relevance score of $d^+$ is normalized and fed into a binary cross entropy function (Figure 3.3).

$$L(q, d^+, D^-) = -\log \frac{e^{rel(q,d^+)} e^{rel(q,d^-)}}{\sum_{d^- \in D^-} e^{rel(q,d^-)}}$$ (3.2)

Stochastic gradient descent is enforced using a mini batch of 32 samples. Gradient descent algorithms are iterative and work on the principle of changing the parameters in the direction of minimizing the total loss function. By minimizing this loss function, the model learns to increase the probability of $d^+$ and thus $rel(q,d^+)$.

Adam optimizer [53] is used for faster convergence. According to a recent survey, executed by Ruder [54], Adam is a gradient descent formulation which has an excellent convergence rate in comparison to other algorithms as it seeks to merge two: AdaGrad [55], which works well with sparse gradients and RMSProp [56], which works well in on-line and non-stationary settings.

To construct each sample of the batch, for each query $q$, a document $d^+$ is randomly sampled with rank $x$, as long as it has any $D^-$ documents with rank $x-1$. This sampling procedure allows the model to better differentiate documents with levels of relevance that are close to each other. The relevance group $x$ is chosen with probability proportional to the number of documents in the group within the training set,
so that the final training corpus keeps a distribution of labels similar to the initial.

3.2 Proposed changes

The previously described framework has some strong assumptions and simplifications that we will explore in this thesis. In this section, we will go through some of these problems and suggest a modification, always with the objective of enhancing a particular building block of the original model that may cause an information bottleneck, i.e., loss or disregard of relevant data.

It is important to note from the beginning that the model depicted in Figure 3.1 has most of its learnable parameters in the Multilayer Perceptron. This is because the two-dimensional convolutions are applied to a matrix that is converted from 2D to 3D, which makes the filters learned by the CNNs relatively small, with only 1 channel dimension. In Deep Learning, less data means more difficulty to learn more weights and biases. All things considered, the proposed changes were also designed having this into account.

3.2.1 Lack of regularization

Deep neural networks contain multiple non-linear hidden layers that allow them to learn very complicated relationships between their inputs and outputs. However, these functions are non-convex and therefore the gradient descent algorithm does not give us a formal guarantee of convergence to the optimal solution. After a few epochs of training any neural network, the loss function of the training set may keep diminishing, while the loss on the validation set starts to get worse. This phenomena is called overfitting, and it means that the network is learning specific information related to the training set, instead of generalizing to unseen data. For this reason it is important to aid the optimization process by imposing regularization terms. These terms help reducing the generalization error, the gap between the error observed while adjusting the parameters to the training set and the validation set.

The original architecture pictured in Figure 3.1 has no kind of regularization mechanisms applied, making the neural network subjective to overfitting. In order to prevent this and make the results more consistent, many methods have been developed. One of them is called cross-validation and Hui et al. [57] report that they used it in their second implementation of this model. Cross-validation is a powerful preventative measure against overfitting. In standard $k$-fold cross-validation the training data is partitioned into $k$ subsets, called folds and the test set is still held out for final evaluation. The model is then iteratively trained on $k - 1$ folds while the remaining fold is used for validation. Having $k$ models at the end of training, the average predictions on the test set serve as final result.

However, due to cross-validation being computationally expensive (as it trains more than one model) it is usually avoided in deep learning and other alternatives to prevent overfitting have been explored, that are now common practice in the field. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization or a simple mechanism called dropout [58] introduced by Srivastava et al.
The key idea of dropout is to randomly drop units from the neural network during training. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. This prevents the weights from co-adapting too much. This technique is implemented at train time by generating random binary streams with a ‘do not keep’ probability \( p \) and multiplying it by a layer output. The outputs that are not dropped out are then scaled by dividing them by the ‘keep’ probability \( 1 - p \). This ensures that for any hidden unit, the expected output is the same as the actual output at test time. At test time, the complete network is used, i.e., \( p \) is set to 0.

![Diagrams](image)

(a) Standard Neural Net.  
(b) After applying dropout.

Figure 3.4: Effects of dropout on the network topology. Extracted from [58].

In this way, we present a way of regularizing the architecture by adding dropout layers after different blocks with trainable weights, that is, following the CNNs (after the max pooling operation) and the hidden layers of the MLP.

### 3.2.2 Parameterless pooling operations

Reducing the dimensionality of 3D and 2D matrices is a fundamental part of the baseline model represented in Figure 3.1. Although using parameterless pooling operations solves this issue without the introduction of new trainable variables, they can also be detrimental to the model's performance since the network is not considering a lot of information that was removed by an aggressive pooling function like maximum.

Considering this a limitation, in this section we will propose changes to the two pooling layers used in the baseline model, as we can see in Figure 3.5.

**Dimensionality reduction of 3D matrices**

Even though max pooling has been a standard operation when applied to images, in fields such as image recognition [59] [60], we argue that a non-linear dimensionality reduction might be more suitable when dealing with similarity matrices. Without increasing the learnable parameters of the network too
much, and motivated by its success in computer vision \cite{29} \cite{61} in decreasing the channels dimension (i.e. the third dimension of the input), a $1 \times 1$ convolution is proposed to replace the max pooling layer.

Applied like a regular convolution layer, $1 \times 1$ convolution is a ‘feature pooling’ technique, that performs transformations in the filter space. As in most of applications, a non-linear action layer is introduced thereafter, in this case a ReLU. The intuition behind the replacement of this pooling layer is that, without modifying the architecture too much, the neural model will be able to learn a more suitable function to reduce 3D matrices into 2D without imposing the max operation as it was done previously.

**Dimensionality reduction of 2D matrices**

Intuitively, the application of k-max pooling layers seems like a necessary step that allows us to employ a parameterless way of reducing the matrices size to something that is feasible for an MLP to handle. Notwithstanding, choosing only the top $k$ entries of each row is a strong assumption that might be an impediment for the neural network to learn a better representation of the problem, and consequently achieving improving its results.

Inspired by the soft attention mechanism and its recent success in other areas, as described in 2.6.4, we propose an architecture that allows the reduction of each row of the similarity matrices, with size $l_d$, into a single score. This score is obtained with the following formula:

$$z(x) = \text{softmax}(v \cdot \tanh(E_q W_q + E_d W_d)) \cdot x$$

(3.3)
This mechanism has three variables that are going to be learned while training: the vector \( v \) with dimension \((1, \sigma)\) and matrices \( W_q \) and \( W_d \), both with size \((300, \sigma)\), where \( \sigma \) is a hyperparameter. The width 300 of the matrices is equal to the length of the Word2Vec embeddings used.

With the final objective of reducing the row’s size from \( l_q \) to 1, the proposed attention mechanism takes as inputs a row of the similarity matrices \( x \), the embeddings of the document terms \( E_d \) and the query term embedding \( E_q \) associated with the given row \( x \), as each row contains the interactions of a query word with all documents words. Note that, per matrix, this layer is applied \( l_q \) times, one for each row.

![Figure 3.6: Using an attention mechanism as an alternative to k-max pooling.](image)

The intuition behind this attention is to perform a weighted average of the whole row, so that the information of it can be summarized into a single score. This is accomplished by first using the embeddings to learn a more suitable representation of the query term \( E_q, W_q \) and the document terms \( E_d W_d \). Those representations are then joined, by summing element-wise the vector \( E_q, W_q \) with all the rows of \( E_d W_d \), and passed through an activation function, \( \text{tanh} \). At this point, the output of the non-linearity has shape \((\sigma, l_d)\) and the vector \( v \) will be responsible to learn a linear transformation to reduce it to the same length of \( x, l_d \). \( \text{softmax} \) is then applied to normalize this final vector and the final score is obtained by calculating the dot product with the row \( x \) in case. An overview of how the attention is performed row-wise is illustrated in Figure 3.6.

Instead of having \( k \) scores per row, as with the former k-max pooling, in this case we have only 1. This number can however be enlarged by using more attention layers, along with other learnable variables \( v, W_q \) and \( W_d \). Also worth noting that, although this attention mechanism gives the model more freedom while selecting matching scores for each matrix row, more data is needed to train all the new introduced parameters, since the previous k-max pooling was parameterless.
3.2.3 Static similarity matrix

In this architecture we consider the input $sim_{l_q \times l_d}$ to be the biggest information bottleneck, since documents and queries are only represented by local interactions and other semantic information about them is lost in the process. Furthermore, the use of only the first $l_d$ words of the document is definitely another loss of information that is never considered again by the model. Hui et al. [50] explored different methods to select these $l_d$ document terms, but with no better results. Therefore, in this work we abstain from trying other approaches and stick with the first $l_d$ document words.

For all the reasons described above, we argue that a considerable improvement to the model’s performance might be achieved by altering the way query and document terms are represented. In addition to modifying the similarity matrix, the influence of the query IDF vector, $IDF_{l_q \times 1}$, is analyzed as well in our experiments.

In this thesis two methods are experimented to extract more information from the text: 1D convolutions and a soft attention mechanism.

1D Convolutions

Not to confuse with $1 \times 1$ convolutions (Section 3.2.2), one-dimensional convolutions act on 2D inputs. As previously mentioned in Section 3.1.2, the similarity matrix is firstly transformed to have a 3D shape and 2D convolutions are then applied to it. 1D convolutions act in a similar way, with a one-dimensional kernel instead. Commonly used in Natural Language Processing, it is normal to employ them after an embedding layer, in order to model word interaction within a context defined by the convolution window.
We argue that, even though 2D convolutions are already applied to detect bi-grams and tri-grams matching patterns between query and document, the similarity matrix, which is the basis of the model, consists of only unigram matches between words. Imagining we have two expressions: ‘feeling blue’ and ‘being sad’. These two expressions have exactly the same meaning. Yet, if we calculated the cosine similarity of every word pair combination, we would probably get a high value for ‘feeling’ and ‘being’ but a low one for ‘sad’ and ‘blue’. In this case, the cosine similarity matrix would not be able to properly detect that the two expressions are identical.

Therefore, we propose to implement an extra similarity matrix that will be constructed the same way as the original one, i.e., by taking the cosine distance of every combination of word vectors from the inputs. However, beforehand, both the document and query embeddings will be passed by a 1D convolution layer that will theoretically learn how to represent more than one word in one single vector. The number of words that it represents depends on the kernel size used. An overview of how this extra matrix will be made is depicted in Figure 3.8. In the example given above, by using this matrix constructed with cosine similarity of 1D convolutions, the network can learn how to properly represent ‘feeling blue’ and ‘being sad’ in two separate single vectors with a low cosine angle between, in their vector space.

![Figure 3.8: Construction overview of the proposed extra similarity matrix using 1D convolutions.](image)

The extra matrix will then be propagated through the network like the unigram matrix, i.e., not passing through the two-dimensional convolution and max pooling layers and contributing to the $P$ matrix after k-max pooling is applied to it (Figure 3.1).

**Soft attention mechanism**

Inspired by the soft attention mechanism previously explained in Section 2.6.4, a new method is proposed to transform what is passed to the CNNs. With the intention of increasing the local interaction information between every query term $t_q$ and $t_d$, in comparison with the previously used cosine similarity, the matrix will now be three-dimensional, with size $l_q \times l_d \times n_c$. To attain this objective, another attention-like mechanism is employed. The word embeddings of both queries $E_q$ and documents $E_d$ are passed
by a hidden layer, with $n_c$ perceptrons (equation 2.13), that first reduces their dimension by applying a linear projection, without using the non-linear activation function $f$. The new projected embeddings, $E_{qp}$ and $E_{dp}$, are in this way obtained and combined to produce the final 3D matrix $\text{sim}_{l_q \times l_d \times n_c}$, so that, given arbitrary positions $i$, $j$ and $k$,

$$\text{sim}_{l_q \times l_d \times n_c}(i, j, k) = E_{qp}(i, j) + E_{dp}(k, j)$$

(3.4)

where $E_{qp}$ and $E_{dp}$ have sizes $(l_q, n_c)$ and $(l_d, n_c)$, correspondingly.

![Figure 3.9: Construction overview of the proposed 3D similarity matrix.](image)

This new input matrix does not change the rest of the architecture and allows the CNNs to develop three dimensional filters and take into account more information for every $t_q$ and $t_d$ combination. Additionally, the use of cosine similarity is eradicated and the network will theoretically learn a more suitable query-document relationship function for the ad-hoc retrieval task. Once again, as in Section 3.2.2, this new richer representation needs more training data to learn the new introduced parameters, both in the hidden layer with length $n_c$ that projects the embeddings, and the CNN filters, that increased their channel size.

### 3.2.4 Loss function

Section 3.1.4 explained the way a binary cross entropy is used in the baseline model as a cost function. In order to minimize this loss, the neural network will update its parameters according to the gradient of this function. Therefore, the way the loss function is constructed is ultimately a decisive factor for a good performance of the model.

In this thesis, a different loss function will be explored, a cross entropy with custom gains, that was designed to have a similar behavior as the evaluation metrics used. Backed up by its previous success, reported by Zamani et al. [3], the new cost function is designed as
where \( Y^- \) are the labels of the negative documents \( D^- \), \( y^+ \) is the label of the positive document \( d^+ \), \( q \) represents the query and \( g \) is a gain function that is also applied in the DCG formula (equation 2.20),

\[
g(a, B) = \frac{2^a - 1}{\sum_{b \in B} 2^b - 1}.
\]

In addition to making the new loss function more proportional to the evaluation metric, this new formulation also allows the use of a custom cross entropy with all the true labels involved, which were not directly employed in the preceding version (equation 3.2).

### 3.3 Final proposed model

Figure 3.10 depicts an overview of the various proposed changes, merged in a single model. It is important to note some differences with the baseline model (Figure 3.1), that are not here presented for simplicity, such as the addition of dropout layers and the new loss function.

Although this new model aims to solve most of the issues previously discussed, there are other things that could have also be considered, such as developing methods to avoid cropping query and document into fixed dimensions, and other approaches that could be tested to solve the same problems. Nonetheless, we considered that the proposed changes here presented were worth trying and aimed to tackle different information bottlenecks existent in the baseline model.
Figure 3.10: Overview of the proposed model.
Chapter 4

Experiments

4.1 Description of the task

The experiments reported in this thesis were made on two years of an IR competition, the TREC Web Track from 2013 [62] and 2014 [63]. TREC stands for Text Retrieval Conference, which is one of the most important, and probably the most famous, multiple set of events and competitions supporting and promoting research in vast fields of Information Retrieval. Competitions are referred to as tracks.

As previously mentioned, this project focus on one of those tracks, the Web Track ad-hoc task. ClueWeb09-B [64] and ClueWeb12 [65] are the datasets used for our experiments, both of them used in this competition from 2009 to 2012 and 2013 to 2014, respectively. The former will only be used for training and validation, since access to the full ClueWeb09 dataset was not obtained, only to its category B, that contains less data than the original. The whole dataset is needed during test time because comparisons are made with other methods that reported results on it as well.

Both datasets, not publicly available, contain millions of HTML pages that were obtained by crawling extensively the World Wide Web. TREC used human annotators to label a subset of these collections. For each of the six years, on average, fifteen thousand judgments were used to evaluate the performance of retrieval systems. These judgments consisted of graded relevance scores between query-document pairs. Documents belong to the described datasets, while queries were provided by TREC, fifty per year to be more precise, totaling three hundred. This number of queries is particularly low to train a deep learning system, which influenced the way the model described in Chapter 3 was constructed. The query-document pairs were labeled in a six-point scale according to Table 4.1.

Every year, the objective of the task was to retrieve the top ten thousand documents, for each of the fifty queries, according to their relevance. This means that the candidates did not have access to the relevance judgments of that year before the conclusion of the event. However, the queries were partially released beforehand. Queries are composed of a main topic (the query itself) and a subtopic, that represents the intent of the query, as many can be multi-faceted, i.e., have more than one interpretation. For instance, one of the actual topics was ‘president of the united states’ and it has multiple subtopics (e.g., ‘Find the homepage of the President of the United States’ or ‘Find a list of past Presidents of the
Table 4.1: Relevance judgments used in the TREC Web Track.

<table>
<thead>
<tr>
<th>Relevance grade</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Junk - This page does not appear to be useful for any reasonable purpose</td>
</tr>
<tr>
<td>0</td>
<td>Non-relevant - The content of this page does not provide useful information on the topic, but may provide useful information on other topics, including other interpretations of the same query</td>
</tr>
<tr>
<td>1</td>
<td>Relevant - The content of this page provides some information on the topic, which may be minimal</td>
</tr>
<tr>
<td>2</td>
<td>HRelevant - The content of this page provides substantial information on the topic</td>
</tr>
<tr>
<td>3</td>
<td>Key - This page or site is dedicated to the topic, it is worthy of being a top result in a web search engine</td>
</tr>
<tr>
<td>4</td>
<td>Nav - This page represents a home page of an entity directly named by the query and the user may be searching for this specific page or site</td>
</tr>
</tbody>
</table>

United States'). During the tournament, topics were provided while subtopics were only known at test time.

Participants’ systems previously indexed ClueWeb documents, in order to retrieve them automatically, which were then evaluated using common metrics in Information Retrieval, like average nDCG (equation 2.21) and ERR (equation 2.22), using scripts provided by TREC [66] that contained probabilities \( p(q,d) \) (used to calculate ERR) proportional to the document's relevance judgment. One could say that the goal of this competition was to simulate a real search engine, being an authentic Web retrieval problem.

Nevertheless, as it was described in Chapter 3, the proposed relevance ranking model does not index all the documents of a corpora. Although it could be possible, using this system to retrieve texts from a collection is infeasible because it would be necessary to compute all the relevance scores for every possible query-document pair. Since the target is to rerank results, the deep learning model is evaluated in another way. As it was previously done in the area [50] [57], experiments of this work are mainly evaluated based on the nDCG@20 and ERR@20 score of reranks done over the TREC query likelihood baseline [66], which is a type of language model (Section 2.4.2). The baseline consists of no more than a file containing, per query, up to ten thousand documents ordered by their relevance, according to the query likelihood model. This type of evaluation fits the way the whole retrieval system is constructed (Figure 1.1), considering that the model can be used to rerank documents provided by a search engine. Moreover, by using metrics @20, only the top twenty documents for each query are evaluated, which means that the assessment hardly penalizes systems who are not able to distinguish relevance grades.
4.2 Implementation details

Throughout all the experiences that will be presented in Section 4.3 we resort to the use of dropout, except for the case when the advantages of using it are being evaluated. The same applies to the use of the query IDF vector, depicted in Figure 3.1, that will only be added before the feedforward layer in Section 4.3.1, where its use will be further discussed.

The choice of hyperparameters is also a crucial step that defines the structure of the model. For reimplementation purposes, Section 4.3.1, those are set according to the original used values by Hui et al. [50]. Some of the most important ones are here defined:

- Document and query lengths in the similarity matrix are set to 800 and 16, respectively, and the first words of them are kept when their dimension exceeds these values.
- The size of the two hidden layers used in the multilayer perceptron at the end of the architecture is 32 and 16.
- Both the $2 \times 2$ and $3 \times 3$ convolution layers are applied with 32 filters.
- k-max pooling is used with $k = 3$.

Some of these values were tuned on the validation set when implementing other proposed changes to the baseline model (Sections 4.3.2 and 4.3.3). For the remaining of this section, more technical details of the implementation will be given.

4.2.1 Framework

The experiments were ran in Python using built-in functionalities of Keras [67], a deep learning library built on top of Tensorflow [68]. Tensorflow is a Python library that allows to define, optimize, and evaluate mathematical expressions, especially ones with multi-dimensional arrays. Operations are built through symbolic expressions and chained in a computation graph, which allows to easily compute complex operations such as gradients, when back-propagating the error for calculating the derivative of the cost function.

In order to accelerate training, all the experiments were performed using an NVidia Titan X graphics processing unit (GPU) to enable faster matrix computations.

4.2.2 Input Pre-processing

As previously explained in Section 4.1, the data is in the form of strings for queries and HTML code for documents. Article extraction needs to be applied to the latter in order to retain only the needed text. This is a crucial step since information can be lost when performing it or parts of code can still remain and will probably affect the relevance matching algorithm negatively.

The Diffbot API [8] was used to convert these Webpages to text strings because of its great performance, formerly reported in Section 2.2, and its out-of-the-box usability with HTML requests.
Considering that the relevance matching model relies on similarities between words, it is important to remove words that might have matches in both query and document but that do not contain much meaning. Therefore, as it was done in the baseline model [50], stopword removal is put into use to every string. Moreover, punctuation is also discarded and the words are lowercased.

To deal with OOV words, the 300-dimensional Word2Vec CBOW vectors [39] are re-trained using Noise Contrastive Estimation with 5 negative samples and a context window size of 10. This step was accomplished by making use of the Python language, Gensim [69], that provides a Word2Vec implementation for learning new word vectors from text.

For each sample, after having both the query and document pre-processed texts, the similarity matrix \( \text{sim}_{l_q \times l_d} \) can then be computed as described in Section 3.1.1.

### 4.2.3 Weight initialization

An important factor of the optimization problem is the starting region in the parameter space, also called the weight initialization. If the weights are too small, then the variance of the input signal starts diminishing as it passes through each layer in the network. The input eventually drops to a really low value and can no longer be useful. On the other hand, if the weights are too large, then the variance of input data tends to rapidly increase with each passing layer, eventually becoming so large that it becomes useless. Proper weight initialization can be the difference between the network converging in a reasonable amount of time and the network loss function not going diminishing even after hundreds of iterations.

A recent paper by Glorot et al. [70] introduced a novel initialization method that guarantees the input signal is not attenuated or grows too large when it propagates along the network. Empirically, this initialization, also called Xavier initialization, performs better than randomly initializing the weights. Moreover, it also samples the layer’s weights in a way that the input variance is preserved and therefore the information flow along the network topology.

For an input \( X \) with \( N_{in} \) components and a linear neuron with random weights \( W \) that outputs \( Y \), we have

\[
Y = \sum_{i=0}^{N_{in}} X_i W_i \tag{4.1}
\]

Taking as an assumption that the input variables are uncorrelated, we can calculate the variance.

\[
\text{Var}(Y) = \text{Var}\left(\sum_{i=0}^{N_{in}} X_i W_i \right) = \sum_{i=0}^{N_{in}} \text{Var}(X_i W_i) \tag{4.2}
\]

If inputs and weights both have mean zero, it can be simplified.

\[
\text{Var}(X_i W_i) = \text{Var}(X_i) \text{Var}(W_i) \tag{4.3}
\]

Then if we make a further assumption that the \( X_i \) and \( W_i \) are all independent and identically distributed, the variance of \( Y \) can be finally obtained.
\begin{align}
Var(Y) &= \sum_{i=0}^{N_{in}} Var(X_i)Var(W_i) = N_{in}Var(X_i)Var(W_i)
\end{align}

(4.4)

The input variance, \( Var(X_i) \), is preserved in the output if the weights have a variance, \( Var(W_i) \) given by \( \frac{1}{N_{in}} \). The proposed sampling distribution is a Gaussian with zero mean and a variance given by \( \frac{2}{N_{in}+N_{out}} \) which corresponds to the geometric mean between preserving the forward, \( \frac{1}{N_{in}} \), and back propagation, \( \frac{1}{N_{out}} \), pass variance.

### 4.2.4 Masking CNNs

Zero padding the similarity matrices \( sim_{l_q \times l_d} \) solves one problem, but creates another one. With the introduction of new rows and/or columns, the CNNs take these extra zeros into consideration because, in spite of the filter weights being multiplied by these zeros (and therefore not backpropagating any gradient), the biases will be updated to non-zero values and influence the error propagation. This unnecessary information may be actually detrimental to the model’s learning process.

To overcome this problem, there is a technique called masking, which is commonly used within the machine learning field, specially with Recurrent Neural Networks. Masking consists of passing an extra input, called a mask. The mask can be thought of as a binary matrix with the same shape as \( sim_{l_q \times l_d} \). This input contains information of what was zero padded and what was not. After applying the convolution layers, their output is multiplied with the mask, so that the non-zero values that might appear at padded positions do not contribute to the gradient update or to posterior phases.

Due to reasons previously pointed, masking is always applied in our experiments, as a way to prevent the similarity matrix’s entries (after the convolutions), that were previously padded, of not getting high values that might allow them to get pooled by the k-max pooling layer. Moreover, when k-max is not used, these positions that are non-zero (but should be) might have a detrimental effect on the model’s ability to learn matching patterns.

### 4.2.5 Gradient norm clipping

When optimizing strongly non-linear functions there is the problem of abrupt and large changes in derivatives that can lead to both exploding and vanishing gradients, previously referred in Section 2.6.3. In the baseline architecture, vanishing gradients are avoided by not using sigmoid or hyperbolic tangents as activation functions but ReLUs instead.

Nonetheless, large gradients are more prone to occur and can have the effect of drastically changing the parameters and hindering the convergence. A common technique for dealing with this problem and which can greatly improve convergence rate is the gradient clipping method.

The gradient clipping method deals with large gradients by setting an upper bound on the gradient norm and scaling the gradient norm down to the threshold if it surpasses this limit. Empirically, this has the effect of avoiding large parameter changes which might hinder convergence to the optima in highly non-linear parts due to exploding gradients.
4.3 Objective Evaluation

In this section, we evaluate the different architectures and compare them using both the mean ERR@20 and nDCG@20. Both the novelties proposed in Chapter 3, the baseline model reimplementation and its original reported results will be presented. It is important to remember that all these results were obtained by reranking the query-document pairs existent in the query likelihood submission. Therefore, Table 4.2 presents the results of both this model and the PACRR baseline model [50] developed by Hui et al., where TrW represents the number of trainable weights. This number will aid the analysis of the different architectures, as we will see that it is a crucial factor. TrW does not account for biases (equation 2.13) for a matter of simplicity. In this dissertation, comparisons of different model architectures are made with the results reported in 4.2, keeping in mind that the original PACRR model [50] reported improvements over numerous anteceding models that are not in here presented.

It is is also worth noting, from Table 4.2, that the results on 2013 and 2014 are really different which leads us to think that the labeled samples relative to each year have a different distribution and, therefore, improvements on the validation set may not be reflected in the test set, as different years were used for them.

<table>
<thead>
<tr>
<th>Model</th>
<th>TrW</th>
<th>2013</th>
<th>2014</th>
<th>mean</th>
<th>2013</th>
<th>2014</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Likelihood</td>
<td>-</td>
<td>0.101</td>
<td>0.131</td>
<td>0.116</td>
<td>0.190</td>
<td>0.231</td>
<td>0.211</td>
</tr>
<tr>
<td>PACRR [50]</td>
<td>6061</td>
<td>0.166</td>
<td>0.221</td>
<td>0.194</td>
<td>0.295</td>
<td>0.339</td>
<td>0.317</td>
</tr>
</tbody>
</table>

Table 4.2: ERR@20 and nDCG@20 values comparison between PACRR [50] and Query Likelihood.

Throughout this section, we will keep referring the baseline model as PACRR and its reimplementation as RPACRR. The authors of PACRR [50], Hui et al., also kindly open-sourced their own similarity matrices [71] which will also be used whenever it is possible. That is, every model that takes as input solely the similarity matrices, will be separately trained with both the data previously described in Section 4.1 (RPACRR-SM) and with these original matrices (RPACRR-OM). A key factor to retain is that the total amount of original matrices is around 120000 while the ones self-constructed in this work are more or less 75000, which is notably lower. Moreover, the matrices are just the result of cosine similarity between query and document terms, which means that we do not have access to the raw text that the authors used and do not know the exact preprocessing applied, that might differ from the one used in this thesis, described in Section 4.2.2.

Judgments with a relevance grade of -2 are removed, as in the original implementation [50]. This is because the documents labeled that way are mostly spam and, in a real retrieval system, spam detection is formerly applied and treated as a different problem.

To construct each training sample, 1 positive document $d^+$ is chosen from a relevance group with probability proportional to the number of documents in that group, within the training set. 6 other documents $D^-$ are chosen from a relevance group with an inferior grade to the chosen document.

Using a batch size of 32, models are trained and tested in a cross validation round-robin manner, using individual years as training, validation, and test data. Specifically, test results are reported on one
year by using combinations of the other five, so that four are used for training and the remaining for validation. The score for each year is therefore obtained by running 5 models, each for 50 epochs, and averaging their test predictions at the end. In practice, this means that the results become more robust and stable.

For each training/validation combination, the epoch on which the model performed better is chosen by maximizing the ERR@20 on the validation set. This metric emphasizes the quality of the top-ranked documents and heavily penalizes relevant documents that are ranked lower by a model when enough relevant documents have been observed earlier [52]. This means that the improvement of the ERR for a model mainly comes from improvements on queries for which search results at the top are not good enough from an initial ranker.

4.3.1 Baseline model reimplementation

Reimplementing our baseline model, PACRR [50], is the first step to confirm our experiments’ validity. Since we have access to the similarity matrices used in [50], the self-made implementation is run with two kinds of data, as previously mentioned. Results of both these runs are summarized in Table 4.3. RPACRR-OM refers to the reimplementation with original matrices and the REPACRR-SM with self-made matrices.

<table>
<thead>
<tr>
<th>Model</th>
<th>TrW</th>
<th>2013</th>
<th>2014</th>
<th>mean</th>
<th>2013</th>
<th>2014</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPACRR-OM</td>
<td>5552</td>
<td>0.163</td>
<td>0.214</td>
<td>0.189</td>
<td>0.451</td>
<td>0.436</td>
<td>0.444</td>
</tr>
<tr>
<td>RPACRR-SM</td>
<td>5552</td>
<td>0.161</td>
<td>0.199</td>
<td>0.180</td>
<td>0.437</td>
<td>0.415</td>
<td>0.426</td>
</tr>
</tbody>
</table>

Table 4.3: ERR@20 and nDCG@20 values comparison between RPACRR trained on the self-made or original similarity matrices.

By comparing these results with the ones from Table 4.2, it is visible that PACRR was successfully reimplemented, if looking solely at the ERR@20 values, that are really similar. However, nDCG@20 scores were higher in our reimplementation. To explain this difference, one has to go back to the metric definitions, equations 2.20 and 2.22. We can see that nDCG penalizes relevant documents that appear after less relevant ones by lowering their total contribution with a denominator of \( \log_2(k+1) \). nDCG only discounts based on the rank of the document, but does not consider any of the documents previously seen at higher ranks. On the other hand, ERR implicitly discounts documents based on the relevance of previously seen documents and adds to the metric score at most \( 1/k \), which is a stronger attenuation than before. In fact, as Chapelle et al. [52] claim, ERR differs from other metrics because it heavily discounts the contributions of documents that appear after highly relevant documents, i.e., the ones that appear at lower ranks.

This leads us to conclude that the differences in nDCG@20 metrics, from the original and the reimplementation versions of the model, mean that RPACRR is actually better at getting the top 20 relevant documents but that the order of which it organizes them is mostly the same. That explains the fact that nDCG@20 is higher, because there are more relevant documents, and ERR@20 is similar, since the
first documents are the ones that contribute the most to this metric.

We believe this increase in the nDCG@20 score is due to some careful implementation details, previously described in Section 4.2, and mostly to the added regularization, as it will be presented in Section 4.3.1.

Additionally, a slight decrease in performance when using the self-made matrices can also be noted. This phenomenon can be easily explained because of different data preprocessing and specially due to the fact that the original similarity matrices consist of way more training data than the self-made ones.

Effect of regularization

Probably the most important change in this reimplementation of PACRR [50], the use of dropout as a regularizer for the model will be analyzed in this section.

Reproduction of PACRR [50] results showed instability from what was reported in the original work [50]. Since the neural architecture had no kind of regularization, an experiment is in here conducted to assert that adding regularization to the network is in fact preventing the model from overfitting, and therefore stabilizing the results. Overfitting is often visualized by comparing validation and training loss and noting that, after a certain point, the latter continues to decrease while the former starts increasing.

However, validation loss is not calculated in this model. Section 3.1.4 detailed how the loss is computed during train in a listwise way, comparing a set of documents to a query, while single query-document pairs are scored during validation and test time. Therefore, overfitting is in here examined with another approach: train loss and ERR@20 are measured while training a model with and without dropout.

Figure 4.1 shows us these results. In here, both architectures were trained in a different way from what was described earlier. Both models were run for 50 epochs only, with all the data available from ClueWeb09-B for training and judgments from 2013 (ClueWeb12) for validation (both self-made matrices). No cross validation is employed since that method has an intrinsic regularization effect and the objective of this experiment was to capture how a single trained model performs without the need of it.

For the model with dropout, it is possible to inspect that the loss stabilizes around 1.8 and the ERR@20 does not change much either, having its peak in epoch 30. When comparing with the model without dropout we can notice that while the train loss keeps decreasing, the model’s ERR@20 on validation does not. Moreover, it is clear that the model without dropout does not achieve as good scores on the validation set as when regularized.

Both these factors are indicators that the model is overfitting when not using dropout, i.e., is adapting its parameters in order to get good results on the train set and not generalizing well enough. Hence, dropout is used in all out other experiments.

Query IDF usage

In the original implementation of PACRR, Hui et al. [50] argue that, prior to the Multilayer Perceptron, concatenating the query IDF vector with the k-max pooled values (Figure 3.1) adds a normalizing effect
to the model, given more or less important to certain query terms.

Nonetheless, that phenomenon is explored in this section. Every value of the query IDF vector is associated with a group of k-max pooled values that belonged to a specific matrix row, meaning that they are all related to a certain query term. Since all the values (k-max pooled and query IDF) are flattened and passed to an MLP (Figure 3.1), the network needs to learn this association during training as the information of the query term that each value is related with is lost during the flattening process.

We believe that the network would need to be trained with a lot of data to be able to perform this association properly. Moreover, the assumption that it acts as a query term normalizer might be wrong, leading the network to interpret this information in an adverse way. Therefore, we conduct an experiment, presented on Table 4.4 where we compare the performance of the model with an without adding the query IDF vector, using once again the original matrices (RPACRR-OM) and the self-made ones (RPACRR-SM).
Table 4.4: ERR@20 and nDCG@20 values comparison between RPACRR with and without query IDF vector.

After examining that table, it is notable that the performance of the model is consistent when the query IDF is used and when it is not. The small changes in ERR@20 and nDCG@20 are not noteworthy as each model run tends to vary slightly in score, due to random weight initialization and shuffling of the data. These results are indicative that, in fact, the use of query IDF is not beneficial to the network’s performance, at least with the amount of training data that is available.

Using this extra vector also adds more parameters to the network, since more values are passed to the first layer of the MLP. For both reasons presented above, the query IDF was not used during the rest of our experiments.

**Custom loss function**

Another characteristic that we wanted to analyze, using the original architecture of PACRR [50], was how the loss function was constructed. As formerly explained in Section 3.2.4, binary cross entropy is used as a surrogate cost function for the actual evaluation metrics, ERR@20 and nDCG@20.

From Table 4.5, it is possible to observe comparisons of using both the custom loss function (CLF) and binary cross entropy (BCE). Both methods were run with the two different types of data previously mentioned.

Table 4.5: ERR@20 and nDCG@20 values comparison between RPACRR with binary cross entropy (BCE) loss and a custom loss function (CLF).

As expected theoretically, making the loss function closer to the metric that is being employed (nDCG in this case) increases the model’s performance. The ERR@20 score also increases, manifesting that applying a custom cross entropy loss with a gain function similar to nDCG is definitely an improvement over the baseline.
4.3.2 Matrices dimensionality reduction

The baseline model depicted in Figure 3.1 applied strong pooling techniques, max and k-max pooling, in order to reduce either the dimensionality of 3D or 2D matrices, respectively. It does this as a simple, parameterless way of having an input vector that is manageable by the Multilayer Perceptron, given the amount of training data, while trying to retain relevant query-document matching patterns.

Nonetheless, in Chapter 3 it was argued that parameterless pooling techniques may be too aggressive and lose important data that is never seen by the model again. Therefore, in this chapter we analyze two different ways of reducing dimensionality, giving some freedom to the model to learn what features to keep. Those methods are the previously described $1 \times 1$ convolution (Section 3.2.2), replacing the need for max pooling, and the row-wise soft attention mechanism (Section 3.2.2), that takes the place of the k-max pooling layers. Results of model runs with these methods are presented in Table 4.6, 1x1conv and rSAtt, respectively. The latter was only run with the self-made matrices as it requires access to the query and document embedding vectors (see equation 3.3), that were not provided by the authors of PACRR [50] [71].

<table>
<thead>
<tr>
<th>Model</th>
<th>TrW</th>
<th>ERR@20 2013</th>
<th>ERR@20 2014</th>
<th>ERR@20 mean</th>
<th>nDCG@20 2013</th>
<th>nDCG@20 2014</th>
<th>nDCG@20 mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPACRR-OM + 1x1conv</td>
<td>5616</td>
<td>0.167</td>
<td>0.223</td>
<td>0.195</td>
<td>0.450</td>
<td>0.433</td>
<td>0.442</td>
</tr>
<tr>
<td>RPACRR-OM</td>
<td>5552</td>
<td>0.163</td>
<td>0.214</td>
<td>0.189</td>
<td>0.445</td>
<td>0.422</td>
<td>0.434</td>
</tr>
<tr>
<td>RPACRR-SM + 1x1conv</td>
<td>5616</td>
<td>0.165</td>
<td>0.207</td>
<td>0.186</td>
<td>0.441</td>
<td>0.420</td>
<td>0.431</td>
</tr>
<tr>
<td>RPACRR-SM + rSAtt</td>
<td>21680</td>
<td>0.110</td>
<td>0.139</td>
<td>0.125</td>
<td>0.205</td>
<td>0.240</td>
<td>0.223</td>
</tr>
<tr>
<td>RPACRR-SM</td>
<td>5552</td>
<td>0.161</td>
<td>0.199</td>
<td>0.180</td>
<td>0.437</td>
<td>0.426</td>
<td>0.126</td>
</tr>
</tbody>
</table>

Table 4.6: ERR@20 and nDCG@20 values comparison between RPACRR methods for dimensionality reduction.

Without using a lot of extra trainable parameters, it is possible to observe that $1 \times 1$ convolutions enhance the model’s performance, as expected, since the model learns how to reduce third dimensions (channels) without applying an aggressive maximum operation. This method is only used after the $2 \times 2$ and $3 \times 3$ convolutions, that were applied with 32 filters, making each $1 \times 1$ convolution to use only one filter with a length of 32 also, used to condense the information of multiple channels into a single value.

In order to get similar results to the Query Likelihood baseline (Table 4.2), a lot of hyperparameter tuning was needed for the row-wise attention mechanism. Replacing the k-max pooling layer was probably the most challenging task of this thesis, since k-max uses no trainable parameters, with the assumption that only the top $k$ values of each row of the matrices are required for the subsequent layers. We acknowledge that this assumption done in PACRR [50] is well conceived, due to the existing difficulty of reducing each matrix row, that has a total of $l_d$ values, that was set to 800.

From our experiments, the row-wise attention mechanism, presented in Table 4.6 as rSAtt, was the non-parameterless method that best worked for replacing k-max pooling. Nevertheless, it is possible to see that its results are similar to the Query Likelihood model. It can also be seen that the number of trainable parameters increases immensely when the k-max pooling layer is replaced, which might be indicative that a lot more train data is required to learn a proper dimensionality reduction with this
row-wise attention mechanism.

4.3.3 Changing the input

As previously discussed in Section 3.2.3, having a single static cosine similarity matrix is a clear information bottleneck of the whole architecture. In this section, methods that either try to provide more additional information to the model (1D convolutions) or improve the way that similarity matrix is constructed (soft attention mechanism) are analyzed.

Table 4.7 shows the results of those experiments, with the evaluated ERR@20 and nDCG@20 metrics, of both methods: 1D convolutions (1Dconv) and the soft attention mechanism for constructing the 3D similarity matrix (3DSAtt). Different network architectures were only trained using the self-made matrices (RPACRR-SM) since we did not have access to the original embeddings used in baseline model [50] [71]. It is important to note that the 3DSAtt method did not actually use 2D similarity matrices but constructed new three-dimensional ones. However, the same nomenclature, RPACRR-SM, is used as a way of referring that models were trained with the same data.

<table>
<thead>
<tr>
<th>Model</th>
<th>TrW</th>
<th>ERR@20 2013</th>
<th>ERR@20 2014</th>
<th>ERR@20 mean</th>
<th>nDCG@20 2013</th>
<th>nDCG@20 2014</th>
<th>nDCG@20 mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPACRR-SM + 1Dconv</td>
<td>16688</td>
<td>0.142</td>
<td>0.175</td>
<td>0.159</td>
<td>0.252</td>
<td>0.293</td>
<td>0.273</td>
</tr>
<tr>
<td>RPACRR-SM + 3DSAtt</td>
<td>19456</td>
<td>0.127</td>
<td>0.159</td>
<td>0.143</td>
<td>0.220</td>
<td>0.256</td>
<td>0.238</td>
</tr>
<tr>
<td>RPACRR-SM</td>
<td>5552</td>
<td>0.161</td>
<td>0.199</td>
<td>0.180</td>
<td>0.437</td>
<td>0.426</td>
<td>0.126</td>
</tr>
</tbody>
</table>

Table 4.7: ERR@20 and nDCG@20 values comparison between RPACRR methods for changing the model’s input.

As in Section 4.3.2, the curse of the amount of learnable weights (TrW) is also present in here. In both methods, it is possible to note a decrease of the model’s performance as this number of trainable parameters increases. Even though convolutions are known for not having a lot of learnable units, this 1D convolution has filters that act over embedding vectors with a dimension of 300, that increases the number of parameters drastically, even when using a kernel size of 2, when a reasonable number of these filters are applied.

4.3.4 Best performing model

Having the previous experiments concluded, we also want to test a final model, that basically consists of an aggregation of the changes that yielded better results than the baseline model, i.e., superior ERR@20 and nDCG@20 scores than the ones reported for PACRR [50] in Table 4.2. Therefore, this final model (Figure 4.2) is constructed with small modifications over the one depicted in Figure 3.1, those being:

- Replacement of the max pooling layers by convolutions with $1 \times 1$ kernels.
- Introduction of dropout layers after non-linear activations.
- Removal of the query IDF vector as an input to the network.
- Replacement of the binary cross entropy loss by a custom function (equation 3.5).
- Other implementation details previously mentioned in Section 4.2.

Figure 4.2: Simplified final model.

<table>
<thead>
<tr>
<th>Model</th>
<th>TrW</th>
<th>ERR@20 2013</th>
<th>nDCG@20 2013</th>
<th>ERR@20 2014</th>
<th>nDCG@20 2014</th>
<th>ERR@20 mean</th>
<th>nDCG@20 mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACRR [50]</td>
<td>6061</td>
<td>0.166</td>
<td>0.221</td>
<td>0.194</td>
<td>0.295</td>
<td>0.339</td>
<td>0.317</td>
</tr>
<tr>
<td>RPACRRF-OM</td>
<td>5616</td>
<td>0.178</td>
<td>0.228</td>
<td>0.203</td>
<td>0.489</td>
<td>0.466</td>
<td>0.478</td>
</tr>
<tr>
<td>RPACRRF-SM</td>
<td>5616</td>
<td>0.173</td>
<td>0.215</td>
<td>0.194</td>
<td>0.460</td>
<td>0.435</td>
<td>0.448</td>
</tr>
</tbody>
</table>

Table 4.8: ERR@20 and nDCG@20 values comparison between RPACRRF and PACRR.

Table 4.8 shows the results for the final model, RPACRRF, comparing it with the original reported results by Hui et al. [50]. Once again, since it is possible, we show the difference of using the original matrices (OM) or the self-made matrices (SM). As expected, RPACRRF achieved better results than any antecedent model, according to the improvements previously obtained and detailed in Sections 4.3.1 and 4.3.2. Nonetheless, a second version of the baseline model constructed by Hui et al. [57], still shows better results in terms of ERR@20. According to the results reported by Hui et al. [50], the final architecture also shows gains in performance over other models, such as the work developed by Guo et al. [43], Pang et al. [47] and Xiong et al. [49]. It is also interesting to compare the results from Table 4.8 with the ones reported on the competition itself, during 2013 [62] and 2014 [63], although the methods for retrieving the top documents were different: the model developed in this thesis is reranking a prior selection of top documents while the models used during the competition can be viewed like
search engines (as illustrated in Figure 1.1). In terms of ERR@20, RPACRR-OM would be classified in second place for both years, by solely reranking the Query Likelihood submission [66].

The final model can be described as an extraction-distillation-combination sequence, with CNN kernels extracting relevance matches, pooling layers and $1 \times 1$ convolutions distilling the matches into a series of small vectors for each query term, and a final block combining the query term signals into an ultimate relevance score for each query-document pair. Following this framework, we attempt to better understand the functionality of the convolution layers by visualizing their outputs. Figure 4.4 displays markups for different kernels sizes on the same document snippet, showing the strongest signal among all query terms, i.e., choosing the highest value for each column from the 2D matrix input to the different k-max pooling layers. The higher the opacity is (i.e., darkness of the text), the higher the output value is, in comparison with the other values from the same output matrix. Note that the presented document text was preprocessed as described in Section 4.2.2. The query that is being analyzed can be viewed in Figure 4.3 and the document example was chosen randomly from a pool of highly relevant documents, i.e., with a relevance grade of 4 (Table 4.1).

**Topic:** symptoms of heart attack

**Subtopic:** What are the symptoms of a heart attack in both men and women?

Figure 4.3: Query example, from the 2014 TREC Web Track [63].

The use of real valued cosine similarity in the input matrices allows the model to match related terms beyond exact matches, thereby expanding the query. For example, in Figure 4.4(a), terms such as "coronary", "cardiac" and "health" have relatively high weights though they do not appear in the query. Looking at the inputs that were passed through CNNs, Figure 4.4(b) and 4.4(c), we can see that most unigram signals still keep their relatively high weights and that other bigram and trigram matches increase the weights of other terms, with word combinations such as ‘muscle strain’ or ‘coronary artery blockage’. For the higher dimension kernel signals, it is possible to notice that almost all terms have at
least some weight, reducing the difference between the salient text and the remaining text. This is due to the way CNN kernels work when combined with real valued similarity. Taking the dot product of all terms in a window generally produces non-zero values and acts as a smoothing effect.

4.4 Discussion

During the initial stage of these experiments we have successfully reproduced the baseline model and proposed modifications that proved to be advantageous and improved the model’s performance in terms of ERR@20 and nDCG@20 scores. Nevertheless, it is important to point out that despite these deep learning systems showing good efficiency in reranking search results, they are limited by the retrieval system that selected the top documents for each query in the first place, which was the Query Likelihood model [66] in our experiments and also on the original baseline model (Table 4.2).

We have shown how regularization can be used to achieve better and more stable results as well as the importance of adapting the training scheme, approximating it more to the way these systems are evaluated. We consider these key improvements over the baseline and that more work can be focused on this direction, as we will further discuss in Chapter 5.

Letting the model learn how to perform pooling operations can also be a beneficial factor, to some extent. I.e., we have shown how a slight change of the max pooling layer, without adding too many additional trainable parameters, results in a performance increase. On the other hand, alternative methods for replacing the k-max pooling did not work so well due to the amount of newly introduced variables.

It is also worth noting that the best performing model, depicted in Figure 4.2, does not have a notable computational overhead at test time when compared to the baseline model, since it ends up switching a maximum operation by a convolution but balances by having less input weights to the Multilayer Perceptron.

In spite of achieving significant gains in terms of ERR@20 and nDCG@20, when compared to other models tested on the same dataset [50], the applicability of this final system in real life is still questionable. This is mostly due to information bottlenecks, some previously mentioned in Section 3.2, that are still inherent to the model depicted in Figure 4.2.

**Topic:** cheap internet  
**Subtopic:** I'm looking for cheap (i.e. low-cost) internet service.

Figure 4.5: Query example, from the 2009 TREC Web Track [72].

Figure 4.6: Examples of similar documents with different relevance grades.
To exemplify some of these limitations in an intuitive way, we show in Figure 4.6 how two really similar texts can have different relevance grades for the same query, depicted in Figure 4.5. Both texts represented in Figure 4.6 are the output of the article extractor, also used in the previous experiments, that extracted those texts from HTML code. Assuming that the dataset was well labeled, we can see in this example why the final system fails to distinguish both documents. The information present in the texts is not even enough for a human to successfully assign different relevance grades to them. Therefore, the issue in this example is that either the article extractor was not efficient at obtaining the text from the HTML pages associated with both documents, or that the information present in the raw body text was simply not sufficient to differentiate those documents. This dissertation, as other work done in this field of study, considered that the Web page document could be represented by its raw body text. In the case of the example depicted in Figure 4.6, this assumption is wrong, the reason being that important information was contained in a table embedded in the HTML code, that the article extractor did not retrieve because it does not consist of article text. Parts of the table present in both documents can be viewed in Figure 4.7. We can conclude from this analysis of the example that the article extractor was in fact successful at retrieving the body text, although that was a limited representation of the Web page and that more information should be considered, since the relevance of a document can also be associated with other elements of it, for instance, tables or images.

Moreover, from Figure 4.6, we can imagine how the similarity matrices could practically contain the same signals, even for distinct texts, making them also a limited representation of query-document interactions, even though that was not the information bottleneck illustrated in this example.

Influenced by their recent success in Natural Language Processing [41], soft attention mechanisms have been employed extensively in this thesis. However, these mechanisms require a lot of data to train properly, which was not the case in here. Lack of training data was definitely a hard obstacle to overcome in this project, which conditioned how PACRR [50] was designed in the first place. Having only available 200 queries associated with the training set was definitely not enough to learn more complicated associations between query and documents that could then be generalized in the test set. Nonetheless, this was not clear at the beginning of this project, and it was only during our experiments that it became more evident that this was a transfer learning approach that constructs a soft matching function over fixed pre-trained word representations, which means that the pre-trained model plays a fundamental role. Besides this deficiency of training data, the poor performance of these methods (as well as the 1D convolutions used in Section 4.3.3) can be caused by a variety of other factors normally present in neural networks, that were described in Section 2.6.3. In addition, there is no report of use of these type of mechanisms in the area of ad-hoc retrieval, which increased the motivation for our investigation. Although not performing as good as other architectures used in the first experiments, these models are still able to get a score increase over the Query Likelihood submission that they are reranking. This leads us to think that more work about the use of these methods still has to be explored and that they can eventually produce notable performance gains.

There are a few more techniques that could be experimented with our model. However, due to the limited time a Master thesis imposes, this research has to be left as future work. With this in mind,
Our experiments do not provide enough evidence to discard any presented method just yet, and rather provide extra motivation to explore possible future solutions to solve the problem of reranking search results in ad-hoc retrieval.
Chapter 5

Conclusions

5.1 Achievements

The objective of this thesis was to contribute to the development of a reranking system for ad-hoc retrieval with deep learning techniques that eliminate the requirement for handcrafted query and document features.

We started by exploring previous work, selecting the state of the art model on the dataset available, and implemented it. The baseline algorithm consists of multiple convolution layers applied over a cosine similarity matrix with the intent of catching different query-document matching patterns that can be further passed to a Multilayer Perceptron that finally produces a single relevance score.

By analyzing this network topology, it was evident that boosting the model’s performance would be a challenging task since the architecture had strong assumptions and the stochastic gradient descent procedure was mostly used to learn the parameters within the MLP, since the number of CNN parameters was sensibly smaller in comparison. Moreover, the model consisted mostly of a transfer learning approach since the Word2Vec embeddings were not retrained (along with the rest of the model) due to the reduced amount of labeled query-document pairs. New building blocks were then designed and optimized, always keeping in mind that optimization over the baseline model might be stuck in a particular local minima and that even small changes would require a lot of work to achieve similar results.

As is the case of most deep learning projects, there is a lot of trial and error associated with this thesis. There is plenty of work done in background that was not included in this document since these ideas did not yield better results than the Query Likelihood baseline, mostly because there was simply not enough data to train some kind of architectures, as is the case of Recurrent Neural Networks. Lack of training data was definitely an issue throughout this project as it was pointed out in Section 4.4, specially due to the reduced number of available queries that made it only possible to learn shallow query-document matching patterns, captured by CNNs. Recent work done on ad-hoc retrieval use bigger datasets, labeled automatically using clickthrough data from commercial search engines [3] [48] [73]. Despite seeking extensively, no access to this type of data was found.

There are still some limitations with the final enhanced baseline model (Figure 4.2), as discussed in
Section 4.4, most of them related with how the input to the system is represented. Besides not being sufficient information about the document in some cases, the body text is always extracted from HTML code, which is not a process with guaranteed success due to the different ways a Web page can be structured. It is safe to assume that the model would achieve a better performance if the dataset was labeled using only this extracted body text, although that would not reproduce properly the use of the system in a real world scenario. This extracted body text of the document is then represented as a matrix of cosine similarities between query and document words, using retrained Word2Vec embeddings, which we consider a limited representation since semantic information is lost during this transformation process. For that reason, we tried different alternatives to combat this information bottleneck of the system, that yielded no improvements over the baseline model though.

Nonetheless, this thesis managed to achieve good results in the reimplementation of a contemporary relevance matching model, on a known dataset, in the area of Information Retrieval. Moreover, it also shows some improvements by applying some techniques as regularization, $1 \times 1$ convolutions or tweaks in the loss function, as well as some proposed methods on how the model could potentially be adapted to retain or learn more information to ultimately produce a query-document relevance score.

We expect that the work presented in this document will help future researchers in the areas of both Information Retrieval and Natural Language Processing. To aid scientific reproducibility, the full code base associated with this thesis is available at [74] for future use. This source code is the sole authorship of the author of this thesis.

5.2 Future Work

There are a few possible future work directions this thesis can take. Taking into account the further development of the baseline model we studied, the following suggestions can be considered:

- Explore different ways of redefining the training scheme for reranking models, with the intention of diminishing the differences between the loss function and the evaluation metric.

- Associate the interaction focused model examined in this thesis with a representation focused, as done previously by Mitra et al. [48]. Recurrent neural networks could be used in this model to learn a semantic representation of both query and document, separately, that would afterwards help producing the final relevance score. Architectures similar to Hierarchical attention networks [75] could be explored to represent long document texts.

- Explore the use of recent contextualized word embeddings (e.g., ELMo [76] and CoVe [77]) in the analyzed architecture and compare its performance with the used Word2Vec word representations.

- Combine multiple document fields, instead of only considering document body text as input to the system, as done by Zamani et al. [3], along with other metrics that characterize a particular website or its domain, such as PageRank [4].
Bibliography


